TpaNestingOEM

TpaNestingOEM Library

User manual

V. 1.0.0

23/04/2021

[redacted] INTRODUCTION

TpaNestingOEM.dll (aka 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

…

1.2. License

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

- however, this functionality is only available with *TrueShape* optimization enabled.
- the accessory authorizations are *DWG* format and Spline curve import.

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

1.3. Ownership and copyright

No part of this document may be reproduced, changed, integrated, or translated without prior written permission of the copyright owner.
The information contained in this document is subject to change and does not represent a commitment on the part of TPA Srl.

Although every effort has been made to ensure the accuracy of the information presented in this document, TPA Srl assumes no liability of any kind for any loss or damage caused by errors, omissions, or statements of any kind in this document, its updates, its additions, or special editions, regardless of whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Furthermore, TPA Srl does not assume any liability deriving from the use of the information here described; nor any liability for accidental or consequential damages resulting from the use of this document.

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: +390236527550
e-mail: info@tpaspa.it

or visit our web site:
www.tpaspa.com

USER GUIDE

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

TPA_N calculates the *shape* 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.

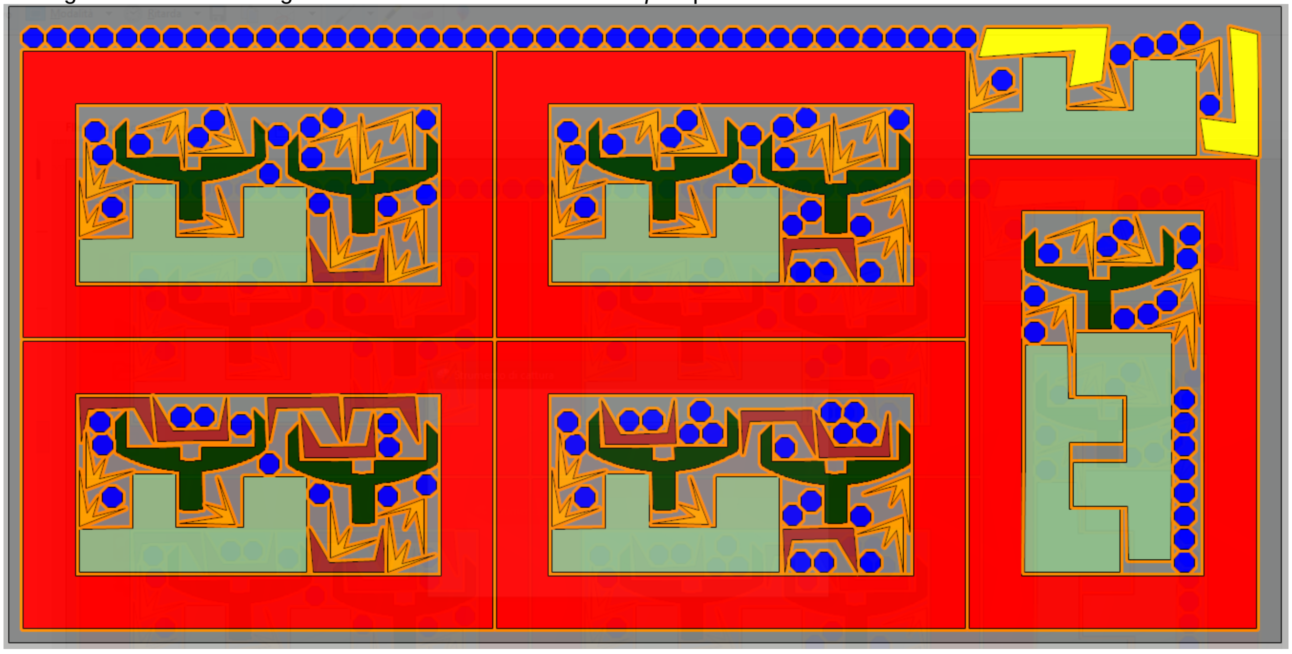
The assignment of *parts* allows defining transforms of the basic geometry, like rotation and/or mirroring, as well as generic (available quantity, positioning priority) or generally technological assignments (thickness, material, colour, grain, …).

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

The generally technological data allow 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 *TrueShape* optimization.



2.1. Nesting optimization typologies

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

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).

If all the *parts* and *sheets* are assigned as simple rectangles, the library assumes 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 profile itself, and the profiles assigned as islands are ignored.
- the *sheets* assigned with a geometric profile are considered for the bounding box of the profile itself 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.

TrueShape 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 mirroring 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.

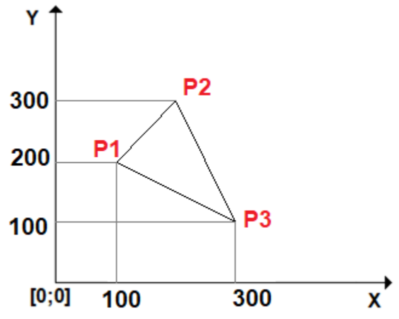
2.2. Unit and coordinates

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 coordinate point [0;0] is the bottom left angle.
- X is the horizontal axis, positive towards angle.
- 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)



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

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

2.3. Profile definition

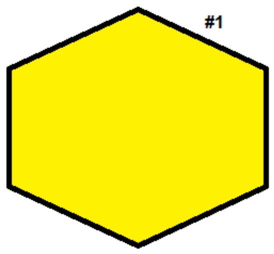
A *part* or a *sheet* is assigned through a primary geometric profile and eventual secondary profiles (islands or scrap areas): the simplest geometric profile is a rectangle, that can be directly assigned with the *length* and *height* dimensions.

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 that connects P3 and P1.

Each profile assigns a *net* area:

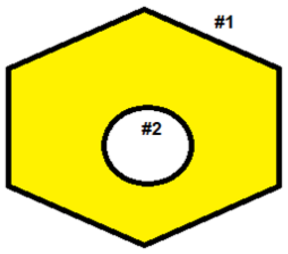
- cut area, for the parts.
- edge area, for the sheets.

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:

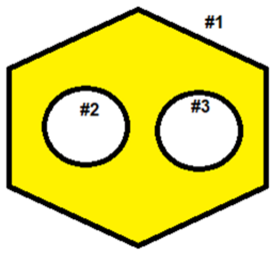
- #1 marks the primary external profile.



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 sheet: 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. Particular attention should be paid to two specific situations:

- assignment of manual clusters of the parts
- assignment of sheet geometry: as a matter of fact, the placements are calculated on the overall dimension actually assigned to the sheet.

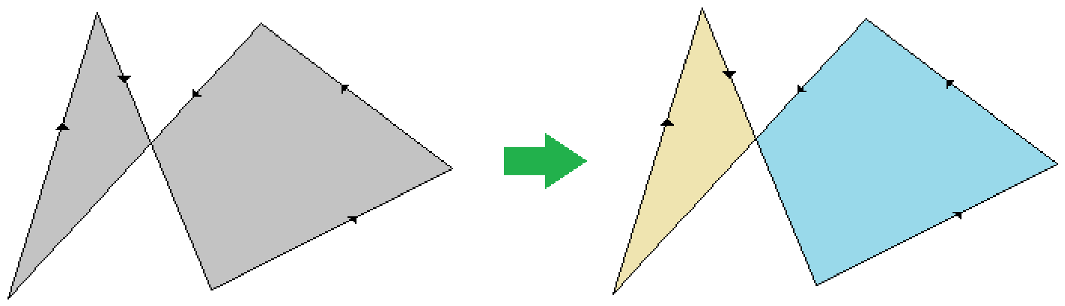
A geometric profile is valid if it assigns a *clean* geometry: no *null* or *intersecting* segments, and it closes a non-null area.

In order to have a *clean* geometry, each profile is drawn up by TPA_N with:

- removal of coincident (except for linear resolution epsilon) or aligned vertices.
- removal of intersecting situations, 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 clean geometry.

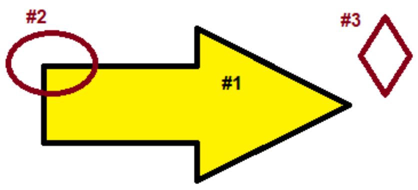


The nesting procedure discards the profiles that cannot be attributed to a valid geometry, and possibly excludes them from the placement calculation.

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 internal area's colour is: yellow)
- #2 marks a scrap profile that *intersects* the primary profile
- #3 marks a scrap profile that is *external* to the primary profile

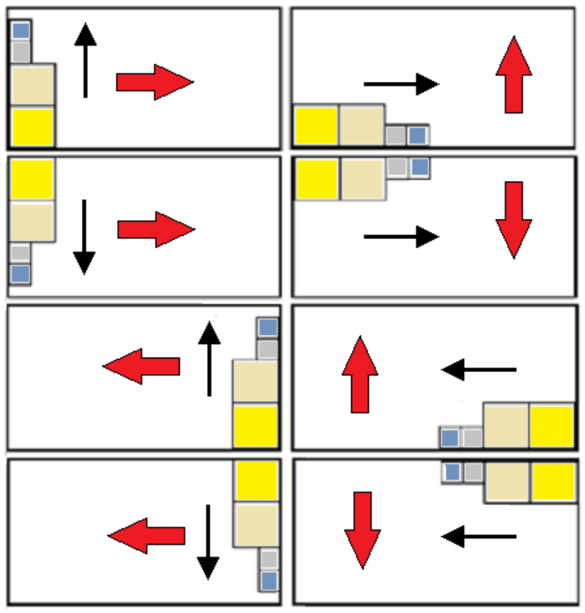
The nesting procedure will ignore both scrap profiles. In particular: the area used by the two scrap profiles can in no way be observed during the evaluation of the part placement on a *sheet*.
A situation like the one shown here entails the certain placement overlapping of other parts in the area of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

2.4. Nesting development Direction and Vertex

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 Vertex assigns the placement starting vertex among four possible values:



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

Information corresponds to the project general assignments Direction and Corner.

2.5. Matching filters and groups

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 review the settings concurring in applying the matching conditions.

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

- *S*: double field for thickness
- *Fiber*: integer field for generic assignment of material
- *Colour*: integer field for generic assignment of colour.

As for *S* field (double type), its match is evaluated on the equality of the set values, unless an epsilon of linear comparison matches the MinResolution setting.
Two more settings enable the application to use the fields *Fiber* and *Colour* (MatchType and MatchColor).

Examples of matching group assignment 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 group1 can generate a Nesting solution, allowing the association between parts and sheets: the parts with identifier (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. An 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 structures NestPart and NestSheet, matching the assignment of direction and grain:

- the assignment takes place in the *Grain* field of the structures.
- the technological meaning of this information depends on the material of the sheets (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 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.

Let us see which criteria are applied:

- 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:

- a privileged placement on the sheet with the same assigned grain is not guaranteed.
- the active *RctMinimize* setting is not applied, only if there is the possibility to use the part on a type of sheet with assigned grain.

2.6. Transforms applied to parts

For the parts, it is possible to assign information on two geometric transforms:

- mirroring
- rotation.

The application features are substantially different:

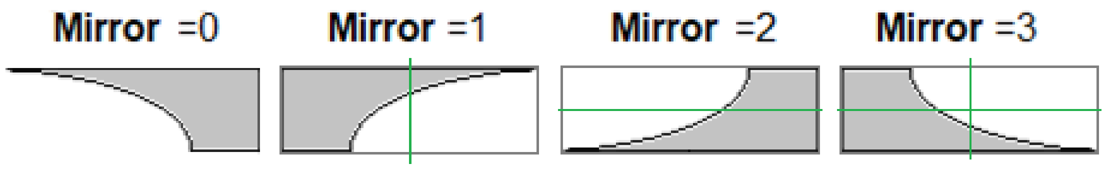
- the *mirroring* functionality specifies that a prior symmetry of the assigned geometry is **required** for the part.
- the *rotation* functionality specifies if and with what rules it is **possible** to perform rotations of the part, in order to optimize the placements.

Part mirroring

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 centreline axis of its own bounding rectangle.
- *2 (Y)*: the part is mirrored around the horizontal centreline axis of its own bounding rectangle.
- *3 (X+Y)*: sum of the two previous options.

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



The selection is irrelevant to calculate the placements in *Square* optimization, where the only important characteristic is the overall rectangle of the part. In the *TrueShape* optimization, the symmetry is applied before using the part.

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
- *TrueShape*: 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 *TrueShape*) or counter-clockwise 90° step rotation (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 the StepAngle general assignment:
- the property only accepts values between 5.0 and 90.0
- the minimum value of rotation actually applied is reduced to an integer submultiple 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°).

Option to Scale down a part bounding box

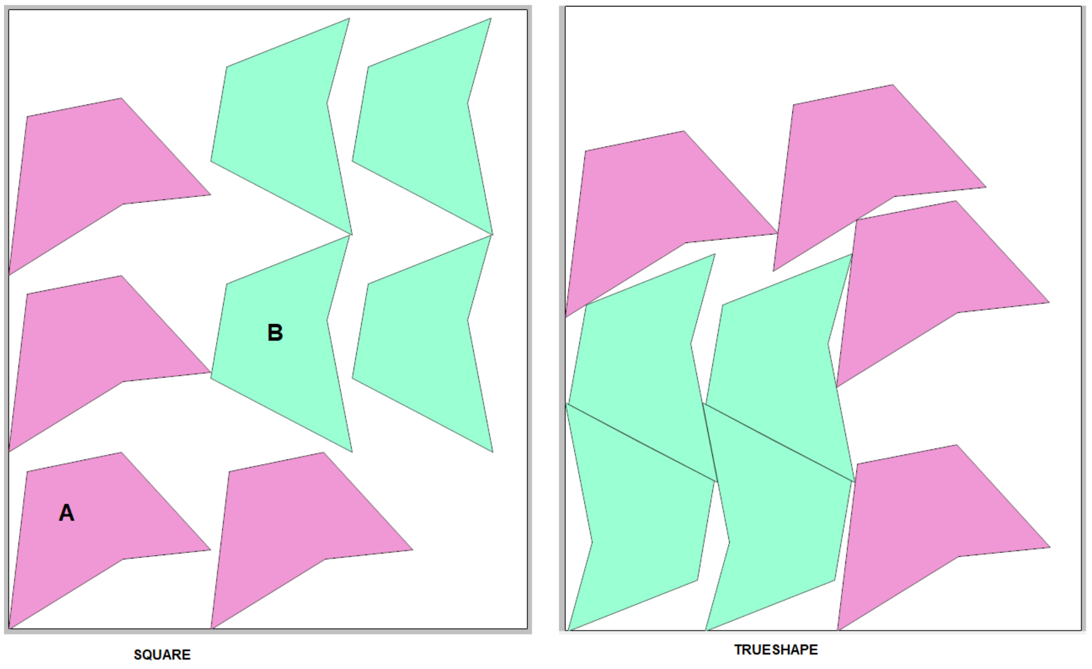
Another general ***boolean*** setting can affect the rotation logics of the parts (RctMinimize).
The parts with an assigned geometry can use this setting.

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 preventively modified, compared to the original, with assigned rotation, so to scale down its overall bounding rectangle. This can only happen if the part:
    - can rotate and
    - is not affected by a manual cluster.

  and the placement of the part itself can apply a further 90° rotation.

In case of *TrueShape* optimization:
- *false:* if the part allows for an *any* rotation, the rotation that minimizes the bounding rectangle is determined anyway.
- *true*: equivalent setting, also for the parts that allow for 90°-step rotations.

The figure corresponds to executing the two optimizations with placement of a same shape in *Square* optimization (on the left), *TrueShape* (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 rectangle of the same shape, consequently with a higher possibility of placement optimization.



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 *TrueShape* optimization.
For each part it is possible to enable the placement within its islands, corresponding to the *UseIsle* field in the NestPart structure.

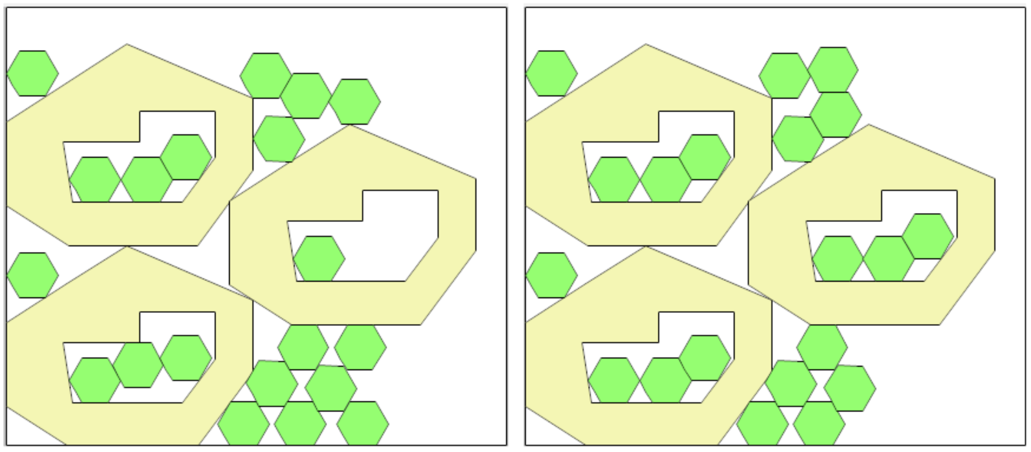
Some general ***boolean*** settings control the placements in the islands:

- *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 managing terms of this functionality should be evaluated considering technological factors about the sheet fastness and the cut of parts.

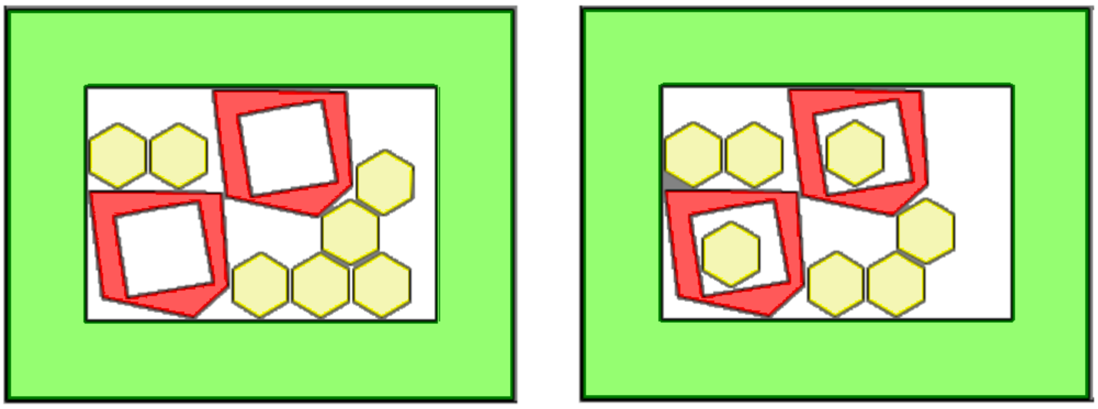
The figure compares the effect of the *PartInHoleBefore* setting:

- 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 occur within the part shown in *red*.
- on the right, the case of *true* value: the placements are carried out in the island of the part shown in red.



2.8. Control of the spacing between placements

Different data contribute to determine the spacing between the placements and between placements and sheet edges.

Part cut diameter

This piece of information stands for the diameter of the tool used to cut a part: it corresponds to the CutterDiameter setting.
The value can be null (=0.0).
The tool is applied around the profile of each part:

- on the outer part of the external profile, to increase the effective area of the profile.
- on the inner part of the scrap cut profiles, to decrease the scrap area.

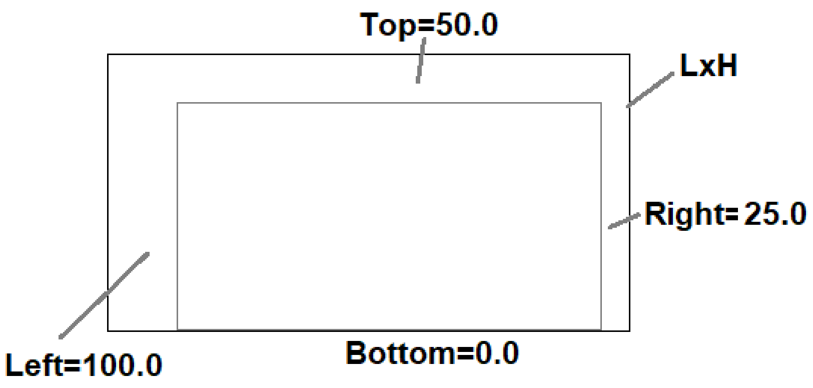
The figure shows two generic parts:

- on the left, a part without islands
- on the right, a part with island

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

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, a sheet of size LxH and where the assignments of the four external edges are all different.



Setting the external edges corresponds to these settings:

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

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

The *minimum spacing* there may be between a sheet edge and a placement, intended as net area, is determined as (*CutterDiameter*/2):

- *CutterDiameter* is the diameter of the part cutting technology, and it is set with the setting of the same name (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 it to the minimum distance from the edges at the value of (*CutterDiameter*)
    - *true*: allows the tool to get out of the sheet margins for half of its value (*CutterDiameter*/2)
- the assessment of the minimum distance of placements from the edges can add a further spacing if the external edges are already insufficient.

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
- the OverrunSecurity setting assigns the *Safety distance* added to the placements with an applied overrun 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

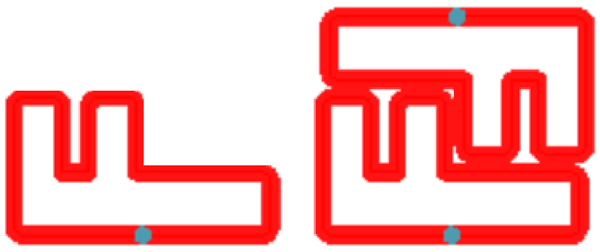
2.9. Part clusters

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 of the generation of 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 *TrueShape* 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 always comes before the single placement if the use efficiency of the *group* is equal at least to 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 clustering mechanism requires the part to be rotatable.

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* with 90° steps.
The part can also be placed individually, where it is no longer possible to apply it in a *group*.

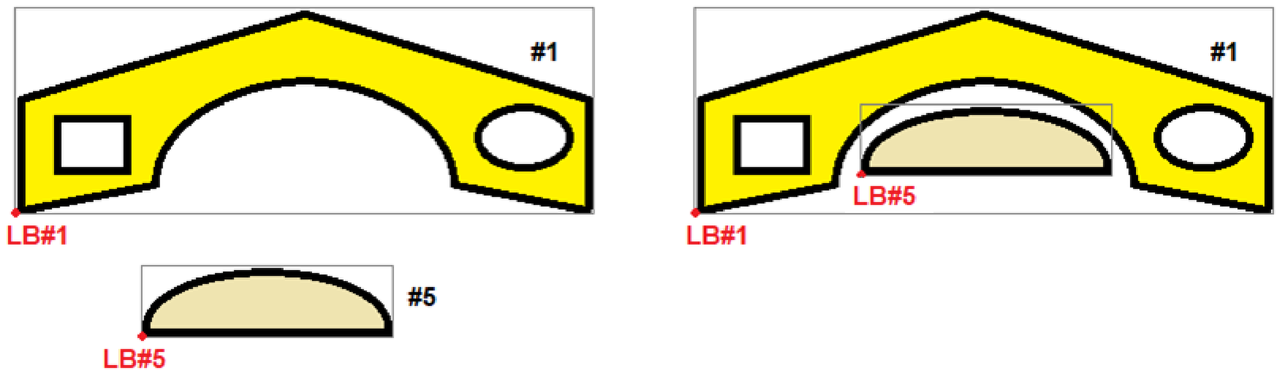
The *AutomaticCluster* setting assigns the total enabling to carry out automatic clusters.

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 the possible islands: the drawing so obtained is then used as a single entity to be placed on the sheets.

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 primary profile only.

The vertex LB (bottom left) of the primary profile bounding rectangle is indicated for both parts.
On the right, the parts are shown on the same representation plane, showing their reciprocal positioning.



The correct construction of a cluster comes from the correct assignment of the parts. E.g.: 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, field *Enable=true*).

A manual cluster is assigned in correspondence to the field *ID_parent* in the structure NestPart:
- identify a *main* part of the group, for which assign *ID_parent* to the default value (=0). In our graphical example: #1
- for the other parts of the group, assign *ID_parent* to the identifier of the main part. In our example: for the part with ID 5, set (*ID_parent*=1).

Continuing with our example, the part with ID 1 assigns the features of the manual cluster. In particular:
- requested and maximum quantity available
- allowed and requested transforms (rotation, mirroring)
- matching assignments (thickness, material, colour, grain)
- enabling assignments (automatic cluster, placement in islands, grid placements)
- priority.

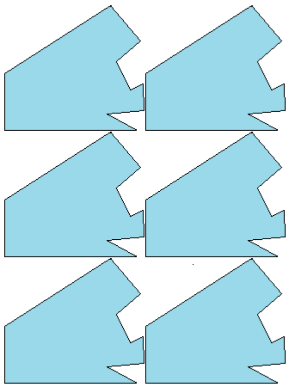
Grid placements

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

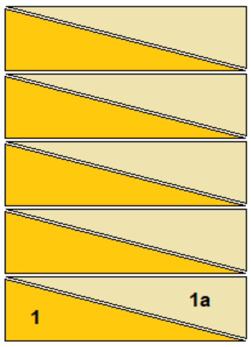
The *AutomaticGrid* setting assigns the total enabling to carry out the grid placements.
The function is managed only in *TrueShape* optimization.

The workpieces for which a grid placement is requested are used before the others and are placed according to a *row*column* arrangement, considering the space available on the sheet. To have the placements optimized, each part can be analysed by applying autonomous clustering strategies:
- a part can be placed with repetition of a unit that can correspond to a single part, always repeated with the same rotation, or to 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.

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

The optimization procedure places the workpieces on the fewest 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 one enabled element only. 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 panel
0	>0	the procedure places the highest number of workpieces on the total available of the sheet (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 a *Maximum quantity > Requested quantity* is set for the part, the placement fills the last used panel up, up to the maximum value set.
>0	>0	the procedure places the Requested quantity of the part on the fewest available sheets. If the procedure has placed the available quantity and *Maximum quantity > Requested*

	quantity is set for the part, the placement fills up the last used panel, up to the highest set value.

Multiple parts and Single sheet

The *part* list has more than one enabled element, 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 panels required to place all the requested parts.
>0	the procedure places the highest number of requested parts on the lowest number of available panels

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

Extra placements

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

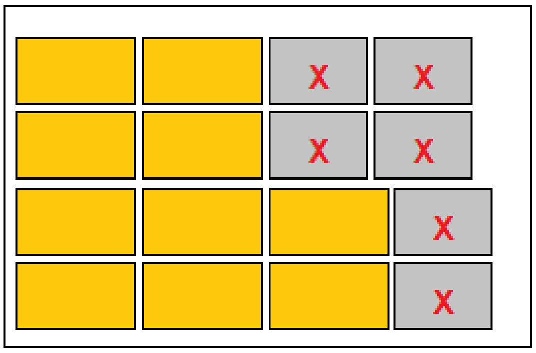
- a strictly positive value is assigned for the requested placements (in the *N* field of the structure)
- the set value is higher than that of the requested placements.

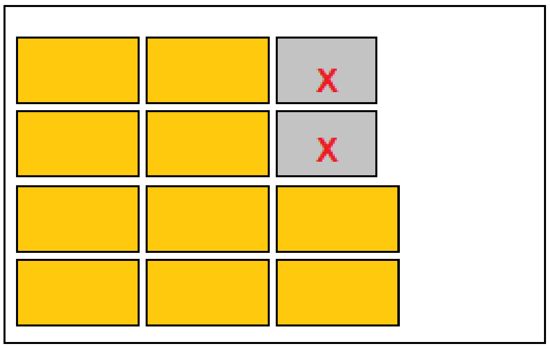
In these conditions, the number of *extra placements* is calculated to be: *(Nmax – N)*.
On a sheet, the extra placements are used only after checking that it is not possible to place any more requested parts.
Based on what was here said, it is clear how in no case a sheet can be used only to apply extra placements.
Using the extra placements, it is possible to choose between two modes, as in the setting ExtraFiller:

- *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:

- the extra placements are marked with an 'X'
- on the left, the case of *ExtraFiller = False*
- on the right, the case of *ExtraFiller = True*





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 more immediate understanding *parameters*:

- changes in the part placement order
- changes in the sheet filling logics
- 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 *TrueShape* optimization applies changes of both types.

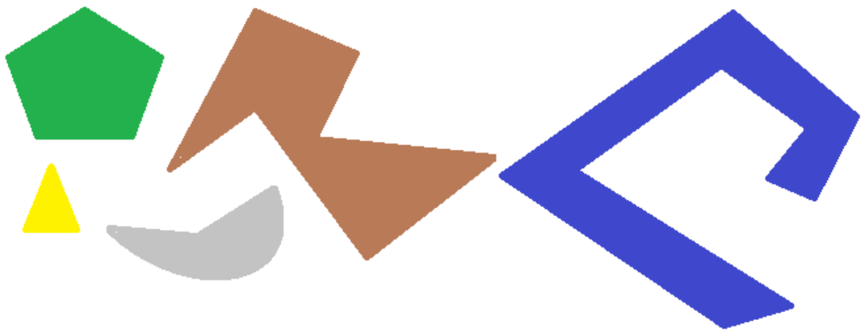
Square optimization

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 predefined attempts.

TrueShape optimization

The *TrueShape* 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 internal holes exploitable*, *in number and possibility of variable rotations*. The possibilities of interlocking among shapes are almost endless. The figure suggests 5 different shapes.



It is evident 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 it is necessary to put a time limit, 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 *TrueShape* optimization, it is possible to calculate more solutions, up to a maximum of 20.
While the first *TrueShape* optimization returns the first calculated solution, for the ones after it is possible to select between two functions (see function: RetryCompute)

- the optimization stops according to the maximum assigned time.
- it returns the first calculated solution, activating a *StepByStep* function.

Next optimizations can be calculated if:

- a first optimization in *TrueShape* mode has been performed.
- all settings stay unchanged (settings, part 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.
- nothing guarantees that it can be considered *better* than the previous one.

All the optimizations calculated stay available until

- a new total optimization (see function: Compute)
- a request to delete the solutions found (see function: ClearSolution).

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 different 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 *TrueShape* optimization:

- on a first optimization (function: *Compute*) only one attempt is run, 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 favours 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 cutting bit.

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 and sheets.
Part list checks:

- unenabled parts are excluded from the optimization (*Enable* field in the *NestPart* structure)
- parts that have no correspondence with valid *sheets* 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 rectangle dimensions <= minimum resolution value are excluded from the optimization (general setting MinResolution)
- parts with invalid manual cluster assignment are excluded from the optimization (see: How to assign a manual cluster).

Sheet list checks:

- unenabled sheets are excluded from the optimization (*Enable* field in the *NestSheet* structure)
- sheets with one or both the bounding rectangle dimensions <= minimum resolution value * 50.0 are excluded from the optimization (MinResolution general setting)
- sheets with no correspondence with valid *parts* are excluded from the optimization (e.g.: sheet with *Fiber*=1 and no part with the same setting).

The result of the reported analysis must be able to use at least one element a list:

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

The start of the nesting procedure applies specific criteria to assign the use order of parts and sheets, with some distinctions between the two optimization modes.

Use of sheets

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

- sorting by decreasing priority (*Order* field in the *NestSheet* structure)
- sorting by increasing type (*ID* field in the *NestSheet* structure), with the same or unmanaged priority.

The general UseOrderSheet setting enables applying the *sheet* priority.

Use of parts (Square)

Using the parts follows an order that can consider two different situations:

- the general setting UseOrderPart enables applying the *part* priority.
- 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 table checks the possible situations:

UseOrderPart	*PartSortMode*	Sorting of parts by…
False	0	decreasing area (large workpieces first)
False	1	increasing area (small workpieces first)
True	0	- sorting by decreasing priority - same priority: by decreasing area
True	1	- sorting by decreasing priority - same priority: by increasing area

When applying the cases listed in the table, with the same conditions, it privileges (in the sequence) a part if:

- not rotatable
- *square*
- with greater perimeter
- larger requested quantity
- Increasing type (*ID* field in *NestPart* structure).

Use of parts (TrueShape)

The use of parts follows an order that considers a larger number of different situations compared to the *Square* case.
Here as well, the general setting UseOrderSheet enables the application of the priority of the *parts*.
Before delving in 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 now the main criteria applied to the initial part order. The points addressed below are applied following the entered order, going to the next one in case it was not possible to operate a dominant choice on the previous point:

- sort by decreasing priority (if *UseOrderSheet*=*true*)
- parts with grid placement request (*AutoGrid* field in *NestPart* structure)
- part with concave shape or with islands, with possible inclusion in the concave area or in the islands
- part of larger area
- part with less possibility of rotation
- part with larger requested quantity
- part with increasing type (*ID* field in *NestPart* structure).

2.12. Known limits of the library

Assignment of parts

- It is possible to assign up to 300 different *parts.*
- for each *part*, the maximum placeable quantity that can be assigned is 999.
- the maximum total of placements requested for all the assigned *parts* is 10000.

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 100.

Square optimization

- placing a *part* with assigned profile and grain does not apply the *RctMinimize* setting as active, if it is possible to be placed on a sheet with assigned grain
- placing *manual clusters* does not apply the *RctMinimize* setting as active.

GUIDE TO THE LIBRARY FUNCTIONS

This chapter details the properties and functions of TPA_N.

3.1. License management

IsSquareEnabled

Boolean 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 library action can be performed if the basic function key is not valid.
The key existence and status check is autonomously and regularly carried out by the library.

IsShapeEnabled

Boolean properly testing presence and status of the key for advanced functions (*TrueShape* 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.
The key presence and status check is autonomously and regularly carried out by the library.

3.2. 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, …)
- *ErrorIOfile*: error managing temporary files (path and/or file access error, …)
- *ErrorNoneSolution*: no solution is assigned
- *ErrorBusy*: the component is busy optimizing
- *ErrorContext*: it indicates that the current context is not valid
- *ErrorUnexpected*: general or unmanaged error.

3.3. 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 ="") {*default*=""}
- *int ID_parent*: numerical identifier of the part to which the placement is to be associated {*default*=0}
  - a strictly positive value must match the identifier of a part different from the current one
  - the field manages the *Manual cluster* mode of the parts.

- *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 *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}
  - 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 value that is greater than *N* and with a strictly positive *N*. In this case: *(Nmax – N)* = usable quantity to fill the assigned *sheets*, only after trying to place the requested quantities

of all the *part* typologies. The placements corresponding to the application of the *Nmax* field are also indicated as *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
- *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).

- *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 grain itself (with possible application with rotation); a part without grain can be applied on sheets with any *Grain* field

- *short Order*: use priority (>=0) {*default*=0}
  - *parts* with higher priority have placement priority in the nesting solution.
  - the field application is conditioned by the general function setting ***UseOrderPart***

- *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 a step of considerable angle (assigned through the ***StepAngle*** property) and it is used only in case of *TrueShape* nesting. In case of *Square* nesting, the interpretation is the same as for value 1
  - value 1 corresponds to enabling a 90° rotation: in case of *TrueShape* nesting, it corresponds to the possibility of a 90°-step rotation.

- *short Mirror*: mirroring 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*: setting to make placements in the islands of the part {*default*=false}
  - this field is significant only in case of *TrueShape* 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 *TrueShape* 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 *TrueShape* nesting.
  - the field application is conditioned by the general function setting ***AutomaticGrid***

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 ="") {*default*=""}

- *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)
  - the values are in the following units: Unit

- *double S*: thickness (>=0.0) {*default*=0.0}.
  - assign the field differentiated values if it is necessary to apply a match with the corresponding *part* field
  - the 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
- *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).

- *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*: use priority (>=0) {*default*=0}
  - *sheets* with higher priority have use priority in the nesting solution.
  - the field application is conditioned by the general function setting ***UseOrderSheet***

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= the 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* = clockwise arc
  - *3* = counter-clockwise arc

- *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) vertex of the original bounding rectangle of the part
- *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 *TrueShape* nesting)
- *int CountSlave*: number of secondary placements
  - a positive value corresponds to the solution of a *manual cluster* (see paragraph: Example code: acquisition cycle of the geometry of a placement)

3.4. CallBack functions

The management of the available call-back functions is not necessary for TPA_N to function.

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 determines the closure of the optimization phase with complete cancellation of the procedure.
- interruption determines the closure of the optimization phase once the first valid solution is completed.

In both cases, it is possible that the optimization is not immediately closed, as the safe termination modes are active anyhow.

3.5. Definition of the functionalities

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 TrueShape nesting*: assignments to be applied to TrueShape optimization.

IniSettings

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

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

Arguments

- *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 function allows continuing with the total or partial assignment of the general function settings. On TPA_N's initialization, all settings are assigned to the default values.

- if a *part* or *sheet* assignment phase is active: this is pre-emptively terminated.
- if it is *Clear=true*: the function initializes the settings with default values, except for some properties that stay unchanged: ***DirectoryTemp***

- 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* and *sheet* lists.

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 negative result occurred while verifying the presence and status of the basic function key
- 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:

LastError

Property of ***NestErrors*** type 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 returns "" (empty string).

TimeNesting

Integer property that assigns/returns the maximum calculation time of a 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).

Functionalities regarding general assignments of a nesting project

DirectoryTemp

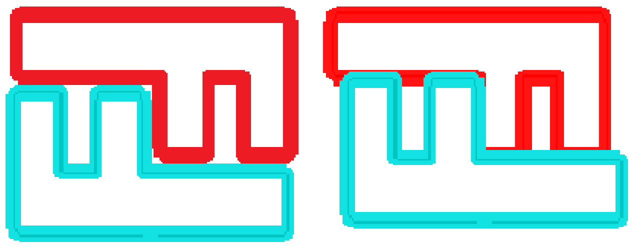
String property that assigns/returns the folder that manages the temporary files {*default*=""}
If no valid folder was assigned (it assigns empty string or inaccessible folder), it uses the storage path of the TPA_N library.

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

MinResolution
Double property that assigns/returns the minimum resolution value {*units*: 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).

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



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

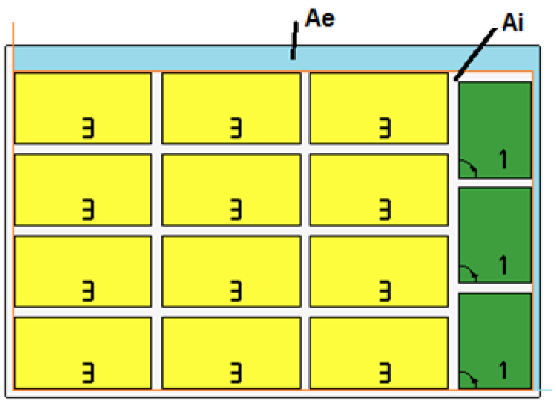
OverrunSecurity
Double type property that assigns/returns the *Safety distance* added to the placements, with the application of the cutting area overlap of the parts {*unit*: Unit; *default*= 0.1 [mm]; *range of validity*: 0.0/10.0 [mm]}

OverrunSheet
Boolean property that assigns/returns the activation of the sheet cutting area overlap {*default*= True}.
- *True:* minimum distance between an edge of the sheet and a placement, corresponding to half of the cut diameter (CutterDiameter * 0.5)
- *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 cut diameter (CutterDiameter).

The assessments of minimum distance of the placements from the margins can add a further spacing if the assigned edges are not sufficient (see: MarginLeft(…),…)

SolutionMaxScrap
Double type property that assigns/returns the value of the Maximum difference of internal scraps {*units*: %; *default*=2.5; *range of validity*: 0.5/50.0}.
The assessment is done by calculating the scraps as a percentage of the bounding rectangle that includes the placements of a sheet. This information is used combined with the other criteria considered to determine the best choice among different solutions.
Using this information in these terms can scarcely 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 concerning 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*.
- **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 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 setting):
  - if Horizontal: the filling is applied along the length of the sheet, while no limit in height is applied
  - if Vertical: the filling is applied along the height of the sheet, while no limit in length is applied.

UseOrderPart

Boolean setting that assigns/returns the activation of the application of the *part* priority {*default*=True}.
- *true*: it activates the application of the *Order* field, assigned in the ***NestPart*** structure, used to assign a part
- *false*: the structure field is ignored.

UseOrderSheet

Boolean setting that assigns/returns the activation of the application of the *sheet* priority {*default*=True}.
- *true*: it activates the application of the *Order* field, assigned in the ***NestSheet*** structure, used to assign a sheet
- *false*: the structure field is ignored.

MatchType

Boolean setting that assigns/returns the activation of the *material* match application between parts and sheets {*default*=True}.
- *true*: it activates the application of the *Fiber* field assigned in the structures ***NestPart*** and ***NestSheet***
- *false*: the structure fields are ignored.

MatchColor

Boolean setting that assigns/returns the activation of the *colour* match application between parts and sheets {*default*=True}.
- *true*: it activates the application of the *Colour* field assigned in the structures ***NestPart*** and ***NestSheet***
- *false*: the structure fields are ignored.

RctMinimize

Boolean type setting 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.

MarginOuter

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

MarginLeft, MarginRight, MarginBottom, MarginTop

Double type property that assigns/returns the edges external to the sheets, respectively on the left, right, bottom, and top sides {*unit*: Unit; *default*= 0.0; *range of validity*: >= 0.0}.
The use of differentiated edges can be applied to rectangular *sheets*. In case of generically-shaped sheets, a unique value is applied that corresponds to the highest value set of the four.

MarginInner

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

Direction

Short type property that assigns/returns the placement feed direction {*default*= 0; *range of validity*: 0/1}:

- 0=horizontal direction
- 1=vertical direction.

Corner

Short type property that assigns/returns the placement starting vertex among the valid values {*default*= 0; *range of validity*: 0/3}

- 0=Left-Bottom
- 1=Left-Top
- 2=Right-Bottom
- 3=Right-Top.

Square nesting functionalities

PartSortMode

Short type property that assigns/returns the sorting criteria for the parts prior to the placements {*default*= 0, *range of validity*: 0/1}:

- *0*: sort by decreasing area (large workpieces first). The area is assessed in conformity with the nesting direction:
    - if horizontal: it sorts by decreasing height
    - if vertical: it sorts by decreasing length
- *1*: sort by increasing area (small workpieces first).

True-Shape nesting functionalities

StepAngle

Double type property that assigns/returns the minimum rotation angle {*default*= 10.0; *unit*= degree and decimals of degree; *range of validity*: 5.0/90.0}.
The set value corresponds to value 2 in the *Rotate* field assigned in the ***NestPart*** structure used to assign a part.

PartInHole:

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

- *true*: it activates the application of the field *UseIsle=true*, assigned in the ***NestPart*** structure, used to assign a part.

PartInHoleMulti:

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

PartInHoleBefore:

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

AutomaticCluster

Boolean setting that assigns/returns the enabling to apply *Automatic clusters* {*default*= True}.

- *true*: it activates the application of the field *AutoPair=true*, assigned in the ***NestPart*** structure, used to assign a part.

ClustersExpected

Double type setting that assigns/returns the area minimum use value (in %) of an automatic cluster of parts relative to the single placements {*default*= 75.0; *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)

AutomaticGrid

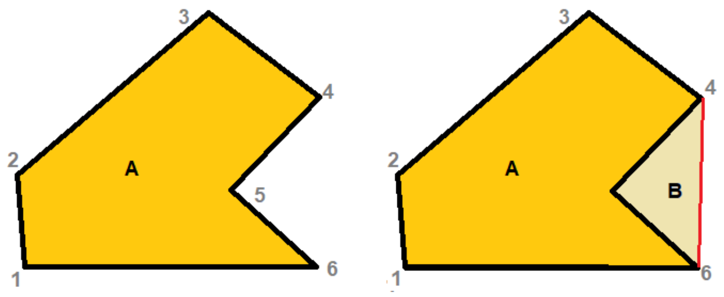
Boolean setting that assigns/returns the activation of the application of *Grid placements* {*default*= True}.

- *true*: it activates the application of the field *AutoGrid*=*true*, assigned in the ***NestPart*** structure, used to assign a part.

ExploreConcave

Boolean property that assigns/returns the enabling 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 eliminating 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 eliminates point 5, and the overall dimension of the part adds the area indicated with the letter B.



ConcaveDimension

Double type setting 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 that connects the (4, 6) vertices is less than the set value.

3.6. 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 ***EndSetParts()*** function.

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

- a negative result occurred when verifying the existence and status of the basic function key
- a 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 ***IniSetParts***, or the part list is not valid).

AddPart

The function adds a part to the list.

NestErrors AddPart (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.ErrorPartsTomany*) the part list is already at the allowed maximum (300 parts)
- (*NestError.ErrorUnexpected*) the part is assigned no valid identifier (the ***Item.ID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) a part with number identifier corresponding to ***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 (the ***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with number identifier corresponding to ***ItemID*** results as assigned.

WritePart

The function changes a part of the list.

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 number identifier corresponding to ***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*** property gets the number of parts in the list.

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 (the ***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with number identifier corresponding to ***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*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry* → *EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid index is assigned.

3.7. 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 negative result occurred when verifying the existence and status of the basic function key
- a 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
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.ErrorSheetsTomany*) the sheet list is already at the allowed maximum (100 sheets)
- (*NestError.ErrorUnexpected*) the sheet is assigned no valid identifier (the ***Item.ID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) a sheet with number identifier corresponding to ***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 eliminates a sheet or the geometries of a sheet.

NestErrors RemoveSheet (int ItemID, bool OnlyGeometry)

Arguments
- *ItemID:* sheet 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 (***IniSetSheet* → *EndSetSheet***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (the ***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no sheet with number identifier corresponding to ***ItemID*** results as assigned.

WriteSheet
The function changes a sheet of the list.

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 with number identifier corresponding to ***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
The ***integer*** property gets the number of sheets in the list.

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*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry* → *EndGeometry***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (the ***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no sheet with number identifier corresponding to ***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*: 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 (***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*) a 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 polygonal geometries

We will examine now how to assign the part and/or sheet geometries (*shape*).
The functions are operating even if no advanced license for the *TrueShape* functioning is recognized.
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 part polygonal geometry.

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 an already assigned primary geometry)
- *X, Y*:	starting geometry coordinates (the unit of the values is: 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 the ***EndGeometry ()*** function.

The function returning *False* corresponds to one of the error situations:
- (*NestError.ErrorLicense*) a negative result occurred when verifying the existence and status of the basic function key
- (*NestError.ErrorContext*) the function is not used within a section (***IniSetPart → EndSetPart***, or ***IniSetSheet → EndSetSheet***)
- (*NestError.ErrorUnexpected*) no valid identifier is assigned (the ***ItemID*** value must be strictly positive)
- (*NestError.ErrorUnexpected*) no part with number identifier corresponding to ***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 a part polygonal geometry.

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 a 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 - final radius|> ***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 a 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 a polygonal geometry.

NestErrors AddToGeometry (NestGeometry Item)

Arguments

- *item*:	geometric element assignment structure

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:*	identifier of the element (part or sheet) (> 0)
- *IndexItem*:	(zero base) index in the assigned geometric profile list
- *IndexElement*:	(zero base) index of the geometric element in the geometric profile
- *Item*:	geometric element assignment structure

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.

Corresponding to the assigned values of (*List*, *ItemID*), the first call must use (*IndexItem=0, IndexElement=0*). Any return of this first call 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*) the element is assigned no geometry (neither primary nor secondary).

Use of external geometries

ShapeExtension

String property that returns the supported file extension list for the interpretation of geometries from external file. Here are the possible cases now:

- ="":	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*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within a section (***IniGeometry→ EndGeometry***)
- (*NestError.ErrorLicense*) a negative result occurred when verifying the existence and status of the advanced function key
- (*NestError.ErrorIOfile*) an error occurred in accessing or interpreting the *pathOpen* file
- (*NestError.ErrorIOfile*) an error occurred in the writing procedure of the *pathResult* file.

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.

Now we will see the general interpretation rules of a drawing file:
- a 2D drawing is interpreted
- only profile-assigning geometric entities are interpreted (polylines, circles, ellipses, splines)
- the interpretation of the *splines* is conditioned by the verification of a specific activation of the hardware key
- only closed profiles are recognized as valid (the condition of closed profile is assessed by applying a maximum distance equal to the cut diameter: see ***CutterDiameter***)
- only 1 master profile (the one with the greatest area) and its eventual isles are kept, assigned at the first internal layer.

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

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

Arguments

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

Return value
NestError.ErrorNone if the result is positive

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

3.9. Nesting solution

Compute
The function starts the optimization procedure

NestErrors Compute (int OptiSelect, bool OnlyTest)

Arguments

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

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*) a 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 *TrueShape* optimization is requested and the key status does not match the advanced function
- the sheet and/or part list is not valid
- (*NestError.ErrorReset*) the procedure was annulled by the calling application.

At the start, the function
- closes an eventual open assignment section (settings, parts, sheets)
- runs the preliminary checks (as mentioned above)
- runs checks on the part and sheet lists
- assesses the type of required optimization, 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 goes on based on ***OnlyTest***:
- ***true***: it does not change the current optimization status and simply returns the result (positive or error)
- ***false***: it annuls 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 it is *OptiSelect=-1*
- if it is *OptiSelect=0*
- if the advanced function key existence and status check fails

Running the *Square* optimization:
- the *parts* assigned with a geometric profile are considered for the bounding rectangle of the profile itself, and the profiles assigned as islands are ignored.
- the *sheets* assigned with a geometric profile are considered for the bounding rectangle of the profile itself 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
- it is not possible to calculate progressive optimizations.

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

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

TimeOut
Boolean property that returns the information on if the computing procedure has ended for having reached the time-out.
The value of this property is significant after the execution of an optimization.

CanRetry
Boolean property that returns the information on if it is possible to request a new optimization attempt, starting from the last found solution.
The conditions to be able to calculate more solutions are:
- an optimization in *TrueShape* mode must result executed and complete
- the component must have no changes assigned (general settings, part and/or sheet 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*: it selects the type of optimization working
- *RetryBest*: true= it 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 annulled by the calling application.

The value of *RetryBest* selects between two possible functioning:
- *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 functioning of the optimization procedure:
- *false*: the optimization stops according to the maximum assigned time
- *true*: the optimization stops at the first calculated solution.

ClearSolution
The function resets any calculated solutions

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*) a nesting optimization is running.

3.10. The nesting results

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

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

Solution
Integer 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 an already calculated one
- no assignment is done if the value is not valid for the definition of a solution.

OfSolution
Integer 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 ***RetryCompte()*** function).

Fitness

Double type property that returns the information on efficiency of the *current solution*.
The value of this property is significant after the execution of an optimization.
The total efficiency is evaluated in % of the ratio between the area used for the placements and the total area of the used panels. If the solution generates more sheets, the last sheet is ignored in the efficiency evaluation, if assigned with non-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 of 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 corresponding to a sheet of the current solution and/or to a 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 corresponding to the assigned arguments:

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

In case of functioning with negative *Index* (-1), the returned value considers identical repeated sheets.
In case of functioning with Index>= 0, the returned value counts the placements of a single sheet, not considering the eventual repetitions of the sheet itself.

ReadResult

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

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*: number 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
It is recommended to not use the (*NumPlaces*) value returned by the ***ReadResult*** function as maximum iterator for (*IndexPart*), as (*NumPlaces*) includes the eventual secondary placements deriving from the application of a manual cluster.

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

- placement (X, Y) coordinates of the LB (bottom left) vertex of the original bounding rectangle of the part
- *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
    - A assigns a relative angle: the part rotates A around (X, Y)
- *int CountSlave*: number of secondary placements
    - a positive value corresponds to the solution of a *manual cluster* (see paragraph: How to assign a manual cluster)

ReadAddedPartInResult

The function returns the information concerning a secondary placement deriving from the application of a manual cluster

bool ReadAddedPartInResult (int IndexInPart, ref NestBox Item)

Arguments
- *IndexInPart*: zero base index of the secondary placement, corresponding to a previous call to the ***ReadPartInResult*** function
- *Item*: placement assignment structure

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, in case of ***Item.CountSlave***> 0: the value of the field assigns the number of secondary placements deriving from the application of a manual cluster.
For further details, see paragraph: How to assign a manual 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 in the assigned geometric profile list
- *IndexElement*:	(zero base) index of the geometric element in the geometric profile
- *Item*:	geometric element assignment structure

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 or the ***ReadAddedPartInResult*** 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:
- a 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:
- mirroring, as set in the *Mirror* field of the part
- translation and rotation, as calculated by the optimization procedure.

In case of part shape 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*) a nesting optimization is running
- (*NestError.ErrorNoneSolution*) no current solution resulting
- (*NestError.ErrorIOfile*) an error occurred in accessing or interpreting the file to write.

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 (placement (X, Y) coordinates and rotation angle).

3.11. 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 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*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings* → *EndSettings***, ***IniSetPart* → *EndSetPart***, ***IniSetSheet* → *EndSetSheet***)
- (*NestError.ErrorIOfile*) an error occurred in accessing or interpreting the file to write.

Use *bMode=true* to save the general function settings as well (corresponding to the ***IniSettings* → *EndSettings*** section). The usefulness of a saving that also includes the settings is evident for testing situations.

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*) a nesting optimization is running
- (*NestError.ErrorContext*) the function is used within an assignment section (***IniSettings* → *EndSettings***, ***IniSetPart* → *EndSetPart*, *IniSetSheet* → *EndSetSheet***)
- (*NestError.ErrorIOfile*) an error occurred in accessing or interpreting the file to read.

Use *bMode=true* to read the general function settings as well (corresponding to the ***IniSettings* → *EndSettings*** section): only the settings read from file will be assigned. The usefulness of a reading that also includes the settings is evident for testing situations.

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

GUIDE TO THE USE OF THE LIBRARY

The aim of this chapter is to offer the information to integrate TPA_N library in your 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. run the general preliminary function checks (key existence check and verification of the enabled functioning level: *Square* or *TrueShape*)
3. recording of the *call-back* functions

4. open the assignment session of the general function settings with call to the *IniSettings(..)* function
5. assignment of the general function settings (e.g.: *DirectoryTemp*, *TimeNesting,…*)
6. closing the session with call to the *EndSettings()* function

7. opening the *part* assignment session with call to the *IniSetPart (..)* function
8. assignment of the *parts* with call to the *AddPart (..)* function: the shape assignment can occur in *IniGeometry (..)*, …, *EndGeometry (..)* sessions
9. closing the *part* assignment session with call to the *EndSetPart ()* function

10. opening the *sheet* assignment session with call to the *IniSetSheet (..)* function
11. assignment of the *sheets* with calls to the *AddSheet (..)* function
12. closing the *sheet* assignment session with call to the *EndSetSheet ()* function

13. start of the optimization procedure with call to the *Compute (..)* function
14. acquirement of the results with call to functions: *ReadNumResult (..), ReadResult (..), ReadPartInResult (..)*
15. autonomous acquirements/elaborations of the information on the efficiency of the nesting procedure.

The assignment order of *settings*, *parts*, and *sheets* is irrelevant: the one here suggested is only an example.
The totality of parts and sheets form what is indicated as *nesting project*, as it identifies the totality of the data needed to execute an optimization. It is then obvious that the solution of a project also depends on the general settings.

4.2. Preliminary checks

A preliminary check of the key presence, of the enabled functioning level, and of the additional options can be useful and needed to adapt the functions of your application:
- check the presence and validity of the key with query to IsSquareEnabled
- check the advanced functioning level of TPA_N with query to IsShapeEnabled
- having checked the advanced functioning level, check the available options when reading a drawing from external file, with query to ShapeExtension.

It is here reminded that the key check is run according to time rate (i.e.: every X seconds) and context (i.e.: with call to specific functions): therefore, in order to guarantee a normal functioning, 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 function 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 aforementioned functions.
It is here reminded that the same settings can be read in a different context, too.

Part of the general function settings are 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 move on to 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), number 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.

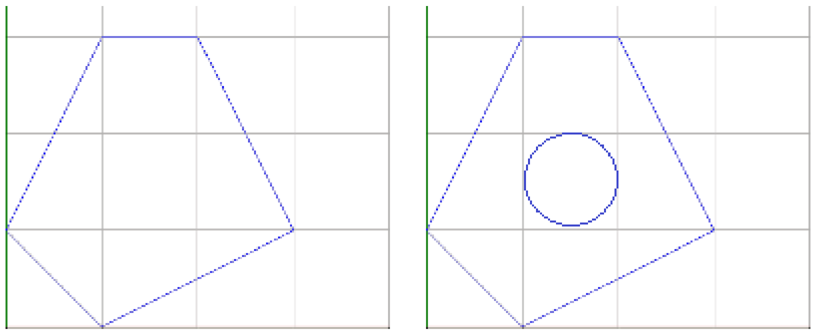
To add a part, call the function *AddPart(..)*.
To cancel a part, call the function *RemovePart(..)* with argument *OnlyGeometry=false.*
To change a part, call the function *WritePart(..)*.
To change the *shape* of an already inserted part, call the function *RemovePart(..)* with argument *OnlyGeometry=true.*

How to assign the shape of a part

It is possible to define a shape for an already assigned *part*.
The part shape assignment must be within a section (***IniSetPart → EndSetPart***).

In the figure are two examples of shape:

- 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 *IniGeometry* function indicates:

- on which part the assignment is to be made
- if it is an external or an internal geometry.

Only one external geometry can be assigned for each part, before the internal ones.

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 assignment section of the parts
    nestObj.EndSetPart();
}
```

Example code: external geometry acquisition cycle

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

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

     //acquired geometry cycle
     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;
}
```

How to assign a manual cluster

A manual cluster is assigned in correspondence to the *ID_parent* field in the NestPart structure:
- identify a *main* part, for which to assign *ID_parent* the default value (=0)
- assign then the *ID_parent* field of all the secondary parts to the main part ID.

If the assignment of the manual clusters is not correct, some parts will be automatically discarded from the nesting procedure. Let us consider an instance of part assignment with manual clusters.

Example
The table shows a list of 5 parts, assigned with an identifier, from 1 to 5.
Two manual clusters result assigned
- *A cluster:* primary ID= 1; secondary IDs (3, 5)
- *B cluster:* primary ID= 4; secondary IDs (2)

NestPart.ID	*NestPart.ID_parent*	
1	0	*main part* of **A** cluster
2	4	secondary part of **B** cluster
3	1	secondary part of **A** cluster
4	0	*main* part of **B** cluster
5	1	secondary part of **A** cluster

Let us see some particular situations:
- the part that corresponds to ID= 4 is *deactivated* → the part corresponding to ID= 2 is excluded from the nesting, too
- the part that corresponds to ID= 4 assigns N=0 and the *single part optimization* condition is not verified → the part corresponding to ID= 2 is excluded from the nesting, too
- the part that corresponds to ID= 4 has no possibility of match with the list sheets (as for: material/colour/grain checks) → the part corresponding to ID= 2 is excluded from the nesting, too
- the part that corresponds to ID= 4 assigns *ID_parent> 0* → the part corresponding to ID= 2 is excluded from the nesting
- the part that corresponds to ID= 2 assigns *ID_parent= 6* (invalid ID) → the part corresponding to ID= 2 is excluded from the nesting.

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), number 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.

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 a sheet

It is possible to define a shape for an already assigned *sheet*.

The *sheet* shape assignment must be within a section (***IniSetSheet*** **→** ***EndSetSheet***).

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

4.6. 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:

- there have to be requested parts for the available placements and sheets
- 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 call-back functions and eventually request the interruption of the optimization itself.

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 *TrueShape*)
- *TimeOut*: information on if the computing procedure is over due to reached time-out
- *CanRetry:* possibility to request a new optimization.

A group of functions allows acquiring instead all the information on the optimization solution:
- *Fitness*: solution efficiency
- *ReadNumResult*: information on the total number of sheets of the solution, or of a type of sheet
- *ReadNumPartInResult*: information on the number of placements of the solution or on a sheet and/or of a type of part
- *ReadResult*: general information about a single solution sheet
- *ReadPartInResult*, *ReadAddedPartInResult, ReadGeoInResult*: information on a single placement on a solution sheet
- *SaveSolution:* saves the solution to file (XML format).

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

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

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

//query cycle 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 cycle of a sheet placements
        //……
        IndexSheet ++; // it increases the sheet index
}
```

Example code: acquisition cycle of sheet placements

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

while (nestObj.ReadPartInResult (IndexSheet, IndexBox, ref nestBox))
{
        // ……
        // see: “Acquisition cycle of placement geometry”
        //…..
        If (nestBox.CountSlave >0)
        {
            //… acquisition cycle of the secondary placements deriving from the application of a manual cluster
            int IndexInPart=0; //secondary placement index
            while (nestObj.ReadAddedPartInResult (IndexInPart, ref nestBox)
            {
                //……
                // see: “Acquisition cycle of placement geometry”
                IndexInPart ++; // it increases the secondary placement index
            }
        }
        IndexBox++; // it increases the placement index
        // ……
}
```

Example code: acquisition cycle of placement geometry

The reading cycle must be run after a call to function ***ReadPartInResult***, ***ReadAddedPartInResult***.

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

//part geometry cycle
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 *TrueShape* optimization, it is possible to calculate more solutions, up to a maximum of 20:

- the CanRetry property gets the information on 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.

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;
}
```

4.7. Processing the CallBack functions

The functions are used to monitor the progress of the optimization phase. At the moment, only one function is available.

Progres function

Managing the event allows cancelling or interrupting the optimization procedure.

- cancelling it, causes the shutting of the optimization phase, with the total deletion of the procedure
- interrupting it, causes the shutting of the optimization phase once 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 selection:
        // CancelCondition=true -> condition of optimization interruption
        // StopCondition=true -> condition of quick exit from optimization

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

4.8. Saving and reading a TPA_N project

It is possible to save and retrieve a project through the functions SaveProject and LoadProject.
The two functions also allow saving and retrieving 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 requested retrieving a critical situation.

After the execution of *LoadProject*, an external application has to update parts and sheets, reading the 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();

// It reads the project
if (nestObj.LoadProject ("C:\projects\one.xml", false) == NestErrors.ErrorNone)
{
      //---------- It reads the parts
      nestObj.IniSetPart (true);
      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 cycle
          while (nestObj.ReadGeometry (0, ItemPart.ID, IndexItem, IndexElement, ref ItemGeo))
          {
                // ……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))
                {
                          // ……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)
          }
      }
      nestObj.EndSetPart ();

      //---------- It reads the sheets
      nestObj.IniSetSheet (true);

      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 cycle
          while (nestObj.ReadGeometry (1, ItemShet.ID, IndexItem, IndexElement, ref ItemGeo))
          {
                // …
          }
```

```
        }
        nestObj.EndSetSheet ();
}
```