TpaCutOEM

TpaCutOEM Library

User manual

v. 1.1.0

05/02/2024

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

1. PRESENTATION 5

1.1. VERSIONS 5
1.2. LICENSE 5
1.3. OWNERSHIP AND COPYRIGHT 5

2. USER GUIDE 5

2.1. CUT OPTIMIZATION 6
2.2. UNITS AND COORDINATES 6
2.3. DIRECTION AND DEVELOPMENT CORNER 7
2.4. MATCHING GROUPS AND FILTERS 7
2.4.1. Grain control 8
2.5. CUTS 8
2.5.1. Cutting Levels 9
2.5.2. Trims 11
2.5.3. Tension Free Cut 11
2.5.4. Sheet margins 11
2.6. OPTIMIZATION IN TPA_C 11
2.6.1. Progressive optimizations 12
2.6.2. Optimization criteria and priorities 12
2.6.2.1. Use of Sheets 13
2.6.2.2. Use of Rectangles 13
Extra placements 13
2.7. SUPPORT FOR MANAGING CUTTING PROJECTS 13
2.8. KNOWN LIMITS OF THE LIBRARY 13

3. GUIDE TO THE LIBRARY FUNCTIONS 14

3.1. LICENSE MANAGEMENT 14
3.1.1. IsValidLicense 14
3.2. CONSTANTS 14
3.3. ENUMERATIONS 14
3.3.1. CutError 14
3.4. STRUCTURES 15
3.4.1. OneRect 15
3.4.2. OneSheet 16
3.5. DEFINITION OF FUNCTIONS 16
3.5.1. General Functions 17
3.5.1.1. Version 17
3.5.1.2. LastError 17
3.5.1.3. ErrorMessage 17
3.5.2. Functions corresponding to general settings of cut optimization 17
3.5.2.1. NumberCustom 17
3.5.2.2. NumberRectParam 17
3.5.2.3. EnableRectEdge 17
3.5.2.4. OrderSheetDim 17
3.5.2.5. OrderBeforeScrap 18
3.5.2.6. OrderStrips 18
3.5.2.7. OrderSubStrips 18
3.5.3. Functions related to general assignments of a project 19
3.5.3.1. Unit 19

3.5.3.2. CutterDiameter 19
3.5.3.3. Direction 19
3.5.3.4. Corner 19
3.5.3.5. MaxCutLevels 19
3.5.3.6. TensionGap 20
3.5.3.7. PreCut 20
3.5.3.8. Longcut 20
3.5.3.9. Transvcut 20
3.5.3.10. Zcut 20
3.5.3.11. Custom1, Custom2, …, Custom10 20
3.5.3.12. ClearAll 20
3.6. DEFINITION OF RECTANGLES 20
3.6.1. AddRct 20
3.6.2. ClearRct 21
3.6.3. CountRect 21
3.6.4. ReadRect 21
3.6.5. ReadRectIndex 21
3.7. DEFINITION OF SHEETS 21
3.7.1. AddSheet 22
3.7.2. ClearSheet 22
3.7.3. CountSheet 22
3.7.4. ReadSheet 22
3.7.5. ReadSheetIndex 22
3.8. CUTTING SOLUTION 23
3.8.1. Compute 23
3.9. OPTIMIZATION RESULTS 23
3.9.1. NumberOfSolutions 23
3.9.2. SelectSolution 23
3.9.3. SolutionSheets 24
3.9.4. GetSheetId 24
3.9.5. NumberOfRepetitions 24
3.9.6. UsedSheets 24
3.9.7. UsedRects 24
3.9.8. FitnessSheet 25
3.9.9. ReadResolRect 25
3.9.10. NumberOfCuts 26
3.9.11. ReadResolCut 26
3.9.12. CutLinear 27
3.9.13. CutArea 27
3.9.14. MarginArea 27
3.9.15. GetWaste 28
3.9.16. Estimated time 28
3.9.17. Estimated cost 28
3.9.18. Export 29
File structure corresponding to a cutting pattern 29
<GENERALSETTINGS> node 30
<TECHSETTINGS> node 30
<DIM> node 30
<DIMTRIMS> node 30
<DATA> node 30
<PIECESLIST> node 30

<DRAW> node 31
3.10. PROJECT SERIALIZATION FUNCTIONS 34
3.10.1. SaveProject 34
3.10.2. LoadProject 35
3.10.3. ImportProject 35
4. GUIDE TO USING THE LIBRARY 37
4.1. TYPICAL FLOW CHART 37
4.2. PRELIMINARY CHECKS 37
4.3. ASSIGNMENT OF GENERAL OPERATION SETTINGS 37
4.4. ASSIGNMENT OF GENERAL PROJECT SETTINGS 37
4.5. ASSIGNMENT OF RECTANGLES 38
4.6. ASSIGNMENT OF SHEETS 38
4.7. PERFORMANCE OF THE CUT OPTIMIZATION 38
4.7.1. Acquiring the result of the solution 38
4.7.1.1. Example code: How to acquire general information of the solution sheets 38
4.7.1.2. Example code: Acquiring cycle of sheet placements 39
4.7.1.3. Example code: Acquiring cycle of sheet cuts 39
4.7.2. Managing multiple solutions 39
4.7.2.1. Example code: Navigation between Multiple solutions 39
4.8. SAVE AND READ A TPA_C PROJECT 40
4.8.1. Example code 40

1. PRESENTATION

TpaCutOEM.dll (aka TPA_C) is a product designed for software developers and can be integrated as a library for 32- and 64-bit Windows architecture.
TPA_C is a library developed for the automatic optimization of linear cutting of rectangular shapes and allows developing applications of optimization of 2D placements in various application sectors.

The typical use of the library is related to the programming of cutting machines.

The library must be integrated as part of your product.

1.1. VERSIONS

Version	*Release date*	*Comments*
1.1.0	05.02.2024	

1.2. LICENSE

TPA_C operation requires hardware key verification.
The key also allows to save the optimization result in XML format.

1.3. OWNERSHIP AND COPYRIGHT

For further information, please contact:

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

or visit our website:
www.tpaspa.com

2. USER GUIDE

TPA_C assigns the ***TpaRctCut*** class in the ***TpaCutOEM*** namespace.

TPA_C calculates the linear cut of rectangular shapes in rectangular placement areas.
A *rectangle* is a single entity that can be made by making multiple cuts over a sectional area.
Sectional areas are called *sheets* and they too are always rectangular in shape.

The assignment of *rectangles* allows defining information of a generic (available quantity, placement priority) or generally technological nature (thickness, material, grain, possibility of rotation, …). For each type of rectangle, both the requested quantity to be placed and an extra quantity that can be placed in addition, to fill the sheets already used, are defined.

The assignment of *sheets* allows defining settings of a generally technological nature (thickness, material, grain, …). For each type of sheet, you define the quantity available.

Information of a generally technological nature makes it possible to apply matching and/or filters conditions between *rectangles* and *sheets* to the optimization process.

The *result* of the optimization provides the user with detailed information about the best way to cut *rectangles* from the *available sheets*. The optimization *result* can assign one or more *sheets*: each *sheet* contains the cut of at least one *rectangle*.

The figure shows an example of optimizing a sheet, otherwise called *cutting pattern*:

2.1. CUT OPTIMIZATION

The rectangles indicated on the cutting pattern of a sheet are made by sequencing horizontal and vertical cuts, alternated in subsequent nesting operations.
The result of the cutting pattern that is computed for a sheet is to cut the single parts by alternating splitting into horizontal and vertical cuts, until all the required partitions are achieved.

The set of calculation processes leading to a cutting solution is referred to as *optimization step*.
Cutting a rectangle can apply 0° or 90° rotations (counterclockwise) to the same rectangle.
Optimization may propose up to a maximum of 5 solutions, which the user may evaluate for choosing the best solution.

2.2. UNITS AND COORDINATES

In TPA_C metric values (coordinates, dimensions) are expressed in the unit defined as [mm] or [inch] in the *Unit* property (0= [mm], 1= [inch]).

TPA_C works in a 2D Cartesian coordinate system:

- the coordinate point [0;0] is placed in the bottom left corner of the sheet
- X is the horizontal axis, positive to the right
- Y is the vertical axis, positive upwards.

The coordinates of the cuts that are calculated always refer to the bottom left corner of the sheet.

2.3. DIRECTION AND DEVELOPMENT CORNER

Direction assigns the direction for the first cut of a sheet:

- *horizontal direction*: cuts will start horizontally, with the initial cutting level of 1
- *vertical direction*: cuts will start vertically, with an initial cutting level of 0 (otherwise called: head cuts)
- *direction non set*: each single optimization sheet may have either vertical or horizontal direction, depending on which gets the best result for that sheet.

Corner selection assigns the starting vertex for the sheet placements, with scraps accumulated around the opposite corner. There are four values:

- 0 = left-bottom
- 1 = left-top
- 2 = right-bottom
- 3 = right-top

TPA_C always uses the Corner setting as assigned.
Information corresponds to Direction and Corner project general assignments.

2.4. MATCHING GROUPS AND FILTERS

The assignment of *rectangles* and *sheets* has generally technological settings to apply matching and/or filter conditions between *rectangles* and *sheets*.

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

The affected settings correspond to fields in OneRect and OneSheet structures:

- *Thickness*: (double type) thickness
- *Material*: (string type) generic material assignment
- *Grain*: (integer type) assigns grain or grain direction.

Thickness field: matching is assessed on the equality of set values, if their difference is smaller than the epsilon of linear comparison corresponding to 0.1 mm.

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

- two rectangles are assigned (identifiers: 1, 2) with field *Thickness* =18.0
- a rectangle is assigned (identifier: 3) with field *Thickness* =25.0
- a sheet is assigned (identifier: 1) with field *Thickness* =0.0
- a sheet is assigned (identifier: 2) with field *Thickness* =18.0

this situation determines the assignment of 3 matching groups:

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

Each group is optimized separately. From the example above, it is clear that only group 1 can determine a cutting solution, making possible the association between rectangles and sheets: identification rectangles (1, 2) can be placed on sheets with identifier 2.

When multiple values are assigned (for example, also for material), matching associations between rectangles and sheets can become more complex. An example:

- group 1, matching *Thickness* = 18.0 and *Material*= "mahogany"
- group 2, matching *Thickness* = 18.0 and *Material*= "birch"
- group 3, matching *Thickness* = 25.0 and *Material*= "mahogany"

2.4.1. GRAIN CONTROL

The field appears in OneRect and OneSheet structures, corresponding to the grain or grain direction assignment:

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

There are three assignable values:

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

The grain assignment does not necessarily lead to determining matching separate groups: rectangles with the same value in the *Grain* field can be placed in sheets of different groups and/or with the same or different value in the *Grain* field.

Let us see which criteria are applied, if rotation to the rectangle is allowed:

- a rectangle with *Grain=(1, 2)* can be placed with any rotation in sheet with *Grain=0*
- a rectangle with *Grain=(1, 2)* can be placed in sheet with *Grain=(1, 2)* rotating in such a way that the assigned direction for both is respected
- a rectangle with *Grain=0* can be placed with any rotation regardless of the *Grain* field of the sheets.

A rectangle with Grain=(1, 2), but excluding the possibility of rotation, can only be placed on sheets with the same Grain.

If a rectangle with rotation enabled and with *Grain=(1, 2)* can be placed in different sheets for grain, no privileged placement is guaranteed in sheet with the same grain assigned.

2.5. CUTS

Various information helps to determine the placement and sequence of cuts, both in relation to the rectangles to be cut and to the sheet edges.

General information assigns the diameter of the tool used for the cuts: it corresponds to the CutterDiameter setting.
The tool is used for all cuts required by the cutting pattern.

There are two types of cuts:

- **Rip cuts**: the cut runs through the panel along X dimension.
- **Cross cuts**: the cut runs through the panel along Y dimension.

The cuts that are assigned in a cutting pattern are differentiated based on specific codes, referred to as *cutting levels* below:

- **Head cut**: cross cut type that generates a *panel* (level 0): a head cut can only result from optimization of a sheet with *Vertical direction*
- **Rip cut**: longitudinal cut that generates a *strip* along the entire length of the panel (level 1)
- **Cross cut**: transverse cut that generates an element by sectioning a strip (level 2)
- **Z**: longitudinal cut that generates an element by sectioning a portion of the panel obtained with level 2 cut (level 3)

- **W**: transverse cut that generates an element by sectioning a portion of the panel obtained with level 3 cut (level 4)
- **Cut #5**: longitudinal cut that generates an element by sectioning a portion of the panel obtained with level 4 cut (level 5)
- **Cut #6**: transverse cut that generates an element by sectioning a portion of the panel obtained with level 5 cut (level 6).

The sequence of levels matches the number of times a base material has to be turned from longitudinal cut to cross cut and vice versa to complete a piece. Each turn of a panel or strip will increase the level by one. This can be illustrated as follows, let's say for the case of *Horizontal direction*:

- the first cut direction is level 1 (longitudinal)
- the material strips obtained will have to be cut in the other direction (level 2, cross)
- if necessary, a single portion thus obtained may be divided by further longitudinal cuts (level 3)
- and so on, up to a maximum of 6 cutting levels.

2.5.1. CUTTING LEVELS

The sequence of cuts in a pattern obtained from the optimization respects a maximum level of cuts, as assigned in the MaxCutLevels property:

- **Level 1**: only head cuts (if optimization in *Vertical direction*) or rip cuts (if optimization in *Horizontal direction*) are inserted into a pattern
- **Level 2**: only cross cuts are inserted into a strip
- **Level 3**: only Z cuts are inserted into a cross element
- **Level 4**: only W cuts are inserted into a z element
- **Level 5**: only level 5 cuts are present in a W element
- **Level 6**: no cuts beyond level 6 are inserted in the optimizer

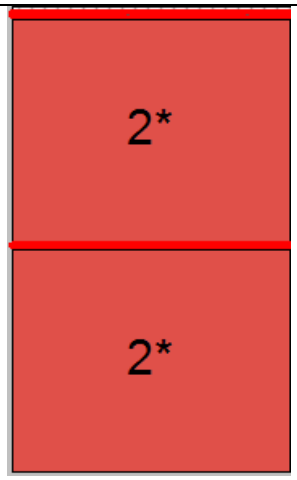
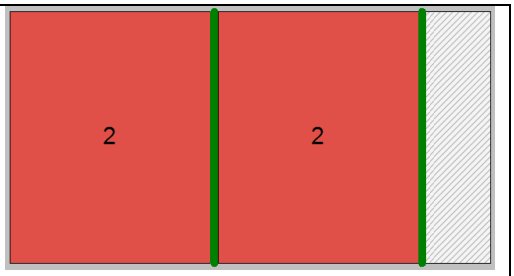
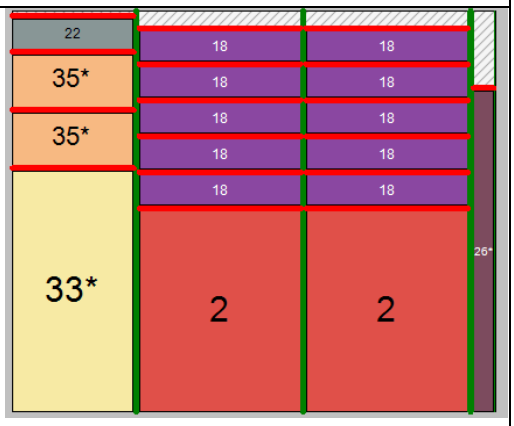
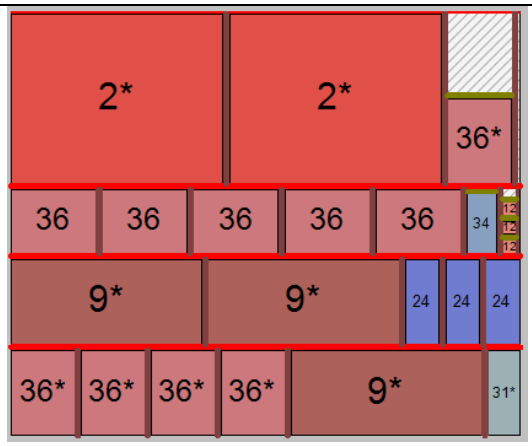
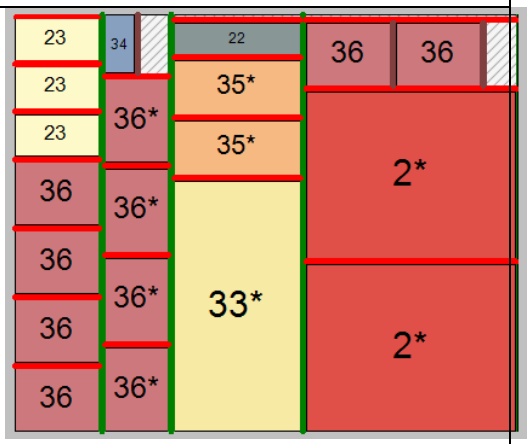
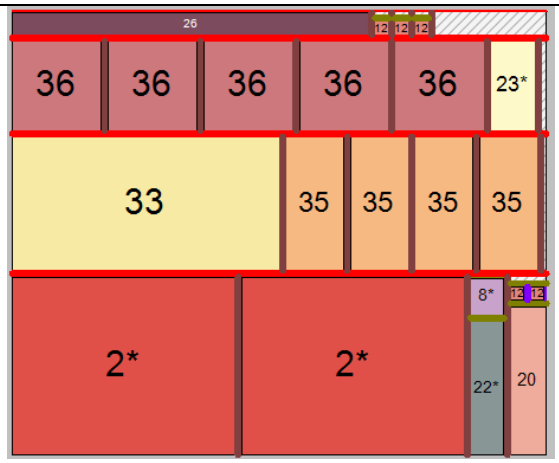
The maximum cutting level is 6. Level 5 and 6 cuts do not have a specific name.

The number of levels allowed will determine the complexity of the cutting patterns:

- a limitation on the number of levels will simplify the cutting patterns
- simplified cutting patterns will reduce cutting operation time
- a limitation on the number of levels can lead to greater waste of material.

The following pictures correspond to different values of MaxCutLevels property:

MaxCutLevels	*Direction*= Horizontal	*Direction*= Vertical

1	 2* 2*	 2 2
2	 36 36 36 36 36 2* 2* 18* 18* 2* 2* 18* 18*	 22 35* 35* 33* 18 18 18 18 18 18 18 18 18 18 2 2 26*
3	 2* 2* 36* 36 36 36 36 36 34 12 12 12 9* 9* 24 24 24 36* 36* 36* 36* 9* 31*	 23 23 23 36 36 36 36 34 36* 36* 36* 36* 22 35* 35* 33* 36 36 2* 2*
4	 26 12 12 12 36 36 36 36 36 23* 33 35 35 35 35 2* 2* 8* 12 12 22* 20	 34 12 12 12 36 36 34 36 36 34 36* 36* 36* 36* 2* 2* 2* 2*

Different colors are used to represent the different cutting levels:

- **Head cuts**: green color
- **Rip cuts** (level 1): red color
- **Cross cuts** (level 2): brown color
- **Z cuts** (level 3): olive green color
- **W cuts** (level 4): purple color.

2.5.2. TRIMS

You can add spacing based on the cutting levels using the following *double* type properties:

- *PreCut:* Quantity added to all head cuts (zero level cuts)
- *LongCut:* Quantity added to all first level cuts
- *TransvCut:* Quantity added to all second level cuts
- ZCut: Quantity added to all third level cuts.

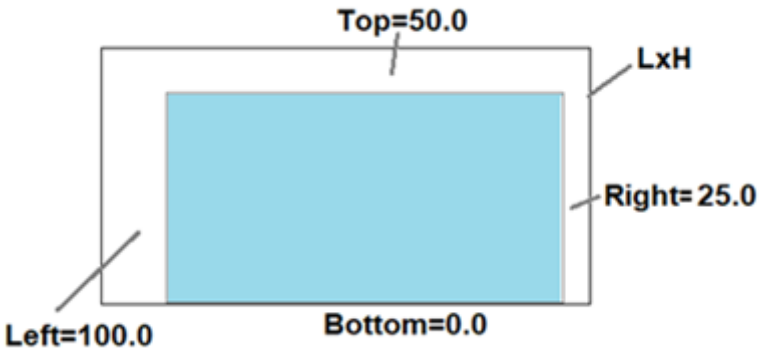
The set values are added to the cutting diameter.
For example: with (CutterDiameter = 5, LongCut = 8), all first level cuts have thickness of 13.

2.5.3. TENSION FREE CUT

The functionality referred to as Tension Free Cut (TFC) allows additional cuts to be added at the longitudinal cuts, so as to delete the "banana" effect due to tensions in the sheet.
The assignment corresponds to the property of *double* type *TensionGap:* the value is added to both the cutting diameter and the *LongCut* parameter.

2.5.4. SHEET MARGINS

Given the *sheet* dimensions, it is possible to assign side areas that are not useful for placement purposes, differentiated by side. The picture shows a sheet of dimensions LxH and with four different outer margins assigned: the useful area for placements is the inner coloured one.



The outer margins setting corresponds to OneSheet structure fields: BorderLeft, BorderRight, BorderTop and BorderBottom.
For all fields it is possible to assign a null or positive value.

2.6. OPTIMIZATION IN TPA_C

Optimization in TPA_C has the primary aim of achieving the best overall use of the available material (*sheets*) with the placements requested (*rectangles*) and with the conditions set (*general settings*).

The optimization essentially runs a number of *attempts* and chooses the result of the attempt offering the best use and, therefore, *the best solution*.
The sequence of these attempts modifies certain *parameters* as to generate different placement situations.
The nature of these that we generally define as *parameters* concerns:

- changing the filling logic of a sheet
- changing the placement order of rectangles
- changing the rotation of a rectangle
- changing the cut direction (horizontal or vertical; only if enabled with the Direction property).

But what qualifies a situation as better than another one?

Let us take a look at some very general criteria:

- the largest placement area is preferred. A solution that has pieces occupying 93.0% of the sheets is better than a solution that results in a 88.00% filling;
- the "most reusable" solution is preferred (it means having a single scrap area large enough to be reused);
- additional assessment criteria are be applied in respect of the arrangement of pieces, as well as the number and size of the pieces placed and the internal scraps.

The optimization procedure leads to the definition of a solution that can be determined in a repetitive manner, keeping unchanged all boundary settings that may affect its development.

2.6.1. PROGRESSIVE OPTIMIZATIONS

The optimizer may calculate and make more solutions accessible, up to a maximum of 5.
All proposed solutions use the same number of sheets.

All calculated optimizations remain available until:

- a new total optimization (see function: Compute)
- a total initialization request (see function: ClearAll).

2.6.2. OPTIMIZATION CRITERIA AND PRIORITIES

Upon optimization request, TPA_C starts to analyse the assigned lists and the next optimization step.
The list analysis may encounter error situations, which may result in the optimization being cancelled.

Let us take a look at this first step of analysis, distinguishing between rectangles and sheets.

Checks on the list of *rectangles*:

- Non-enabled rectangles are excluded from optimization (*Enable* field in *OneRect* structure)
- Rectangles with both null requested quantity (*N* field) and extra (*Extra* field*)* are excluded from the optimization
- Rectangles with one or both dimensions assigned < 1.0 mm are excluded from optimization
- Rectangles that do not match any valid *sheets* are excluded from optimization (example: rectangle with *Material*="abc" and no sheet with the same setting)

Checks on the list of *sheets*:

- Non-enabled sheets are excluded from optimization (*Enable* field in *OneSheet* structure)
- Sheets with null available quantity (*N* field) are excluded from optimization
- Sheets with one or both dimensions < 1.0 mm are excluded from optimization
- Sheets that do not match any valid *rectangles* are excluded from optimization (example: sheet with *Material*= "abc" and no rectangle with the same setting).

The reported result must be able to use at least one element for each list of rectangles and sheets.
Starting the optimization process applies specific criteria to assign the order in which rectangles and sheets are used.

2.6.2.1. USE OF SHEETS

The use of sheets follows an order that can take different situations into account.

Now let us take a look at the main criteria that are applied to the initial sorting of sheets. Points are applied according to the order given, moving on to the next if it has not been possible to make a choice prevailing over the previous point:

- sort by ascending priority (except 0, which runs last)
- sort the sheets marked as scrap first (*IsScrap* field in *OneSheet* structure), with equal priority
- the OrderSheetDim general setting enables sheet sorting by size (sort by descending area)
- sort by ascending type (*ID* field in *OneSheet* structure) in the same way as the other conditions in the previous points.

The UseBeforeScrap general setting enables the application of the *sheet* qualification field to be applied as scrap.

2.6.2.2. USE OF RECTANGLES

Now let us look at the main criteria that are applied to the initial sorting of rectangles. Points are applied according to the order given, moving on to the next if it has not been possible to make a choice prevailing over the previous point:

- sort by ascending priority (except 0, which runs last)
- rectangles with requested quantity assigned (positive). I.e.: sort after rectangles only requiring extra placements
- sort by descending area
- rectangle with no rotation possible
- rectangle with larger requested quantity
- rectangle with ascending type (*ID* field in *OneRect* structure)

EXTRA PLACEMENTS

It is possible to assign extra placements for rectangles, in addition to or as an alternative to the ones actually requested. That is to say, for a rectangle it is possible to assign also or only extra placements.
Information is assigned in the *Extra* field of OneRect structure: a positive value directly assigns the number of extra placements.

Extra placements are used on a sheet only after checking that it is no more possible to place the rectangles requested. It is therefore clear that under no circumstances a sheet can be used only to apply extra placements.

2.7. SUPPORT FOR MANAGING CUTTING PROJECTS

A cutting project is understood as the set of all information assigned to the TPA_C library:
- overall enable settings
- lists of rectangles, sheets.

TPA_C exposes methods of serialization of a project. These methods are particularly useful for:
- testing situations
- TPA_C integration cases in which information provided here in the list assignment structures is sufficient.

2.8. KNOWN LIMITS OF THE LIBRARY

Assignment of rectangles

- up to 500 different *rectangles* can be assigned
- for each *rectangle*, a maximum placeable quantity of 999 can be assigned.

Assignment of sheets
- up to 100 different *sheets* can be assigned
- for each *sheet*, a maximum usable quantity of 999 can be assigned.

3. GUIDE TO THE LIBRARY FUNCTIONS

This chapter describes in detail the properties and functions of TPA_C.

3.1. LICENSE MANAGEMENT

3.1.1. ISVALIDLICENSE

Boolean type property testing the key presence and status.

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

Notes
The property query performs an actual key reading, but only if no cut optimization is running.
No optimization can be performed if the key is not valid.
The presence and status of the key are verified by the library on a time basis.

3.2. CONSTANTS

The following constants are available in the *TpaCutOEM.TpaRctCut* class:

MAX_ROW_ITEMS	500	Maximum number of rectangle types
MAX_ROW_N	999	Maximum value of placements requested for one rectangle
MAX_SHEET_ITEMS	100	Maximum number of sheet types
MAX_SHEET_N	999	Maximum quantity of placements available for one sheet
MAX_CUSTOM_SETTINGS	10	Maximum number of project general settings
MAX_LL_SETTINGS	250	Maximum length of single general setting of project (string length)
MAX_ROW_PARAMS	15	Maximum number of general settings in a single rectangle
MAX_LL_PARAMS	50	Maximum length of single general setting of rectangle (string length)
MAX_CUT_LEVELS	6	Maximum number of cutting levels

3.3. ENUMERATIONS

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

3.3.1. CUTERROR

Assignment of library-managed errors:

- *ErrorNone*: no error
- *ErrorKey*: license not verified
- *ErrorMemory*: memory error
- *ErrorRectEmpty*: empty rectangle list or no rectangle enabled or placeable
- *ErrorSheetEmpty*: empty sheet list or no sheet enabled
- *ErrorNoMatch*: no corresponding match with the sheet list
- *ErrorIOfile*: error managing project files (path and/or file access error, …)
- *ErrorFileNotValid*: error managing project files (not valid format)
- *ErrorNoSolutions*: no solution is assigned
- *ErrorUnexpected*: general or unmanaged error (e.g., invalid assigned index).

3.4. STRUCTURES

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

3.4.1. ONERECT

General assignment of a *rectangle*. The structure exposes methods for initializing fields to default values.

- *int ID*: numeric identifier of *rectangle* (univocal, strictly positive) {*default* = 0}
- *bool Enable*: enabling the use of the *rectangle* (false = does not place the rectangle) {*default* = true}
- *string Label*: descriptive name of the rectangle (can be =""; maximum length 50 characters) {*default* =""}
- *double Length*: length (>= 0.0) {*default* = 0.0}
- *double Height*: height (>= 0.0) {*default* = 0.0}
  - values are in units of: Unit
- *double Thickness*: thickness (>= 0.0) {*default* = 0.0}.
  - assign the field with differentiated values if you need to match the corresponding *sheet* field
  - values are in units of: Unit
- *int N*: quantity requested for placements (>= 0) {*default* = 0; maximum value = 999}
- *int Extra*: extra quantity available (significant if > 0) {*default* = 0; maximum value = 999}
  - The *N field* sets the quantity requested for the rectangle. *Extra* field sets the extra quantity. Extra pieces are only inserted after trying to place the requested quantities of all *rectangle* types.
- *string Material*: material {*default* = "”}
  - assign the field with differentiated values if you need to match the corresponding *sheet* field
  - the real meaning of the field lies with the external application
- *int Grain*: grain direction (0 = none; 1 = horizontal; 2 = vertical) {*default* = 0}
  - assign the field with a value of 1 or 2 if you need to match the corresponding *sheet field*. A rectangle with field (1, 2) can be placed on a *sheet* with grain only if the direction of the grain itself can be observed (with possible application of rotation); a rectangle without grain is applicable to sheets with any *Grain* field
- *int Priority*: priority of use (>= 0) {*default* = 0}
  - *rectangles* with lower number priority have placement priority in the cutting pattern solution
  - *rectangles* with priority 0 will be inserted last
- *bool Rotate*: rotation (*false* = not allowed; *true* = allowed) {*default* = *true*}
  - *true* value is to enable 90° counterclockwise rotation
- *string MatEdge1, MatEdge2, MatEdge3, MatEdge4*: Material of the edges, differentiated on the 4 sides of the rectangle (top, bottom, right, left, respectively) {*default* = "”}
- *double ThickEdge1, ThickEdge2, ThickEdge3, ThickEdge4*: Thickness of the edges, differentiated on the 4 sides of the rectangle {*default* = 0}
- *bool BoolEdge1, BoolEdge2, BoolEdge3, BoolEdge4*: enabling the rectangle to reduce the thickness corresponding to edges, differentiated on the 4 sides of the rectangle, to be used in cut optimization {*default* = false}

- cut optimizer uses the edge information to apply a possible reduction in the size of the rectangle, as obtained from the cutting pattern. The reduction is applicable on each side of the rectangle, with verification of corresponding information (*ThickEdge >0.0, BoolEdge =true)*
- edge-related fields will be usable in the edgebanding station, after a cut optimization
- edge-related fields are only meaningful if EnableRectEdge=true.

- *string Param1, Param2…, Param15*: 15 generic parameters that do not affect optimization {*default* = "”}
  - fields are available for assigning management or generally technological information. Examples: information on the order (customer, order number, notes), added information on the material.

3.4.2. ONESHEET

General assignment of a *sheet*. The structure exposes methods for initializing fields to default values.

- *int ID*: numerical identifier of the *sheet* (univocal, strictly positive) {*default* = 0}
- *bool Enable*: enables the use of the *sheet* (false = does not place the sheet) {*default* = true}
- *string Label*: descriptive name of the *sheet* (can be =""; maximum length 50 characters) {*default* =""}
- *bool Scrap*: identifies the sheet as retrieved (e.g., previously optimized sheet scrap) {*default* = false}
  - *sheets* of this type may have precedence of use
  - field application is assigned by the *OrderBeforeScrap* setting

- *double Length*: length (>= 0.0) {*default* = 0.0}
- *double Height*: height (>= 0.0) {*default* = 0.0}
  - values are in units of: Unit
- *double Thickness*: thickness (>= 0.0) {*default* = 0.0}.
  - assign the field with differentiated values if you need to match the corresponding *rectangle* field
  - values are in units of: Unit

- *int N*: available quantity (>= 0) {*default* = 0; maximum value = 100}

- *string Material*: material {*default* = "”}
  - assign the field with differentiated values if you need to match the corresponding *rectangle* field
  - the real meaning of the field lies with the external application
- *int Grain*: grain direction (0 = none; 1 = x = horizontal; 2 = y = vertical) {*default* = 0}
  - assign the field with a value of 1 or 2 if you need to match the corresponding *rectangle* field

- *int Priority*: priority of use (>= 0) {*default* = 0}
  - lower number priority *sheets* have priority of use
  - *sheets with priority = 0* will be used last

- *double Cost*: sheet cost per unit area {*default* = 0}
  - information unit shall be assigned by the client as its use is delegated to the client
  - typical units can be: €/sq.m. (euro/square meter), $/ft2 (dollar/square foot)

- *double BorderBottom, BorderTop, BorderRight, BorderLeft*: sheet margins {*default* = 0}
  - values are in units of: Unit

3.5. DEFINITION OF FUNCTIONS

Let us examine here how to assign all the functions of TPA_C: with reference to the functions, the wording of settings is also used.
When initializing TPA_C, all settings are assigned to the default values, indicated below with locution {*default* = …}.

In order to provide a more comprehensive framework, assignments are divided into four groupings:

- *General functions*: assignments of generic use

- *Functions corresponding to general settings of cut optimization*: assignments that mean general customizations of projects and cut optimization. Information of this kind usually remains unchanged and is not serialized into a file corresponding to a project
- *Functions related to general assignments of a cutting project*: assignments that mean specific assignments of a project. Information of this kind is usually serialized into a file corresponding to a project.

3.5.1. GENERAL FUNCTIONS

3.5.1.1. VERSION

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

3.5.1.2. LASTERROR

CutError type property that returns the last error encountered. The value is updated in all procedures that can diagnose an anomalous situation.

3.5.1.3. ERRORMESSAGE

The function returns the assigned message for the error indicated.

string ErrorMessage (CutError Error)

Arguments
- *Error*: enumeration value

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

3.5.2. FUNCTIONS CORRESPONDING TO GENERAL SETTINGS OF CUT OPTIMIZATION

This information does not change as the project changes: it concerns settings usually attributable to general operation options of an application.

3.5.2.1. NUMBERCUSTOM

Integer type property that assigns/returns the number of general settings of the project {*default* = MAX_CUSTOM_SETTINGS, min=0, max = MAX_CUSTOM_SETTINGS}.
The value can limit the number of general settings of the project that TPA_C manages (for example: while reading and/or recording the project to and from file).

3.5.2.2. NUMBERRECTPARAM

Integer type property that assigns/returns the number of parameters in each OneRect {*default* = MAX_ROW_PARAMS, min=0, max = MAX_ROW_PARAMS}.
The value can limit the number of general settings of the project that TPA_C manages (for example: while reading and/or recording the project to and from file).

3.5.2.3. ENABLERECTEDGE

Boolean type property that assigns/returns the activation to assign the edges of rectangles {*default* = true}.
In particular:
- *true*: the optimizer can use a small rectangle, due to the application of edges
- *false*: the fields related to the edges of a rectangle are not serialized (on project files).

3.5.2.4. ORDERSHEETDIM

Boolean type property that assigns/returns the enablement to sort sheets by size {*default* = false}.
Value =*true:* enables the optimization to apply sheet sorting by decreasing area.

3.5.2.5. ORDERBEFORESCRAP

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

Value =*true:* enables optimization to use the scrap sheets before the others.

3.5.2.6. ORDERSTRIPS

Boolean type property that assigns/returns the enablement to sort the strips in a cutting pattern {*default* = *true*}.

Please note that a strip corresponds to a portion of the panel obtained by longitudinal cut of level 1.

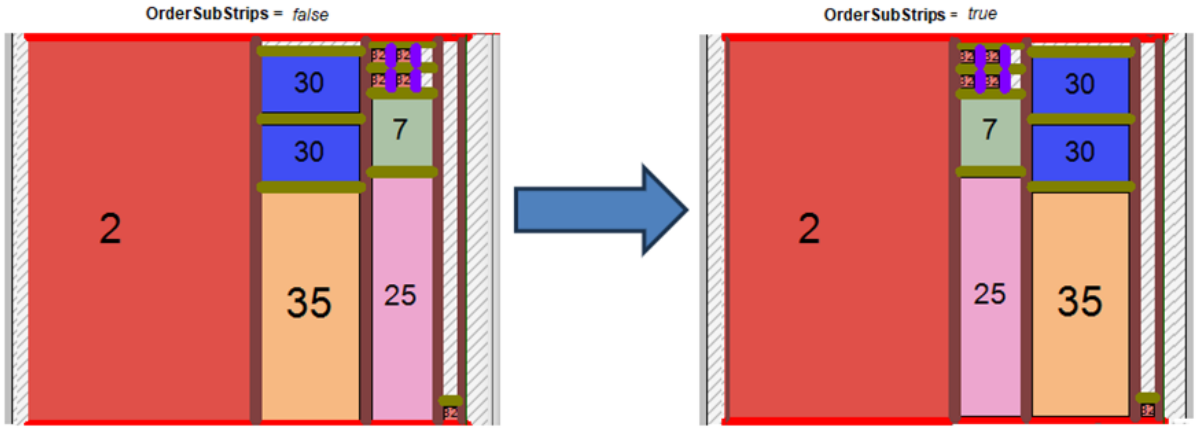
The figure shows the same cutting pattern as obtained in both cases:

- *false*: (left) the strips are reported in calculation order
- *true*: (right) the order of the strips is such as to respect:
  - increasing values of wasted material inside the strip
  - increasing values by strip height (equal waste).

3.5.2.7. ORDERSUBSTRIPS

Boolean type property that assigns/returns the enablement to sort pieces within a strip or strip portion of a cutting pattern {*default* = *true*}.

The figure shows the same strip as obtained in both cases:



- *false*: (left) the strip cuts are listed in order of calculation
- *true*: (right) the order in each portion of the strip is such as to respect
    - increasing values of wasted material within the portion.

3.5.3. FUNCTIONS RELATED TO GENERAL ASSIGNMENTS OF A PROJECT

This information can change as the project changes.

3.5.3.1. UNIT

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

3.5.3.2. CUTTERDIAMETER

Double type property that assigns/returns the diameter of the cutting tool {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.
The placement between *rectangles* applies a minimum spacing matching the set value.

3.5.3.3. DIRECTION

Short type property that assigns/returns the direction of the first cutting level applied to sheets {*default* = 0; *validity range*: 0/1/2}:

- 0 = horizontal direction
- 1 = vertical direction
- 2 = direction determined by optimizer.

In case of value 2, the optimizer will always search for the optimal cutting pattern for each sheet type. In this case: as a result of the optimization, some sheets may have the first cut direction that matches the length dimension and others may have the first cut direction that matches the height dimension.

3.5.3.4. CORNER

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

- 0 = left-bottom
- 1 = left-top
- 2 = right-bottom
- 3 = right-top.

3.5.3.5. MAXCUTLEVELS

Integer type property that assigns/returns the maximum cutting level that can be used in optimization {*default* = MAX_CUT_LEVELS; min = 1; *max* = MAX_CUT_LEVELS}.

Value 1 corresponds to the possibility to perform only one cutting level:

- Level 0 (head cut) in case of optimization in vertical direction
  - requires rectangles with the same height as the raw panel
- Level 1 (longitudinal cut) in case of optimization in horizontal direction
  - requires rectangles with the same length as the raw panel.

3.5.3.6. TENSIONGAP

Double type property that assigns/returns an additional distance between the first level cuts in order to relieve the sheet tension {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.

3.5.3.7. PRECUT

Double type property that assigns/returns a trim added to all zero level cuts {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.

3.5.3.8. LONGCUT

Double type property that assigns/returns a trim added to all first level cuts {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.

3.5.3.9. TRANSVCUT

Double type property that assigns/returns a trim added to all second level cuts {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.

3.5.3.10. ZCUT

Double type property that assigns/returns a trim added to all third level cuts {*Unit*: Unit; *default* = 0.0; *validity range*: >= 0.0}.

3.5.3.11. CUSTOM1, CUSTOM2, …, CUSTOM10

String type properties that assign/return generic project information that does not affect the optimization process {default = ""; maximum number of properties = NumberCustom; maximum length of a parameter = MAX_LL_SETTINGS}.

3.5.3.12. CLEARALL

The function clears the project lists (rectangles, sheets) and any calculated solution.

bool ClearAll ()

Return value
true if result is positive.
Currently no situation can determine return as *false*.

3.6. DEFINITION OF RECTANGLES

3.6.1. ADDRCT

The feature adds a rectangle to the list.

bool AddRct (OneRect Item)

Arguments
- *Item*: structure of rectangle assignment

Return value
true if result is positive, *false* if the rectangle has not been inserted

Notes
A *false* return corresponds to one of these error situations:

- the list of rectangles is already at the maximum allowed (500 elements)
- the rectangle has an invalid identifier assigned (***Item.ID*** must be strictly positive)
- a rectangle with ***Item.ID*** numerical identifier is already assigned
- the rectangle has invalid dimensions, or invalid edge dimensions. Specifically: subtracting thickness of the edges from the rectangle dimensions would reduce one or both dimensions to value < 1.0 mm.

3.6.2. CLEARRCT

The function clears the list of rectangles.

bool ClearRct ()

Return value
True if the list was deleted correctly

3.6.3. COUNTRECT

Int type property gets the number of rectangles in the list.

3.6.4. READRECT

The function searches for a rectangle matching the specified ID.

CutError ReadRect (*int ItemID, ref *OneRect Item)

Arguments
- *ItemId*: rectangle identifier (> 0)
- *Item*: structure of rectangle assignment

Return value
CutError.ErrorNone if result is positive.

Notes
A return *other than CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorUnexpected*) either an invalid identifier is assigned (***ItemID*** must be strictly positive) or no rectangle with an ***ItemID*** numerical identifier is assigned.

3.6.5. READRECTINDEX

The function searches for a rectangle matching the specified index.

CutError ReadRectIndex (*int Index, ref *OneRect Item)

Arguments
- *Index*: index (zero base) on the list of rectangles (>= 0)
- *Item*: structure of rectangle assignment

Return value
CutError.ErrorNone if result is positive

Notes
The primary use of the function is for acquiring rectangles after executing the ***LoadProject*** function.

A return *other than CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorUnexpected*) an invalid index is assigned.

3.7. DEFINITION OF SHEETS

3.7.1. ADDSHEET

The feature adds a sheet to the list.

bool AddSheet (OneSheet Item)

Argument
- *Item*: structure of sheet assignment

Return value
true if result is positive

Notes
A return other than *true* corresponds to one of these error situations:
- The sheet list is already at the maximum allowed (100 elements) or the quantity requested in the structure exceeds the maximum allowed (999)
- The sheet has an invalid identifier assigned (***Item.ID*** must be strictly positive)
- A sheet with ***Item.ID*** numerical identification is already assigned
- The sheet assigns invalid margins. Specifically: subtracting margins from the sheet dimensions reduces one or both dimensions to value < 1.0 mm

3.7.2. CLEARSHEET

The function clears the list of sheets.

bool ClearSheet ()

Return value
true if the result is positive

3.7.3. COUNTSHEET

Int type property gets the number of sheets in the list.

3.7.4. READSHEET

The function searches for a sheet matching the specified ID.

CutError ReadSheet (*int ItemID, ref *OneSheet Item)

Arguments
- *ItemId*: sheet identifier (> 0)
- *Item*: structure of sheet assignment

Return value
CutError.ErrorNone if the resut is positive.

Notes
A return other than *CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorUnexpected*) either an invalid identifier is assigned (***ItemID*** must be strictly positive) or a sheet with an ***ItemID*** numerical identifier is not assigned

3.7.5. READSHEETINDEX

The function searches for a sheet matching the specified index.

CutError ReadSheetIndex (*int Index, ref *OneSheet Item)

Arguments
- *Index*: index (zero base) on the sheet list (>= 0)
- *Item*: structure of sheet assignment

Return value
CutError.ErrorNone if the result is positive.

Notes
The primary use of the function is for acquiring sheets after running the ***LoadProject*** feature.

A return other than *CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorUnexpected*) an invalid index is assigned.

3.8. CUTTING SOLUTION

3.8.1. COMPUTE

The function starts the optimization process

bool Compute ()

Return value
true if the result is positive.

Notes
In case of *false* return, you can query the *LastError* property to see what prevented the optimization:
- (*CutError.ErrorKey*) the key presence and status check failed
- (*CutError.ErrorRectEmpty*) the list of rectangles is empty or does not assign enabled items
- (*CutError.ErrorSheetsEmpty*) the list of sheets is empty or does not assign enabled items
- (CutError.ErrorNoMatch) the list of parts and sheets do not have common matching groups

At start-up, the function:
- carries out preliminary checks (mentioned above)
- checks lists of rectangles and sheets.

If the execution of the listed checks leads to an error situation, the function still returns without running any optimization.

3.9. OPTIMIZATION RESULTS

All properties and features listed here fail if a valid optimization is not computed.

3.9.1. NUMBEROFSOLUTIONS

Integer type property that returns the total number of computed solutions.
The return value is:
- 0: no solution is calculated
- >0: number of solutions calculated

3.9.2. SELECTSOLUTION

Integer type property that assigns/returns the *current solution* number.
The return value is:
- -1: no solution is calculated
- >=0: the current solution matches a computed one with a ***Compute()*** function call

Under assignment:
- the property allows you to confirm or change the *current solution*

- no assignment is made if the value is invalid.

3.9.3. SOLUTIONSHEETS

Integer type property that returns the number of sheets of the *current solution*.
The return value is:

- 0: the solution is empty or does not exist
- >0: the solution is valid

A solution sheet is also referred to as: *cutting pattern*.

The returned value does not take into account any repeated sheets.

3.9.4. GETSHEETID

The function returns the ID of a sheet given its index in the *current solution*.

Int GetSheetID (int idx)

Arguments

- *idx:* sheet index (>= 0)

Notes
The function returns 0 in case of invalid index or solution.

3.9.5. NUMBEROFREPETITIONS

The function returns how many times a particular cutting pattern is repeated in the *current solution*.

int NumberOfRepetitions (int idx)

Arguments

- *idx:* sheet index (>= 0)

Notes
The value of the property is significant after running an optimization.
The function returns 0 if there is an invalid index or solution.
The feature returns 1 if the cut scheme is assigned only once (no patterns).

3.9.6. USEDSHEETS

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

int UsedSheets (int ID_Sheet)

Arguments

- *ID_Sheet*: sheet identifier (> 0)

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

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

3.9.7. USEDRECTS

The feature reads the number of placements in a sheet of the current solution or the entire current solution, with the option to match only one type of rectangle.

int UsedRects (int Index, int ID_Part)

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

Notes
The function reads the number of placements corresponding to the assigned arguments:

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

In cases of operation with negative *Index* (-1), the return value takes into account the same repeated sheets. In cases of operation with *Index* >= 0, the return value counts the placements of a single sheet, without taking into account any repetitions of the sheet itself.

3.9.8. FITNESSSHEET

The function returns the efficiency information of a sheet of the *current solution*.

double FitnessSheet (int idx)

Arguments
- *idx:* sheet index (>= 0)

Notes
The value of the property is significant after running an optimization.
The overall efficiency is assessed as a % of the ratio of the area used for placements to the total area of the sheets used. If the solution generates multiple sheets, the efficiency evaluation of the last sheet can appropriately limit the useful area for placements
The function returns fitness as a percentage of one sheet or all current solution:

idx	
= -1	gets the fitness of the whole solution (takes repetitions into account)
>=0	gets the fitness of the idx index sheet

3.9.9. READRESOLRECT

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

int ReadResolRect (int idx, int idxrct, ref double qx, ref double qy, ref bool rotate)

Arguments
- *idx*: index (zero base) of the solution sheet
- *idxrct*: index (zero base) of the placement on the solution sheet
- *qx*: x coordinate of the lower left corner of the rectangle
- *qy*: y coordinate of the lower left corner of the rectangle
- *rotate*: indicates whether the rectangle is rotated (*true* = rotated)

Return value
The identification of the rectangle whose data was read in case of a positive result, 0 otherwise.

Notes

In case of return 0, you can query the LastError property to find out the error:
- (*CutError.ErrorKey*) the key presence and status check failed
- (*CutError.ErrorNoSolutions*) no solution was found
- (*CutError.ErrorUnexpected*) one or both indices (*idx, idxrct)* are invalid

3.9.10. NUMBEROFCUTS

The function returns the number of cuts calculated for a sheet of the *current solution* matching the specified index

int NumberOfCuts(int idx)

Arguments
- *idx*: index (zero base) of the solution sheet

Return value
The number of cuts in a cutting pattern. The function returns 0 in case of invalid index or solution.

3.9.11. READRESOLCUT

The function returns information about a cut on a sheet of the *current solution*

int ReadResolCut (int idx, int idxcut, ref double xStart, ref double yStart, ref double xEnd, ref double yEnd, ref int orientation, ref double thickness)

Arguments
- *idx*: index (zero base) of the solution sheet
- *idxcut*: index (zero base) of cut on solution sheet
- *xStart*: x coordinate of the start point of the cut
- *yStart*: y coordinate of the start point of the cut
- *xEnd*: x coordinate of the end point of the cut
- *yEnd*: y coordinate of the end point of the cut
- *orientation*: cut orientation (0 = vertical, 1 = horizontal)
- *thickness*: cut thickness

Return value
The cutting level whose data was read (-1 in case of error).

Notes
In case of return -1, you can query the *LastError* property to find out the error:
- (*CutError.ErrorKey*) the key presence and status check failed
- (*CutError.ErrorNoSolutions*) no solution was found
- (*CutError.ErrorUnexpected*) one or both indices (*idx, idxcut)* are invalid

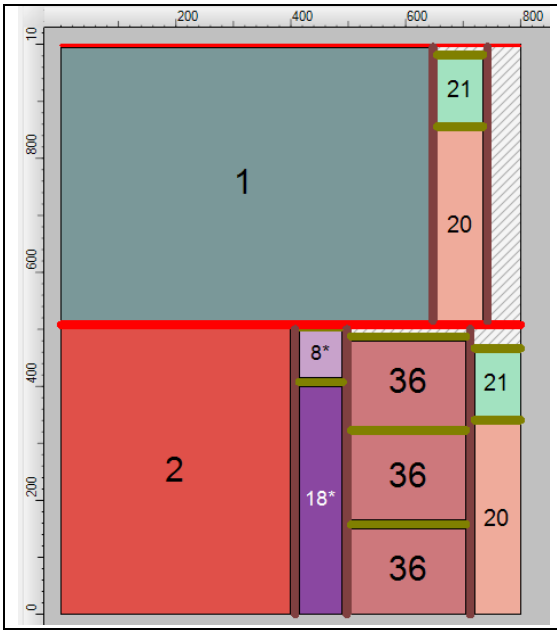
The function return indicates the *cutting level* and is from 0 to 6.
The coordinates of the cut are always referred to the pure Cartesian system (origin at the left-bottom corner).
The *thickness* argument can assign thickness different from the thickness of the cutting tool:
- *bottom*, if the cut is partly outside the useful area of material
- *top*, if the cut is to be repeated (for example: for adding a trim).

The cut sequence follows the division of the sheet into strips and the strips into further parts: the sequence respects the order in which the cuts are arranged on the raw panel, starting from the vertex used for the placements towards the opposite vertex.
No cuts are assigned to separate the sheet edges.

Here is an example of a cutting pattern obtained with optimization in horizontal direction:
- the vertex used for placements is *left-bottom*
- there are 2 longitudinal cuts (colour cuts: red)
- the maximum cutting level is 3 (Z cuts).

The returned cut sequence corresponds to the sequence on the right of the image.

Rulers on the sides of the sheet help to locate the cuts graphically.
The raw sheet has the following dimensions: 800 x 1000 mm.



Longitudinal cut: [0;500] →[800;500]
Cross cut:[400;0] →[400;500]
Cross cut:[490;0] →[490;500]
Z Cut: [415;400] →[490;400]
Z Cut: [415;497] →[490;497]
Cross cut:[705;0] →[705;500]
Z Cut: [505;150] →[705;150]
Z Cut: [505;315] →[705;315]
Z Cut: [505;480] →[705;480]
Cross cut: … not assigned
Z Cut: [720;333] →[800;333]
Z Cut: [720;459] →[800;459]

Longitudinal cut: [0;995] →[800;995]
Cross cut:[640;515] →[640;995]
Cross cut:[735;515] →[735;995]
Z Cut: [655;848] →[735;848]
Z Cut: [655;974] →[735;974]

The last *cross cut* reported for the first *longitudinal cut* is indicated as "not assigned" as it is completely outside the right edge of the sheet.
The *longitudinal cut* at the top of the sheet is clearly less thick than the first one, as it overlaps with the top edge of the sheet.

3.9.12. CUTLINEAR

The function returns the linear development corresponding to the calculated cuts for a sheet of the *current solution* at the specified index

double CutLinear(int idx)

Arguments
- *Idx*: index (zero base) of the solution sheet

Return value
The function returns 0 in case of invalid index or solution; otherwise: the overall linear development of the cuts.

3.9.13. CUTAREA

The function returns the area corresponding to the calculated cuts for a sheet of the *current solution* at the specified index

double CutArea(int idx)

Arguments
- *idx*: index (zero base) of the solution sheet

Return value
The function returns 0 in case of invalid index or solution; otherwise: the area occupied by the cuts.
Area is calculated by applying the thickness of the cuts, as returned by ReadResolCut function.

3.9.14. MARGINAREA

The function returns the area corresponding to the assigned margins for a sheet of the *current solution* at the specified index

double MarginArea(int idx)

Arguments
- *idx*: index (zero base) of the solution sheet

Return value
The function returns 0 in case of invalid index or solution or null assigned margins; otherwise: the area corresponding to the margins.
Margins are assigned in the OneSheet structure of the sheet.

3.9.15. GETWASTE

The function returns the unused area for a sheet of the current solution at the specified index

double GetWaste (int idx)

Arguments
- *idx*: index (zero base) of the solution sheet

Return value
The function returns 0 in case of an invalid index or solution; otherwise: The unused area of the sheet.

The unused area of a sheet shall be calculated by subtracting from the original area of the sheet:

- the area corresponding to the margins
- the area corresponding to the cut rectangles
- the area of the cuts.

3.9.16. ESTIMATED TIME

It assigns the time estimated by the client to execute the cutting pattern of the *current solution* at the specified index

public bool EstimatedTime(int idx, long valueH, long valueM, long valueS)

Arguments
- *idx*: sheet index. If -1: assigns the total time of the solution.
- *ValueH*: number of hours (>=0)
- *ValueM*: number of minutes (>=0)
- *ValueS*: number of seconds (>=0)

Return value
true if the result is positive.

Notes
A return other than *true* corresponds to one of these error situations:
- no solution calculated
- invalid index (idx)
- negative values for time values.

The allocation of time in (hours, minutes, seconds) allows for greater flexibility: the total time in seconds, for example, can be allocated.
The function allows to assign information that can only be determined on the basis of assessments depending on the specific application.

3.9.17. ESTIMATED COST

It assigns the cost estimated by the client to execute the cutting pattern of the *current solution* at the specified index

public bool EstimatedCost(int idx, double value)

Arguments
- *idx*: sheet index. If -1: assigns the total cost of the solution
- *value*: cost (of the sheet or solution)

Return value
true if the result is positive.

Notes
A return other than *true* corresponds to one of these error situations:
- no solution computed
- invalid index (idx)
- negative value for the cost

The function allows to assign information that can only be determined on the basis of assessments depending on the specific application.

3.9.18. EXPORT

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

CutError Export (string pathName)

Arguments
- *pathName*: File path

Return value
CutError.ErrorNone if the result is positive.

Notes
A return other than *CutError.ErrorNone* corresponds to one of these error situations:
- (*CutError.ErrorNoSolutions*) no solution has been computed
- (CutError.ErrorKey) invalid license or key not found
- (*CutError.ErrorIOfile*) there was an error accessing the file to be written
- (CutError.ErrorUnexpected) an unexpected error has occurred.

The saved files can be interpreted by an external application as an alternative to querying exposed functions. In particular:
- the files have "XML" format: any extension ".XML" is added to *pathName,* if necessary
- one file is saved for each calculated cutting scheme
- each file includes information about any repetition of the cutting pattern
- the first file is saved as *pathName+"_001"*
- the second file is saved as *pathName+"_002"*
- until cutting patterns are exhausted
- existing files are overwritten.

FILE STRUCTURE CORRESPONDING TO A CUTTING PATTERN

The master node is the **MAIN** node and contains general information distributed in nested elements:

```
<MAIN>
 <GENERALSETTINGS Info1="Client ABC" Info2="ABC12345" />
 <TECHSETTINGS Unit="0" Direction="0" Corner="0" MaxCutLevel="3" />
 <DIM Code="3.0" L="800.0" H="1000.0" T="0.0" G="N" Scrap="0" Priority="0" />
 <DIMTRIMS BladeThickness="15.0" PreCut="0.0" LongCut="0.0" TransvCut="0.0" ZCut="0.0" />
 <DATA Rep="2" />
 <PIECESLIST>
  <PIECE Code="1" L="640.0" H="480.0" T="0.0" Used="1" Rotation="1" G="N" Priority="0" …/>
  <PIECE Code="2" L="400.0" H="500.0" T="0.0" Used="1" Rotation="1" G="N" Priority="1" … />
  …
 </PIECESLIST>
 <DRAW>
  …
 </DRAW>
```

</MAIN>

<GENERALSETTINGS> NODE

provides general project information (see: Custom1, Custom2, …, Custom10)

- attributes: *Info1*,… , *Info10*

<TECHSETTINGS> NODE

lists some project assignments

- *Unit* attribute: unit of measure of the project (see: Unit)
- *Direction* attribute: forward direction for placements (see: Direction)
- *Corner* attribute: starting vertex for placements (see: Corner)
- *MaxCutLevel* attribute: the maximum cutting level that can be used in optimization (see: MaxCut-Levels)

<DIM> NODE

provides general sheet information (see structure: OneSheet)

- *Code* attribute: *ID* field of structure
- *L* attribute: *Length* field of structure
- *H* attribute: *Height* field of structure
- *T* attribute: *Thickness* field of structure
- *Descr* attribute: *Label* field of structure
- *Mat* attribute: *Material* field of structure
- *G* attribute: *Grain* field of structure (values: "N"=no grain, "X"=x grain, "Y"=y grain)
- *Scrap* attribute: *Scrap* field of structure
- *Priority* attribute: *Priority* field of structure
- *TopMargin* attribute: *BorderTop* field of structure
- *BottomMargin* attribute: *BorderBottom* field of structure
- *RightMargin* Attribute: *BorderRight* field of structure
- *LeftMargin* Attribute: *BorderLeft* field of structure
- *Cost* Attribute: *Cost* field of structure

<DIMTRIMS> NODE

reports the assignments of the project trims

- *BladeThickness* attribute: thickness of the cutting tool (see: CutterDiameter)
- *PreCut* attribute: trim added to all head cuts (see: PreCut)
- *LongCut* attribute: trim added to longitudinal cuts (see: LongCut)
- *TransvCut* attribute: trim added to cross cuts (see: TransvCut)
- *ZCut* attribute: trim added to Z cuts (see: Zcut)
- *LongCutTFC* attribute: additional distance between first level cuts in order to relieve sheet tension (see: TensionGap)

<DATA> NODE

provides general information about the cutting pattern

- *Rep* attribute: sheet repetition
- *Cost* attribute: estimated cost of execution (as assigned by the client)
- *Time* attribute: estimated time of execution (unit: seconds)- (as assigned by the client)

<PIECESLIST> NODE

lists the rectangles that are cut by the current cutting pattern.
Each one has a <**PIECE**> node, with the general information about the rectangle (see structure: OneRect):

- *Code* attribute: *ID* field of structure
- *L* attribute: *Length* field of structure
- *H* attribute: *Height* field of structure
- *T* attribute: *Thickness* field of structure
- *Descr* attribute: *Label* field of structure
- *Mat* attribute: *Material* field of structure
- *G* attribute: *Grain* field of structure (values: "N"=no grain, "X"=x grain, "Y"=y grain)
- *Rotation* attribute: *Rotate* field of structure
- *Priority* attribute: *Priority* field of structure
- *MatEdge1, ThickEdge1, EnableEdge1* attributes: *MatEdge1, ThickEdge1, BoolEdge1* fields of structure
- *MatEdge2, ThickEdge2, EnableEdge2* attributes: *MatEdge2, ThickEdge2, BoolEdge2* fields of structure
- *MatEdge3, ThickEdge3, EnableEdge3* attributes: *MatEdge3, ThickEdge3, BoolEdge3* fields of structure
- *MatEdge4, ThickEdge4, EnableEdge4* attributes: *MatEdge4, ThickEdge4, BoolEdge4* fields of structure
- *Param1,…,Param15* attributes: fields *(Param1,…,Param15)* of structure

in addition to:
- *Used* attribute: number of placements in the current sheet

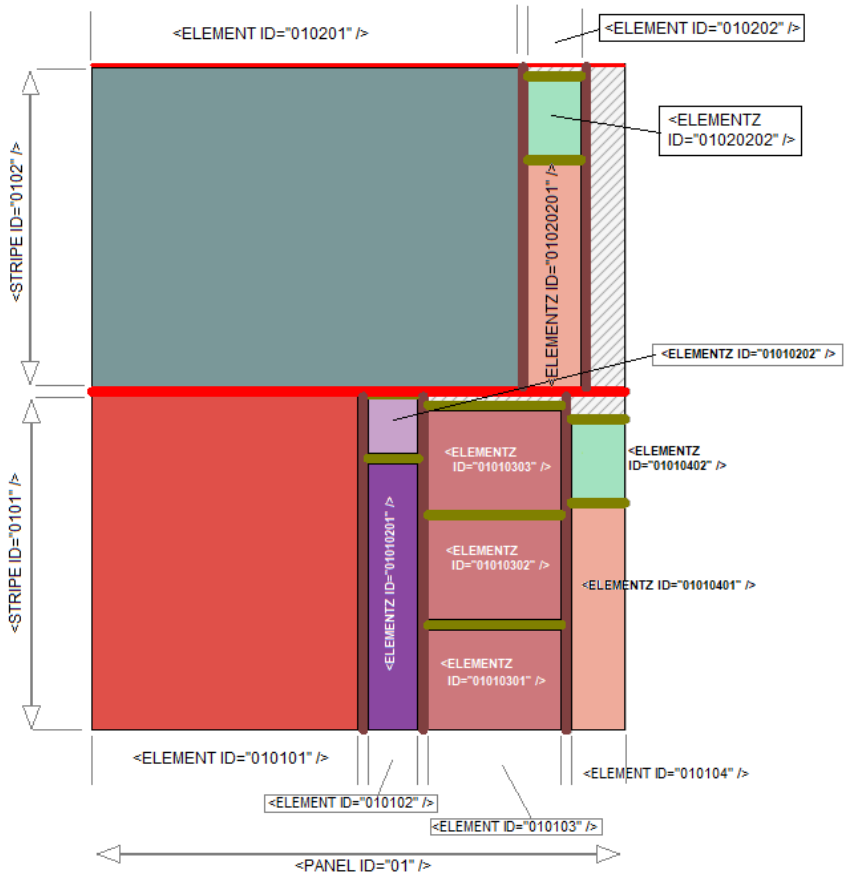
<DRAW> NODE

returns the cutting pattern corresponding to the sheet, but in a different way than previously seen.
Below is the cutting pattern corresponding to the example in paragraph ReadResolCut

```
<DRAW>
 <PANEL ID="01" REP="1" L="800.00" H="1000.000">
  <STRIPE ID="0101" REP="1" L="800.000" H="500.000">
   <ELEMENT ID="010101" REP="1" L="400.000" H="500.000">
    <LABEL Code="2" Rep="1" Rotated="0" />
   </ELEMENT>
   < ELEMENT ID="010102" REP="1" L="75.000" H="500.000">
    <ELEMENTZ ID="01010201" REP="1" L="75.000" H="400.000">
     <LABEL Code="18" Rep="1" Rotated="1" />
    </ELEMENTZ>
    <ELEMENTZ ID="01010202" REP="1" L="75.000" H="82.000">
     <LABEL Code="8" Rep="1" Rotated="1" />
    </ELEMENTZ>
   </ELEMENT>
    <ELEMENT ID="010103" REP="1" L="200.000" H="500.000">
    <ELEMENTZ ID="01010301" REP="1" L="200.000" H="150.000">
     <LABEL Code="36" Rep="1" Rotated="0" />
    </ELEMENTZ>
    <ELEMENTZ ID="01010302" REP="1" L="200.000" H="150.000">
     <LABEL Code="36" Rep="1" Rotated="0" />
    </ELEMENTZ>
    <ELEMENTZ ID="01010303" REP="1" L="200.000" H="150.000">
     <LABEL Code="36" Rep="1" Rotated="0" />
    </ELEMENTZ>
   </ELEMENT>
   <ELEMENT ID="010104" REP="1" L="80.000" H="500.000">
    <ELEMENTZ ID="01010401" REP="1" L="80.000" H="333.000">
     <LABEL Code="20" Rep="1" Rotated="0" />
    </ELEMENTZ>
    <ELEMENTZ ID="01010402" REP="1" L="80.000" H="111.000">
     <LABEL Code="21" Rep="1" Rotated="0" />
    </ELEMENTZ>
   </ELEMENT>
  </STRIPE>
  <STRIPE ID="0102" REP="1" L="800.000" H="480.000">
   <ELEMENT ID="010201" REP="1" L="640.000" H="480.000">
    <LABEL Code="1" Rep="1" Rotated="0" />
   </ELEMENT>
   <ELEMENT ID="010202" REP="1" L="80.000" H="480.000">
    <ELEMENTZ ID="01020201" REP="1" L="80.000" H="333.000">
     <LABEL Code="20" Rep="1" Rotated="0" />
    </ELEMENTZ>
    <ELEMENTZ ID="01020202" REP="1" L="80.000" H="111.000">
     <LABEL Code="21" Rep="1" Rotated="0" />
```

```
      </ELEMENTZ>
    </ELEMENT>
  </STRIPE>
 </PANEL>
</DRAW>
```

The figure highlights the meaning given to each individual node in the "xml" scheme:



<PANEL> NODE

Given the raw panel, it can be separated by head cuts (cuts in vertical direction, covering the height of the raw panel) into several panels: as already mentioned, this is the case of an optimized panel by applying *vertical direction*.
Each element thus obtained corresponds to a section of <**PANEL**> type: in case of panel optimized by applying *horizontal direction*, the section is always one (as in the example).
The order of the <**PANEL**> type sections follow the order in which they are placed on the raw panel, starting from the vertex used for the placements towards the opposite vertex (in the example: from *left-bottom* vertex to *right-top* vertex*)*.

Node attributes are common to all subsequent node types, both in name and meaning.

```
<PANEL ID="01" REP="1" L="800.00" H="1000.000">
```

- *ID* attribute: univocal identifier
- *REP* attribute: element repetition (default= 1)
- *L* attribute: horizontal dimension of the element (net of cuts)
- *H* attribute: vertical dimension of the element (net of cuts)

In our example: the element has the same dimensions as the raw panel (800 x 1000 mm).

<STRIPE> NODE

Nodes inserted inside a <**PANEL**> node, describing the *strips* generated by the longitudinal cuts (cuts in the horizontal direction, covering the length of the element to which it belongs <**PANEL**>). The order of

the <**STRIPE**> type sections follows the order in which they are arranged on the <**PANEL**> element, in accordance with the direction imposed by the vertex used for placements (in the example: from bottom to top).
The node attributes are the same as those examined for the parent node.
In this example, 2 <**STRIPE**> sections are assigned.

```
<DRAW>
 <PANEL ID="01" REP="1" L="800.00" H="1000.000">
  <STRIPE ID="0101" REP="1" L="800.000" H="500.000">
   …

  </STRIPE>
  <STRIPE ID="0102" REP="1" L="800.000" H="480.000">
   …
  </STRIPE>
 </PANEL>
</DRAW>
```

<ELEMENT> NODE

Nodes inserted within a <**STRIPE**> node, describing the elements generated by the cross cuts (cuts in the vertical direction, covering the height of the element to which it belongs <**STRIPE**>). The order of the <**ELEMENT**> type sections follows the order in which they are placed on the <**STRIPE**> element, in accordance with the direction imposed by the vertex used for placements (in the example: from left to right).

Node attributes are the same as those examined for the parent node.
In this example, the following are assigned:

- 4 sections for the first strip
- 2 sections for the second strip.

```
<PANEL ID="01" REP="1" L="800.00" H="1000.000">
   <STRIPE ID="0101" REP="1" L="800.000" H="500.000">
    <ELEMENT ID="010101" REP="1" L="400.000" H="500.000">
     <LABEL Code="2" Rep="1" Rotated="0" />
    </ELEMENT>
    <ELEMENT ID="010102" REP="1" L="75.000" H="500.000">
    …
   </ELEMENT>
   <ELEMENT ID="010103" REP="1" L="200.000" H="500.000">
   …
    </ELEMENT>
    <ELEMENT ID="010104" REP="1" L="80.000" H="500.000">
   …
    </ELEMENT>
   </STRIPE>
   <STRIPE ID="0102" REP="1" L="800.000" H="480.000">
    <ELEMENT ID="010201" REP="1" L="640.000" H="480.000">
   …
    </ELEMENT>
    <ELEMENT ID="010202" REP="1" L="80.000" H="480.000">
   …
    </ELEMENT>
   </STRIPE>
  </PANEL>
```

<ELEMENTZ> NODE

Nodes inserted within an <**ELEMENT**> node, describing the elements generated by Z cuts (cuts in the horizontal direction, covering the length of the <**ELEMENT**> item to which it belongs). The order of <**ELEMENTZ**> type sections follows the order in which they are placed on the <**ELEMENT**> item, in accordance with the direction imposed by the vertex used for placements (in the example: from bottom to top).

<ELEMENTW> NODE

Nodes inserted inside an <**ELEMENTZ**> node, describing the elements generated by W cuts (cuts in the vertical direction, covering the height of <**ELEMENTZ**> item to which it belongs). The order of <**ELEMENTW**> type sections follows the order in which they are placed on <**ELEMENTZ**> item, in accordance with the direction imposed by the vertex used for placements (in the example: from left to right*)*.
In this example, no such nodes are assigned.

<ELEMENT5> NODE

Nodes inserted inside an <**ELEMENTW**> node, describing the elements generated by cuts #5 (cuts in the horizontal direction, covering the length of <**ELEMENTW**> item to which it belongs). The order of <**ELEMENT5**> type sections follows the order in which they are placed on <**ELEMENTW**> item, in accordance with the direction imposed by the vertex used for placements (in the example: from bottom to top*)*.
In this example, no such nodes are assigned.

<ELEMENT6> NODE

Nodes inserted inside an <**ELEMENT5**> node, describing the elements generated by cuts #6 (cuts in the vertical direction, covering the height of <**ELEMENT5**> item to which it belongs). The order of <**ELEMENT6**> type sections follows the order in which they are placed on the element <**ELEMENT6**>, in accordance with the direction imposed by the vertex used for placements (in the example: from left to right*)*.
In this example, no such nodes are assigned.

<LABEL> NODE

A node of this type can be inserted inside each node of type (PANEL, STRIPE, ELEMENT, ELEMENTZ, ELEMENTW, ELEMENT5, ELEMENT6), by definition of the rectangle that is placed in the corresponding element. A <**LABEL**> node terminates a cutting branch.

```
<PANEL ID="01" REP="1" L="800.00" H="1000.000">
    < STRIPE ID="0101" REP="1" L="800.000" H="500.000">
     <ELEMENT ID="010101" REP="1" L="400.000" H="500.000">
      <LABEL Code="2" Rep="1" Rotated="0" />
     </ELEMENT>
     …
    </STRIPE>
    < STRIPE ID="0102" REP="1" L="800.000" H="480.000">
     …
    </STRIPE>
  </PANEL>
```

The node assigns the following attributes:

- *Code* attribute: rectangle identifier (ID field in OneRect structure)
- *REP* attribute: element repetition (default= 1)
- *Rotated* attribute: indicates if the rectangle is placed rotated.

3.10. PROJECT SERIALIZATION FUNCTIONS

Functions handle project serialization to/from files.

3.10.1. SAVEPROJECT

The function saves the list assignments of *rectangles* and *sheets* to files (XML format).

CutError SaveProject (string pathName, bool bMode)

Arguments

- *pathName*: file path

- *bMode*: *true* requires saving general project settings

Return value
CutError.ErrorNone if result is positive.

Notes
A return other than *CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorIOFile*) an error occurred in accessing the file to be written.

Use *bMode=true* to save general project settings as well.

3.10.2. LOADPROJECT

The function reads the list assignments of *rectangles* and *sheets* from files (XML format).

CutError LoadProject (string pathName, bool bMode)

Arguments
- *pathName*: file path
- *bMode*: *true* enables reading general project settings

Return value
CutError.ErrorNone if result is positive.

Notes
A return other than *CutError.ErrorNone* corresponds to one of the error situations:
- (*CutError.ErrorIOFile*) Error in accessing the file.

Use *bMode=true* to also read general project settings: settings that are not read by the file are initialized to default values.
If *bMode=false*: the settings remain unchanged.

Executing the function does not change the general optimization settings.

3.10.3. IMPORTPROJECT

The function imports the assignments of *sheets and rectangles* lists from CSV file

CutError ImportProject (string pathName, string stFormatSheet, string stFormatRect)

Arguments
- *pathName*: file path
- *stFormatSheet*: formatting string of sheet assignment line (in CSV file)
- *stFormatRect*: formatting string of rectangle assignment line (in CSV file)

Return value
CutError.ErrorNone if result is positive.

Notes
A return other than *CutError.ErrorNone* corresponds to one of these error situations:
- (*CutError.ErrorIOFile*) Error in accessing the file
- (*CutError.ErrorFileNotValid*) data read is inconsistent

The function initializes the current project by assigning lists of *sheets and rectangles* from CSV file and leaving the remaining general project settings unchanged.

A CSV file is a text file used for assigning a data table. Each row of the file matches a table row and is in turn subdivided into fields (individual columns) by means of a separator character (comma and semicolon are recognized as valid).

The function assigns two strings to format the assignment lines of each item type, which are used if the same CSV file does not assign any more.

Let us see an example of CSV file:

***panel;d;l;h;n;m;g;p ***part;d;l;h;n;x;m;g;p;r	element formatting lines • optional • not necessarily both present • recognized heading for lines is highlighted: ○ "***panel;" for sheets ○ "***part;" for rectangles
panel;MDF;2000;1000;1; panel;MDF2;2000;1500;1;	sheet assignment lines - recognized heading for lines is highlighted: "panel;" - subsequent fields are interpreted according to the formatting assigned here in the first line of the file
part;A1;1000;300;30;0;;;1 part;A2;750;250;30;0;;;1 part;A3;300;250;30;0;;;1 part;A4;150;130;50;10;;;1 part;A5;180;150;0;20;;;0	rectangle assignment lines - recognized heading for lines is highlighted: "part;" - subsequent fields are interpreted according to the formatting assigned here in the second line of the file

Formatting string is considered valid if it assigns the fields corresponding to: length, height, quantity. Each field is marked with a univocal string of up to two characters in length.
Let us see the full list:

- ***e*** enable line (interprets: 0/1, yes/no, on/off, true/false) – (default=1)
- ***d*** element description – (default="")
- ***l*** element length dimension
- ***h*** element height dimension
- ***s*** element thickness dimension
- ***n*** quantity of the element
- ***x*** extra quantity (only for rectangles)
- ***m*** material
- ***g*** grain (interprets: 1/2, x/y) – (default=0)
- ***r*** rotation possible (only for rectangles) (interprets: 0/1, yes/no, on/off, true/false) – (default=0)
- ***p*** priority
- ***et*** top edge of rectangle (maximum format: "thickness|name|cut flag". E.g. "3.0|abc456|0")
- ***eb*** bottom edge of rectangle (format: see previous field)
- ***er*** right edge of rectangle (format: see previous field)
- ***el*** left edge of rectangle (format: see previous field)
- ***a1*** Param1 field of rectangle
- ***a2;a3;a4;a5;a6;a7;a8:a9;a10;a11;a12;a13;a14;a15***
  Fields *(Param2,…,Param15)* of rectangle
- ***mt*** top margin of sheet
- ***mb*** bottom margin of sheet
- ***mr*** right margin of sheet
- ***ml*** left margin of sheet
- ***y*** scrap sheet information (interprets: 0/1, yes/no, on/off, true/false).

In the example above:

- the formatting string of sheets assigns "d;l;h;n;m;g;p":
  - "d": assigns the *Label* field in OneSheet structure
  - "l": assigns the *Length* field in structure
  - "h": assigns the *Height* field in structure
  - "n": assigns the *N* field in structure
  - "m": assigns the *Material* field in structure
  - "g": assigns the *Grain* field in structure
  - "P": assigns the *Priority* field in structure

- the formatting string of rectangles assigns "d;l;h;n;x;m;g;p;r":
  - "d": assigns the *Label* field in OneRect structure
  - "l": assigns the *Length* field in structure
  - "h": assigns the *Height* field in structure
  - "n": assigns the *N* field in structure
  - "x": assigns the *Extra* field in structure
  - "m": assigns the *Material* field in structure
  - "g": assigns the *Grain* field in structure
  - "p": assigns the *Priority* field in structure

- "r": assigns the *Rotate* field in structure.

4. GUIDE TO USING THE LIBRARY

The purpose of this chapter is to provide you with the necessary information for integrating the TPA_C library into your application.

4.1. TYPICAL FLOW CHART

The following sequence describes a basic use of the library:

1. Create an instance of the *TpaCutOEM.TpaRctCut()* class
2. Carry out preliminary checks of general operation (key presence check)
3. Assignment of general operation settings (see: *NumberCustom*, *NumberRectParam*, …)
4. Assignment of general project settings (see: *Unit*, *Direction*, …)
5. Assignment of *rectangles* with *AddRct()* function calls
6. Assignment of *sheets* with *AddSheet()* function calls
7. Start optimization procedure with *Compute()* function call
8. Acquisition of results with call to functions: NumberOfSolutions, SelectSolution, …, …
9. Autonomous acquisition/processing of efficiency information about class optimization and assignments (EstimatedTime, EstimatedCost)
10. Saving optimization report (Export).

The order in which *settings*, *rectangles* and *sheets* are assigned is indifferent: the one proposed is only indicative.
The set of general settings, rectangles and sheets is what is referred to as the *cutting pattern*, as it identifies the set of data required to perform an optimization.

4.2. PRELIMINARY CHECKS

Preliminary check of the key presence can be useful and necessary to adapt your application's functionalities:

- Check the key presence and validity by querying *IsValidLicense*

Please note that the key check is performed both on time basis (i.e., every X seconds) and contextually (i.e., with call to specific functions): in order to ensure normal operation, it is therefore recommended not to remove the key during the execution of your application.

4.3. ASSIGNMENT OF GENERAL OPERATION SETTINGS

It is generally necessary to make a preliminary assignment of the general operation settings. This information does not change as the project changes, as this is usually attributable to options of general operation of an application.

4.4. ASSIGNMENT OF GENERAL PROJECT SETTINGS

It is necessary to make a preliminary assignment of general project settings, which may change for each project.
Preliminary assignment of settings can also be made by reading from a previously saved file: see *SaveProject()*, *LoadProject()* functions.

4.5. ASSIGNMENT OF RECTANGLES

We move on to assign the list of rectangles.
There are a few key points to note when assigning rectangles:

- Each rectangle has a univocal, strictly positive numerical identifier (> 0): *ID* field in *OneRect* structure
- It is therefore not possible to assign the same *ID* for several rectangles
- Identifiers can be assigned in any order, not necessarily in a row
- Note that IDs play an important role in sorting rectangles: as other notable settings (e.g. priority, area of rectangle) it is the ID that decides which rectangle has the highest placement priority in the optimization process.

To add a rectangle, call the *AddRct()* function.
To delete rectangles, call the *ClearRct()* function.

4.6. ASSIGNMENT OF SHEETS

We move on to assign the sheet list.
As with rectangles, there are a few key points to note when assigning sheets:

- Each sheet has a univocal, strictly positive numerical identifier (> 0): *ID* field in *OneSheet* structure
- Therefore, it is not possible to assign the same *ID* for multiple sheets
- Identifiers can be assigned in any order, not necessarily in a row
- Keep in mind that IDs play an important role in sorting sheets: as other notable settings (e.g., priority, scrap information, size) it is the ID that decides which sheet has the highest priority of use in the optimization process.

To add a sheet, call the *AddSheet()* function.
To delete the list of sheets, call the *ClearSheet()* function.

4.7. PERFORMANCE OF THE CUT OPTIMIZATION

To start a project optimization you need to call the Compute function.
Rectangle and sheet checks are executed prior to optimization, which can also cause the function to return with error and cancel the optimization:

- there must be rectangles requested for placements and sheets available
- checking the matching filters (thickness, material) must be able to match at least one rectangle to a sheet.

4.7.1. ACQUIRING THE RESULT OF THE SOLUTION

After optimization, you can acquire the results of the cutting solution.

Instead, a set of functions captures all the information related to the optimization solution:

- *NumberOfSolutions:* number of solutions calculated by the optimizer
- *FitnessSheet*: efficiency of the solution
- *SolutionSheets*: information about the number of cutting patterns of the solution
- *UsedRects*: information about the number of placements of the solution or a sheet and/or type of rectangle
- *ReadResolRect*: information about rectangles for a single sheet of the solution
- *ReadResolCut*: information on cuts for a single sheet of the solution
- *Export*: saves the solution to file (XML format)

4.7.1.1. EXAMPLE CODE: HOW TO ACQUIRE GENERAL INFORMATION OF THE SOLUTION SHEETS

```
TpaCutOEM.TpaRctCut optimizeObj=new TpaCutOEM.TpaRctCut ();

…
//Assignment fields on component query
OneRect OneRect = new OneRect();

//query cycle on solution sheets
int IndexSheet = 0; //sheet index
for (IndexSheet = 0; IndexSheet < optimizeObj.SolutionSheets; IndexSheet++)
{
    //index sheet ID (IndexSheet)
    int IdSheet =  optimizeObj.GetSheetID(IndexSheet);
    //sheet repetition
    int  sheetRepetition  =  optimizeObj.NumberOfRepetitions(IndexSheet);
    // …
    //acquisition cycle of sheet placements
    //……
}
```

4.7.1.2. EXAMPLE CODE: ACQUIRING CYCLE OF SHEET PLACEMENTS

```
// IndexSheet = sheet index (see: previous cycle)
int IndexRct = 0; //placement index on sheet
OneRect oneRect=new OneRect();

for (IndexRct = 0; IndexRct < UsedRects(IndexSheet, 0); IndexRct ++)
{
      double qx = 0.0, qy = 0.0;
      bool rotate = false;
      //reads the position and rotation of the inserted workpiece
      int ID = optimizeObj.ReadResolRect(IndexSheet, IndexRct, ref qx, ref qy, ref rotate);
      //reads the dimensions and features of the inserted piece
      OptimizeObj.ReadRect(ID, ref OneRect);
      //…
}
```

4.7.1.3. EXAMPLE CODE: ACQUIRING CYCLE OF SHEET CUTS

```
// IndexSheet = sheet index (see: previous cycle)
for (idxP = 0; idxP < optimizeObj.NumberOfCuts(IndexSheet); idxP++)
{
      double xStart = 0, yStart = 0, xEnd = 0, yEnd = 0, thickness = 0;
      int orientation = 0;
      int levelCut = optimizeObj.ReadResolCut(i, idxP, ref xStart, ref yStart, ref xEnd, ref yEnd,
            ref orientation, ref thickness);
      //…
}
```

4.7.2. MANAGING MULTIPLE SOLUTIONS

It is possible that the optimizer finds more solutions, up to a maximum of 5.
When optimization is performed, the first solution among the calculated ones is automatically set as the *current solution*.
All calculated solutions remain available, with the ability to navigate between them by changing the current solution to an appropriate specification.
The current solution can be saved to a file by calling the Export function.

4.7.2.1. EXAMPLE CODE: NAVIGATION BETWEEN MULTIPLE SOLUTIONS

```
TpaCutOEM.TpaRctCut optimizeObj=new TpaCutOEM.TpaRctCut();

// GoToNextSolution: Move to the next calculated solution
Bool GoToNextSolution()
{
    If (optimizeObj.SelectSolution < optimizeObj.NumberOfSolution)
    {
        optimizeObj.SelectSolution = optimizeObj.SelectSolution+1;
        return true;
    }
    return false;
}

// GoToPrevSolution: Moves to the previous calculated solution
bool GoToPrevSolution()
{
    If (optimizeObj.SelectSolution >0)
    {
        optimizeObj.SelectSolution = optimizeObj.SelectSolution-1;
        return true;
    }
    return false;
}
```

4.8. SAVE AND READ A TPA_C PROJECT

You can save and retrieve a project using the SaveProject e LoadProject functions.
The two functions also allow saving and retrieving the general settings of a project.
After executing *LoadProject*, an external application needs to update project settings, rectangles and sheets, reading information from TPA_C.

4.8.1. EXAMPLE CODE

```
TpaCutOEM.TpaRctCut optimizeObj=new TpaCutOEM.TpaRctCut();

//Structures
OneRect ItemRect=new OneRect();
OneSheet ItemSheet=new OneSheet();

// Reads the project
if (optimizeObj.LoadProject ("C:\projects\one.xml", true) == CutError.ErrorNone)
{
    //---------- Project settings
    myUnit= optimizeObj.Unit;
    myDirection= optimizeObj.Direction;
    myCorner= optimizeObj.Corner;
    myFiTool= optimizeObj.CutterDiameter;
    myTensionGap= optimizeObj.TensionGap;
    myMaxCutLevels= optimizeObj.MaxCutLevels;
    myPreCut= optimizeObj.PreCut;
    myLongCut= optimizeObj.LongCut;
    myTransvCut= optimizeObj.TransvCut;
    myZCut= optimizeObj.ZCut;
    myInfoCst[ni = 0]= optimizeObj.Custom1;
    myInfoCst[++ni]= optimizeObj.Custom2;
    ….
    myInfoCst[++ni]= optimizeObj.Custom10;
```

```
    //---------- Reads rectangles
    for (int idxElement=0; idxElement< optimizeObj.CountRect; idxElement++)
    {
        optimizeObj.ReadRectIndex (idxElement, ref ItemRect);
        // …
        // Reading and processing data
       //…
    }

    //---------- Reads the sheets
    for (int idxElement=0; idxElement< optimizeObj.CountSheet; idxElement++)
    {
        optimizeObj.ReadSheetIndex (idxElement, ref ItemSheet);
        // …
        // Reading and processing data
       //…
    }
}
```