TpaNestingOEM

TpaNestingOEM Library

User Manual

V. 3.0.0

29/09/2023

Tecnologie e Prodotti per l'Automazione

Indice

1 Introduction

TpaNestingOEM.dll (alias TPA_N) is a product designed for software developers and can be integrated as a library for 32 and 64-bit Windows architectures.
TPA_N is a library for the automatic nesting of complex 2D shapes and allows the development of optimizer applications for 2D placements or shape cuts in disparate application fields.

The library must be integrated as component of your product.

A library demo app is available on the web site www.tpaspa.com/oem-nesting-library. It should however be noted that this application does not display all the available TPA_N functions.

1.1 Versions

Version	*Release date*	*Comments*
1.0.0	14.04.2021	
1.5.0	11-01-2022	Method *Compute*: modified prototype Property added: *Version* Method added: *SaveSolutionDXF*
1.8.0	14-06-2022	Added properties: *TimeCompute*, *TimerProgres*
1.9.0	29-09-2022	Product maintenance changes
3.0.0	20-11-2023	Overall product review

1.2 License

To operate, TPA_N requires a hardware key verification.
The key may assign optimization authorizations on two levels: *Square* and *True Shape*.
Additionally, the key can assign specific authorizations for the accessory functionality that imports geometries from DXF or DWG files:

- however, this functionality is only available with *True Shape* optimization enabled.
- accessory authorization is *DWG* format import.

Unless otherwise stated, what written below is to be intended as operating with full authorization.

1.3 Ownership and copyright

For further information, please contact:

T.P.A. Srl Tecnologie e Prodotti per l'Automazione - Via Carducci, 221 - 20099 Sesto S. Giovanni (MI) - ITALY
Phone: +393666507029
e-mail: info@tpaspa.it

or visit our web site:

www.tpaspa.com

2 User guide

TPA_N assigns the ***Nesting*** class in the ***TpaNestingOEM*** namespace.

TPA_N calculates the *shapes* placements in rectangular or irregular positioning areas.
A *shape* is a polygonal geometry:

- the simplest shape is defined as a rectangle
- a shape may define an external geometry and one or more internal geometries (islands).

A *part* is a single entity that can result from a placement in an area.
The placement areas are called *sheets*.

A *part* has a geometry, generally assigned as *shape*. The shape assigned to a *part* has only one external path and can have islands: internal scrap areas, that can possibly be used to position other smaller *parts*.
A *sheet* assigns a sheet of material, where the parts are positioned. The sheets have a geometry, as well, which is generally assigned as *shape*: generally speaking, *sheets* are simple rectangles, but it is possible to assign irregular shapes and possible inner constraint areas that cannot be used to place parts.

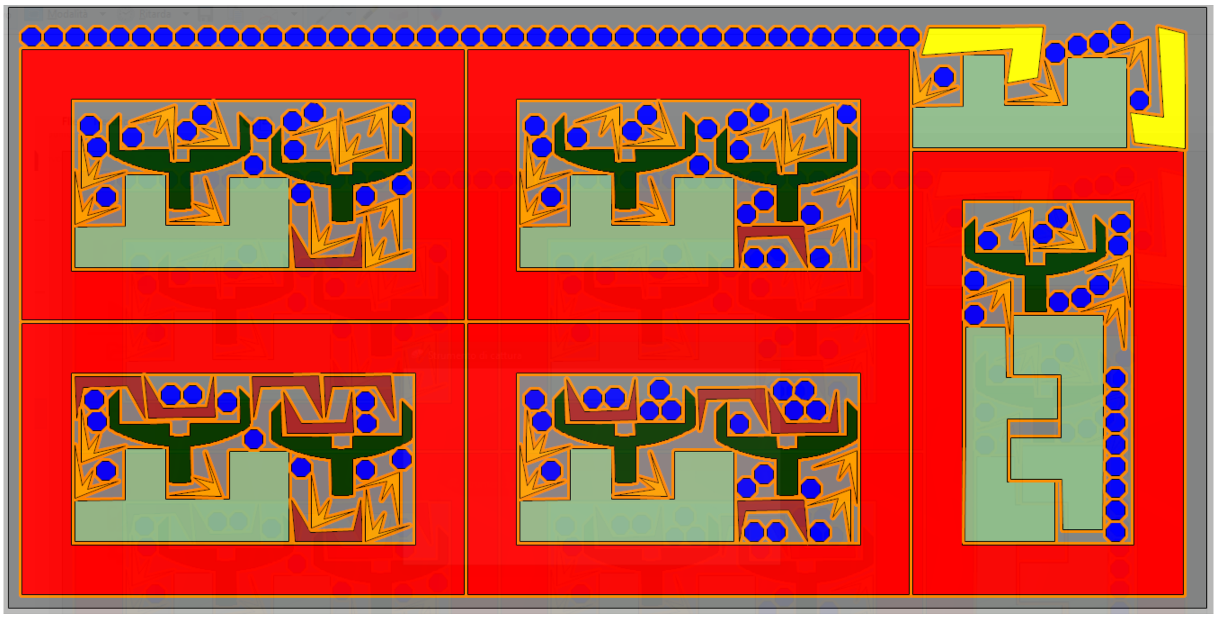
The assignment of *parts* allows defining transforms of the basic geometry, like rotation and/or mirroring, as well as generic (available quantity, positioning priority) or generally technological assignments (thickness, cutter diameter, material, colour, grain, …). For each type of part, the required quantity to be placed and a possible maximum quantity that can be placed are defined to fill the sheets already used.

The assignment of *sheets* allows defining generally technological settings (thickness, material, colour, grain, …).
The available quantity is defined for each type of sheet.

The generally technological data allows applying matching conditions and/or filters between *parts* and *sheets* to the nesting procedure.

The *result* of the nesting optimization assigns the position and rotation of all the *parts* that are placed on the *available sheets*. A nesting *result* can assign one or more *sheets*: each *sheet* contains the placement of at least one *part*.

The figure shows a rectangular sheet obtained with *True Shape* optimization.



2.1 Nesting optimization typologies

The totality of calculation procedures that bring to a nesting solution is called *optimization phase*.
TPA_N manages two optimization modes: *Square* and *True Shape*.

Square optimization

- It manages the placement of rectangular *parts* in rectangular *sheets*.
- the placement of one part can apply 0° or 90° rotations (counter-clockwise) and mirror transforms
- the result of the optimization is unique.

If all the *parts* and *sheets* are assigned as simple rectangles, the library takes the *Square* optimization as default.

In case that *Square* optimization is forced:

- the *parts* assigned with a geometric profile are considered for the bounding box of the same profile, and the profiles assigned as islands are ignored.
- the *sheets* assigned with a geometric profile are considered for the bounding box of the same profile and the profiles assigned as constraint areas are ignored: in this case, the optimization solution can make placements outside of the assigned sheet and/or within the constraint areas.

True Shape optimization

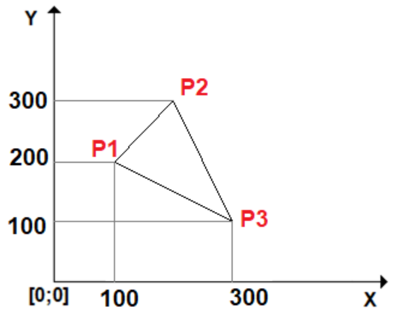
- It manages the placement of *parts* and *sheets* assigned in any way.
- a *part* may contain one or more islands (internal scrap areas), that can possibly be used for the placement of other *parts*.
- a *sheet* may contain constraint areas where the placement of parts is excluded.
- the placement of a *part* can apply variable rotations and possible mirror transforms.
- the placement of *parts* on a *sheet* follows the shapes of parts and sheet, with the result of optimizing the use of the material.
- it is possible to calculate consecutive optimizations, valuable as equivalent alternatives.

2.2 Units and coordinates

In TPA_N the metric values (coordinates, dimensions) are expressed in the unit defined as [mm] or [in] in the call to function IniSettings(...).

TPA_N works in a 2D Cartesian system:
- the point of coordinates [0;0] is located on the lower left corner.
- X is the horizontal axis, positive to the right.
- Y is the vertical axis, positive upwards.

In the figure, a simple example of polygon assigned on 3 points: P1(100;200), P2(200;300), P3(300;100)



Reference will be made to remarkable geometric characteristics of a polygon. Let's see how they are defined:
- *bounding box*: rectangular area that completely contains the profile on the XY plane
- *LB (Left Bottom) corner*: lower left corner of the bounding box:
  - we indicate the corner opposite to LB as RT (Right Top)
  - the words (left, right) refer to the horizontal axis (X axis)
  - the words (bottom, top) refer to the vertical axis (Y axis)
  - as shown in the figure: the extreme corners of the bounding box do not necessarily belong to the profile.

As shown in the figure:
- the bounding box is identified by the two extreme corners: LB={100; 100}, RT={300; 300}.

All angles are expressed in units of degree and decimals of degree. Where the interpretation of rotation is significant:

- counter-clockwise rotation is associated to a positive angle
- clockwise rotation is associated to a negative angle.

2.3 Nesting dimensions

TPA_N can operate in two different types of sheets, based on the setting *OneNestingDimension*
- *(false)* 2D: the sheets assign a part of the plane on which the nesting is two-dimensional
- *(true)* 1D: the nesting is on a linear support and the sheets are bars, profiles, tubes.

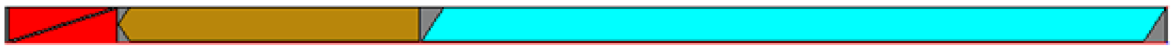
Nesting 2D

The optimization operates on 2D space as bounded by the sheets. The selection corresponds to the default operation (*OneNestingDimension=false*).
Typical applications concern the cutting of sheets of regular or irregular shapes in a variety of materials: metal, plexiglass, wood, leather, paper, fabric.

Nesting 1D

Typical applications concern the cutting of:
- bars, tubes or metal profiles
- wooden beams.

The selection forces the use of *Vertical direction* but limits neither consecutive placements in horizontal progression nor the height of the sheets at the height of the parts.
The figure shows an example of a beam optimization solution: the two triangles shown on the left side show how the optimization procedure still keeps active the generic logic of placement on a two-dimensional plane.



The selection directs TPA_N in the choice of criteria which are studied to obtain better results in case of placements on linear support. Inappropriate selection can lead to suboptimal results, especially from the point of view of calculation time.
Settings to be associated with the selection may concern:
- selection of the type of part cutter (*OneCutterDimension* general setting)
- settings relating to the overlapping of the cut profiles at the sheet edges (general settings: OverrunSheet, MaximizeOverrunSheet)
- mirror application criterion of the parts (ModeMirrorPart general setting)

The above example:
- can work with a linear type cutter (e.g., blade)
- the cuts of parts exclude the horizontal trims, along the upper and lower part of the beam: the cuts can then be applied completely outside the edges of the beam
- the beam does not have a good side (top and bottom are equal): a part can therefore be applied in normal or mirrored mode, based on the greater efficiency.

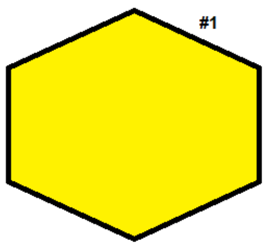
2.4 Profile definition

A *part* or a *sheet* is assigned through a primary geometric profile and possible secondary profiles (islands or scrap areas): the simplest geometric profile is a rectangle, that can be directly assigned with the *length* and *height* dimensions.

A geometric profile can consist of linear or curved (arcs) segments, and anyhow it is used as *closed*, with possible closing by a linear segment. As in the previous figure: the profile is assigned with the three points (P1, P2, P3) and it is closed with a linear segment joining P3 to P1.

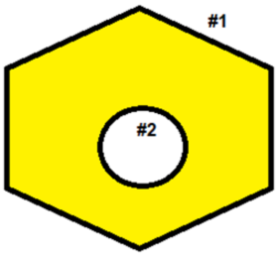
Each profile assigns a *net* area:
- cut area, for the parts
- border area, for the sheets.

If a part (or sheet) assigns a geometry, the primary 2D dimensions of the part are not used (*L* and *H* fields in the ***NestPart*/*NestSheet*** structure of part/sheet assignment).



The figure represents the case of a part without islands:

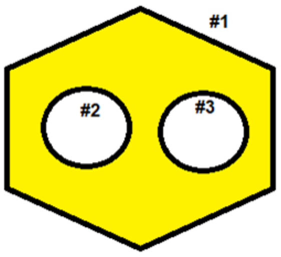
- #1 marks the primary external profile.



The figure represents the case of a part with 1 island:

- #1 marks the external primary profile
- #2 marks an internal scrap profile.

Assigning the part: the scrap profiles can be used to place other parts.
Assigning the sheets: the scrap profiles represent areas that cannot be used for placements.



The figure represents the case of a part with 2 islands:

- #1 marks the external primary profile
- #2 and #3 mark the two internal scrap profiles.

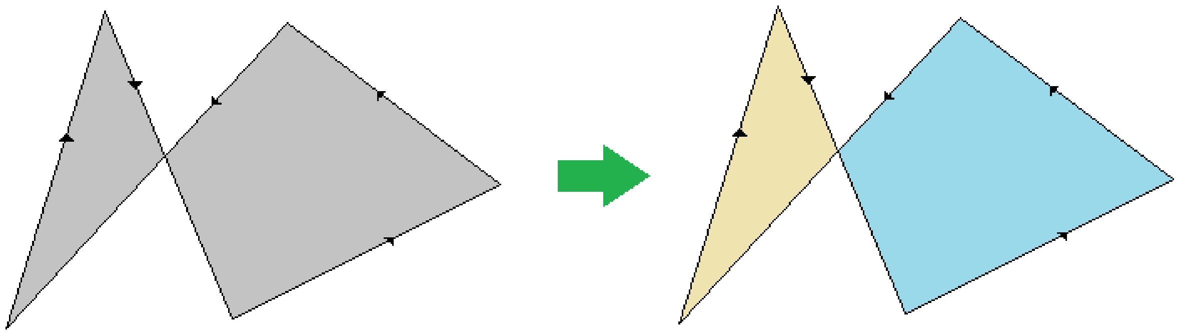
The bounding box of a geometry can be assigned anywhere, in X and/or Y positive or negative area: assigning a geometry in positive XY area and a minimum overall dimension in (0;0) can be a programming convenience, but it does not represent a necessary condition.

Simplification of profiles

A geometric profile is valid if it assigns a *simple* geometry: no null or intersecting segments and closing a non-null area.

In order to have a *simple* geometry, each profile is drawn up by TPA_N with:
- removal of coincident (less than one epsilon of linear resolution) or aligned vertices.
- removal of situations of intersections, with possible splitting into more profiles.

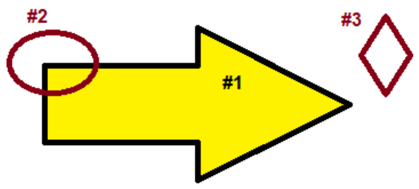
In the figure we can see an example of profile with intersecting segments:
- on the left: the original profile
- on the right, the two-coloured areas highlight how the profile is split into two profiles, each of which is then to define a *simple* geometry
- by assigning:
  - a part: the two simple profiles obtained are appropriately joined and used as a single element
  - a sheet: separate placements will be possible within each simplified geometry



Profiles that cannot be traced back to a valid geometry are discarded by the nesting procedure, possibly resulting in exclusion from the placement calculation.

Particular attention must be given to assigning internal profiles (part islands or sheet constraint areas):
- they **must** be fully inside their corresponding primary profile.
- in case of islands of a part: they can be used to place other parts only if in turn they do not contain more islands.

The figure shows a wrong assignment situation:



- #1 marks the primary profile (the inner area is yellow)
- #2 marks a scrap profile that *intersects* the primary profile
- #3 marks a scrap profile that is *external* to the primary profile

The nesting procedure will ignore both scrap profiles. In particular: the area used by the two scrap profiles can in no way be observed during the evaluation of the part placement on a *sheet*.
A situation such as the one illustrated here involves the certain overlapping of placements of other parts in the area of the profiles indicated as (#2, #3): it is the task of the calling application to check and/or report similar situations.

Approximations of profiles

The curved segments of a profile are approximated in a series of linear segments, with a defined maximum tolerable inaccuracy of 0.3 mm:
- that means the maximum distance that the flattened path will deviate from the *real* arc will not be more than 0.3 mm
- the number of segments in which an arc is approximated varies according to the arc radius: it increases as the radius decreases.

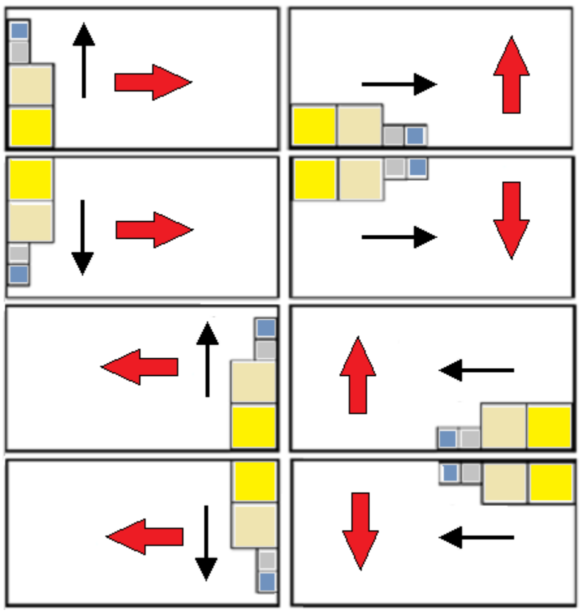
The tolerance value introduced here is indicated as *Chordal error*.

The approximation of one or more segments of a profile can lead to the addition of a safety distance (at least equal to the *Chordal error*) between contiguous placements or between parts and the sheet edge.

2.5 Nesting Direction and Development Corner

The Nesting Direction assigns the placement feed direction, filling up the sheets:
- *horizontal direction*: the placements are performed first vertically (in the picture: cases on the left, with horizontal red arrow)
- *vertical direction*: the placements are performed first horizontally (in the picture: cases on the right, with vertical red arrow)

The Nesting Corner assigns the placement starting corner among four possible values:



- 0 = Left-Bottom (in the picture: cases in the first row from top)
- 1 = Left-Top (in the picture: cases in the second row from top)
- 2 = Right-Bottom (in the picture: cases in the third row from top)
- 3 = Right-Top (in the picture: cases in the fourth and last row from top)

Information corresponds to the project general assignments Direction and Corner.

TPA_N always uses the Corner setting as assigned.
Instead, it is possible that TPA_N attempts optimization with a modified value of Direction, in order to obtain better solutions.

Matching Groups and Filters

The assignment of *parts* and *sheets* calls for generally technological settings, in order to apply matching conditions and/or filters between *parts* and *sheets*, even though this is not the normal condition of use.

Let us first examine the settings that help to apply the matching conditions.

These settings correspond to the fields in the NestPart and NestSheet structures:

- *S*: (double type) thickness
- *Fiber*: (integer type) generic assignment of material
- *Colour*: (integer type) generic assignment of colour
- *Grain*: (integer type) assignment of grain direction.

S Field: two values are equal, if their difference is smaller than the epsilon of comparison (MinResolution setting).
Further settings enable the application to use the fields *Fiber*, *Colour* and *Grain* (MatchType, MatchColor and MatchGrain).

Examples of assigning matching groups on thickness of parts and sheets:

- two parts are assigned (identifiers: 1, 2) with field *S*=18.0
- one part is assigned (identifier: 3) with field *S*=25.0
- one sheet is assigned (identifier: 1) with field *S*=0.0
- one sheet is assigned (identifier: 2) with field *S*=18.0

the situation causes the assignment of 3 matching groups:

- group 1, matching *S*=18.0
- group 2, matching *S*=25.0
- group 3, matching *S*=0.0.

Each group is optimized separately. It is evident from the example above how only group 1 can generate a Nesting solution, allowing the association between parts and sheets: the parts with identifiers (1, 2) will be placed on sheets with identifier 2.

In case of assignment of multiple values (for instance: for the material as well), the matching associations between parts and sheets can increase in complexity. For example:

- group 1, matching *S*= 18.0 and *Fiber*=0
- group 2, matching *S*= 18.0 and *Fiber*=1
- group 3, matching *S*= 25.0 and *Fiber*=0.

Grain Control

A further field shows up in the NestPart and NestSheet structures, matching the assignment of grain direction:

- the assignment takes place in the *Grain* field of the structures
- information technological meaning depends on the sheet material (e.g., wood, veneer, metal).

The assignable values are three:

- *0 (None)*: it does not assign grain
- *1 (X)*: it assigns grain along the horizontal direction
- *2 (Y)*: it assigns grain along the vertical direction.

The grain assignment does not necessarily lead to the determination of separate matching groups: parts with the same value in the *Grain* field can be placed in sheets of different groups and/or with the same or a different *Grain* field value. As seen in the previous paragraph the behavior is governed by the MatchGrain property.

If it is *MatchGrain=true*: each *Grain* value results in a separate match:
- a part with *Grain=(1, 2)* can only be placed in sheets with same grain
- a part with *Grain=0* can only be placed in grainless sheets.

Let us see which criteria are applied in case of *MatchGrain=false*:
- a part with *Grain=(1, 2)* can be placed with any rotation on a sheet with *Grain=0*
- a part with *Grain=(1, 2)* can be placed on a sheet with *Grain=(1,2)* with such a rotation as to comply with the direction assigned to both
- a part with *Grain=0* can be placed with any rotation, regardless of the sheets' *Grain* field.

If a part with *Grain=(1, 2)* can be placed on different sheets by grain:
- a privileged placement on the sheet with the same assigned grain is not guaranteed
- the active *RctMinimize* setting is not applied, if there is the possibility to use the part on a type of sheet with assigned grain.

2.6 Transforms applied to the parts

For the parts, it is possible to assign information on two geometric transforms:
- mirror
- rotation.

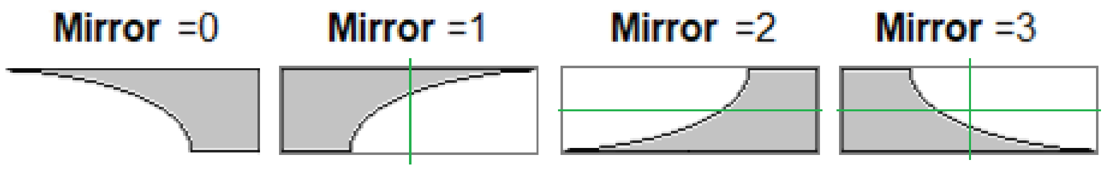
Part Mirror

TPA_N supports two ways to apply mirror parts, according to the *ModeMirrorPart* setting (***boolean*** type).

Force Mirror

The function corresponds to *ModeMirrorPart=false*.
It is possible to request the mirrored placement for every part, along one or both the coordinated axes, with possible selection among four choices:
- *0 (None)*: the part is not mirrored
- *1 (X)*: the part is mirrored around the vertical centerline axis of its own bounding box
- *2 (Y)*: the part is mirrored around the horizontal centerline axis of its own bounding box
- *3 (X+Y)*: sum the two previous options.

The part assignment corresponds to the *Mirror* field in the NestPart structure:



The symmetry is applied before using the part and before any rotation of the part.
The mode corresponds to the case of a sheet with a good *side* of cutting: typically the upper side.
Specific applications use non-linear supports: wood or metal plates, leather, fabric.

Mirror as an option

The function corresponds to *ModeMirrorPart=true*.
Mirroring can be applied to a part if placement is not possible in normal mode, even by trying different rotations.
The selection is on four choices, even if for the purposes of use the choice between the first two values may be sufficient:
- 0 (None): it is not possible to mirror the part
- 1 (X): it is possible to mirror the part.

If the size of linear nesting is selected (*OneNestingDimension=true*): attempts to mirror the part are still made, even if a part can be placed in normal mode.

The mode corresponds to the case of a sheet without a good *side* of cutting: the upper and lower sides are equivalent. Specific applications may concern linear supports: bars, beams.

Rotation Control

TPA_N manages the possibility to apply a rotation to the parts. The application differs according to the optimization modes:
- *Square*: the parts can rotate 90° counter-clockwise
- *True Shape*: the parts can rotate in angular steps, with the possibility to change this step.

For each part it is possible to assign the degree of freedom allowed for rotation, with possible selection among three choices:
- *0 (None)*: no rotation is allowed for the part
- *1 (90°)*: the part allows for 90° step rotation, (if it optimizes *True Shape*) or 90° counter-clockwise (if it optimizes *Square*)
- *2 (Any)*: the part allows for rotation in variable steps. If it optimizes *Square*: the selection corresponds to the previous one (*90°*).

The part assignment corresponds to the *Rotate* field in the NestPart structure.

Information about the rotation steps allowed with *Any* selected corresponds to StepAngle general assignment:
- the property accepts values between 1.0 and 90.0
- the minimum value of rotation actually applied is reduced to an integer sub-multiple of 360°.

The lower the set value is, the more challenging the placement calculation phase will be, both in terms of required memory and in terms of time needed to find a placement solution:
- set value 45.0 to allow rotations of up to 45.0° minimum steps: one part has 8 possible placement solutions, determined by applying all the multiples of 45.0°, i.e., 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°;
- set value 15.0 to allow rotations of up to 15.0° minimum steps: one part has 24 possible placement solutions, determined by applying all the multiples of 15.0° (0°, 15°, 30°, …, 345°).

Minimizing the bounding box of a part

Another general ***boolean*** type setting can affect the rotation logic of the parts (RctMinimize).
The parts with a non-rectangle assigned geometry can use this property.

In case of *Square* optimization:
- *false*: the parts are used as assigned. In case of assigned geometry: this is placed as in the original or with 90° rotation (if allowed)

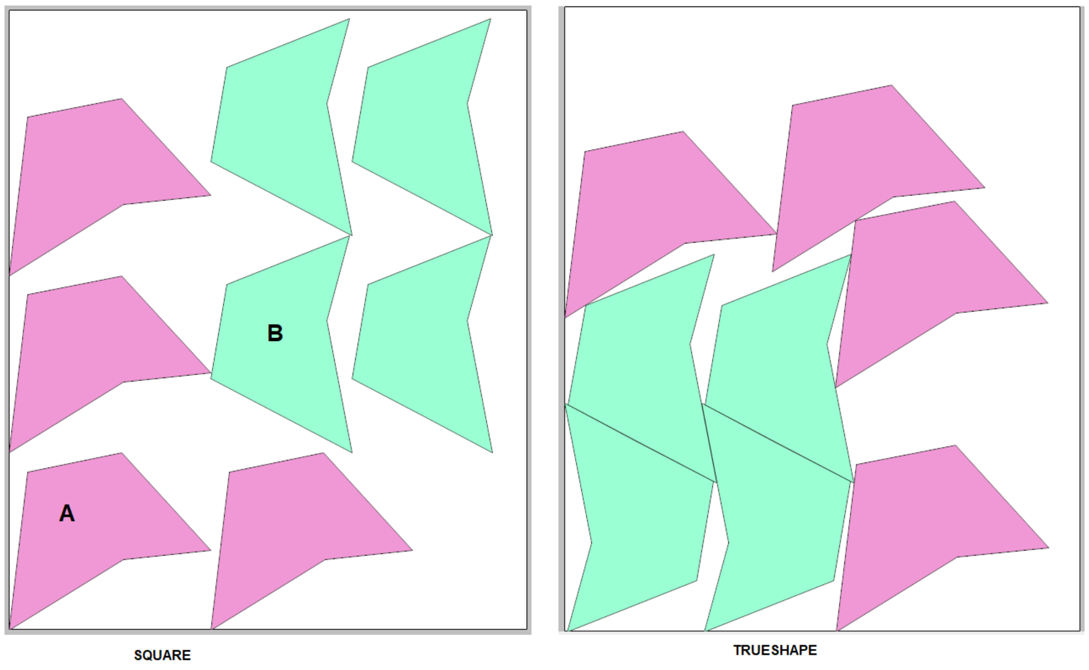
- *true*: the initial placement of the part can be changed in advance, compared to the original, with a rotation determined so as to minimize its overall bounding rectangle. This can happen only if the part can rotate.

In case of *True Shape* optimization:

- *false*: if the part allows for *any* rotation, the rotation minimizing the bounding box is determined anyway
- *true*: similar qualification also for the parts that allow rotations in 90° steps.

The figure corresponds to executing the two optimizations with placement of a same shape in *Square* optimization (on the left), *True Shape* (on the right) and with *RctMinimize=true*:

- **A**: the shape is used with no possibility of rotation. The placements match the original geometry
- **B**: the shape is used with possibility of rotation (90.0°). It is now clear how each placement starts from a rotation that scales down the bounding box of the same shape, consequently with a higher possibility of placement optimization.



Rotation and Grain

The grain assignment (X or Y) of a part can limit its ability to rotate or prevent it from being placed in grained sheets:

- as a first point, the application of rotation (*Any*) is excluded: if assigned, it is reduced to (*90°*)
- if part and sheet have the same grain (e.g., X+X; Y+Y): the part can be used with rotations (*0°*; *180°*) and only (*0°*) if the part cannot rotate
- if part and sheet have different grain (e.g., X+Y; Y+X): the part can only be used if it can rotate and with rotations (*90°*; *270°*).

2.7 Placement in the islands

The *parts* can assign islands that can be used to place other smaller parts.
The functionality is managed only in *True Shape* optimization.
For each part it is possible to enable the placement within its islands, corresponding to the *UseIsle* field in the NestPart structure.

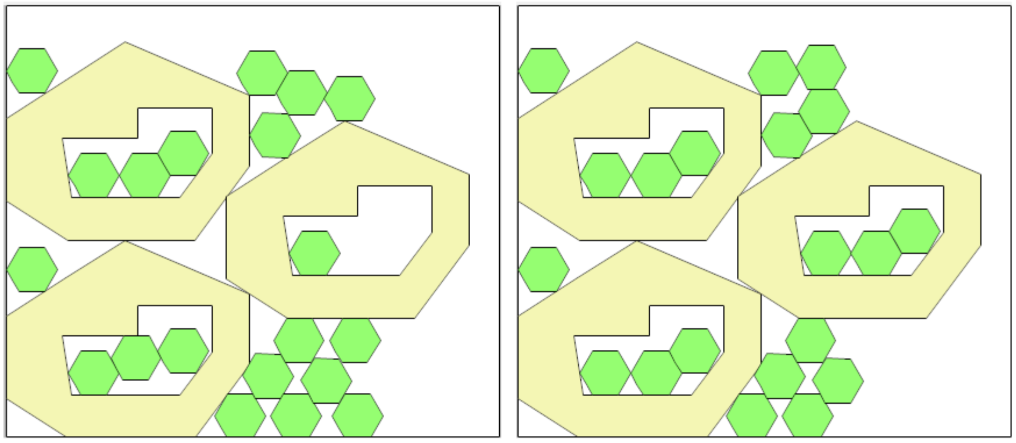
Some general ***boolean*** type settings control the placements in the islands:

- *PartInHole*: fully enables performing placements within the parts' scrap areas.
- *PartInHoleMulti*: enables performing recursive placements within the parts' scrap areas
- *PartInHoleBefore*: enables prioritizing the placements within the parts' scrap areas.

The terms of managing this functionality must be evaluated considering technological factors about the sheet fastness and the cut of parts.

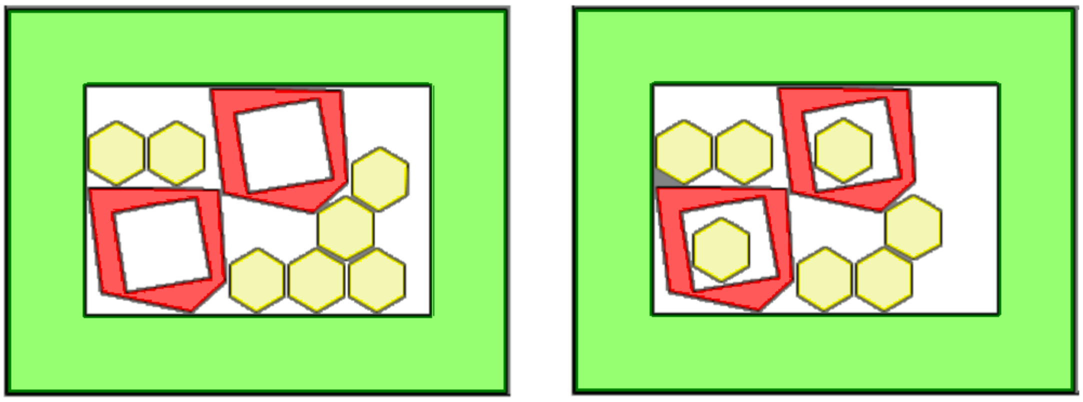
The figure compares the effect of the *PartInHoleBefore* setting:

- on the left, the case of *false* value: the placements are made according to general criteria of maximum filling of the used part of the sheet
- on the right, the case of *true* value: the possible placements within the islands are prioritized.



The figure compares the effect of the *PartInHoleMulti* setting:

- on the left, the case of *false* value: the part shown in *red* is placed in the island of the *green* part; no more placements are performed within the part shown in *red*
- on the right, the case of *true* value: the placements are carried out in the island of the *red* part.



2.8 Control of the spacing between placements

Various information contributes to determining the spacing between the placements and between placements and sheet edges.

Part Cutter diameter

General information assigns the diameter of the tool used to cut a part: it corresponds to the CutterDiameter setting.
The value can be null (=0.0) and is used for parts that do not assign their own tool diameter.
For each part it is possible to assign a tool diameter for the primary profile and one for cutting the islands, corresponding to the fields (*PartDiameter*, *IsleDiameter*) in NestPart structure:

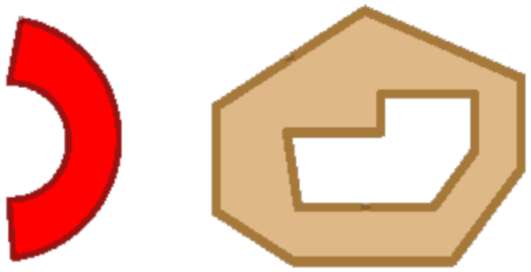
- if both fields have significant (positive) value: the part uses specific diameters for primary profile and islands
- *IsleDiameter=0.0*: the part uses *PartDiameter* for inner profiles
- *PartDiameter=0.0*: the part uses CutterDiameter for primary profile and islands.

Below we will refer generically to *CutterDiameter* to indicate the cutter diameter of a part: actually, the value used takes into account the settings shown here.

The tool is applied around the profile of each part:
- on the outer part of the external profile, to increase the real area of the profile.
- on the inner part of the scrap cut profiles, to decrease the scrap area.

The figure shows two generic parts:
- on the left, a part without islands
- on the right, a part with island

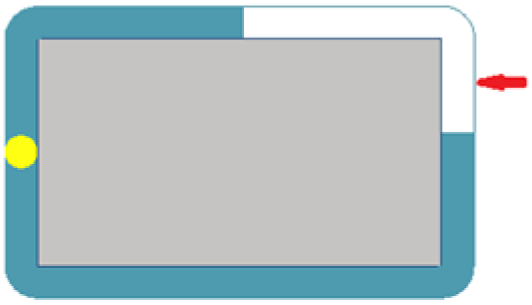
The solid colored part marks the paths assigned to the parts: the net area of a part.
The dashed paths often mark the real overall dimension of the parts, with the added overall dimension of the cutting tool.



Typology of part cutter

It is possible to select the typology of the cutter, by configuring OneCutterDimension setting (***boolean*** type):
- *(false)* the tool has cutting dimension in XY (e.g., milling cutter, laser)
- *(true)* the tool has only one cutting dimension (e.g., blade, cutter).

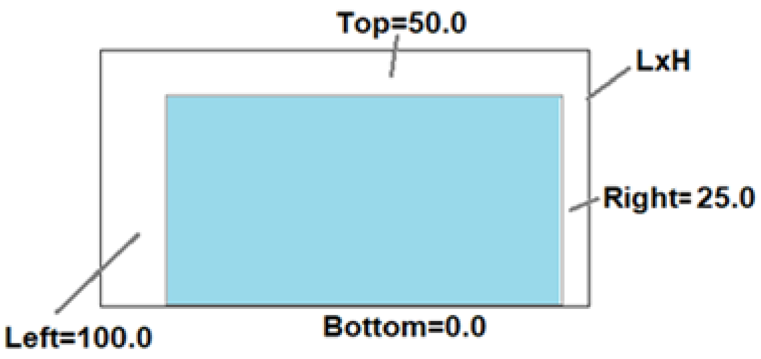
The setting is used in *True Shape* optimization as one of the factors causing spacing among contiguous placements or between the parts and the sheet edges:
- the use of tools (milling cutter, laser) automatically introduces a minimum spacing equal to the *Chordal error*, since the movement of the tool around an edge traces a curved external path (in the figure: indicated with an arrow), which must be approximated in a series of linear segments



- the use of tools (such as blade, cutter) introduces a minimum spacing equal to the *Chordal error* only in view of how the profile is.

Spacing from the sheet edges

Given the size of a rectangular *sheet*, it is possible to assign side areas that are useless for placements and that can be differentiated by side. In the figure, there is a sheet of size LxH and with assignment of the four external margins, all different: the useful area for placements is the colored inner area.



The setting of the external margins corresponds to these settings:

- MarginOuter: for a combined assignment of the four margins
- MarginLeft, MarginRight, MarginBottom, MarginTop: for a differentiated assignment of the margins.

In all cases it is possible to assign either null or positive value.
In case of *True Shape* optimization and with a non-rectangular sheet: a single external margin is used. It takes the highest value set for the four differentiated-by-side margins. No margin is added, instead, to any constraint areas inside the sheet.

The *minimum spacing* there may be between a sheet edge and a placement, intended as net area, is 0.0mm. The real distance is determined on a case-by-case basis, based on multiple aspects:

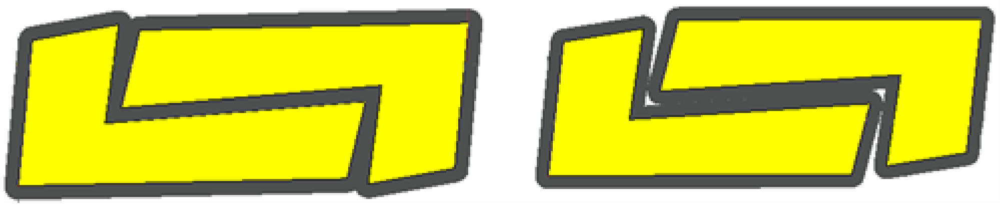
- *CutterDiameter* is the diameter of the part cutting technology (the minimum valid value is 0.0)
- the OverrunSheet setting
  - *false*: it forces the cutting tool not to get out of the sheet margins, thus taking the minimum distance from the edges to the value (*CutterDiameter*)
  - *true*: it allows the tool to get out of the sheet margins, based on the MaximizeOverrunSheet setting in full or half its value

  - the assessment of the minimum distance of placements from the edges can add a spacing in compensation of:
    - insufficient external margins
    - modification of the profiles with application of the *chordal error*
- if different technology diameters are assigned to the parts, the diameter to be considered in the above points is the minimum one.

Spacing between placements

The *minimum spacing* there can be between adjacent placements amounts to (CutterDiameter + OverrunSecurity):

- the OverrunPolygon setting manages enabling the overrun of the part cutting areas. In the figure an example of placements with overrun (on the left) and without overrun (on the right): the grey outline corresponds to the path of the cutting tool



- the OverrunSecurity setting assigns the *Safety distance* added to the placements with the overrun applied of the part cutting areas
- the MarginInner setting assigns an added spacing between adjacent placements.

Now we will see some examples of possible settings

CutterDiameter	*OverrunPolygon*	*OverrunSecurity*	*MarginInner*	**Spacing between placements**
10.0	False	0.0	0.0	20.0

10.0	False	0.0	20.0	40.0
10.0	True	0.5	0.0	10.5
10.0	True	0.0	20	30.0

What has been said here is simplified with respect to the assignment of different diameters for the parts. In this case:

- the maximum overlap between cutting profiles is equal to half of the lower diameter assigned for the parts (all those that can be placed)
- the assessment of the minimum distance between the parts can add a spacing to compensate for changes in profile segments with application of the *Chordal error*.

Overall dimensions external to the parts

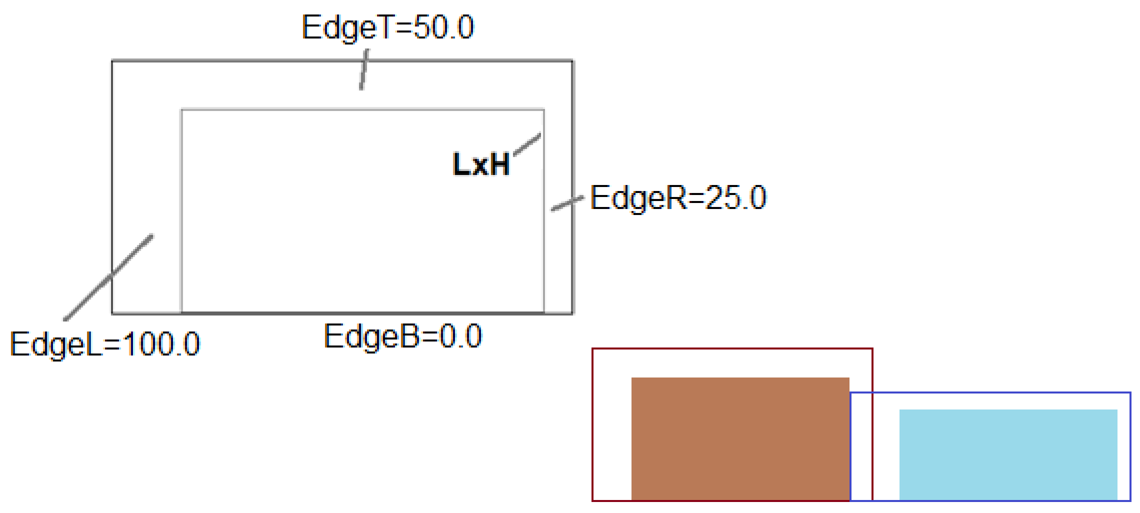
For parts, it is possible to assign four fields interpreted as external dimensions, differentiated by side, to be added to the part profile:

- the assignments are in the fields (*EdgeL*, *EdgeR*, *EdgeB*, *EdgeT*) in NestPart structure
- these overall dimensions assign areas considered to be scrap:
  - contiguous external dimensions overlap
  - useful placements are excluded in all the external dimensions.

In *True Shape* optimization the external overall dimensions are generally applied in excess compared to the *Square* optimization:

- if the part assigns a non-rectangular geometry: in this case it uses a single value, equal to the maximum dimension set.
- increase the spacing of the parts from the sheet edges

On the upper part of the figure there is a part (identified by the external dimensions of the LxH profile) with assigned all different external dimensions.
On the lower part an example of maximum horizontal approach between two parts with their outer edges



In case of symmetry and/or rotation of a part, the margins *follow* the applied transforms. Let's assume the case of a part with applied mirror X:

- the fields (*EdgeL*, *EdgeR*) are applied exchanged.

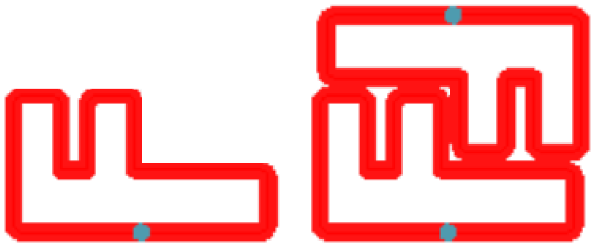
2.9 Part clusters

TPA_N manages the possibility to activate part clusters to be used for placements instead of the single parts.

Automatic cluster

An automatic *cluster* consists in generating a group obtained for a single part by matching the part with a copy of itself rotated by 180°.
The function is managed only in *True Shape* optimization.

The figure shows an example of single part on the left and, on the right, the *group* that can be obtained by applying the automatic cluster.



The placement of a part in *Automatic cluster* mode comes before the single placement if the use efficiency of the *group* is equal to at least the value assigned with the *ClustersExpected* property, where the use efficiency of an automatic cluster is calculated as:

(Area of the single part * 2 * 100) / (Group length * Group height)

For each part it is possible to request the automatic cluster placement, corresponding to the *AutoPair* field in the NestPart structure. The automatic cluster mechanism requires the part to be rotatable.

If possible TPA_N searches for the most efficient automatic cluster by applying the rotation that minimizes the overall dimension of the part.
The placement of an automatic cluster can apply further rotations of the *group*, as a first attempt with 90° steps and subsequent to lower steps (45°, 30°, …).
The part can also be placed individually, where it is no longer possible to apply it in a *group*.

The *AutomaticCluster* setting assigns the total activation to carry out automatic clusters.

The *ClustersAbsolute* property assigns the efficiency of using an automatic cluster independently both for the involved parts and the *AutomaticCluster* setting: if an automatic cluster has an efficiency at least equal to the set value, the cluster placement is activated anyway.

The application in automatic cluster can be applied forcibly active or not in case of recognition of profiles of remarkable shapes. For example:
- does not apply in case of: rectangle, circle, ellipse, some types of regular polygons
- forces the application in some types of regular polygons.

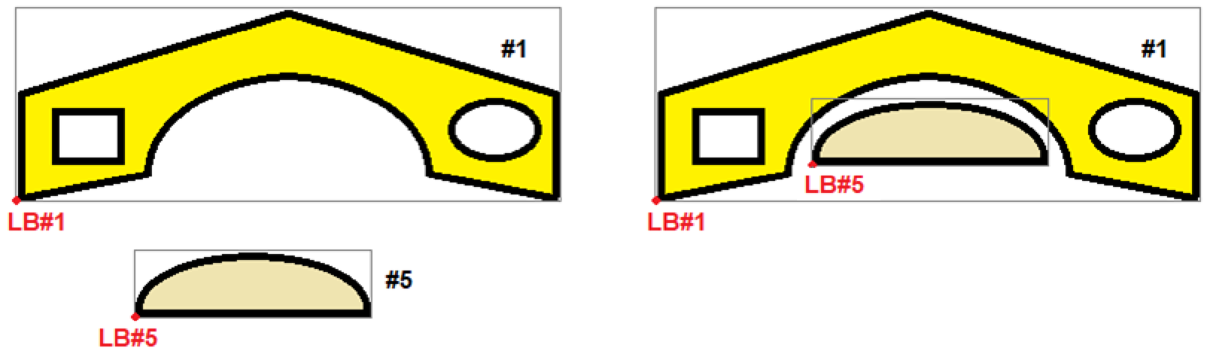
Manual cluster

This mechanism allows creating clusters of individual *parts* that have been assigned independently: the parts grouped in this way will be placed as a single group, keeping their reciprocal position unchanged.
In case of manual cluster among parts, TPA_N creates a general drawing of all the parts, including possible islands: the drawing so obtained is then used as a single entity to be placed on the sheets.

The assignment of manual clusters is managed in both *True Shape* and *Square* optimization, with their own differences relating to the two specific procedures.
Below, the *cluster* word will be used to identify a generic manual cluster.

Two parts are represented in the figure:
- #1 is the part with ID 1: primary profile and 2 islands.
- #5 is the part with ID 5: one single primary profile.

The corner LB (Left Bottom) of the bounding box of the primary profile is indicated for both parts.
On the right, the parts are shown on the same representation plane, showing their mutual positioning.



The correct construction of a cluster comes from the correct assignment of the parts. For example, the calling application is responsible for assigning the parts so that they have intersections or that they are geometrically distinct.
The parts that are actually taken into consideration are only those enabled (NestPart structure, *Enable=true* field).

A manual cluster is treated by the nesting procedure as a single part: cluster placement is not optional, but requested.
To all intents and purposes, a cluster is to be understood as similar to a part only with a different geometric construction method.
The considerations made up to now regarding the assignment and use of single parts can be extended to clusters (e.g., grain control, application of transforms, placement in automatic cluster): this also applies to the considerations that will follow, unless otherwise specified.

A manual cluster is assigned through specific methods and structures.
The use of a part in the composition of one or more clusters does not exclude the possibility of obtaining direct placements for it: in this case we talk about *residual placements*.

Similarly to a part, a cluster can assign a set of general information:
- requested and maximum quantity available
- allowed and requested transforms (rotation, mirror)
- matching assignments (thickness, material, colour, grain)
- enabling assignments (automatic cluster, placement in islands, grid placements)
- priority.

The geometric construction of the cluster takes place by indicating which parts contribute to the creation of the cluster scheme:
- all the parts assigned are usable
- each part can be used once or more times, in the construction of a cluster or several clusters
- a cluster can indicate up to a maximum of 100 distinct uses of parts
- each use of part assigns the transforms to be applied: translation, mirror, rotation.

Matching Groups and Filters

What has already been said regarding the matching conditions and/or filters between *parts* and *sheets* is also valid between *clusters* and *sheets*.
No match is required between a *cluster* and the *parts* used in it, with the only exception concerning the grain.
Let's see what this entails:

- a cluster with a thickness of 20 mm is assigned: each part can be used in the construction of the cluster, regardless of the part thickness. In particular a part that does not verify any match with the sheets can also be used
- a cluster with grain X is assigned: a part can be used only if assigned with grain other than Y
- a cluster with grain Y is assigned: a part can be used only if assigned with a grain other than X
- a cluster with grain OFF is assigned: a part can be used only if assigned with grain OFF.

Grid placements

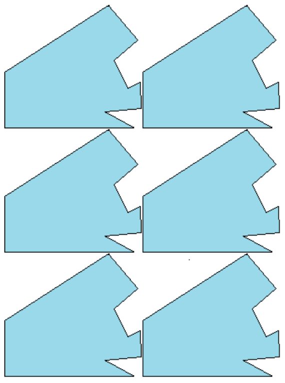
For one part it is possible to request placements that follow a grid pattern, corresponding to the *AutoGrid* field in the NestPart structure.

The *AutomaticGrid* setting assings the total enabling to carry out the grid placements.
The function is managed more effectively in *True Shape* optimization: it is indicated below as a prerogative of *True Shape* optimization, but is now partially working also in *Square* optimization.

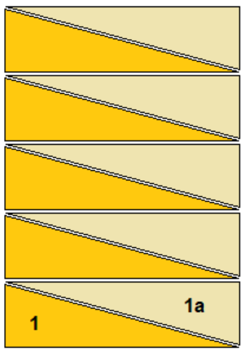
The workpieces for which a grid placement is requested are used before the others and are placed according to a *row * column* layout, considering the space available on the sheet. To have the placements optimized, each part can be analyzed by applying autonomous cluster strategies:

- a part can be placed with repetition of a unit that can match a single part, always repeated with the same rotation, or two parts, with the application of the automatic cluster.
- the repetition unit, of single or double part, can then be placed assessing a 0° or 90° rotation variation.

Below are examples of grid placement of a part.



- no automatic cluster is applied
- the grid development is 3 rows * 2 columns



- the automatic cluster of the part is applied, consisting of pairing the parts (**1;1a**)
- the grid development is 5 rows * 1 column

In *True Shape* optimization, the application of grid placements can be applied forcibly active or not in case of recognition of profiles of remarkable shapes.
In *Square* optimization, grid placement is limited to only the first type of element placed.

Groupings of parts

Grouping functions of single parts are being studied to define logic for subdividing placements on areas of sheets or in a single sheet.

The *Range* field in the NestPart structure and the UseRangePart and DimRangePart settings are set up for these functions.

2.10 Part and Sheet repetitions

The normal assignment condition of *parts* and *sheets* coincides with having lists with more enabled elements (*Enable* field in the NestPart and NestSheet structures).

In this situation: only the elements that assign a strictly positive requested / available quantity are considered (*N* field in *NestPart* and *NestSheet* structures).

Everything we say refers to a single matching group between sheets and parts: if there are more than one group, the considerations must be applied to each single group.
What has been said for the use of the parts may partly differ for the cluster placement, for which please refer to the paragraph Cluster placements.

The optimization procedure places the workpieces on the least number of panels available. If the procedure has placed the available quantity of workpieces and for some of them a **Maximum quantity > Requested quantity** is set, the placement fills up the panels already used, up to the maximum set value.
As for the management of the Maximum quantity field of a part, please refer to the subject Extra placements.

We will indicate this situation with the words: *Multiple parts and Multiple sheets*.
Let us now examine below the particular situations that can derive from this general one.

Single Part and single Sheet

Both the *part* and the *sheet* lists have only one element enabled. According to your needs, it is possible to select among specific optimizations:

NestPart.N	*NestSheet.N*	
0	0	the procedure places the highest number of workpieces on 1 sheet
0	>0	the procedure places the highest number of workpieces on the total available of the sheets (all the resulting sheets are the same)
>0	0	the procedure calculates the number of sheets required to place the requested Quantity of the part. If *Maximum quantity > Requested quantity* is set for the part, the last sheet used is filled up to the maximum value set
>0	>0	the procedure places the Requested quantity of the part on the least number of sheets available. If the procedure has placed the available quantity and *Maximum quantity > Requested quantity* is set for the part, the last sheet used is filled up to the maximum value set.

Multiple Parts and single Sheet

The *part* list has more than one element enabled, while the *sheet* list only has one. The *Available quantity of parts* must be strictly positive (>0).
According to your needs, it is possible to select among specific optimizations:

NestSheet.N	
0	the procedure calculates the number of sheets required to place all the requested parts
>0	the procedure places the highest number of requested parts on the least number of sheets available

In both cases: if the procedure has placed the available quantity of workpieces and if for some of them a *Maximum quantity> Requested quantity* is set, the placement fills the sheets already used up to the highest set value.

Single Part and multiple Sheet

The *part* list has only one enabled element, while the *sheet* list has more than one. The *Available quantity of sheets* must be strictly positive (>0).
According to the needs, it is possible to select among specific optimizations:

NestPart.N	
0	the procedure places the highest number of pieces on the sheets available

>0	the procedure places the requested number for the part on the least number of sheets available. If the procedure has placed the available quantity and *Maximum quantity > Requested quantity* is set for the part, the placement fills up the last sheet already used, up to the highest set value.

Extra placements

It is possible to assign extra placements for parts and clusters, in addition to those actually requested.
Information is assigned in the *Nmax* field of the NestPart structure.
The value is used if:

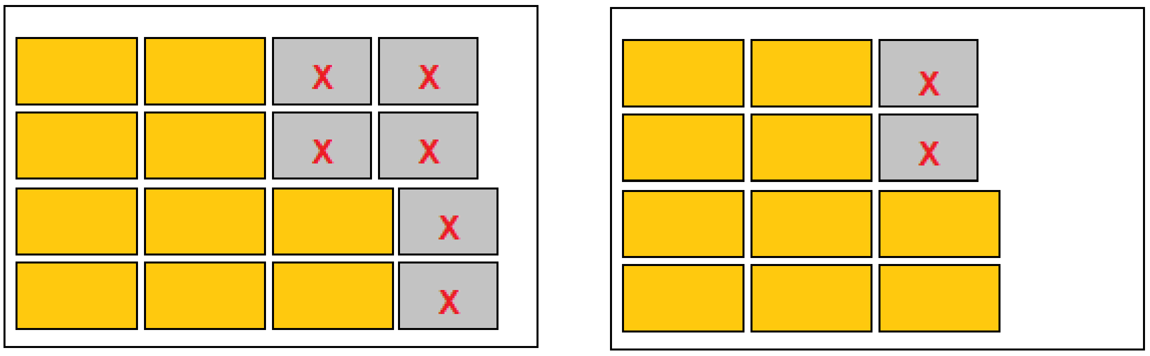
- a strictly positive value is assigned for the requested placements (in the *N* field of the structure)
- the meaning attributed to the value is determined by the *ModeExtraPart* setting (***boolean*** type*)*:
    - if it is *ModeExtraPart=false (default)*: the set value is greater than the one of the placements requested. In these conditions, the number of *extra placements* is calculated to be: *(Nmax - N)*.
    - if it is *ModeExtraPart=true*: the set value is positive and directly assigns the number of extra placements.

Unless otherwise specified, the situation corresponding to *Mode* is assumed as valid in the document.
On a sheet, the extra placements are used only after checking that it is not possible to place any more requested part.
Based on what was here said, it is clear how in no case a sheet can be used only to apply extra placements.
Using the extra placements, it is possible to choose between two modes, as in the ExtraFiller setting:

- *False*: the extra placements are applied to fill the sheets
- *True*: the extra placements are applied only to fill up the length or height that is already being used by the requested workpieces. The filling direction is chosen according to the feed direction of the placements (see Direction setting):
    - if Horizontal: the filling is applied along the sheet length, while no height limit is applied.
    - if Vertical: the filling is applied along the sheet height, while no length limit is applied.

The figure shows the result obtained with a different property value:

- it is assumed to work with: *Direction*=0, *Corner*=0
- the extra placements are marked with an 'X'
- on the left, the case of *ExtraFiller = False*
- on the right, the case of *ExtraFiller = True*



Manual Cluster placements

Some additional considerations are needed for the use of elements in manual cluster. Requested quantity assignments are now in *NestCluster* structure, *N* and *Nmax* fields:

- for clusters the management of null requested quantity is excluded (*N* field=0): in this case, the cluster is automatically excluded

- if <u>all</u> the parts used in a cluster assign null available quantity (*N* field of *NestPart* structure equal to 0)
    - the quantities that can be placed for the cluster are those assigned in fields (*N* and *Nmax*) of *NestCluster* structure

  - the parts used in the construction of the cluster cannot be used for direct placements
- otherwise: the quantities (*N* and *Nmax*) assigned for the cluster are compared with those of the single parts
  - the quantities that can be placed for the cluster are limited by the quantities of the parts, with the extreme possibility of excluding the application of the same cluster
  - if a part is available in sufficient quantity, it is possible that the same is then used for direct placements.

In evaluating the above, it is necessary to keep in mind some general aspects:
- a part can be used several times in a cluster and also in different clusters
- the list of clusters is scanned in sequence of a primary identifier and each one *books* the quantity required for each part: this can lead to insufficient quantity for a subsequent cluster.

A particular use of clusters may correspond to what we have already reported: assign null available quantity (=0) for all the parts used in the clusters. The parts list actually assigns a library of available shapes, while the real nesting project is assigned directly on the clusters.

Let's explain with some examples.

A. Assign a cluster with required quantities: *N=10*, *Nmax=20*. The cluster uses 3 parts:
   - part with ID=2, for which we have: *N=5*, *Nmax=10*
   - part with ID=3, for which we have: *N=7*, *Nmax=10*
   - part with ID=5, for which we have: *N=5*, *Nmax=15*

   - the quantities that can be placed for the cluster are: *N=5*, *Nmax=10*
   - part with ID=2 cannot be used for direct placements
   - part with ID=3 can be used for direct placements in quantities: *N=2*, *Nmax=0*
   - part with ID=5 can be used for direct placements only of the extra type (*Nmax=5*).

B. Assign the same cluster as in the previous example, but with changed quantities for the parts used:
   - part with ID=2, for which we have: *N=0*
   - part with ID=3, for which we have: *N=7*, *Nmax=10*
   - part with ID=5, for which we have: *N=5*, *Nmax=15*

   - the cluster cannot be placed
   - part with ID=2 can be used only as a single part in a matching group with sheets
   - the remaining parts can be used for direct placements based on the set quantities.

C. The same cluster as in the previous example is assigned, but with quantities for all null parts:
   - the quantities that can be placed for the cluster are those required: *N=10*, *Nmax=20*
   - the parts cannot be used for direct placements.

In case of residual placements of the parts used in clusters, each part is used according to the match group that the part verifies. Let us see an example:

- assign a part with thickness= 20.0 mm
- by using the part in a cluster the cluster thickness is applied
- by using the part in direct placements the part thickness is applied.

2.11 Optimization in Tpa_N

Optimization in TPA_N has the main purpose of getting the best overall use of the available material (*sheets*) with the requested placements (*parts*) and under the given conditions (*general settings*).

The optimization essentially performs several *attempts* and selects the result of the attempt that offers the best use and, therefore, the *best solution*.
The sequence of the different attempts modifies some *parameters*, so as to generate situations of different placements.
The nature of what we generally define as *parameters* is various and can concern even very complex aspects.
Some examples of *parameters* that can be immediately understood:
- changes in the part placement order
- changes in the sheet filling logic
- changes in the rotation angle of a part
- changes in the Nesting direction (horizontal or vertical).

One aspect of fundamental importance is the mode with which these parameter changes are determined:

- deterministic
- random.

One change performed in deterministic mode can be replicated unchanged over time.
One change performed in random mode can be replicated over time only in terms of probability.

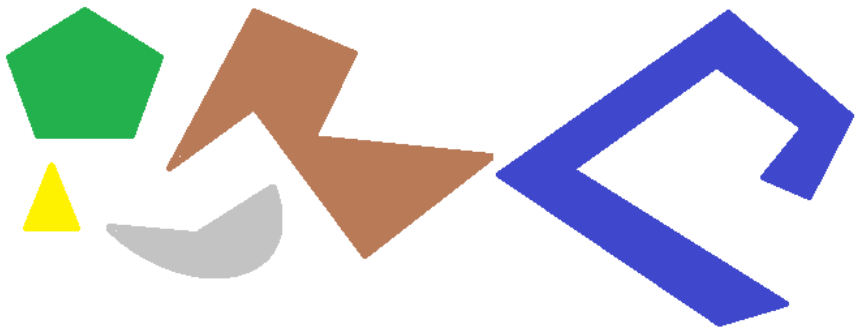
The *Square* optimization applies only deterministic changes.
The *True Shape* optimization applies changes of both types.

Square optimization

The *Square* nesting procedure is such as to lead to the definition of a solution that can be determined repeatedly, keeping unchanged all the initial settings that may affect its development.
The attempts made for this type of optimization are studied in order to examine any possible improvement.
The time set to determine the solution does not limit the number of default attempts.

True Shape optimization

The *True Shape* nesting procedure applies changes to the solution criteria that also have a part of random variability and is therefore such that it leads to the definition of variable solutions, and so that are not necessarily achievable over time, although keeping unchanged all the initial settings that may affect its development. Anyhow, the randomness is generally applied in a *pseudo-random* way: this means that for each random change, such conditions are given as to facilitate the merging towards better solutions, and that, as a matter of fact, limit the number of possible changes.
The nesting solution is now complicated by the fact that the placement concerns *free, concave, or convex shapes*, *with exploitable internal holes*, *in variable number and possibility of rotations*. The possibilities of interlocking among shapes are almost endless. The figure suggests 5 different shapes.



It is clear that the problem of placing a single shape by type, with possible 90°-step rotation, and occupying the smallest and most compact area as possible, is not easy at all: each shape has 4 possible placement solutions (0°, 90°, 180°, 270°) and each one must be assessed for every possible placement of the remaining figures and for every possible pairing combination.
Let us now assign several repetitions for each shape (e.g., 10): every shape repetition must be assessed with any other placeable shape repetition, for a total number of 50 shapes.
Let us assign, now, an angular step of 45°: each shape now has 8 possible placement solutions (0°, 45°, 90°, …, 315°).
It is clear that the problem has become very complicated: it is universally accepted that the solution of a problem like this one cannot rely just on reasoning, but it is necessary to rely also on *randomness*.

A first consequence of this is that it cannot be guaranteed that a solution will be suggested repeatedly. Another consequence is that it is always possible that a new attempt may improve a solution: that is why a time limit should be set, and why it is possible to add more time to determine the solution, starting from the last one determined, in order to make it possible to evaluate among different solutions.

Consecutive optimizations

With *True Shape* optimization, it is possible to calculate more solutions, up to a maximum of 20.

For the first *True Shape* optimization you can select between two modes (see function: Compute)

- it returns the best solution calculated in the time available
- it returns the first solution calculated, activating a *StepByStep* function

Also for the next optimizations it is possible to select between different modes (see function: RetryCompute)

- timed or step-by-step
    - the optimization stops according to the maximum assigned time
    - it returns the first solution calculated, activating a *StepByStep* function.
- It returns a solution only if it is better than the previous one.

Next optimizations can be calculated if:
- a first optimization in *True Shape* mode has been performed
- all settings stay unchanged (settings, part, cluster and sheet lists).
- less than 20 optimizations have been calculated.
- TPA_N has not independently deactivated the possibility, based on its own feasibility and effective usefulness assessments.

Each consecutive optimization:
- starts from the status left by the last optimization executed.
- as said, nothing guarantees that it can be considered *better* than the previous one.

All the optimizations calculated stay available until:
- a new total optimization (see Compute function)
- a request to delete the solutions found (see ClearSolution function).

The best solution

It has already been said that *the solution* of an optimization is the result of a choice made among the solutions of all the attempts that have been made during the optimization phase. The number of attempts fulfilled changes according to several factors.

In *Square* optimization:
- the number of attempts is defined by the procedure
- the time assigned for the optimization is generally enough to run the expected attempts.

In *True Shape* optimization:
- on a first optimization (function: *Compute*): even if the step-by-step functionality is not required, only one attempt is made in the event of a difficult procedure, so as to return a solution in the shortest time as possible.
- on consecutive optimizations (function: *RetryCompute*), the operating mode is selected by the same function:
    - *timed*: more attempts are activated, up to the maximum time available. When the time is almost over, the procedure works to return the best solution among the ones calculated in the attempts that have been completed. It is obvious that there is a minimum time to close the optimization, consisting of the time needed to complete an attempt;
    - *step-by-step*: only one attempt is activated.

Each optimization returns the solution that has been considered the best for the optimization. This means that consecutive optimizations may bring to solutions that are not necessarily better than those calculated before.

But what qualifies a situation as better than another one?

We will say immediately that the answer is not always obvious, and that the required assessment criteria may partly differ between two different optimizations.
Let us see some very general criteria. The points listed below are applied following the entered order, going to the next one in case it was not possible to operate a prevailing choice on the previous point:

- the largest placement area is privileged. A solution that arranges workpieces to occupy the 93.0% of the sheets is better than one that determines an 88.00% filling;
- the solution that favors the arrangement along the selected direction is privileged: along the Y axis in case of Horizontal direction, along the X axis in case of Vertical direction.
- the "neatest" solution is privileged (the assessment is based on the comparison of the off-cuts within the bounding rectangle of the placed workpieces).
- additional assessment criteria about the workpiece arrangement grid are applied, in addition to number and size of the placed workpieces and of the internal off-cuts.

Each point listed above is applied with a relative, variable weight, and with some margins of tolerance, partly fixed and partly adapted to the specific nesting project, so as to allow the combined assessment of the largest number of solutions.

A very practical example: the comparison between placement areas is not absolute (92.7<93.0), but it applies a tolerance area assessed on the minimum size of the workpieces to be placed, in addition to the diameter of the milling cutting.

Optimization criteria and priorities

Upon optimization request, Tpa_N starts an analysis of the assigned lists and the ensuing optimization phase.

The list analysis can come upon error situations, with consequent optimization annulment.

Let us consider this first analysis phase, making a distinction between parts, clusters and sheets.

Checks on the list of *parts*:

- not enabled parts are excluded from the optimization (*Enable* field in *NestPart* structure)
- parts that have no correspondence with valid *sheets* and that are not used in clusters are excluded from the optimization (e.g., part with *Fiber*=1 and no sheet with the same setting)
- parts with one or both the bounding box dimensions <= minimum resolution value are excluded from the optimization (general setting MinResolution)

Checks on the list of *clusters*:

- not enabled cluster (*Enable* field in the *NestCluster* structure) or with null requested quantity (*N* field in the *NestCluster* structure: with value 0) are excluded from the optimization
- clusters that have no correspondence with valid *sheets* are excluded from the optimization (e.g., cluster with *Fiber*=1 and no sheet with the same setting)
- clusters assigned with an insufficient number of parts (less than 2) or using parts that are not identified or excluded or not available in the minimum required quantity are excluded from the optimization
- some checks on the parts result in the exclusion of the cluster:
  - a cluster with grain (e.g., X) cannot use part with different grain (e.g., Y)
  - a cluster without grain cannot use part with grain (X or Y).

Actually, some situations indicated do not necessarily lead to the simple exclusion of the cluster but to an error situation that prevents the procedure from continuing. It is possible to recover the error situations automatically, leading to the exclusion of the cluster, by assigning the general setting FixComputeError=*true*.

Checks on the list of *sheets*:

- not enabled sheets are excluded from the optimization (*Enable* field in the *NestSheet* structure)
- sheets with one or both the bounding box dimensions <= minimum resolution value * 50.0 are excluded from the optimization MinResolution general setting)
- sheets with no correspondence with valid *parts or clusters* are excluded from the optimization (e.g., sheet with *Fiber*=1 and no part or cluster with the same setting).

The result of the reported analyses must be able to use at least one element for each list of parts and sheets:

- in case of single element, even a null requested/available quantity may be requested (*N* field in *NestPart, NestSheet* structures), with ensuing recognition of situations of single part and/or sheet
- otherwise, the elements with null requested/available quantity are excluded from the optimization.

A nesting procedure does not necessarily require the use of clusters.
The start of the nesting procedure applies specific criteria to assign the order of use of parts, clusters and sheets, with some distinctions between the two optimization modes.

Use of sheets

The use of sheets follows an order that can consider different situations:

- sorting the sheets marked as scrap first (*IsScrap* field in the *NestSheet* structure), with same or unmanaged priority
- sorting by increasing type (*ID* field in the *NestSheet* structure), with the same conditions as in the previous points.

The general setting UseBeforeScrap enables applying the qualification field of the *sheets* as scrap.
The general setting UseOrderSheet enables applying the *sheet* priority.
The general setting ModeOrderItem assigns the rule for the interpretation of priority:

- *false*: sort by decreasing priority (sheets with higher priority first)
- *true*: sort by increasing priority (sheets with lower priority first).

- sorting the sheets marked as scrap first (*IsScrap* field in the *NestSheet* structure).

Use of parts (Square)

The term of part covers both individually placeable parts and clusters.

The use of parts follows an order that can consider different situations:
- the general setting UseOrderPart enables applying the *part* priority
- the general setting ModeOrderItem assigns the rule for the interpretation of priority (see above)
- the general setting PartSortMode defines the *part* sorting criteria:
  - *0*: sort by decreasing area (large workpieces first)
  - *1*: sort by increasing area (small workpieces first).
- the general setting *AutomaticGrid* enables grid placement of parts.

Let us see now the main criteria that are applied to the initial sorting of parts. The points are applied in the order shown, moving on to the next one if it was not possible to make a prevailing choice on the previous point:

- sort by decreasing/ increasing priority (if *UseOrderPart=true*)
- parts with grid placement request (if *AutomaticGrid=true*)
- clusters placed before the component parts
- sort by area descending/ increasing (as from: *PartSortMode*)
- part without possibility to rotate
- *square* part
- part with greater perimeter
- part with larger requested quantity
- part with increasing type (*ID* field in *NestPart* structure): cluster IDs are appended in ascending order to those of the parts.

The assessments concerning the comparison between areas, dimensions, perimeters are actually more complex than a list can show: it is interesting to provide a general framework here.

Use of parts (True Shape)

The term part covers both individually placeable parts and clusters.
The use of parts follows an order that considers a larger number of different situations compared to the *Square* case.

Also in this case:
- the general setting UseOrderSheet enables the application of the priority of *parts*
- the general setting ModeOrderItem assigns the rule for the interpretation of priority
- the general setting AutomaticGrid enables grid placement of *parts*.

Before examining the possible situations, it should be clarified that the assigned part order is now to be considered only as an initial situation that can be changed as the optimization attempts progress on.

Let us see what are the main criteria applied to the initial part order. The points are applied in the order shown, moving on to the next one if it was not possible to make a prevailing choice on the previous point:

- sort by decreasing/increasing priority (if *UseOrderPart=true*)
- parts with grid placement request (if *AutomaticGrid=true*)
- clusters placed before the parts they use
- part with concave shape or with islands, with possible inclusion in the concave area or in the islands
- part of larger area
- evaluations of known shapes (rectangles, polygons, circles, …)
- part with less possibility of rotation
- part with larger requested quantity
- part with increasing type (*ID* field in *NestPart* structure): cluster IDs are appended in ascending order to those of the parts.

2.12 Support for managing nesting projects

A nesting project is intended as the set of all the information assigned to the Tpa_N library:
- overall enable settings
- lists of parts, sheets, manual clusters.

Tpa_N exposes serialization methods of a nesting project. These methods are especially useful for:
- nesting situations

- cases of integration of Tpa_N where information provided here in the list allocation structures is sufficient.

However, it is believed that the usual integration of Tpa_N requires customization in this sense.

Let's see how the serialization methods of a nesting project operate in Tpa_N, in particular for the overall enable settings.
By default, all settings are directly associated with a nesting project:
- with the exception of those grouped above in the paragraph of *General functions*
- saving a project to file and loading it include all settings.

This type of operation is very useful for testing situations: it is necessary to test a specific optimization and the project file must be made available complete with all the settings. For this reason, it is advisable for an external application of the Tpa_N library to provide a similar functionality.

Tpa_N also exposes serialization methods for general settings only.

2.13 Known limits of the library

Assignment of parts
- It is possible to assign up to 500 different *parts*.
- for each *part*, the maximum placeable quantity that can be assigned is 999.
- the maximum total placements requested for all the assigned *parts* and *clusters* is 99999.

Assignment of clusters
- It is possible to assign up to 500 different *clusters*
- for each *cluster*, the maximum placeable quantity that can be assigned is 999
- the maximum total placements requested for all the assigned *parts* and *clusters* is 99999
- a *cluster* can assign a minimum of 2 parts and up to a maximum of 100.

Assignment of sheets
- It is possible to assign up to 100 different *sheets.*
- for each *sheet*, it is possible to assign a maximum usable quantity of 999.

Square optimization
- Placing a *part* with assigned grain does not apply the *RctMinimize* setting as active, if it is possible to be placed on a sheet with assigned grain
- only partially resolves the *Grid placements*.

True Shape optimization
- Juxtapose the external margins assigned to *parts*.

3 Guide to the library functions

This chapter details the properties and functions of Tpa_N.

3.1 License management

IsSquareEnabled

Boolean type property testing presence and status of the key for basic functions (*Square* optimization).

Value
True if the module is enabled, *False* otherwise.

Notes
Querying this property runs an actual key reading, but only if no nesting optimization is running.
No optimization can be performed if the basic function key is not valid.
The key existence and status check is regularly carried out by the library.

IsShapeEnabled

Boolean type property testing presence and status of the key for advanced functions (*True Shape* optimization).

Value
True if the module is enabled, *False* otherwise.

Notes
Querying the property runs an actual key reading, but only if no nesting optimization is running.
Some library actions are not performed if the advanced function key is not valid.

3.2 Constants

The following constants are available in the TpaNestingOEM.Nesting class:

MAX_ROW_ITEMS	500	Maximum number of part types
MAX_ROW_N	500	Maximum value of placements requested for a part
MAX_ROW_ITEMSxN	99999	Maximum value of total placements requested (parts and clusters)
MAX_SHEET_ITEMS	100	Maximum number of sheet types
MAX_SHEET_N	999	Maximum value of placements available for a sheet
MAX_CLUSTER_ITEMS	500	Maximum number of cluster types
MAX_CLUSTER_N	999	Maximum value of placements requested for a cluster
MAX_ROW_INCLUSTER	100	Maximum number of parts added in a cluster assignment
SOLUTION_NUMBER	20	Maximum number of solutions that can be calculated in *True Shape* optimization

3.3 Enumerations

The following enumerations are available in the ***TpaNestingOEM*** namespace.

NestErrors

Assignment of the errors managed by the library.

- *ErrorNone*: no error
- *ErrorLicense*: unverified basic license
- *ErrorLicenseHight*: unverified advanced license
- *ErrorReset*: optimization procedure interrupted by the user
- *ErrorMemory*: memory error
- *ErrorPartsEmpty*: empty workpiece list or no enabled or placeable workpiece
- *ErrorPartsTomany*: too many part placements requested
- *ErrorSheetsEmpty*: empty sheet list or no enabled sheet
- *ErrorSheetsTomany*: too many requested sheets
- *ErrorSheetsMatch*: no corresponding match with the sheet list
- *ErrorInGeometry*: error assigning a geometric element (null line, invalid arc, …)
- *ErrorClustersTomany*: too many cluster placements requested
- *ErrorInCluster*: error in manual cluster assignment (part or minimum quantity not available)
- *ErrorClusterMatch*: error in assignment of manual cluster, in checking specific matches (cluster with assigned grain different from grain of parts)
- *ErrorIOProject*: error managing project files (path and/or file access error, invalid format)
- *ErrorIOfile*: error managing temporary files (path and/or file access error, …) or accessories (files: DXF, DWG)
- *ErrorNoneSolution*: no solution is assigned
- *ErrorBusy*: the component is busy in optimization
- *ErrorContext*: it indicates that the current context is not valid
- *ErrorUnexpected*: general or unmanaged error.

3.4 Structures

The following structures are available in the ***TpaNestingOEM*** namespace.

NestPart

General assignment of a *part*. The structure shows initialization methods for fields with default values.

- *int ID*: numerical identifier of the *part* (unique, strictly positive) {*default* = 0}
  - the wording *part typology* is also used for the field
- *bool Enable*: enables the use of the *part* (false= it does not place the part) {*default* = true}
- *string Label*: descriptive name of the part (it can be =""; maximum length 50 characters) {*default* =""}
- *double L*: length (>= 0.0) {*default* = 0.0}
- *double H*: height (>= 0.0) {*default* = 0.0}
  - it is possible to leave the fields (L, H) at 0.0 if a polygonal geometry is assigned for the *part* (e.g., circle, polygon, polycurve)
  - the values are in the following units: Unit
- *double S*: thickness (>= 0.0) {*default* = 0.0}.
  - assign the field with differentiated values if it is necessary to apply a match with the corresponding *sheet* field
  - the values are in the following units: Unit
- *int N*: requested quantity for placements (>= 0) {*default* = 0; maximum value= 999}
- *int Nmax*: maximum available quantity (significant if > N) {*default* = 0; maximum value= 999} (see *ModeExtraPart* general setting)
  - the *N* field sets the requested quantity for the part. The *Nmax* field sets the maximum usable quantity and it is significant if it assigns a strictly positive value. In this case:
    - if it is *ModeExtraPart=false*: *Nmax* must assign a value greater than *N* and (*Nmax* - *N*) = usable quantity to fill the assigned *sheets*
    - if it is *ModeExtraPart=true*: usable quantity to fill the assigned *sheets*

  only after having tried placing the requested quantity of all the typologies of *parts*. The placements that correspond to the application of *Nmax* field are also called *extra placements*
- *short Fiber*: material (>= 0) {*default* = 0}

- assign the field with differentiated values if it is necessary to apply a match with the corresponding *sheet* field
- the real meaning attributed to the field belongs to the external application

- *int Colour*: generic colour information (>= -1) {*default* = -1}
  - assign the field with differentiated values if it is necessary to apply a match with the corresponding *sheet* field. The field assigns a default value =-1 to allow the assignment of an integer value that actually corresponds to a colour (in this case: 0 can correspond to *Black* colour).
  - the real meaning attributed to the field belongs to the external application
- *short Grain*: grain direction (0 = none; 1 = horizontal; 2 = vertical) {*default* = 0}
  - assign the field with value 1 or 2, if it is necessary to apply a match with the corresponding *sheet* field. A part with field (1, 2) can be placed on a *sheet* with grain only if it is possible to follow the direction of the same grain (with possible application with rotation); a part without grain can be applied on sheets with any Grain field
- *short Order*: priority of use (>= 0) {default = 0}
  - parts with higher/lower priority have placement priority in the nesting solution
  - the field application is conditioned by the general function setting ***UseOrderPart***
  - the rule for interpreting the field is assigned by the general setting ***ModeOrderItem***

- *short Rotate*: rotation (0 = none; 1 = 90°; 2 = any) {*unit*= degree and decimals of degree; *default* = 0}
  - value 2 corresponds to activating the possibility of rotation with steps of considerable angle (assigned through the ***StepAngle*** property) and it is used only in case of nesting *True Shape*. In case of nesting *Square*, the interpretation is the same as for value 1
  - value 1 corresponds to enabling a 90° rotation: in case of nesting *True Shape*, it corresponds to the possibility of a 90°-step rotation.

- *short Mirror*: mirror placement (0 = none; 1 = x, 2 = y, 3 = x+y) {*default* = 0}
  - this field assigns a possible transform to be assigned to the original part.

- *bool UseIsle*: enables to make placements in the islands of the part {*default* = false}
  - this field is significant only in case of *True Shape* nesting and only if islands are assigned for the *part*
  - the field application is conditioned by the general function setting ***PartInHole***

- *bool AutoPair*: enables the application of the *part* in *automatic cluster* mode {*default* = false}
  - this field is significant only in case of *True Shape* nesting
  - the field application is conditioned by the general function setting ***AutomaticCluster***
- *bool AutoGrid*: enables the application of the *part* by a grid development {*default* = false}
  - this field is significant only in case of *True Shape* nesting
  - the field application is conditioned by the general function setting ***AutomaticGrid***

- *double PartDiameter*: specific cutter diameter of the part {*default* = 0.0}
- *double IsleDiameter*: specific cutter diameter of the islands of the part {*default* = 0.0}
- *short Range*: grouping value between elements (parts and/or clusters) (>= 0) {*default* = 0}
  - the field is not interpreted
- *double EdgeL*, *EdgeR*, *EdgeB*, *EdgeT*: external dimensions that can be used in *Square* optimization {*default* = 0.0}

NestSheet

General assignment of a *sheet*. The structure shows initialization methods for fields with default values.

- *int ID*: numerical identifier of the *sheet* (unique, strictly positive) {*default* = 0}
  - the wording *sheet typology* is also used for the field
- *bool Enable*: enables the use of the *sheet* (false= it does not place the sheet) {*default* = true}
- *string Label*: descriptive name of the *sheet* (it can be=""; maximum length 50 characters) {*default* =""}
- *bool IsScrap*: identifies the sheet as recovered (e.g., scrap of previously optimized sheet) {*default* = false}
  - *sheets* of this type can have priority of use in the nesting solution
  - the application of the field is assigned by the setting ***UseBeforeScrap***

- *double L*: length (>= 0.0) {*default* = 0.0}
- *double H*: height (>= 0.0) {*default* = 0.0}
  - it is possible to leave the (L, H) fields at 0.0 if a polygonal geometry is assigned for the sheet (e.g.: circle, polygon, polycurve)
  - values are in the following units: Unit

- *double S*: thickness (>= 0.0) {*default* = 0.0}.
  - assign the field with differentiated values, if it is necessary to apply a match with the corresponding *part* field
  - values are in the following units: Unit
- *int N*: available quantity for placements (>= 0) {*default* = 0; maximum value= 100}
- *short Fiber*: material (>= 0) {*default* = 0}
  - assign the field with differentiated values if it is necessary to apply a match with the corresponding *part* field
  - the real meaning attributed to the field belongs to the external application
- *int Colour*: generic colour information (>= -1) {*default*= -1}
  - assign the field with differentiated values if it is necessary to apply a match with the corresponding *part* field. The field assigns a default =-1 value to allow the assignment of an integer value that actually corresponds to a colour (in this case: 0 can correspond to *Black* colour).
  - the real meaning attributed to the field belongs to the external application
- *short Grain*: grain direction (0=none; 1=x=horizontal; 2=y=vertical) {*default* = 0}
  - assign the field with value 1 or 2 if it is necessary to apply a match with the corresponding *part* field
- *short Order*: priority of use (>= 0) {*default* = 0}
  - *sheets* with higher/lower priority have use priority in the nesting solution
  - the field application is conditioned by the general function setting ***UseOrderSheet***
  - the rule for interpreting the field is assigned by the general setting ***ModeOrderItem***

NestCluster

General assignment of a *manual cluster*. The structure shows initialization methods for fields with default values.

- *int ID*: numerical identifier of the *cluster* (unique, strictly positive) {*default* = 0}
  - the wording *cluster typology* is also used for the field
- *bool Enable*: enables the *cluster* use (false = it is not used) {*default* = true}
- *string Label*: descriptive name of the *sheet* (it can be =""; maximum length 50 characters) {*default* =""}
- *double S*: thickness (>= 0.0) {*default* = 0.0}.
- *int N*: requested quantity for placements (>= 0) {*default* = 0; maximum value= 999}
- *int Nmax*: maximum available quantity (significant if > N) {*default* = 0; maximum value= 999}
- *short Fiber*: material (>= 0) {*default* = 0}
- *int Colour*: generic colour information (>= -1) {*default* = -1}
- *short Grain*: grain direction (0 = none; 1 = horizontal; 2 = vertical) {*default* = 0}
- *short Order*: priority of use (>= 0) {*default*= 0}
- *short Rotate*: rotation (0 = none; 1 = 90°; 2 = any) {*unit*= degree and decimals of degree; *default* = 0}
- *short Mirror*: mirror placement (0 = none; 1 = x, 2 = y, 3 = x+y) {*default* = 0}
- *bool UseIsle*: enables to make placements in the islands of the cluster {*default* = false}
- *bool AutoPair*: enables the application of the *cluster* in *automatic cluster* mode {*default* = false}
- *bool AutoGrid*: enables the application of the *cluster* by a grid development {*default* = false}
  - see: analogous fields in *NestPart* structure
- *short Range*: grouping value between elements (parts and/or clusters) (>= 0) {*default* = 0}
  - the field is not interpreted.

ItemCluster

General assignment of a single item in *manual cluster*. The structure shows initialization methods for fields with default values.

- *string NameID*: part identifier
  - numerical (strictly positive): corresponds to the *ID* field in the *NestPart* structure
  - otherwise: corresponds to the *Label* field in the *NestPart* structure
- *double X*, *Y*: positioning (X, Y) dimensions of the LB (left bottom) corner of the original bounding box of the part

- the dimensions are related to the XY system of Cartesian coordinates

- *short Mirror*: mirror required for the part
  - 0 = none; 1 = x; 2 = y; 3 = x+y
  - (X, Y) position the axis to apply the transform
  - the transform is applied before any rotation
- *double A*: rotation angle required for the part (unit: degrees and decimals of degree)
  - the sign assigns the rotation: positive = counter-clockwise; negative = clockwise
  - (X, Y) is the rotation centre.

NestSize

Assignment of the overall dimension of *a part*

- *double LX, LY*: minimum overall (X,Y) dimensions
- *double HX, HY*: maximum overall (X,Y) dimensions

NestGeometry

Assignment of a single geometric element in polygonal geometry

- *bool Isle*: true= geometry corresponds to an island
- *int Type*: element typology of geometry
  - *0* = geometry initial element ((X;Y)= starting point of the polygon)
  - *1* = linear element
  - *2* = arc with clockwise rotation
  - *3* = arco with counter-clockwise rotation
- *double X, Y*: final (X, Y) coordinates of the element
- *double Xc, Yc*: (X, Y) coordinates of the centre of the element (if arc).

NestBox

General assignment of a single placement on a *sheet*. The structure shows initialization methods for fields with default values

- *int ID*: numerical identifier of the *part* corresponding to the placement
- *double X, Y*: positioning (X, Y) dimensions of the LB (bottom left) corner of the original bounding rectangle of the part
- *short Mirror*: mirror required for the placement
  - 0 = none; 1 = x; 2 = y; 3 = x+y
  - (X, Y) position the axis to apply the transform
  - the transform is applied before any rotation
- *double A*: rotation angle corresponding to the placement (unit: degrees and decimals of degree)
  - the sign assigns the rotation: positive=counter-clockwise; negative=clockwise
  - (X, Y) is the rotation centre.
- *bool InIsle*: true = the placement is in an island (only in case of *True Shape* nesting)
- *string InCluster*: indicates if the placement derives from the application of a cluster
  - "": placement is single
  - otherwise: it results from the explosion of a cluster. Format: "#;ID"
    - #: progressive application of clusters (>0). Placements with the same value result from the same cluster placement
    - ID: numerical identifier of the cluster (ID field of *NestCluster* structure).

3.5 CallBack Functions

The management of the available callback functions is not necessary for the operation of TPA_N.

Progres

Function that allows managing the optimizer progression.

void Progres (int nValue, ref bool bCancel, ref bool bPause)

Arguments

- *nValue*: optimization time

- *bCancel*: return *true* to cancel the procedure
- *bPause*: return *true* to interrupt the procedure

Notes
Managing the event allows cancelling or interrupting the optimization procedure:
- cancellation results in the closure of the optimization phase with total cancellation of the procedure
- interruption results in the closure of the optimization phase once the first valid solution is completed.

In both cases, it is possible that the optimization closure may not be immediate, as the safe termination modes are activated anyhow.

The interval between consecutive occurrences of the event is governed by the TimerProgres property.

3.6 Definition of the functionalities

Let us see how to assign all the functions of Tpa_N: with reference to the functions, the wording of settings is also used.
On TPA_N's initialization, all settings are assigned to the default values, indicated here with locution {*default* = …}.

Unless otherwise specified, all the assignments described in this chapter require to be used within a section (***IniSettings -> EndSettings***), except for the properties listed in the *General functions* group.

The use of the properties in reading mode is not subject to limitation.

In order to provide a more organic picture, the assignments are divided into four groupings:

- *General functions: assignments of generic use*
- *Functions regarding the general assignments of a nesting project: assignments that have value of general customization of a nesting project.*
- *Functions regarding Square nesting: assignments to be applied to Square* optimization.
- *Functions regarding True Shape nesting*: assignments to be applied to *True Shape* optimization.

IniSettings

This function opens the assignment section of the general function settings.

bool IniSettings (int Unit, bool Clear, bool Autoconv)

Argoments
- *Unit*: linear unit (0 = mm; 1 = inch) {*default* = 0}
- *Clear*: true = assigns the settings to the default values
- *AutoConv*: true = runs the automatic conversion of the dimensional settings

Return value
True if the result is positive, *False* otherwise

Notes
A *True* return of the function allows continuing with the total or partial assignment of the general operation settings.

- if a *part*, *cluster* or *sheet* assignment phase is active: this is terminated beforehand.
- if it is *AutoConv=true*: the function runs the automatic conversion of the dimensional settings, but only if it is *Clear=false* and if *Unit* changes the unit compared to the current one. Nothing is done automatically for parts and sheets.

As for the argument *Unit*, value 0 (zero) is assessed: Unit=0 assigns [mm] unit; otherwise, it assigns [inch] unit.
A change of the linear unit requires a total assignment of:
- dimensional settings (if it is not: *AutoConv=true)*
- *part, clusters* and *sheet* lists

If it is *Clear=true*: all the settings are pre-set to default values, with the exception of some settings that remain unchanged: FixComputeError, TimerProgres, DirectoryTemp.

The assignment phase is completed with the call of the ***EndSettings()*** function.

A *False* return of the function corresponds to one of the error situations:

- a nesting optimization is running.

Query the *LastError* property to evaluate the specific error situation.

EndSettings

This function closes the assignment section of the general function settings.

bool EndSettings ()

Return value
True if the result is positive, *False* otherwise (the call has no match with ***IniSettings***).

General Functionalities

Version

String type property that returns the library version.
The string reports the major, minor, build, and revision number of the library.

IsSquareEnabled

Boolean type property that checks the validity of the basic license (*Square* optimization).

IsShapeEnabled

Boolean type property that checks the validity of the advanced license (*True Shape* optimization).

LastError

NestErrors type property that returns the last found error. The value is updated in all the procedures that can diagnose an abnormal situation.

ErrorMessage

The function returns the message assigned for the reported error.

string ErrorMessage (NestErrors Error)

Arguments
- *Error*: enumeration value

Return value
If it is *Error = NestError.ErrorNone*, the function return is "" (empty string).
Internal messages are assigned in English.

FixComputeError

Boolean type property which assigns/returns the activation to automatically resolve error situations encountered during the verification phase of the assigned lists.
The setting is currently used to solve manual cluster list checks and avoid reporting error: *NestErrors.ErrorInCluster*, *NestErrors.ErrorClusterMatch*.

TimeNesting

Integer type property that assigns/returns the maximum calculation time of nesting optimization {*unit*=seconds; *default*=30; *range of validity*: 20/600}.

The assignment is not made if a nesting optimization is running.

Particular cases are noted:
- value <=0 or =600: it assigns the maximum value (600)
- value in interval (1-20): it assigns the minimum value (20).

TimerProgres

Integer type property that assigns/returns the time interval for the generation of ***Progres*** events {*unit*=seconds; *default*=8; *range of validity*: >= 5}.
The assignment is not made if a nesting optimization is running.

A value (<=0) disables the management of the ***Progres*** event.

DirectoryTemp

String type property that assigns/returns the folder managing the temporary files {*default*=""}
If no valid folder is assigned (it assigns empty string or the folder is not accessible), it uses the storage path of the TPA_N library.
Assignment of property requires the use within a section (***IniSettings -> EndSettings***).

Functionalities regarding general assignments of a nesting project

Unit

Integer type property that returns the unit assigned with *IniSettings* function {*default* = 0}.

MinResolution

Double type property that assigns/returns the minimum resolution value {*unit*: Unit; *default* = 0.1 [mm]; *range of validity*: 1.0E-5/ 1.0 [mm]}.
The field is used to assess the validity of a rectangle (minimum value of length/height value) or of a geometric element (e.g., null linear segment or invalid arc).

OneNestingDimension

Boolean type property that assigns/returns the selection related to two different sheet types {*default* = *False*}:
- (*false*) 2D: the sheets assign a part of the plane on which nesting is two-dimensional
- (*true*) 1D: <nesting is on a linear support and the sheets assign bars, profiles, tubes.

OneCutterDimension

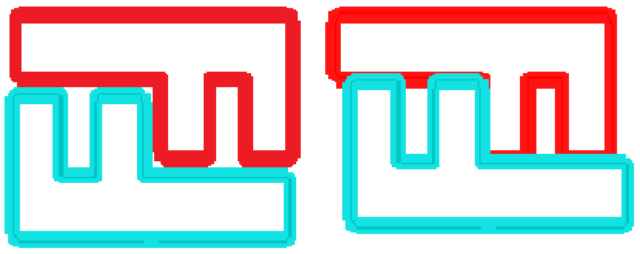
Boolean type property that assigns/returns the type of the part cutter {*default* = False}:
- (*false*) the tool has size in XY (e.g., milling cutter, laser)
- (*true*) the tool has only one size (e.g., blade, cutter).

CutterDiameter

Double type property that assigns/returns the part cutter diameter in the absence of assignments valid for the single parts {*unit*: Unit; *default*= 0.0; *range of validity*: >= 0.0}.
The placement between adjacent *parts* applies a minimum spacing corresponding to the set value.

OverrunPolygon

Boolean type property that assigns/returns the activation of the overlapping of part cutting areas {*default*= False}.
The maximum overrun allowed corresponds to the cutter diameter, with subtraction of the *Safety distance* (assigned with ***OverrunSecurity*** property).
The figure shows the difference in behavior as the property value changes.



- on the left, the placements with value *False*
- on the right, the placements with value *True*.

OverrunSecurity

Double type property that assigns/returns the *Safety distance* added to the placements, with the application of the cutting area overlap of the parts {*unit*: Unit; *default* = 0.1 [mm]; *range of validity*: 0.0/10.0 [mm]}.

OverrunSheet

Boolean type property that assigns/returns the activation for the cutting profiles of the parts to overlap the margins of the sheets {*default* = True}.

- *True*: the cutting profiles of the parts can overlap the outer margins of the sheets. The calculation of the minimum distance between a sheet edge and a placement is equal to half the cutter diameter (CutterDiameter * 0.5) or the same diameter, according to the setting MaximizeOverrunSheet.
- *False*: the cut profiles of the parts are applied within the external margins of a sheet. The minimum distance between a margin of the sheet and a placement corresponds to the cutter diameter (CutterDiameter).

As already mentioned, the determination of the minimum distance of the placements from the edges can add a further spacing considering several factors:
- assignment of margins for the use of sheets (see: MarginLeft(), ...)
- modification of profiles by applying the *chordal error*
- assignment of different technology diameters for parts.

MaximizeOverrunSheet

Boolean type property that assigns/returns the activation to maximize the overrun of the cutting profiles at the sheet edges. The setting is significant if *OverrunSheet=true*.

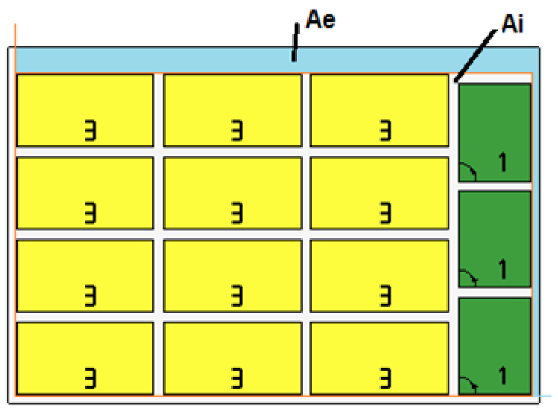
SolutionMaxScrap

Double type property that assigns/returns the value of the Maximum difference of internal scraps {*unit*: %; *default* = 2.5; *range of validity*: 0.5/50.0}.

The assessment is done by calculating the scraps as a percentage of the bounding box including the placements of a sheet. Information is used combined with the other criteria considered to determine the best choice among different solutions.

The use of information in these terms can poorly be efficient in case of a non-rectangular sheet, as the bounding rectangle of the placements may also include areas that are not useful for the placements, as external to the sheet profile.

With reference to the figure:



- the external rectangle represents the sheet
- **Ai** is the area available for the placements: it is bounded by the limit coordinates of the parts. The difference between the **Ai** area and the area of all the placements corresponds to the nesting *internal off-cuts* area
- **Ae** is the area outside the placements and corresponds to the area of the nesting *external off-cuts*

SolutionExpected

Double type property that assigns/returns the minimum expected sheet use value {*unit*: %; *default* = 75.0; *range of validity*: 25.0/95.0}.
The assessment is done by calculating the placement area as a percentage of the bounding rectangle that includes all the placements of a sheet (**Ai** in the previous image). Reaching the set value represents a condition of completion of the calculation phase, as an alternative to reaching the maximum calculation time.

ExtraFiller

Boolean type property that assigns/returns the application mode of the extra placements {*default* = *False*}.
- *False*: the extra placements are applied to fill up the sheets
- *True*: the extra placements are applied only to fill up the length or height already used by the requested workpieces. The filling direction is chosen according to the feed direction of the placements (see: Direction property):
  - if Horizontal: the filling is applied along the length of the sheet, while no limit in height is applied
  - if Horizontal: the filling is applied along the length of the sheet, while no limit in height is applied.

ModeExtraPart

Boolean type property that assigns/returns the interpretation criterion of the extra placements for parts and clusters {*default=False*}:
- *False*: the setting corresponds to the maximum total of placements that can be made (requested placements included)
- *True*: the setting corresponds to the maximum number of the placements that can be made, to be added to the requested placements.

ModeMirrorPart

Boolean type property that assigns/returns the application criterion of the mirror of parts or clusters {*default* = *False*}:
- *False*: transform application is requested and applied
- *True*: transform application is a consequence of the impossibility of making placements in non-mirror mode.

ModeOrderItem

Boolean type property that assigns/returns the application criterion for interpreting the priority of parts, clusters, sheet {*default= False*}:
- *False*: sorts by decreasing priority (elements with higher priority first)
- *True*: sorts by increasing priority (elements with lower priority first).

UseOrderPart

Boolean type property that assigns/returns the activation for the application of the *part* and *cluster* priority {*default= True*}.

- *True*: activates the application of the *Order* field, assigned in the ***NestPart*** and ***NestCluster*** structures, used to assign a part or cluster
- *False*: the structure field is ignored.

UseOrderSheet

Boolean type property that assigns/returns the activation for the application of the *sheet* priority {*default= True*}.

- *True*: activates the application of the *Order* field, assigned in the ***NestSheet*** structure, used to assign a sheet
- *False*: the structure field is ignored.

UseBeforeScrap

Boolean type property that assigns/returns the activation to use first the scrap marked sheets {*default=* False}.

- *True*: activates the use of the *IsScrap* field, assigned in the ***NestSheet*** structure, used to assign a sheet
- *False*: the structure field is ignored.

MatchType

Boolean type property that assigns/returns the activation for the *material* match application between parts and sheets {*default=* True}.

- *True*: activates the application of the *Fiber* field assigned in the structures (***NestPart***, ***NestCluster***, ***NestSheet***)
- *False*: the structure fields are ignored.

MatchColor

Boolean type property that assigns/returns the activation for the *colour* match application between parts and sheets {*default = True*}.

- *True*: activates the application of the *Colour* field assigned in the structures (***NestPart***, ***NestCluster***, ***NestSheet***)
- *False*: the structures fields are ignored.

MatchGrain

Boolean type property that assigns/returns the activation for the *grain* match application between parts and sheets {*default= False*}.

- *True*: activates the application of the *Grain* field assigned in the structures (***NestPart***, ***NestCluster***, ***NestSheet***)
- *False*: the structure fields are ignored.

UseRangePart

Short type property that assigns/ returns the criterion for the recognition of groupings of single parts and/or clusters {*default=* 0}.
***** Selection is not interpreted.

DimRangePart

Double type property that assigns/ returns an accessory value to the optimization with application of significant *UseRangePart* {*default=* 0.0}.
***** Selection is not interpreted.

RctMinimize

Boolean type property that assigns/returns the activation of the rotation search corresponding to the minimum bounding box of the *parts* {*default*= *True*}. This activation can be used for those parts with a non-rectangular primary geometry assigned and with the ability to rotate.

MarginOuter

Double type property that assigns/returns the external edges to the sheets {*unit*: Unit; *default*= 0.0; *range of validity*: >= 0.0}.
The value assigns a total of four outer margins to the sheets.

MarginLeft, MarginRight, MarginBottom, MarginTop

Double type properties that assign/return the outer margins to the sheets, on the left, right, bottom, and top respectively {*unit*: Unit; *default*= 0.0; *range of validity*: >= 0.0}.
The use of differentiated margins can be applied to rectangular *sheets*. In case of generically shaped sheets, a unique value is applied and corresponds to the maximum set among the four.

MarginInner

Double type property that assigns/returns an additional distance applied between placements {*unit*: Unit; *default*= 0.0; *range of validity*: >= 0.0}.

Direction

Short type property that assigns/returns the feed direction for placements {*default*= 0; *range of validity*: 0/1}:
- 0 = horizontal direction
- 1 = vertical direction

In case of *OneNestingDimension*=*true* setting: the direction is applied with value 1 (vertical direction).

Corner

Short type property that assigns/returns the starting corner for placements among the valid values {*default*= 0; *range of validity*: 0/3}
- 0 = Left-Bottom
- 1 = Left-Top
- 2 = Right-Bottom
- 3 = Right-Top.

Nesting Square functionalities

PartSortMode

Short type property that assigns/returns the sorting criterion for the parts prior to the placements {*default*= 0, *range of validity*: 0/1}:
- *0*: sorts by decreasing area (large workpieces first). The area is evaluated according to the nesting direction:
  - if horizontal: it sorts by decreasing height
  - if vertical: it sorts by decreasing length
- *1*: sorts by increasing area (small workpieces first).

Nesting True Shape functionalities

StepAngle

Double type property that assigns/returns the minimum rotation angle {*default*= 5.0; *unit*= degree and decimals of degree; *range of validity*: 1.0/90.0}.

The set value corresponds to value 2 in the *Rotate* field assigned in the ***NestPart*** structure used to assign a part and in the NestCluster structure used to assign a cluster.

PartInHole

Boolean type property that assigns/returns the activation to make placements within the scrap areas of the parts {*default*= *True*}.

- *True*: activates the application of the field *UseIsle=true* assigned in the ***NestPart*** structure, used to assign a part and in *NestCluster* structure of cluster assignment.

PartInHoleMulti

Boolean type property that assigns/returns the activation to make *recursive* placements within the scrap areas of the parts {*default*= *False*}.

PartInHoleBefore

Boolean type property that assigns/returns the activation to *privilege* the placements within the scrap areas of the parts {*default*= *False*}.

AutomaticCluster

Boolean type property that assigns/returns the activation to apply *Automatic clusters* {*default*= *True*}.

- *True*: activates the application of the field *AutoPair=true* assigned in the ***NestPart*** structure used to assign a part and in the *NestCluster* structure used to assign a cluster.

ClustersExpected

Double type property that assigns/returns the area minimum use value (in %) of an automatic cluster of parts with respect to the single placements {*default*= 50.0; *range of validity*: 50.0/95.0}.

This setting assigns the minimum efficiency assessed for the *Automatic cluster* of a part, which is calculated as:

(Area of the single part * 2 * 100) / (Group length * Group height)

ClustersAbsolute

Double type property of similar meaning to the previous one (*ClustersExpected*) for the autonomous application of automatic clusters {*default*= 50.0; *range of validity*: 50.0/100.0}.

AutomaticGrid

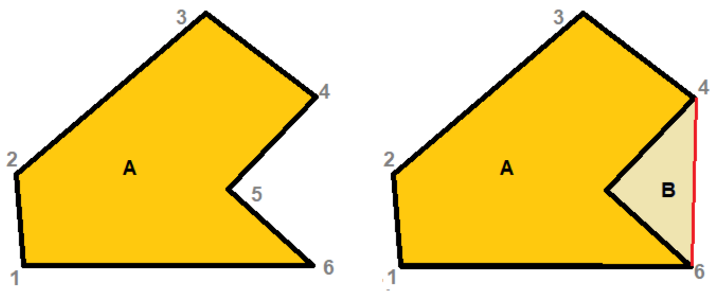
Boolean type property that assigns/returns the activation of the application of *Grid placements* {*default*= *True*}.

- *True*: activates the application of the field *AutoGrid=true* assigned in the ***NestPart*** structure, used to assign a part and in the *NestCluster* structure sued to assign a cluster
- *False*: excludes the application of programmed grid placements.

ExploreConcave

Boolean type property that assigns/returns the activation to explore the concavities of a part {*default*= True}.
In the figure, the case of a concave part:

- *True*: the part is used as assigned (left of the figure: vertices indicated from 1 to 6)
- *False*: the part is used by deleting the concavity. For the purpose of the nesting procedure, the part is modified as shown on the right of the figure: the list of vertices deletes point 5, and the overall dimension of the part adds the area indicated with the letter B.



The exclusion to the use of the concavity zones speeds up the calculation procedures, to the detriment of a lower efficiency of use of the material.

ConcaveDimension

Double type property that assigns/returns a length used to reduce the concavity of a part {*unit*: Unit; *default*= 1.0; *range of validity*: >= 0.0}.
The value is used in case of ***ExploreConcave***=*true*: the nesting procedure explores the concave areas, but it simplifies them.
With reference to the concavity shown in the previous figure, the elimination of the concavity is conditioned to verifications:

- the area of part B is small compared to the area of the circle whose radius is equal to the set value; or
- the distance between vertex 5 and the segment connecting the (4, 6) vertices is less than the set value.

Settings serialization functions

SaveSettings

The function saves all settings to file (XML format)

NestErrors SaveSettings (string pathName, bool bMode)

Arguments

- *pathName*:	file path
- *bMode*:	*** argument available

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings***, ***IniSetPart***, ***IniSetSheet***, ***IniSetCluster***)
- (*NestError. ErrorIOProject*) an error occurred when accessing the file to be written.

However, some settings are excluded from saving: FixComputeError, TimerProgres, DirectoryTemp.

LoadSettings

The function reads all settings from file (XML format).

NestErrors LoadSettings (string pathName, bool bMode)

Arguments

- *pathName*:	file path
- *bMode*:	*** argument available

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) a nesting optimization is running

- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings***, ***IniSetPart***, ***IniSetSheet***, ***IniSetCluster***)
- (*NestError.ErrorIOProject*) an error occurred when accessing the file to be read.

All settings are previously assigned to default values.
The reading excludes some settings which remain unchanged: FixComputeError, TimerProgres, DirectoryTemp.

3.7 Definition of parts

IniSetPart

This function opens the assignment section of the parts.

bool IniSetPart (bool Clear)

Arguments

- *Clear*: true= resets the part list

Return value
True if the result is positive, *False* otherwise

Notes
This function returning *True* allows continuing with the total or partial assignment of the parts.
The assignment phase is completed with the call of the ***EndSetPart()*** function.

This function returning *False* corresponds to one of the error situations:
- a serious system status occurred resulting in a memory allocation error
- nesting optimization is running.

Query the *LastError* property to evaluate the specific error situation.

EndSetPart

This function closes the assignment section of the parts.

bool EndSetPart ()

Return value
True if the result is positive, *False* otherwise (the call has no match with ***IniSetPart*** or the part list is not valid).

AddPart

The function adds a part to the list.

NestErrors AddPart (NestPart Item)

Argomenti

- *Item*: part assignment structure

Return value
NestError.ErrorNone if the result is positive

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorContext*) the function is not used within a section (***IniSetPart -> EndSetPart***)
- (*NestError.ErrorPartsTomany*) the part list is already at the allowed maximum (500 items) or the quantity requested in the structure exceeds the maximum allowed (999)
- (*NestError.ErrorUnexpected*) a part is assigned a no valid identifier (***Item.ID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) a part with numerical identifier ***Item.ID*** results as already assigned.

The assignment of the part can use values that are modified compared to the structure, in case of invalid data.

RemovePart

The function eliminates a part or the geometries of a part.

NestErrors RemovePart (int ItemID, bool OnlyGeometry)

Arguments
- *ItemID*: part identifier (>0)
- *OnlyGeometry*: true = it eliminates only the additional geometry assignments (if assigned)

Return value
NestError.ErrorNone if the result is positive

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetPart -> EndSetPart***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with numerical identifier ***ItemID*** results as assigned.

WritePart

The function changes a part already assigned.

NestErrors WritePart (NestPart Item)

Arguments
- *Item*: part assignment structure

Return value
NestError.ErrorNone if the result is positive

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetPart -> EndSetPart***)
- (*NestError.ErrorUnexpected*) no part with numerical identifier ***Item.ID*** results as already assigned.

The assignment of the part can use values that are modified compared to the structure, in case of invalid data.

CountPart

The ***integer*** type property gets the number of parts in the list.
Using the function does not require working within a section (***IniSetPart -> EndSetPart***).

ReadPart

The function searches for a part corresponding to the specified ID.

NestErrors ReadPart (**int ItemID, ref** ***NestPart Item, ref NestSize ItemSize)***

Arguments
- *ItemID*: part identifier (>0)
- *Item*: part assignment structure
- *ItemSize*: assignment structure of the bounding rectangle used by the optimization procedure

Return value
NestError.ErrorNone if the result is positive

Notes
Using the function does not require working within a section (***IniSetPart -> EndSetPart***).

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorBusy*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with numerical identifier ***ItemID*** results as assigned.

ReadPartIndex

The function searches for a part corresponding to the specified index.

NestErrors ReadPartIndex (*int Index, ref *NestPart Item, ref NestSize ItemSize)

Arguments

- *Index*: (zero base) index in the part list (>= 0)
- *Item*: part assignment structure
- *ItemSize*: assignment structure of the bounding rectangle used by the optimization procedure

Return value

NestError.ErrorNone if the result is positive

Notes

Using the function does not require working within a section (***IniSetPart -> EndSetPart***).
The primary use of this function is to acquire the parts after executing the ***LoadProject*** function.

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid index is assigned.

3.8 Definition of sheets

IniSetSheet

This function opens the assignment section of the sheets.

bool IniSetSheet (bool Clear)

Arguments

- *Clear*: true = resets the sheet list

Return value

True if the result is positive, *False* otherwise

Notes

The function returning *True* allows continuing with the total or partial assignment of the sheets.
The assignment phase is completed with the call of the ***EndSetSheet()*** function.

The function returning *False* corresponds to one of the error situations:

- a serious system status occurred resulting in a memory allocation error
- nesting optimization is running.

Query the LastError property to evaluate the specific error situation.

EndSetSheet

This function closes the assignment section of the sheets.

bool EndSetSheet ()

Return value

True if the result is positive, *False* otherwise (the call has no match with ***IniSetSheet***, or the sheet list is not valid).

AddSheet

The function adds a sheet to the list.

NestErrors AddSheet (NestSheet Item)

Arguments

- *Item*: sheet assignment structure

Return value
NestEError.ErrorNone if the result is positive

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetSheet -> EndSetSheet***)
- (*NestError.ErrorSheetsTomany*) the sheet list is already at the allowed maximum (100 items) or the quantity requested in the structure exceeds the maximum allowed (999)
- (*NestError.ErrorUnexpected*) the sheet is assigned a no valid identifier (***Item.ID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) the sheet with numerical identifier ***Item.ID*** results as already assigned.

The assignment of the sheet can use values that are modified compared to the structure, in case of invalid data.

RemoveSheet

The function deletes a sheet or the geometries of a sheet.

NestErrors RemoveSheet (int ItemID, bool OnlyGeometry)

Arguments

- *ItemID*:	sheet identifier (> 0)
- *OnlyGeometry*:	true= deletes only the additional geometry assignments (if assigned)

Return value
NestError.ErrorNone if the result is positive

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetSheet -> EndSetSheet***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no sheet with number identifier ***ItemID*** results as assigned.

WriteSheet

The function changes a sheet already assigned.

NestErrors WriteSheet (NestSheet Item)

Arguments

- *Item*:	sheet assignment structure

Return value
True if the result is positive, *False* otherwise

Notes
The function returning *False* corresponds to one of the error situations:

- (*NestError.ErrorContext*) the function is not used within a section (***IniSetSheet -> EndSetSheet***)
- (*NestError.ErrorUnexpected*) no sheet assigned with numerical identifier ***Item.ID*** results as already assigned.

The assignment of the sheet can use values that are modified compared to the structure, in case of invalid data.

CountSheet

Integer type property gets the number of sheets in the list.
Using the function does not require working within a section (***IniSetSheet -> EndSetSheet***).

ReadSheet

The function searches for a part corresponding to the specified ID.

NestErrors ReadSheet (**int ItemID, ref** ***NestSheet Item, ref NestSize ItemSize)***

Arguments

- *ItemID*: sheet identifier (> 0)
- *Item*: sheet assignment structure
- *ItemSize*: assignment structure of the bounding rectangle used by the optimization procedure

Return value
True if the result is positive, *False* otherwise.

Notes
Using the function does not require working within a section (***IniSetSheet -> EndSetSheet***).

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no sheet with numerical identifier ***ItemID*** results as assigned.

ReadSheetIndex

The function searches for a sheet corresponding to the specified index.

NestErrors ReadSheetIndex (**int Index, ref** ***NestSheet Item, ref NestSize ItemSize)***

Arguments
- *Index*: (zero base) index in the sheet list (>= 0)
- *Item*: sheet assignment structure
- *ItemSize*: structure assigning the bounding rectangle used by the optimization procedure.

Return value
NestError.ErrorNone if the result is positive.

Notes
Using the function does not require working within a section (***IniSetSheet -> EndSetSheet***).
The primary use of this function is to acquire the sheets after executing the ***LoadProject***. function.
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid index is assigned.

3.9 Definition of polygonal geometries

We will examine now how to assign the part and/or sheet geometries (*shape*).
As a simplification, it will be assumed below we are working in the part list assignment.

IniGeometry

This function opens the assignment section of a polygonal geometry of a part.

bool IniGeometry (int ItemID, bool IsIsle, double X, double Y)

Arguments
- *ItemID*: part identifier (> 0)
- *IsIsle*: true = the assignment corresponds to a scrap area (of primary geometry already assigned)
- *X, Y*: starting coordinates of geometry (values are in unit of: Unit)

Return value
True if the result is positive, *False* otherwise

Notes
The function returning *True* allows continuing with the assignment.
The assignment phase is completed with the call of ***EndGeometry()*** function.

The function returning *False* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetPart -> EndSetPart***, or ***IniSetSheet -> EndSetSheet***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with numerical identifier ***ItemID*** results as assigned

- (*NestError.ErrorUnexpected*) if it is *IsIsle=false*: no primary profile must result as already assigned

Query the *LastError* property to evaluate the specific error situation.

If it is *IsIsle=true* and no primary profile is assigned, this is automatically generated with rectangular shape and dimensions (length * height) assigned for the part.

EndGeometry

This function closes the assignment section of polygonal geometries of a part.

bool EndGeometry ()

Return value
True if the result is positive, *False* otherwise.

Notes
The function returning *False* corresponds to one of the error situations:
- the call has no correspondence with ***IniGeometry***
- the list of geometries assigned to the part is not valid.

The function runs a general validity control of all the geometries assigned to the part and not only of the single geometry assigned in the current session, for each geometry:
- at least one curved element or two linear elements must be assigned.

AddToGeometry_Line

The function adds a linear element to a polygonal geometry.

NestErrors AddToGeometry_Line (double Xend, double Yend)

Arguments

- *Xend*:	final X coordinate of the segment (unit: Unit)
- *Yend*:	final Y coordinate of the segment (unit: Unit)

Return value
NestError.ErrorNone if the result is positive.

Notes
Any function return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorInGeometry*) the linear element has null length (<=***MinResolution***).

The start of the linear segment corresponds to the final point of the previous segment.

AddToGeometry_Arc

The function adds a curved element (arc) to polygonal geometry.

NestErrors AddToGeometry_Arc (double Xend, double Yend, double Xcentre, double Ycentre, bool IsCCW)

Arguments

- *Xend, Yend*:	final (X, Y) coordinates of the segment (unit: Unit)
- *Xcentre, Ycentre*:	centre (X, Y) coordinates of the segment (unit: Unit)
- *IsCCW*:	true = counter-clockwise rotation

Return value

NestError.ErrorNone	if the result is positive.

Notes
The function returning *False* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorInGeometry*) the curved element is not valid (radius<= ***MinResolution*** or |initial radius - raggio finale| > ***MinResolution***).

The start of the curved segment corresponds to the final point of the previous segment.

AddToGeometry_Circle

The function adds a circle element to polygonal geometry.

NestErrors AddToGeometry_Circle (double Xcentre, double Ycentre, bool IsCCW)

Arguments

- *Xcentre, Ycentre*:	centre (X, Y) coordinates of the segment (unit: Unit)
- *IsCCW*:	true = counter-clockwise rotation

Return value
NestError.ErrorNone if the result is positive.

Notes
See the ***AddToGeometry_Arc*** function.

AddToGeometry

The function adds a generic element to polygonal geometry.

NestErrors AddToGeometry (NestGeometry Item)

Arguments

- *item*:	assignment structure of the geometric element.

Return value
NestError.ErrorNone if the result is positive.

Notes
See the ***AddToGeometry_**** functions.

Reading the geometries

ReadGeometry

The function recursive call reads the geometries that result as assigned for a part

NestErrors ReadGeometry (int List, int ItemID, int IndexItem, int IndexElement, ref NestGeometry Item)

Arguments

- *List*:	selects the list: 0= parts; 1= sheets
- *ItemID*:	element identifier (part or sheet) (> 0)
- *IndexItem*:	(zero base) index on the list of the geometric profiles assigned
- *IndexElement*:	(zero base) index of the geometric element in geometric profile
- *Item*:	assignment structure of the geometric element.

Return value
NestError.ErrorNone if the result is positive.

Notes
The primary use of this function is to acquire the geometries after executing the ***LoadProject*** function.

In correspondence to the assigned values for (*List*, *ItemID*), the first call mus use (*IndexItem=0, IndexElement=0*). Any return of this first call different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorUnexpected*) invalid element identifier or element without geometry assigned (neither primary nor secondary).

Use of external geometries

ShapeExtension

String type property that returns the list of supported file extensions for interpreting geometries from external file. Let us see the possible cases at the moment:

- ="":	no valid recognition. A call to the *ReadShape* function fails
- =*".DXF"*:	DXF file reading is recognized
- =*".DXF;.DWG"*	DXF and DWG file reading is recognized.

ReadShape

The function interprets a geometry from DXF or DWG files.

NestErrors ReadShape (string pathOpen)

Arguments
- *pathOpen*: complete path of the file to be read (e.g., "C:\PATTERN\A.DXF").

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ERR_NONE* corresponds to one of the error situations:
- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry -> EndGeometry***)
- (*NestError.ErrorLicense*) negative result occurred when verifying the existence and status of the advanced function key
- (*NestError.ErrorIOfile*) error occurred when accessing or interpreting the *pathOpen* file
- (*NestError.ErrorIOfile*) error occurred when managing temporary files.

The reading of DWG files is furthermore conditioned by the verification of a specific activation of the hardware key.

The function does not change any assignment of the part geometries: it runs a format interpretation and loads the read geometries, making them available for a later reading.

Let us see now the general rules of interpretation of a drawing file:
- a 2D drawing is interpreted
- only profile-assigning geometric entities are interpreted (polylines, circles, ellises, splines)
- only closed profiles are recognized as valid (the condition of closed profile is assessed by applying a maximum distance equal to the cutter diameter: see ***CutterDiameter***)
- only 1 master profile (the one with the greatest area) and its possible isles are kept, assigned to the first internal layer.

ReadGeoInShape

The function recursive call reads the geometries assigned with call to the *ReadShape* function.

bool ReadGeoInShape (int IndexItem, int IndexElement, ref NestGeometry Item)

Arguments

- *IndexItem*:	(zero base) index on the list of the geometric profiles assigned
- *IndexElement*:	(zero base) index of the geometric element in geometric profile
- *Item*:	assignment structure of the geometric element.

Return value
True if the result is positive, *False* otherwise.

Notes
The first call must use (*IndexItem=0, IndexElement=0*). Any return to this first *false* call corresponds to one of the error situations:
- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorUnexpected*) no geometry is assigned (neither primary nor secondary).

Query the *LastError* propperty to evaluate the specific error situation.

3.10 Definition of manual clusters

IniSetCluster

This function opens the assignment section of the clusters.

bool IniSetCluster (bool Clear)

Arguments
- *Clear*: true = resets the cluster list.

Return value
True if the result is positive, *False* otherwise.

Notes
This function returning *True* allows continuing with the total or partial assignment of the clusters.
The assignment phase is completed with the call of the ***EndSetCluster ()*** function.

This function returning *False* corresponds to one of the error situations:
- a serious system status occurred resulting in error with memory allocation
- nesting optimization is running.

Query the *LastError* property to evaluate the specific error situation.

EndSetCluster

This function closes the assignment section of the clusters.

bool EndSetCluster ()

Return value
True if the result is positive, *False* otherwise (the call has no match with ***IniSetCluster*** or the cluster list is not valid).

AddCluster

The function adds a cluster to the list.

NestErrors AddCluster (NestCluster Item)

Arguments
- *Item*: cluster assignment structure

Return value
NestError.ErrorNone if the result is positive.

Notes
A positive return of the function allows continuing with the assignment of the geometric composition of the cluster.

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetCluster -> EndSetCluster***)
- (*NestError.ErrorClustersTomany*) the cluster list is already at the allowed maximum (500 items) or the quantity requested in the structure exceeds the maximum allowed (999)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned to the cluster (***Item.ID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) a cluster with numerical identifier ***Item.ID*** results as already assigned.

The assignment of the cluster can use values that are modified compared to the structure, in case of invalid data.
The new cluster has no geometric composition assigned.

AddToCluster

The function adds an element to define the geometric composition of the cluster.

NestErrors AddToCluster (int ItemID, ItemCluster Item)

Arguments
- *ItemID*: cluster identifier (> 0)
- *Item*: part assignment structure.

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorContext*) the function is not used within a section (***IniSetCluster -> EndSetCluster***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no cluster with numerical identifier ***ItemID*** results as assigned
- (*NestError.ErrorInCluster*) no part with numerical identifier ***Item.NameID*** results as assigned (please remember that the field can assign a numerical identifier or text alias)
- (*NestError.ErrorInCluster*) the list of elements that assign the composition of the cluster is already at the maximum allowed (100).

RemoveCluster

The function deletes a cluster or the elements that define its composition.

NestErrors RemoveCluster (int ItemID, bool OnlyGeometry)

Arguments
- *ItemID*: cluster identifier (> 0)
- *OnlyGeometry*: true = deletes only added geometry assignments (if assigned).

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetCluster -> EndSetCluster***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no cluster with numerical identifier ***ItemID*** results as assigned.

WriteCluster

The function changes a cluster already assigned.

NestErrors WriteCluster (NestCluster Item)

Arguments
- *Item*: cluster assignment structure.

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetCluster -> EndSetCluster***).
- (*NestError.ErrorUnexpected*) no cluster with numerical identifier ***Item.ID*** results as assigned.

The assignment of the cluster can use values that are modified with respect to the structure, in case of invalid data.
Added geometry assignments do not change.

CountCluster

The ***integer*** type property gets the number of clusters in the list.
Using the function does not require working within a section (***IniSetCluster -> EndSetCluster***).

ReadCluster

The function searches for a cluster corresponding to the specified ID.

NestErrors ReadCluster (int ItemID, ref NestCluster Item)

Arguments

- *ItemID*: cluster identifier (> 0)
- *Item*: cluster assignment structure.

Return value
NestError.ErrorNone if the result is positive.

Notes
Using the function does not require working within a section (***IniSetCluster -> EndSetCluster***).

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no cluster with numerical identifier ***ItemID*** results as assigned.

ReadClusterIndex

The function searches for a cluster corresponding to the specified index.

NestErrors ReadClusterIndex (int Index, ref NestCluster Item)

Arguments

- *Index*: (zero base) index in the cluster list (>= 0)
- *Item*: cluster assignment structure.

Return value
NestError.ErrorNone if the result is positive.

Notes
Using the function does not require working within a section (***IniSetCluster -> EndSetCluster***).

The primary use of this function is to acquire the clusters after executing the ***LoadProject*** function.

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorUnexpected*) no valid index is assigned.

ReadInCluster

The recursive call of the function reads the composition elements that are assigned to a cluster.

NestErrors ReadInCluster (int ItemID, int IndexItem, ref ItemCluster Item)

Arguments

- *ItemID*: cluster identifier (> 0)
- *IndexItem*: (zero base) index on the list of elements assigned in cluster
- *Item*: element assignment structure.

Return value
NestError.ErrorNone if the result is positive.

Notes
The primary use of this function is to acquire the clusters after executing the ***LoadProject*** function.

In correspondence to the assigned value of (*ItemID*), the first call must use (*IndexItem=0*). Any return to this first call different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorUnexpected*) an invalid cluster identifier is assigned or cluster without elements assigned.

3.11 Nesting solution

Compute

The function starts the optimization procedure.

NestErrors Compute (int OptiSelect, bool OnlyTest, bool StepByStep)

Arguments

- *OptiSelect*: selects the optimization type (-1= automatic; 0= Square; 1= True Shape)
- *OnlyTest*: true = runs only assignments prior to the optimization
- *StepByStep*: selects the type of optimization working.

Return value
NestError.ERR_NONE if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is already running
- (*NestError.ErrorLicense*) a negative result occurred when verifying the existence and status of the basic function key
- (*NestError.ErrorLicenseHight*) a *True Shape* optimization is requested and the key status does not match the advanced function
- (*NestErrors.ErrorPartsEmpty*) empty part list or no enabled items
- (*NestErrors.ErrorSheetsEmpty*) empty sheet list or no enabled items
- (*NestErrors.ErrorInCluster*) error in manual cluster assignment (part or minimum quantity not available)
- (*NestErrors.ErrorClusterMatch*) error in assignment of clusters, in the verification of specific matches
- (*NestError.ErrorReset*) the procedure was cancelled by the calling application.

At the start, the function

- closes a possible open assignment section (settings, parts, sheets, clusters)
- runs the preliminary checks (as mentioned above)
- runs checks on the part, clusters and sheet lists
- evaluates the type of optimization requested, based on the key verification and sheet and part lists.

If the execution of the listed tests brings to an error situation, the function returns anyway, executing no optimization.
Otherwise, it continues according to ***OnlyTest***:

- ***true***: does not change the current optimization status and simply returns the result (positive or error)
- ***false***: cancels the executed optimization status and starts a new optimization.

The library runs a *Square* optimization:

- if all the *parts* and the *sheets* are assigned as simple rectangles and no cluster is assigned, it is *OptiSelect=-1*
- if it is *OptiSelect=0*
- if the advanced function existence and key status check fails

Running the *Square* optimization:

- the *parts* assigned with a geometric profile are considered for the bounding rectangle of the same profile, and the profiles assigned as islands are ignored.
- the *sheets* assigned with a geometric profile are considered for the bounding rectangle of the same profile and the profiles assigned as constraint areas are ignored: in this case, the optimization solution can perform placements in areas outside the sheet and/or within the constraint areas
- the clusters are considered for the overall bounding box of the geometric composition assigned
- it is not possible to calculate progressive optimizations.

IsComputed

Boolean type property that returns the information of optimization executed.
The property value is significant after a call to the ***Compute*** function with ***OnlyTest=false*** and optimization executed.

ModeCompute

Integer type property that returns the information applied to the last executed optimization: 0= *Square* type; 1= *True Shape* type.
The property value is significant after a call to the ***Compute*** function with ***OnlyTest=false*** and optimization executed.

TimeCompute

Double type property that returns the calculation time relating to the last execution of the ***Compute*** function or ***RetryCompute*** {*unit*= seconds}.
The value of this property is significant after executing an optimization.

TimeOut

Boolean type property that returns the information if the computing procedure has ended for reaching the time-out.
The value of this property is significant after executing an optimization.

CanRetry

Boolean type property that returns the information if it is possible to request a new optimization attempt, starting from the last solution found.
The conditions for being able to calculate more solutions are:

- an optimization in *True Shape* mode must result executed and complete
- the component must have no changes assigned (general settings, part and/or sheet and/or cluster list)
- the maximum managed number of solutions has not already been calculated (assigned= 20)
- the optimization procedure has operating margins for new calculation attempts.

RetryCompute

The function starts the optimization procedure starting from the last solution found.

NestErrors RetryCompute (bool StepByStep, bool RetryBest)

Arguments

- *StepByStep*: selects the type of optimization working
- *RetryBest*: true = assigns a new solution only if better than the previous one

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- see cases of ***CanRetry*** property
- (*NestError.ErrorReset*) the procedure was cancelled by the calling application.

The value of *RetryBest* selects between two possible operations:

- *false*: if the optimization is correctly executed, the calculated solution becomes the current one
- *true*: the calculated solution becomes the current one only if it is assessed *better* than the previous one.

The value of *StepByStep* selects between two possible operations of the optimization procedure:

- *false*: the optimization stops according to the maximum time assigned
- *true*: the optimization stops at the first solution that is calculated.

ClearSolution

The function resets any solutions calculated.

NestErrors ClearSolution ()

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running.

3.12 The nesting results

All the properties and functions listed here fail in one of the error situations:

- nesting optimization is running
- no valid optimization is calculated.

Solution

Integer type property that assigns/returns the number of the current solution.
The returned value is:

- 0: no solution is calculated
- 1: the current solution corresponds to the first one (with call to the ***Compute()*** function)
- >1: the current solution corresponds to one calculated with call to the ***RetryCompte()*** function.

While assigning:

- the property allows changing the *current solution* to one already calculated
- no assignment is done if the value is not valid for defining a solution.

OfSolution

Integer type property that returns the total number of calculated solutions.
The returned value is:

- 0: no solution is calculated
- 1: only the solution with call to the ***Compute()*** function is calculated
- >1: more solutions are calculated (***Compute()*** followed by calls to the ***RetryCompute()*** function).

Fitness

Double type property that returns the information on efficiency of the *current solution*.
The value of this property is significant after executing an optimization.
The total efficiency is evaluated in % of the ratio between the area used for placements and the total area of the panels used. If the solution generates more sheets, the efficiency evaluation of the last sheet can suitably limit the useful area for placements, if assigned with a rectangular geometry.

ReadNumResult

The function reads the number of sheets of the *current solution* (sheets with placements).

int ReadNumResult (int ID_Sheet)

Arguments

- *ID_Sheet*: sheet identifier (> 0)

Notes
The function returns the number of sheets of the current solution or sheet type:

ID_Sheet	
<=0	it gets the total number of sheets calculated for the solution
>0	it gets the number of sheets of (*ID_Sheet*) type calculated for the solution

ReadNumPartInResult

The function reads the number of placements in a sheet of the current solution or on the entire solution, with the possibility of matching only one type of part

int ReadNumPartInResult (int Index, int ID_Part)

Arguments
- *Index*: zero base index of the solution sheet
- *ID_Part*: part identifier (> 0)

Notes
The function reads the number of placements matching the arguments assigned:

Index	*ID_Part*	
-1	0	it gets the total number of the solution placements
-1	>0	it gets the number of placements matching the part (*ID_Part*) in all the solution sheets
>=0	0	it gets the total number of placements in the (*Index*) index sheet of the solution
>=0	>0	it gets the number of placements matching the part (*ID_Part*) in the (*Index*) index sheet

In case of operation with negative *Index* (-1), the returned value considers equal repeated sheets.
In case of operation with *Index* >= 0, the returned value counts the placements of a single sheet, without considering any repetitions of the same sheet.

ReadNumClusterInResult

The function reads the number of cluster placements in a sheet of the current solution or on the entire solution, with the possibility of matching only one type of cluster.

int ReadNumClusterInResult (int Index, int ID_Item)

Arguments
- *Index*: zero base index of the solution sheet
- *ID_Item*: cluster identifier (>0).

Notes
The function reads the number of cluster placements matching the assigned arguments:

Index	*ID_Item*	
-1	0	it gets the total number of the solution cluster placements
-1	>0	it gets the number of placements matching the cluster (*ID_Item*) in all the solution sheets
>=0	0	it gets the total number of cluster placements in the (*Index*) index sheet of the solution
>=0	>0	it gets the number of placements matching the cluster (*ID_Item*) in the sheet with (*Index*) index

In case of operation with negative *Index* (-1), the returned value considers equal repeated sheets.
In case of operation with *Index* >=0, the returned value counts the placements of a single sheet, without considering any repetitions of the same sheet.

ReadResult

The function returns the information concerning a sheet of the *current solution* matching the index specified

bool ReadResult (int Index, ref int ID_Sheet, ref int Item_Sheet, ref double AreaPercent, ref int NumPlaces, ref int Repetition)

Arguments

- *Index*: (zero base) index of the solution sheet
- *ID_Sheet*: numerical identifier of the *sheet* (*ID* field in *NestSheet* structure)
- *Item_Sheet*: occurrence of the sheet of sheet number identifier type
- *AreaPercent*: occupied area/total area
- *NumPlaces*: number of parts placed on the sheet
- *Repetition*: number of sheet repetitions.

Return value
True if the sheet is identified as valid.

ReadPartInResult

The function returns the information concerning a placement on a sheet of the *current solution.*

bool ReadPartInResult (int Index, int IndexPart, ref NestBox Item)

Arguments

- *Index*: (zero base) index of the solution sheet
- *IndexPart*: (zero base) index of the placement on the solution sheet
- *Item*: placement assignment structure

Return value
True if the placement on the sheet is identified as valid.

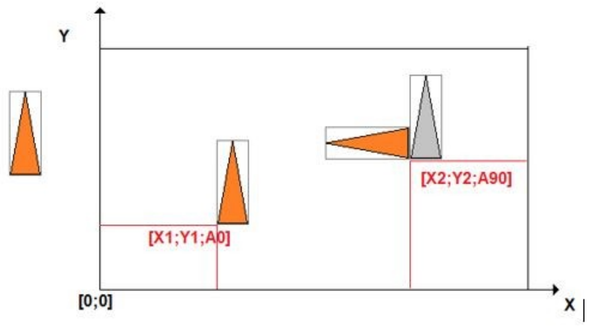
Notes
As already mentioned, the *Item* structure returns the information concerning the placement:

- *int ID*: numerical identifier of the *part* matching the placement
- *double (X, Y)*: placement coordinates of the LB (bottom left) corner of the original bounding rectangle of the part
- *short Mirror*: mirror required for the placement
  - 0=none; 1=x, 2=y, 3=x+y
  - (X, Y) position the axis to apply the transform
  - the transform is applied before any rotation
- *double A*: rotation angle corresponding to the placement (unit: degrees and decimals of degree)
  - the sign assigns the rotation: positive= counter-clockwise; negative= clockwise
  - (X, Y) is the rotation center
  - A assigns a relative angle: the part rotates A around (X, Y)

- *bool InIsle*: true= the placement is in an island (only in case of nesting *True Shape*)
- *string InCluster*: indicates if the placement results from the application of a cluster
  - "": placement is single
  - otherwise: it results from the explosion of a cluster. Format: "#;ID"
    - #: progressive application of clusters (>0). Placements with the same value result from the same cluster placement
    - ID: numerical identifier of the cluster (*ID* field in *NestCluster* structure)

There are many factors that contribute to determine the mirror and rotation transforms of a single placement:

- direct assignment of the source or original cluster: fields (*Mirror*, *Rotate*) in element structures
- ModeMirrorPart setting
- geometric composition in the original cluster.



ReadGeoInResult

The function recursive call reads the geometries that result as assigned for the current placement.

bool ReadGeoInResult (int IndexItem, int IndexElement, ref NestGeometry Item)

Arguments

- *IndexItem*:	(zero base) index on the list of the geometric profiles assigned
- *IndexElement*:	indice (base zero) dell'elemento geometrico in profilo geometrico
- *Item*:	struttura di assegnazione dell'elemento geometrico

Return value
True if the placement on the sheet is identified as valid.

Notes
The function must be called right after a call to the ***ReadPartInResult*** function, and the first call must use (*IndexItem=0, IndexElement=0*).

Any *False* return to this first call corresponds to one of the error situations:
- nesting optimization is running or is not valid
- no current placement results as valid.

The function allows reconstructing the *shape* associated to the current placement, with all the necessary transforms applied: translation, mirroring, rotation.

In case of shape of the part assigned as pure rectangle (size in the *L,H* fields of the part), the function recursive call returns the four linear elements corresponding to the development of the rectangle.

SaveSolution

The function saves the *current solution* to a file (XML format).

NestErrors SaveSolution (string pathName)

Arguments

- *pathName*:	file path

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorNoneSolution*) no current solution resulting
- (*NestError.ErrorIOfile*) an error occurred in accessing the file to be written.

The saved file can be interpreted by an external application as an alternative to the function query.

For the sheet placements, the file saves the basic information of each placement as resulting from ***NestBox*** structure.

SaveSolutionDXF

The function saves the *current solution* in one or more files (DXF format).

NestErrors SaveSolutionDXF (string pathName, bool autoClose, string layerSheet)

Arguments

- *pathName*:	file path
- *autoClose*:	true = generates always closed profiles corresponding to the parts
- *layerSheet*:	name of the assignment layer of the sheet assignment profile(s).

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations already mentioned for the *SaveSolution* function.

If *pathName* assigns an extension other than ".DXF", the same is added to the full file name.

The function saves a file for each sheet of the *current solution* (sheets with placements).
A sheet assigned with repetitions is saved only once.
The first sheet is saved as *pathName*. The remaining sheets modify the original name by adding "_n", where it is n= progressive (1, 2, …).

If *layerSheet* is assigned as significant (i.e., "sheet"):
- the file also assigns the polyline corresponding to the sheet definition, with the layer field corresponding to the string indicated
- additional polylines with the same layer can assign islands inside the sheet
- polylines corresponding to the sheet are always shown closed.

If it *autoClose*=true: the profiles corresponding to the parts are shown closed; otherwise: as programmed.

3.13 Project serialization functions

These functions manage the serialization of the nesting project to/from file.

SaveProject

The function saves the assignments of *part* and *sheet* lists to file (XML format).

NestErrors SaveProject (string pathName, bool bMode)

Arguments
- *pathName*: file path
- *bMode*: *true* requires saving the general operation settings

Return value
NestError.ErrorNone if the result is positive.

Notes
Any return different from *NestError.ErrorNone* corresponds to one of the error situations:
- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings -> EndSettings***, ***IniSetPart -> EndSetPart***, ***IniSetSheet -> EndSetSheet***, ***IniSetCluster -> EndSetCluster***)
- (*NestError.ErrorIOProject*) an error occurred in accessing the file to be written.

Use *bMode=true* to save the general function settings as well (corresponding to the ***IniSettings -> EndSettings*** section). The usefulness of a save that also includes settings is obvious for testing situations.
Some settings are excluded from the saving: FixComputeError, TimerProgres, DirectoryTemp.

LoadProject

The function reads the assignments of *part* and *sheet* lists from files (XML format).

NestErrors LoadProject (string pathName, bool bMode)

Arguments

- *pathName*: file path
- *bMode*: *true* enables reading the general function settings

Return value

NestError.ErrorNone if the result is positive.

Notes

Any return different from *NestError.ErrorNone* corresponds to one of the error situations:

- (*NestError.ErrorBusy*) nesting optimization is running
- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings -> EndSettings***, ***IniSetPart -> EndSetPart***, ***IniSetSheet -> EndSetSheet***)
- (*NestError.ErrorIOProject*) an error occurred in accessing the file to be read.

Use *bMode=true* to read the general operation settings as well (corresponding to the ***IniSettings -> EndSettings*** section): settings that are not read from the file are initialized to the default values. The usefulness of a reading that also includes settings is obvious for testing situations.
Some settings that do not change are excluded from the reading: FixComputeError, TimerProgres, DirectoryTemp.

Se *bMode=false*: the general function settings stay unchanged.

4 Guide to the use of the library

The purpose of this chapter is to provide the information needed to integrate the TPA_N into application.

4.1 Typical flow chart

The sequence shown below describes a basic use of the library:

1. create an instance of the class *TpaNestingOEM.Nesting()*
2. carry out the preliminary checks of general operation (key existence check and operating level enabled: *Square* or *True Shape*)
3. record the *CallBack* functions
4. open the assignment session of the general operation settings with call to the *IniSettings()* function
5. assign the general operation settings (e.g.,: *DirectoryTemp*, *TimeNesting*, …)
6. close the session with call to the *EndSettings()* function
7. open the *part* assignment session with call to the *IniSetPart ()* function
8. assign the *parts* with calls to the *AddPart ()* function: the shape assignment can occur in *IniGeometry ()*, …, *EndGeometry ()* sessions
9. close the *part* assignment session with call to the *EndSetPart ()* function
10. open the *cluster* assignment session with call to the *IniSetCluster ()* function
11. assign the *clusters* with calls to the *AddCluster ()* function: the composition of a cluster occurs with calls the *AddToCluster()* function
12. close the *cluster* assignment session with call to the *EndSetCluster()* function
13. open the *sheet* assignment session with call to the *IniSetSheet ()* function
14. assign the *sheets* with calls to the *AddSheet ()* function
15. close the *sheet* assignment session with call to the *EndSetSheet ()* function
16. start the optimization procedure with call to the *Compute ()* function
17. acquire the results with call to functions: *ReadNumResult ()*, *ReadResult ()*, *ReadPartInResult ()*
18. independently acquire/process the efficient information of the nesting procedure.

The assignment order of *settings*, *parts*, *clusters* and *sheets* is irrelevant: the one here suggested is only an example.

The set of general settings, parts, clusters and sheets constitute what is referred to as the *nesting project*, as it identifies the set of data needed to perform an optimization.

4.2 Preliminary checks

A preliminary check of the key presence, the enabled operating level, and additional options may be useful and required to adapt the functionality of your application:
- check the presence and validity of the key with querying IsSquareEnabled
- check the advanced operating level of Tpa_N with querying IsShapeEnabled
- after checking the advanced operating level, check the options available in drawing reading from external file, with querying ShapeExtension.

Remember that the key verification is performed with both temporal (that is: every X seconds) and contextual expiration (i.e., with call to specific functions): therefore, in order to guarantee a normal operation, we recommend not to remove the key while your application is running.

4.3 Assignment of settings

It is generally necessary to perform a preliminary assignment of the general operation settings:
- the first function to be called is *IniSettings(…)*
- the last function to be called is *EndSettings(…)*.

All the properties concerning the setting assignment must be used between the two functions mentioned above.
It is here reminded that the same settings can be read in a different context, too.
The preliminary assignment of the settings can also be done by reading from a file saved earlier: see functions *SaveSettings()*, *LoadSettings()*.

Part of the general function settings is surely more frequently changed, as directly correlated to a *nesting project*: it is the external application which decides how to organize the settings and by which criteria and at what rate to update them.

A change of the settings hinders the possibility to perform new optimization attempts: a later call to the RetryCompute function fails.

4.4 Assignment of parts

Let us see the assignment of the part list:
- the first function to be called is *IniSetPart()*
- the last function to be called is *EndSetPart()*.

If the part list assignment is total, call *IniSetPart* with argument *Clear=true*.

A change of the parts hinders the possibility to perform new optimization attempts (a later call to the RetryCompute function fails).

Some fundamental aspects in the assignment of parts should be highlighted:

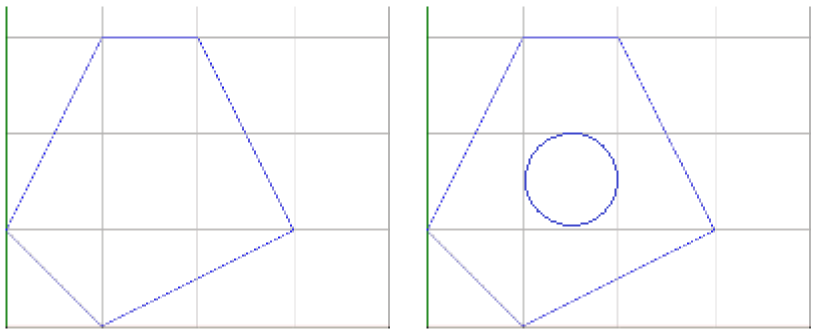
- each part has a unique, strictly positive (>0), numerical identifier: *ID* field in *NestPart* structure
- therefore, it is not possible to assign the same *ID* to more parts
- the identifiers may be assigned in any order and are not necessarily consecutive
- it should be noted that the IDs play an important role for the ordering of the parts: other important settings being equal (i.e., priority, geometry of the part) it is the ID that decides which part has a higher placing priority in the process of placing optimization.

To add a part, call the function *AddPart()*.
To delete a part, call the function *RemovePart()* with argument *OnlyGeometry=false*.
To change a part, call the function *WritePart()*
To change the *shape* of an already inserted part, call the function *RemovePart()* with argument *OnlyGeometry=true*.

How to assign the shape of a part

It is possible to define a shape for a *part* already assigned.
Assignment of part shapes must be made within a section (***IniSetPart -> EndSetPart***).

In the figure are two examples of shapes:
- on the left, a simple shape with only the external geometry assigned
- on the right, a more complex shape, with an internal geometry (island) assigned as well.



Each geometry is assigned independently:
- the first function to be called is *IniGeometry()*
- the last function to be called is *EndGeometry()*.

The function *IniGeometry* indicates:
- on which part the assignment is to be made
- if it is an external or internal geometry.

Only one external geometry can be assigned for each part, before the internal ones.
For a part it is possible to assign only internal geometries: the external part is automatically assigned in rectangular shape, applying the dimensions set for the part.

Example code

```
TpaNestingOEM.Nesting nestObj=new TpaNestingOEM.Nesting();

…
NestPart Item=new NestPart();

//it opens the part assignment section
If (nestObj.IniSetPart(true))
{
    //minimum assignment of the part structure (ID=1, N=20)
    Item.Initialize (1);
    Item.N=20;
    //it adds the part
    nestObj.AddPart(Item);
    //it opens the assignment section of the external geometry of the part with ID=1
    nestObj.IniGeometry(1, false, 0.0, 100);
    nestObj.AddToGeometry_Line(100, 300);
    nestObj.AddToGeometry_Line(200, 300);
    nestObj.AddToGeometry_Line(300, 100);
    nestObj.AddToGeometry_Line(100, 0);
    //it closes the assignment section of the (external) geometry
    nestObj.EndGeometry();

    //it opens the assignment section of the internal geometry of the part with ID=1
    nestObj.IniGeometry(1, true, 100, 150);
    nestObj.AddToGeometry_Circle(150, 150);
    //it closes the assignment section of the (internal) geometry
    nestObj.EndGeometry();

    //it adds the remaining parts
    …
    //it closes the part assignment section
    nestObj.EndSetPart();
}
```

Example code: external geometry acquisition loop

```
bool GeometryFromFile (string pathName)
{
    //error to be managed at the function return
    if (nestObj.ReadShape (pathName) != NestErrors.ErrorNone) return false;

    //initializes the variables used in the reading loop
    int IndexItem = 0;              //geometry index
    int IndexElement = 0;           //geometry element index (IndexItem)
    NestGeometry Item=new NestGeometry();

    //acquired geometry loop
    while (nestObj.ReadGeoInShape (IndexItem, IndexElement, ref Item))
    {
        // ……it treats (Item)=1' element of the index geometry (IndexItem)

        IndexElement++;             //geometry element index increase (IndexItem)
        while (nestObj. ReadGeoInShape (IndexItem, IndexElement, ref Item))
        {
            // ……it treats (Item)= new element of the index geometry (IndexItem)

            IndexElement++;         //geometry element index increase (IndexItem)
        }
```

```
            IndexItem ++;            // it increases the geometry index
            IndexElement=0;          //it initializes the geometry element index (IndexItem)
        }
        return true;
    }
```

4.5 Assignment of sheets

We will move on to the assignment of the sheet list:

- the first function to be called is *IniSetSheet()*
- the last function to be called is *EndSetSheet()*.

If the sheet list assignment is total, call *IniSetSheet* with argument *Clear=true*.

A change of the sheets hinders the possibility to perform new optimization attempts (a later call to the RetryCompute function fails).

As above for the parts, some fundamental aspects in the assignment of sheets are to be highlighted:

- each sheet has a unique, strictly positive (>0), numerical identifier: *ID* field in *NestSheet* structure
- therefore, it is not possible to assign the same *ID* to more sheets
- the identifiers may be assigned in any order and are not necessarily consecutive
- it should be noted that the IDs have an important role for ordering the sheets: other important settings being equal (i.e., priority, scrap information) it is the ID that decides which sheet has a higher priority for use in the optimization process.

To add a sheet, call the function *AddSheet()*.
To remove a sheet, call the function *RemoveSheet()* with argument *OnlyGeometry=false*.
To modify a sheet, call the function *WriteSheet()*.
To change the already inserted *shape* of a sheet, call the function *RemoveSheet()* with argument *OnlyGeometry=true*.

How to assign the shape of sheet

It is possible to define a shape for a *sheet* already assigned.
The *sheet* shape assignment must be made within a section (***IniSetSheet -> EndSetSheet***).

The assignment modes replicate what was already described for the *parts*.

4.6 Assignment of manual clusters

We will move on to the (optional) assignment of the cluster list:

- the first function to be called is *IniSetCluster()*
- the last function to be called is *EndSetCluster()*.

If the cluster list assignment is total, call *IniSetCluster* with argument *Clear=true*.
A change of the clusters hinders the possibility to perform new optimization attempts (a later call to the RetryCompute function fails).

Some fundamental aspects in the assignment of clusters should be highlighted:

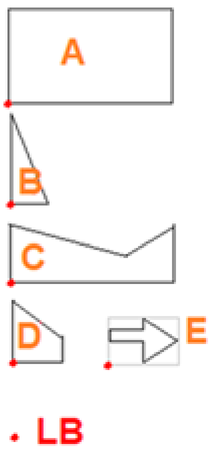
- each cluster has a unique, strictly positive (> 0), numerical identifier: *ID* field in *NestCluster* structure
- therefore, it is not possible to assign the same *ID* to more clusters
- the identifiers may be assigned in any order and are not necessarily consecutive
- it should be noted that the IDs play an important role for ordering the clusters: other important settings being equal (i.e., priority, composition of the cluster) it is the ID that decides which cluster has a higher placing priority in the process of placing optimization.

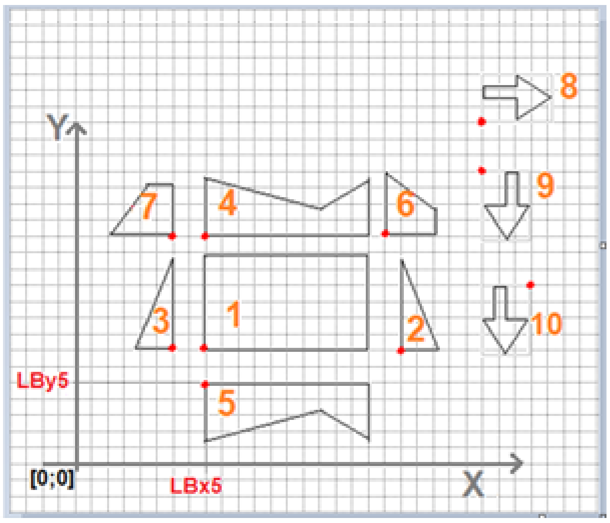
To add a cluster, call the function *AddCluster()*.
To define the composition of a cluster, call the function *AddToCluster()*.
To remove a cluster, call the function *RemoveCluster()* with argument *OnlyGeometry=false*.
To modify a cluster, call the function *WriteCluster()*.
To change the already inserted *composition* of a cluster, call the function *RemoveCluster()* with argument *OnlyGeometry=true*.

How to assign the composition of a cluster

It is possible to define the composition for a *cluster* already assigned.
The cluster composition assignment must be within a section (***IniSetCluster -> EndSetCluster***).
The cluster composition is defined with call to ***AddToCluster*** function.

The figure shows an example of the rules for building clusters:
- on the left we see the placeable parts
  - the letters (A, B, C, D, E) may correspond to the descriptive name of the parts (*Label* field in the *NestPart* structure)
  - the red circle shown on each part is the LB (left-bottom) point of the bounding box of each part
  - only for the (E) part the LB point is not on the profile: the bounding box is shown around (E)

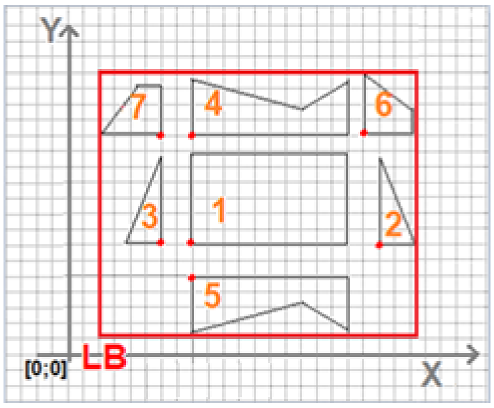




- on the right are possible uses of the parts in a cluster
  - 10 parts are indicated (numbered from 1 to 10)
  - the table shows the cluster construction information (see below: Example code):

#	*Part*	*X*	*Y*	*Mirror*	*A°*
1	A	LBx1	LBy1	0	0
2	B	LBx2	LBy2	0	0
3	B	LBx3	LBy3	x	0
4	C	LBx4	LBy4	0	0
5	C	LBx5	LBy5	y	0
6	D	LBx6	LBy6	0	0
7	D	LBx7	LBy7	0	90
8	E	LBx8	LBy8	0	0
9	E	LBx9	LBy9	0	-90
10	E	LBx10	LBy10	x	90

- with construction of the cluster on the first 7 elements, the figure shows
  - the bounding-box of the cluster (red box)
  - the LB point of the cluster



Example code

```
TpaNestingOEM.Nesting nestObj=new TpaNestingOEM.Nesting();

…
NestCluster Item=new NestCluster();
ItemCluster OneItem=new ItemCluster();


//it opens the cluster assignment section
If (nestObj.IniSetCluster(true))
{
   //minimum assignment of the cluster structure (ID=1, N=20)
   Item.Initialize (1);
   Item.N=20;
   //it adds a cluster
   nestObj.AddCluster(Item);
   //it assigns the cluster composition
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "A", X = 160.0, Y = 140.0, Mirror = 0, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "B", X = 400.0, Y = 140.0, Mirror = 0, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "B", X = 120.0, Y = 140.0, Mirror = 1, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "C", X = 160.0, Y = 280.0, Mirror = 0, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "C", X = 160.0, Y = 100.0, Mirror = 2, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "D", X = 380.0, Y = 280.0, Mirror = 0, A =
   0.0});
   nestObj.AddTocluster(Item.ID, new ItemCluster() { NameID = "D", X = 120.0, Y = 280.0, Mirror = 0, A =
   90.0});
   …

   //it adds the remaining clusters
   …
   //it closes the assignment section of the clusters
   nestObj.EndSetCluster();
}
```

4.7 Execution of the nesting optimization

To start a nesting project optimization, it is necessary to call the Compute function.

Part and sheet checks are run before the optimization, and they may also cause a function return with error and the ensuing cancellation of the optimization:

- parts requested for placements and available sheets must be present
- the match filter check (thickness, material, colour) must be able to match at least a part and a sheet.

During an optimization, the external application can manage the CallBack functions and possibly request the interruption of the same optimization.

Acquiring the solution result

When the optimization is completed, it is possible to acquire the nesting solution results.

Some properties/methods return general information regarding the optimization procedure:
- *IsComputed*: valid optimization status
- *ModeCompute*: information on the executed optimization (*Square* or *True Shape*)
- *TimeOut*: information if the computing procedure is over due to reached time-out
- *CanRetry*: possibility to request a new optimization.

On the other hand, a group of functions allows acquiring all the information on the optimization solution:
- *Fitness*: solution efficiency
- *ReadNumResult*: information on the total number of sheets of the solution or a sheet type
- *ReadNumPartInResult*: information on the number of placements of the solution or a sheet and/or a part type
- *ReadNumClusterInResult*: information on the number of placements of the solution or a sheet and/or a cluster type
- *ReadResult*: general information on a single solution sheet
- *ReadPartInResult*, *ReadGeoInResult*: information on a single placement on a solution sheet
- *SaveSolution*: saves the solution on file (XML format)
- *SaveSolutionDXF*: saves the solution to one or more files (DXF format).

Example code: how to acquire general information on the solution sheets

```
TpaNestingOEM.Nesting nestObj=new TpaNestingOEM.Nesting();

…
//assignment fields on component query
int ID_Sheet = 0, Item_Sheet = 0, NumPlaces = 0, Repetition= 0;
double AreaPercent = 0.0;

//query loop on the solution sheets
int IndexSheet = 0; //sheet index
while (nestObj.ReadResult (IndexSheet, ref ID_Sheet, ref Item_Sheet, ref AreaPercent, ref NumPlaces, ref
Repetition))
{
    // …
    //acquisition loop of the sheet placements
    //……
    IndexSheet ++; //it increases the sheet index
}
```

Example code: acquisition loop of sheet placements

```
// IndexSheet = sheet index (see: previous loop)
int IndexBox = 0; //placement index on sheet
NestBox nestBox=new NestBox();

while (nestObj.ReadPartInResult (IndexSheet, IndexBox, ref nestBox))
{
    // ……
    // see: "Acquisition loop of placement geometry"
    //……

    IndexBox++; //it increases the placement index
    // ……
}
```

Example code: acquisition loop of placement geometry

The reading loop must be run after a call to function ***ReadPartInResult***.

```
int IndexItem = 0;          //part geometry index
int IndexElement = 0;       //geometry element index (IndexItem)
NestGeometry Item=new NestGeometry();

//part geometry loop
while (nestObj.ReadGeoInResult (IndexItem, IndexElement, ref Item))
{
    // ……it treats (Item) = 1' element of the index geometry (IndexItem)

    IndexElement++;         //geometry element index increase (IndexItem)
    while (nestObj.ReadGeoInResult (IndexItem, IndexElement, ref Item))
    {
        // ……it treats (Item)= new element of the index geometry (IndexItem)

        IndexElement++;             //geometry element index increase (IndexItem)
    }
    IndexItem ++;           //it increases the geometry index
    IndexElement=0;         //it initializes the geometry element index (IndexItem)
}
```

Computing and managing progressive solutions

With *True Shape* optimization, it is possible to calculate more solutions, up to a maximum of 20:

- the CanRetry property gets the information if it is possible to start a new optimization
- call the RetryCompute function to run the new optimization.

When the optimization is complete, the calculated solution is automatically set as current.

All the calculated solutions are available, with the possibility to browse them and change the current solution to a specific one.

The current solution can be saved to file calling the SaveSolution function (or SaveSolutionDXF).

Example code: browsing progressive solutions

```
TpaNestingOEM.Nesting nestObj=new TpaNestingOEM.Nesting();

// GoToNextSolution: it goes to the next calculated solution
bool GoToNextSolution()
{
    If (nestObj.Solution < nestObj.OfSolution)
    {
        nestObj.Solution = nestObj.Solution+1;
        return true;
    }
    return false;
}

// GoToPrevSolution: it goes to the previous calculated solution
bool GoToPrevSolution()
{
    If (nestObj.Solution >1)
```

```
        {
            nestObj.Solution = nestObj.Solution-1;
            return true;
        }
        return false;
    }
```

4.8 Processing the CallBack functions

The functions are used to monitor the progress of the optimization phase. Only one function is currently available.

Progres function

Event management allows you to cancel or stop the optimization procedure:

- the cancellation determines the closure of the optimization phase with total cancellation of the procedure
- the interruption causes the closure of the optimization phase when the first valid solution is completed.

```
// Progres function
void ProgresFromNesting (int nValue, ref bool bCancel, ref bool bPause)
{
    //It updates a feed window
    …
    // Interactive management on user selections:
    // CancelCondition=true -> condition of optimization interruption
    // StopCondition=true -> condition of quick exit from optimization

    If (CancelCondition) bCancel=true;
    else if (StopCondition) bPause=true;
}
```

4.9 Saving and reading a TPA_N project

It is possible to save and restore a project through the functions SaveProject and LoadProject.

The two functions also allow saving and restoring the general function settings.
The possibility to request saving a project including also the general settings can be useful for testing purposes, in fact, in case a test is required to restore a critical situation.

After the execution of *LoadProject*, an external application has to update parts and sheets, reading information from Tpa_N.

Example code

```
TpaNestingOEM.Nesting nestObj=new TpaNestingOEM.Nesting();

//Structures
NestPart ItemPart=new NestPart();
NestSheet ItemSheet=new NestSheet();
NestSize ItemSize=new NestSize();
NestGeometry ItemGeo=new NestGeometry();
NestCluster ItemCluster=new NestCluster();
ItemCluster OneItemCluster=new ItemCluster();

// It reads the project
if (nestObj.LoadProject ("C:\projects\one.xml", false) == NestErrors.ErrorNone)
{
    //---------- It reads the parts
    for (int idxElement=0; idxElement< nestObj.CountPart; idxElement++)
```

```
        {
            nestObj.ReadPartIndex (idxElement, ref ItemPart, ref ItemSize);

            int IndexItem = 0;          //part geometry index
            int IndexElement = 0;       //geometry element index (IndexItem)

            //part geometry loop
            while (nestObj.ReadGeometry (0, ItemPart.ID, IndexItem, IndexElement, ref ItemGeo)) ==
                                                                NestErrors.ErrorNone)
            {
                // ......it treats (ItemGeo) = 1' element of the index geometry (IndexItem)

                IndexElement++;         //geometry element index increase (IndexItem)
                while (nestObj ReadGeometry (0, ItemPart.ID, IndexItem, IndexElement, ref ItemGeo)) ==
                                                                    NestErrors.ErrorNone)
                {
                    // ......it treats (ItemGeo)= new element of the index geometry (IndexItem)

                    IndexElement++;     //geometry element index increase (IndexItem)
                }
                IndexItem ++;                       // it increases the geometry index
                IndexElement=0;                     //it initializes the geometry element index (IndexItem)
            }
        }


        //---------- It reads the sheets

        for (int idxElement=0; idxElement< nestObj.CountSheet; idxElement++)
        {
            nestObj.ReadSheetIndex (idxElement, ref ItemSheet, ref ItemSize);
            int IndexItem = 0;          //sheet geometry index
            int IndexElement = 0;       //geometry element index (IndexItem)

            //sheet geometry loop
            while (nestObj.ReadGeometry (1, ItemShet.ID, IndexItem, IndexElement, ref ItemGeo)) ==
                                                                NestErrors.ErrorNone)
            {
                // ...
            }
        }

        //---------- It reads the clusters
        for (int idxElement=0; idxElement< nestObj.CountCluster; idxElement++)
        {
            nesObj.ReadClusterIndex (idxElement, ref ItemCluster);

            int IndexItem = 0;          // cluster assignments index

            //cluster assignments index
            while (nestObj.ReadInCluster ( ItemCluster.ID, IndexItem, ref OneItemCluster) ==
            NestErrors.ErrorNone)
            {
                // ...
                IndexItem ++;           // it increases the assignment index
            }
        }
    }
```

Tecnologie e Prodotti per l'Automazione S.r.l.

Via Carducci 221
I - 20099 Sesto S.Giovanni (MI)
Ph. +393666507029

www.tpaspa.com

info@tpaspa.it

11/17/2023