COMPUTING OBJECTS

CM2 SurfMesh® T3/Q4 for OpenCascade®

Version 5.6

reference manual

Revision February 2025.

https://www.computing-objects.com

Forewords

This manual is a reference manual for the classes `data_type` and `settings_type` of the surface mesh generators for OpenCascade® Technology (OCCT) **CM2 SurfMesh® T3** and **CM2 SurfMesh® Q4** of the **CM2 MeshTools®** SDK.

It is based on the HTML reference manual (Doxygen© generated).

For examples, please refer to the manual **CM2 SurfMesh T3/Q4 - tutorials**.

Table of contents

Table of contents

1. Data of the meshers

All input and output data for a run is gathered into a structure of class `cm2::surfmesh_t3::mesher::data_type` or `cm2::surfmesh_q4::mesher::data_type`:

```
void surfmesh_t3::mesher::run (surfmesh_t3::mesher::data_type& data);
void surfmesh_q4::mesher::run (surfmesh_q4::mesher::data_type& data);
```

The `data_type` class contains the following fields:

Mesh sizes on vertices (user input)

Vector `vertex_H` (IN) of the the mesh sizes (isotropic) at CAD vertices.

The value `vertex_H[i]` specifies the target mesh size near vertex with CAD id `i`.

Only strictly positive values are considered, with a minimum value of `settings_type::min_h`.
Vertices with CAD index greater than or equal to `vertex_H.size()` are assigned a default mesh size derived from the lower precedence mesh sizes (`curve_H`, `surface_H` or `settings_type::target_h`).
`vertex_H` has the highest precedence over all user-defined mesh sizes. The precedence order is:

- `vertex_H` (highest precedence)
- `curve_H`
- `surface_H`
- `settings_type::target_h` (lowest precedence)

☞ The vertex, curve and surface ids must follow the CAD indexing-scheme.
With OCCT, we use the CAD indexing scheme given by `TopTools_IndexedMapOfShape`.
Example:

```
TopTools_IndexedMapOfShape       vertices;
TopExp::MapShapes(root, TopAbs_VERTEX, vertices);  // Index all vertices of root
                                                   // consecutively starting from 1.
unsigned i = vertices.FindIndex(some_vertex);      // `i` is the index we use to refer
                                                   // to `some_vertex`.
```

The index of an edge or face is also its *color*.

☞ As OCCT follows a one-base indexing scheme, `vertex_H[0]` is discarded.

☞ The mesher may use reduced mesh sizes in all directions (isotropic) or in selected directions (anisotropic) to honor the `settings_type::max_chordal_error` criterion.

Mesh sizes on curves (user input)

Vector `curve_H` (IN) of the the mesh sizes (isotropic) on CAD curves.

The value `curve_H[i]` specifies the target mesh size on curve with CAD id `i`.

Only strictly positive values are considered, with a minimum value of `settings_type::min_h`.
Curves with CAD index greater than or equal to `curve_H.size()` are assigned a default mesh size derived from the lower precedence mesh sizes (`surface_H` or `settings_type::target_h`).

☞ As OCCT follows a one-base indexing scheme, `curve_H[0]` is discarded.

☞ The mesh size on a curve is also transferred to its (two) extremity vertices, unless valid mesh sizes are provided for them in `vertex_H`.

☞ The mesher may use reduced mesh sizes in all directions (isotropic) or in selected directions (anisotropic) to honor the `settings_type::max_chordal_error` criterion.

Mesh sizes on surfaces (user input)

Vector `surface_H` (IN) of the the mesh sizes (isotropic) on CAD surfaces.

The value `surface_H[i]` specifies the target mesh size on surface with CAD id `i`.

Only strictly positive values are considered, with a minimum value of `settings_type::min_h`.
Surfaces with CAD index greater than or equal to `surface_H.size()` are assigned the ultimate default mesh size `settings_type::target_h`.

☞ As OCCT follows a one-base indexing scheme, `surface_H[0]` is discarded.

☞ The mesh size on a surface is also transferred to all its boundary curves and vertices, unless valid mesh sizes are provided for them in `curve_H` and `vertex_H`.

☞ The mesher may use reduced mesh sizes in all directions (isotropic) or in selected directions (anisotropic) to honor the `settings_type::max_chordal_error` criterion.

Coordinates of the nodes

Matrix `pos` of the the node coordinates with dimensions 3 x `nods` (OUT).

This matrix stores the coordinates of all nodes. The coordinates are stored column-wise. The column index is the index of the node (zero-based, i.e. from 0 to *N*-1). The *X*-coordinates are in the first row, the *Y*-coordinates in the second row and the *Z*-coordinates in the third row.

☞ As OCCT follows a one-base indexing scheme, there is a left one-shift in the indices.
The coordinates of CAD vertex with id `i` is at column `i-1` in this `pos` matrix.

Edges of curves

Matrix `connectE` (OUT) of the connectivity of the edges along the curves of the model.

This matrix has 2 rows when `settings_type::high_order_type` = 0 (2-node edges), 3 rows otherwise (3-node edges).

The values are node indices and refer to columns in the coordinate matrix `pos`.

Elements

Matrix `connectM` (OUT) of the connectivity of the elements.

For **CM2 SurfMesh T3**, the dimensions are 3 x `nefs` when `settings_type::high_order_type` = 0 (3-node triangles) and 6 x `nefs` otherwise (6-node triangles).

For **CM2 SurfMesh Q4**, the dimensions are 4 x `nefs` when `settings_type::high_order_type` = 0 (4-node quads and 3-node triangles), 8 x `nefs` when `settings_type::high_order_type` = 1 (8-node quads and 6-node triangles) and 9 x `nefs` when `settings_type::high_order_type` = 2 (9-node quads and 6-node triangles).
The leading part of this matrix (from col 0 to col `nefs_Q` - 1) is the connectivity of the quadrangle elements.

The tailing part of this matrix (from col `nefs_Q` to col `nefs` - 1) is the connectivity of the triangles elements, completed with `CM2_NONE` values (i.e. `unsigned(-1)`).

The values are node indices and refer to columns in the coordinate matrix `pos`.

Colors of edges

Vector `colorsE` (OUT) of the colors (same as curve ids) of the edges.

Each edge of `connectE` is assigned a color (from 1 to `nbr_curves`) corresponding to the CAD ids of its curve.
With OCCT, we use the CAD indexing scheme given by `TopTools_IndexedMapOfShape`.

Colors of elements

Vector `colors` (OUT) of the colors (same as surface ids) of the elements.

Each element of `connectM` is assigned a color (from 1 to `nbr_surfaces`) corresponding to the CAD ids of its surface.
With OCCT, we use the CAD indexing scheme given by `TopTools_IndexedMapOfShape`.

Merged vertices

Vector `merged_vertices` (OUT).

The CAD ids of the merged vertices (eliminated due to `settings_type::fix_tolerance`) and their target vertices.
`merged_vertices[i]` is the target vertex CAD id in which `merged_vertices[i+1]` is merged.
`merged_vertices.size()/2` is the number of the merged (eliminated) vertices.

Merged curves

Vector `merged_curves` (OUT).

The CAD ids of the merged curves (eliminated due to `settings_type::fix_tolerance`) and of the target curves.
`merged_curves[i]` is the target curve CAD id in which `merged_curves[i+1]` is merged.
`merged_curves.size()/2` is the number of the merged (eliminated) curves.

Failed surfaces

Vector `failed` of the CAD ids of the failed surfaces, if any (OUT).

Histogram

Histogram `histo_Qs` of the shape qualities (OUT).

This histogram stores also the minimum, the maximum and the average values as data members.

Errors and warnings

Two enums, `error_code` and `warning_code`, give upon exit the error and warning codes (OUT).

A correct run gives `CM2_NO_ERROR` and `CM2_NO_WARNING` (zero values).
String `msg1` holds explanation messages about the error/warning if any.
String `msg2` is for debugging purpose and shouldn't normally be used on production code.

See Section 2 for more detailed explanations of the error and warning codes.

Complementary information

The remaining fields of the data structure (OUT):

- `nbr_vertices:` the number of vertices in the model, including the merged vertices.
- `nbr_curves:` the number of curves in the model, including the merged curves.
- `nbr_surfaces:` the number of surfaces in the model.
- `nefs`: the number of elements in the mesh (also equals to the number of columns in `connectM` upon exit). `nefs = nefs_Q + nefs_T`.
- `nefs_Q`: the number of quadrangles in the mesh (zero with **CM2 SurfMesh T3**).
- `nefs_T`: the number of triangles in the mesh (normally zero with **CM2 SurfMesh Q4** in all-quad mode).
- `nods`: the number of nodes in the mesh.
- `area`: the area of the final mesh. `area = area_Q + area_T`
- `area_Q`: the area of the quadrangles in the final mesh (zero with **CM2 SurfMesh T3**).
- `area_T`: the area of the triangles in the final mesh (normally zero with **CM2 SurfMesh Q4** in all-quad mode).
- `Qmin`: the worst shape quality of the elements in the mesh (also stored in `histo_Qs`).
- `error_code`, `warning_code`: the error and warning codes (see Section 2).
- `read_time`, `meshC_time`, `meshS_time`, `mesh_time`: the times spent in the successive steps.
- `total_time`: the total time spent in the mesher (including read time).
- `speed`: the global meshing speed (number of generated elements per second) excluding the read time.

```
struct data_type
{
   DoubleVec               vertex_H;
   DoubleVec               curve_H;
   DoubleVec               surface_H;
   DoubleMat               pos;
   UIntMat                 connectE;
   UIntMat                 connectM;
   UIntVec                 colorsE;
   UIntVec                 colors;

   UIntVec                 merged_vertices;
   UIntVec                 merged_curves;
   UIntVec                 failed;

   unsigned                nbr_vertices;
   unsigned                nbr_curves;
   unsigned                nbr_surfaces;
   unsigned                nefs;
   unsigned                nefs_Q;
   unsigned                nefs_T;
   unsigned                nods;

   double                  area;
   double                  area_Q;
   double                  area_T;
   double                  Qmin;

   misc::histogram         histo_Qs;

   double                  read_time;
   double                  meshC_time;
   double                  meshS_time;
   double                  mesh_time;
   double                  total_time;
   double                  speed;

   error_type              error_code;
   warning_type            warning_code;
   char                    msg1[256];
};
```

Table 1 – The `data_type` structures.

Function members

The `data_type` class is equipped with constructors, copy operator and some utility members:

```
// Constructors.
data_type();
data_type(const data_type& cpy);

// Copy operator.
const data_type& operator= (const data_type& data);
void shallow_copy (const data_type& data);

// Members.
void clear();
void extract (DoubleMat& P, UIntMat& connectM) const;
void extract (DoubleMat& P, UIntMat& connectQ, UIntMat& connectT) const;
void save (const char* filename, unsigned precision = 16) const;
void print_info (display_handler_type hdl) const;
bool error_raised() const;
bool warning_raised() const;
```

Table 2 – The `data_type` function members.

2. Error and warning codes

Error codes

The error code is a public field of the class `data_type::error_code` as an enum of type `error_type`:.

```
enum error_type
{
   CM2_NO_ERROR,                    // 0
   CM2_LICENSE_ERROR,               // -100
   CM2_SETTINGS_ERROR,              // -101
   CM2_FILE_ERROR,                  // -102
   CM2_DEGENERATED_ELEMENT,         // -103
   CM2_SYSTEM_MEMORY_ERROR,         // -199
   CM2_INTERNAL_ERROR               // -200
};
```

Table 3 – Error codes enum.

`CM2_NO_ERROR`	OK, no problem.
`CM2_LICENSE_ERROR`	The registration must occur before any run of the mesher. Check also your license. You may have to renew it. If so, please, contact <license@computing-objects.com>.
`CM2_SETTINGS_ERROR`	At least one setting is not valid (see Section 3). Check the positivity/range of scalar values such as `target_h`, `max_gradation`...
`CM2_FILE_ERROR`	The input file couldn't be opened/read. Check filename, format...
`CM2_DEGENERATED_ELEMENT`	At least one of the final elements is invalid (null area). Or the all-quad condition couldn't be honored (remaining triangles with **CM2 SurfMesh Q4**). Contact support.
`CM2_SYSTEM_MEMORY_ERROR`	Insufficient memory available. Mesh is too big to be generated.
`CM2_INTERNAL_ERROR`	Unexpected error. Save the data by calling `data_type::save` and send the file zipped to <support@computing-objects.com>.

Table 4 – Error codes.

☞ For error codes `CM2_DEGENERATED_ELEMENT` and `CM2_INTERNAL_ERROR`, please save the data with a call to `data_type::save`, zip the file and send to <support@computing-objects.com>.

Example

```
if (data.error_raised())
{
  // Error, do something according to data.error_code (forward message to a window...)
}
```

Warning codes

The warning code is a public field of the class `data_type::warning_code` as an enum of type `warning_type`:.

```
enum warning_type
{
   CM2_NO_WARNING,                    //   0
   CM2_INTERRUPTION,                  // -10
   CM2_FAILED_SURFACES_WARNING,       // -11
   CM2_SHAPE_QUALITY_WARNING          // -12
};
```

Table 5 – Warning codes enum.

`CM2_NO_WARNING`	OK, no problem.
`CM2_INTERRUPTION`	The user has aborted the run (through the interrupt handler). The final mesh may be empty or with missing parts.
`CM2_FAILED_SURFACES_WARNING`	At least one surface couldn't be meshed. Check the model.
`CM2_SHAPE_QUALITY_WARNING`	The final mesh is valid but at least one of the elements is very bad (shape quality < 0.01). Check the model.

Table 6 – Warning codes.

Example

```
if (data.warning_raised())
{
  // Warning, do something according to data.warning_code
  // e.g. mesh again with lower chordal error...
}
```

3. Settings of the meshers

CM2 SurfMesh T3/Q4 can be controlled by several settings that can change drastically the output meshes. They are gathered into a structure of class `cm2::surfmesh_t3::mesher::settings_type` or `cm2::surfmesh_q4::mesher::settings_type` as a public field of the meshers:

```
cm2::surfmesh_t3::mesher::settings_type   settings;
cm2::surfmesh_q4::mesher::settings_type   settings;
```

Default target mesh size

`target_h`. Positive value. Default = 0. Should be reset by user.

This is the target size for the elements when not constrained by the user-defined `vertex_H`, `curve_H`, `surface_H` or by dimensions of the surfaces, their curvatures or the curvatures of their boundary curves.

☞ The default value (zero) is replaced with an arbitrary strictly positive value (at present computed as 1/20 of the maximum dimension of the model) and `fix_tolerance` is considered as zero (merges only exactly coincident vertices and curves).

Minimum size

The minimum mesh size (security). Default = -1E-3 (i.e. 0.1% of `taget_h`).

The mesh size will not be reduced (by `max_chordal_error` or `max_chordal_error_ratio` below) beyond this limit:

- If negative, this value is relative to `target_h` (min size = - `min_h` * `target_h`).
- If positive, this value is absolute (min size = `target_h`).

Fixing tolerance

`fix_tolerance`. Default = -1E-4 (i.e. 0.01% of `taget_h`).

CAD entities are merged when their maximum distance is below or equal to the merge tolerance:

- If negative, this value is relative to `target_h` (merge tolerance = - `fix_tolerance` * `target_h`).
- If positive, this value is absolute (merge tolerance = `fix_tolerance`).

Vertices within the merge tolerance are merged. Curves approximately within the merge tolerance are merged.

Set to `+-DBL_MAX` to completely disable the merging.

Max chordal error

`max_chordal_error`. Default = -0.02 (i.e. 2% of local radius).

The maximum chordal error allowed between exact curves/surfaces and *linear* elements.

The mesh size will be reduced locally to limit the chordal error between the mesh and the curves or the surfaces:

- If negative, this value is relative to the local radii (for instance -0.02 => max chordal error = 2% of local radii).
- If positive, this value is absolute (for instance 0.1 => max chordal error = 0.1).

A complete circle is meshed with approximately:

- 10 linear elements for `max_chordal_error` = -0.05 (5% of radius).
- 13 linear elements for `max_chordal_error` = -0.03 (3% of radius).
- 16 linear elements for `max_chordal_error` = -0.02 (2% of radius).
- 22 linear elements for `max_chordal_error` = -0.01 (1% of radius).

☞ The considered chordal error is between the curves/surfaces and the *linear* elements.

With linear elements of size h along a curve of radius R, the chordal error is approximately given by

$$\delta = \frac{h^2}{8\,R}$$

The chordal error is divided by 4 when the size h is divided by 2

In case of quadratic elements (`high_order_type` >= 1 and `high_order_mode` >= 1), the chordal error (or rather the maximum distance between the quadratic element and the curve) is well below:

$$\delta = \frac{h^4}{512\,R^3}$$

The chordal error is divided by 16 when the size h is divided by 2.

Max ratio between chordal errors

`max_chordal_error_ratio`. Default = 0.1

The size of the mesh along the smallest curvature direction (largest radius) is reduced (but not below `settings_type::min_h`) so that the chordal error remains below `max_chordal_error_ratio` * size of the mesh along highest curvature (smallest radius).

This parameter limits the anisotropy of the elements.
It is used only with anisotropic control (`chordal_control_type` = 2 or 4) and when the surface has double curvature.

Chordal control type

The type of chordal control. Default = 4 (anisotropic control with exact curvatures).

Possible values are:

- 0: No curvature is computed and the chordal error control is disabled.
- 1: Isotropic chordal control based on approximate curvatures (least-square method using the local bases). Element sizes are limited by the same (isotropic) chordal value.
- 2: Anisotropic chordal control based on approximate curvatures (least-square method using the local bases). Element sizes are limited by two different (anisotropic) chordal values along the two principal curvature directions.
- 3: Isotropic chordal control based on exact curvatures. Element sizes are limited by the same (isotropic) chordal value.
- 4+: Anisotropic chordal control based on exact curvature.
Element sizes are limited by two different (anisotropic) chordal values along the two principal curvature directions.

Maximum gradation of sizes

The maximum gradation of the elements size. Default = 0.5

This parameter controls the gradation of the elements size along the curves and inside the surfaces towards the `target_h` mesh size.

A value close to 0 leads to a more progressive variation of mesh size (smoother).

Flag for all-quad or quad-dominant mode (CM2 SurfMesh Q4)

`all_quad_flag`. Default = `false`.

Flag to force the generation of an all-quad mesh (no triangle):

- false: generates a mixed triangle-quad mesh (the default).
- true: generate an all-quad mesh.

A good all-quad mesh can only be generated in simple cases. If possible, rather use the mixed quad-dominant generation mode.

Setting `all_quad_flag` to true, disables parameters `min_Q4_angle_quality`, `min_Q4_warp_quality` (i.e. same as `min_Q4_angle_quality` = `min_Q4_warp_quality` = 0), `force_even_flag` (same as true) and `quadrangle_weight`.

Parity along the curves

`force_even_flag`. Default = `false`.

Flag to force the number of edges along each curve to be even.

This flag is considered as true with **CM2 SurfMesh Q4** when `all_quad_flag` is set to true.

Optimization level

`optim_level`. Integer between 0 and 10. Default = 3.

Controls the trade-off between speed (low values) and elements quality (high values). The higher the level, the more aggressive the algorithms.

Level 0 disables any optimization.

☞ Optimizations are only intra-surface (no optimization across curves).

Weight on quadrangles (CM2 SurfMesh Q4)

`quadrangle_weight`. Value between 0 and 1. Default = 0.70

Preference for quadrangles vs. triangles.

Weight between 0 and 1 indicating the preference for quadrangles over triangles.
With `quadrangle_weight` = 0, quadrangles are never used. With `quadrangle_weight` = 0.5, quadrangles are used only when they improve the quality of the mesh. For values between 0.5 and 1, quadrangles are more and more used even if this lead to a lesser quality of the mesh. With `quadrangle_weight` = 1, the minimum number of triangles are used (but may not be null).

This parameter is used only when `all_quad_flag` = `false`.

☞ This parameter is not the ratio between quads and triangles. And there is no linearity between this weight and the ratio between quads and triangles.

Minimum acceptable quadrangle quality (CM2 SurfMesh Q4)

`min_Q4_angle_quality`. Value between 0 and 1. Default = 0.20.

Minimum acceptable angle quality for the quadrangles. This threshold is based on the angle quality of the quads (not the geometric quality which takes the length ratios also into account). The angle quality is computed as the minimum of the four angles at summits.

With the default value (0.20), acceptable quads have angle between 0.20 * 90 = 18 deg and 180 - 0.20 * 90 = 162 deg. Others are split into triangles.

Set `min_Q4_angle_quality` = 1 - ε to allow rectangles only (quads with right angles only). In this case, be aware that when curves are not straight very few rectangles may be generated.

This parameter is used only when `all_quad_flag` = `false`.

Minimum acceptable warp quality (CM2 SurfMesh Q4)

`min_Q4_warp_quality`. Value between 0 and 1. Default = 0.50 (i.e. maximum warpage of 90°).

Minimum acceptable warp quality for the quadrangles. Quads with warp quality below this threshold are split into two triangles.

If α is the maximum angle between the normals of two opposite split triangles. The warp quality of a quadrangle is defined as $1 - \alpha/\pi$.
The warp quality of a flat quad equals to 1 (perfect) whereas its *geometric* quality equals to 1 only when the quad is a square.

With the default value (0.50), acceptable quads have warpage below 90°. Others are split into triangles.

Set `min_Q4_warp_quality` = 1 - ε to allow for flat quads only. In this case, be aware that very few rectangles may be generated on non flat surfaces (mostly triangles).

This parameter is used only when `all_quad_flag` = `false`.

High-order type

`high_order_type`. Integer ≥ 0. Default = 0 (i.e. linear elements T3 and Q4).

To convert into high-order elements. Possible value are:

- 0: Keep linear elements (`CM2_FACET3`).
- 1+: Convert to quadratic elements (`CM2_FACET6`).

High-order nodes are located onto the curves and surfaces (i.e. elements are curved) when `settings_type::high_order_mode` > 0.

High-order mode

`high_order_mode`. Default = 2 (high-order nodes on curves/surfaces at middle between linear nodes).

The method for positioning of the high-order nodes (*HON*). Possible values are:

- 0: The *HON* are interpolated between the linear 3D nodes. Elements remain straight. This method is the fastest.
- 1: The *HON* are on the curves/surfaces and have reference coordinates in the middle between the linear nodes but may not be equidistant between the linear 3D nodes. Elements are curved. The speed of this method is between methods 0 and 2.

- 2+: The *HON* are on the curves/surfaces and equidistant between the linear 3D nodes but may not have reference values in the middle between the linear nodes. Elements are curved. This method is the slowest.

Relevant only when `high_order_type` > 0.

Number of threads for parallel processing

`num_threads`. Integer ≥ 0. Default = 0 (= the number of logical processors on the running CPU).

If `num_threads` = 1 the mesh generator runs on a single thread. Value is capped by the number of logical processors on the running CPU.

Display handler

`display_hdl`. Default = `NULL`.

The user-supplied function to handle the messages issued by the mesher.

```
typedef void (*display_handler_type) (void* pass_thru, unsigned level, const char* msg);
```

The `pass_thru` parameter is the pointer set by the user in `settings_type::pass_thru` (see below).
The `level` parameter gives the importance of the message:

- 0: Important (for instance entering a major step of the process)
- 1: Somewhat important (minor step of the process)
- 2+: Not serious (debug messages that should not be printed for end-users).

The `msg` parameter is the string message (length ≤ 255 characters).

☞ This handler is not called in case of error or warning. At the end of the run, user must check for an error or a warning in the fields `data_type::error_code` and `data_type::warning_code` and then (in case of error or warning) process the string `data_type::msg1`.

Example

```
void my_display_handler (void* pass_thru, unsigned level, const char* msg)
{
   window_type*   my_window = static_cast<window_type*>(pass_thru);
   my_window->show(msg);
}

cm2::surfmesh_t3::mesher              my_mesher;
cm2::surfmesh_t3::mesher::data_type   my_data(pos, connectB);
window_type                           my_window;   // A “window” instance.

my_window.init(...);          // Initialize the window somehow.
my_mesher.settings.display_hdl = &my_display_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_window);
my_mesher.run(my_data);       // Will call my_display_handler with “my_window”
                              // in pass_thru parameter.
```

Interrupt handler

`interrupt_hdl`. Default = `NULL`.

The user-provided interruption function. When this function returns true, the tool aborts.

```
typedef bool (*interrupt_handler_type) (void* pass_thru, double progress);
```

An interruption also raises the `CM2_INTERRUPTION` warning.

The `pass_thru` parameter is the `void*` pointer set by the user in `settings_type::pass_thru` (the same parameter is also passed to the display handler).
The parameter `progress` (between 0 and 1) gives a hint about the progress of the meshing process.

Example (time-out)

```
bool my_interrupt_handler (void* pass_thru, double progress)
{
   clock_t*   t_timeout = static_cast<clock_t*>(pass_thru);
   return clock() > (*t_timeout);
}

cm2::surfmesh_t3::mesher            my_mesher;
cm2::surfmesh_t3::mesher::data_type my_data(pos, connectB);
clock_t                             my_t_timeout(::clock() + 1E3 * CLOCKS_PER_SEC);

my_mesher.settings.interrupt_hdl = &my_interrupt_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_t_timeout);
my_mesher.run(my_data);         // Will stop if duration > 1000 s.
```

Example (progress bar)

```
bool my_interrupt_handler (void* pass_thru, double progress)
{
   window_type*         my_window = static_cast<window_type*>(pass_thru);
   std::string          msg("Progress: ");

   msg += std::to_string(static_cast<int>(100.*progress)) + " %";
   my_window->show(msg);
   return false;
}

cm2::surfmesh_t3::mesher             my_mesher;
cm2::surfmesh_t3::mesher::data_type  my_data(pos, connectB);
window_type                          my_window;   // A window instance.

my_window.init(...);          // Initialize the window somehow.
my_mesher.settings.interrupt_hdl = &my_interrupt_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_window);
my_mesher.run(my_data);       // Will call my_interrupt_handler with "my_window"
                              // in pass_thru parameter.
```

Pass-through pointer

`pass_thru`. Default = `NULL`.

Void pointer (to be cast) to pass some user structure/class to the display handler and to the interruption handler.

```
struct settings_type
{
   double                  target_h;
   double                  min_h;
   double                  fix_tolerance;
   double                  max_chordal_error;
   double                  max_chordal_error_ratio;
   unsigned                chordal_control_type;
   double                  max_gradation;
   bool                    force_even_flag;
   unsigned                optim_level;
   unsigned                high_order_type;
   unsigned                high_order_mode;
   unsigned                num_threads;
   display_handler_type    display_hdl;
   interrupt_handler_type  interrupt_hdl;
   void*                   pass_thru;
};
```

Table 7 – The `settings_type` structure (**CM2 SurfMesh T3**).

```
struct settings_type
{
   double                  target_h;
   double                  min_h;
   double                  fix_tolerance;
   double                  max_chordal_error;
   double                  max_chordal_error_ratio;
   unsigned                chordal_control_type;
   double                  max_gradation;
   bool                    all_quad_flag;
   bool                    force_even_flag;
   unsigned                optim_level;
   unsigned                high_order_type;
   unsigned                high_order_mode;
   double                  quadrangle_weight;
   double                  min_Q4_angle_quality;
   Double                  min_Q4_warp_quality;
   unsigned                num_threads;
   display_handler_type    display_hdl;
   interrupt_handler_type  interrupt_hdl;
   void*                   pass_thru;
};
```

Table 8 – The `settings_type` structure (**CM2 SurfMesh Q4**).

Most useful fields are `target_h`, `min_h`, `fix_tolerance, max_chordal_error` (and `all_quad_flag` for **CM2 SurfMesh Q4**) and possibly the handlers.

Function members

The `settings_type` class is equipped with default constructor and some utility members:

```
// Constructor.
settings_type();

// Members.
bool valid() const;
int check() const;
void reset();
void save (const char* filename) const;
int load (const char* filename);
```

Table 9 – The `settings_type` function members.

COMPUTING
OBJECTS

https://www.computing-objects.com