Morph Mesher User Manual

SAND2026-22297O

Unclassified Unlimited Release

June 10, 2026

FAST Team
Sandia National Labs [1]

[1]Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

1 Introduction

Morph is an automatic 3D tetrahedral mesh generator for CAD geometry files. The primary goal of Morph is to build a conformal, analysis-ready mesh without requiring the CAD to be repaired, cleaned up, defeatured, imprinted, or merged prior to meshing. Unlike traditional tetrahedral meshers, Morph captures only the geometric features that are resolvable at the user-specified mesh size (based on a user tolerance) and ignores the rest. In addition, Morph washes over geometry imperfections such as volumetric overlap, non-watertight volumes, etc, creating conformal tetrahedral meshes without requiring the geometry to first be cleaned, repaired, defeatured, imprinted and merged.

Traditional tetrahedral meshers capture every geometry feature in the CAD, regardless of its size, by refining the mesh to a small enough resolution to capture every feature. This results in excessive element counts to capture features that are not important, resulting in unneeded analysis degrees of freedom. In addition, traditional meshers build overlapping meshes if there are overlapping geometric volumes, or fail if the geometric representation is not perfect (for example, non-watertight volumes). To avoid these issues, traditional meshers require the user to first modify the CAD to contain only the CAD features that the user wants to capture, remove all overlaps and unwanted gaps, to remove any errors in the geometry definition, and imprint and merge adjacent volumes. These required geometry operations are, at best, time consuming, repetitive and tedious, and often impossible due to failures of the geometry engine.

In contrast, Morph was designed to be tolerant to these geometry CAD errors, automatically ignore features that are smaller than a user specified tolerance without any modification to the CAD, and build conformal tetrahedral meshes even if adjacent volumes overlap or have small gaps between them.

2 Morph - An Overlay Grid Algorithm

The key to Morph's ability to be geometry tolerant is that Morph uses an overlay grid algorithm, instead of the bottoms-up algorithm that is used by traditional tetrahedral mesh generators.

2.1 Traditional Tetrahedral meshers

Morph uses an overlay grid algorithm as opposed to the bottoms-up algorithms used by traditional tetrahedral meshers. Bottoms-up algorithms build the mesh starting at the bottom of the CAD topology tree (i.e. vertices) working their way up. Bottoms-up algorithms first place a unique node on each vertex (regardless of how close it is to any other vertex). Bottoms-up algorithms then create a unique string of edges on each CAD curve, bound by the nodes on the CAD vertices at either end (regardless of how close it is to any other curve). Bottoms-up algorithms then create a patch of triangles for each CAD surface, bound by the edges on the surrounding CAD curves (regardless of how close it is to any

other surface). Finally, bottoms-up algorithms place tetrahedra in each CAD volume, bound by the triangles on the surrounding CAD surfaces (regardless of any overlap with any other CAD volumes). In contrast to Morph, these bottoms-up algorithms result in every CAD feature being captured regardless of how small it is and non-conforming overlapping meshes where ever the geometry overlaps.

2.2 Morph Behind the Curtain

In contrast Morph is an overlay grid approach, where an overlay grid is constructed of very high quality and space-filling tetrahedral elements conforming to a slightly expanded bounding box of the geometry to mesh. This overlay grid initially ignores the geometry and conforms only to the expanded bounding box. Morph next computes "intersection points" wherever an overlay edge intersects a CAD surface, an overlay triangle intersects a CAD curve, or an overlay tet contains a CAD vertex. Morph then either snaps existing overlay nodes to these intersection points, or cuts new nodes into the overlay grid at intersection point locations. When each intersection point is processed, before cutting it into the overlay grid, a search is performed for nearby nodes in the overlay grid. If a nearby node is found that is within the user-specified geometry tolerance, and that node is already associated (i.e. cut or snapped) to a piece of geometry, then the current intersection point is simply ignored. This enables Morph to wash over small geometry features. The overlay grid nature of this algorithm also ensures that there is never any mesh overlap.

3 Running Morph

Morph is an MPI-parallel batch program, available wherever the Sierra CompSim software suite is available. Sierra is developed and maintained at Sandia National Labs. At Sandia, Morph is available by the user loading one of the Sierra 'modules'. To run the latest official released version of Morph, use:

```
module load sierra
```

To run the latest development sprint version of the day of Morph, use:

```
module load sierra/sprint
```

To run the latest daily development version of the day of Morph, use:

```
module load sierra/daily
```

Loading a Sierra module puts the executable morph_mesher into your PATH environment variable. Morph can then be run with:

```
morph_mesher --help
```

to see the various Morph options.

On the Sandia SRN, both modules sierra and sierra/daily are available. For access on other Sandia networks, contact the Sandia Fast team for support (cubit-help@sandia.gov or sierra-help@sandia.gov)

4 Morph with Cubit

As of June 2026, Morph is not yet distributed with Cubit. However, development is underway to integrate Morph into Cubit as a meshing scheme. In the meantime, Cubit is still a powerful pre and post-processing tool for Morph. To assist in preparing the Morph input file, Cubit is recommended to identify the values for required Morph input, such as target element size, surfaces for sidesets, volumes for blocks, etc. Once integrated into Cubit as a meshing scheme, you'll be able to launch Morph directly from Cubit.

Integrating Morph into Cubit will require significant changes to traditional Cubit workflows that users are accustom to. Cubit has historically only supported traditional tetrahedral mesh generation with the limitations listed in the introduction section of requiring the user to first repair, cleanup, defeature, imprint and merge the geometry. This has enabled Cubit to make assumptions on how the meshes are associated to the geometry (i.e. each geometric vertex has a unique mesh node, each geometric curve has a unique set of mesh edges, only imprinted and merged geometry will result in conformal meshes, etc.). Morph breaks these assumptions. For example, Morph will only build a single node to represent multiple geometric vertices if those vertices are within a user specified tolerance of each other. Further, by default, Morph builds conformal meshes between geometric volumes if they are spatially within a user specified tolerance, even if they are not imprinted and merged. Modifications in Cubit are underway to remove these assumptions in preparation for integrating Morph as a Cubit meshing scheme.

5 Specifying Morph Options

Morph options can be specified either on the command line or in an input file. To see all command line options, issue the command:

```
morph_mesher --help
```

To see all input file options, issue the command:

```
morph_mesher --input_help
```

Only a limited number of options can be specified on the command line. All but a few rarely used options can be specified in the input file.

6 Input File Requirements

The Morph input file must be formatted using the standard YAML format.

7 Minimum Input to Run Morph

The minimum input to run Morph is the name of the geometry file Morph will read, a global element size, and the name of an output file that Morph will

create containing the resulting mesh. The command to run Morph from the command line with this minimum input is:

```
morph_mesher -g path/MyModel.sat
             -s <positive floating point number>
             -o path/MeshFileToWrite.exo
```

The minimal Morph input file would be a YAML text file with the following contents:

```
geometry_file: path/MyModel.sat
output_mesh_file: path/MeshFileToWrite.exo
global_elem_size: 10.2
```

You would then run Morph with this input file with the following command:

```
morph_mesher -i <MyMorphInputFile>
```

To run Morph in parallel, run the command below:

```
mpirun -np <NumProcs> morph_mesher ...options...
```

8 Geometry File

The geometry can be either an ACIS (*.sat) file, a STEP (*.stp, or *.step) file, an STL (*.stl) file, or an SGM (*.sgm) file. The full path to the geometry file must be specified. The geometry file is specified in the input file as follows:

```
geometry_file: path/MyModel.sat
```

9 Output File

The output file is written by Morph and contains the output mesh in exodus format. If a file with the specified name already exists, it will be overwritten. The file will be spread across the number of MPI procs that Morph was run on. The output file is specified in the input file as follows:

```
output_mesh_file: path/MyMeshedModel.exo
```

10 Global Mesh Size

The global element size is the approximate size of the edges in the resulting mesh. The global element size must be a floating point number greater than 0.0. The global element size is specified in the input file as follows:

```
global_elem_size: 10.2
```

11 Referencing Geometry

Any Morph input file option that references a CAD entity (i.e. volume, surface, curve or vertex) can refer to the CAD entity by name or by id. Names are strongly recommended since ids can change unexpectedly. Use of wildcards is supported for specifying CAD entities by name.

CAD volumes can be referenced by name in the Morph input file with any or all of the following options:

```
volume_names: ["Gasket1", "Gasket2", "Gasket3"] # 3 volumes
volume_names: ["Housing*"] # all names starting with Housing
volume_names: [Housing*] # all names starting with Housing
volume_names: [1Bolt,2Bolt] # most names do not need quotes
volume_names: ["*Bolt"] # leading wildcard needs quotes
```

CAD surfaces, curves, and vertices can be referenced by name in the Morph input file with the surface_names, curve_names, and vertex_names keywords using the same syntax as above for volumes.

CAD volumes can be referenced by id in the Morph input file with any or all of the following options:

```
volumes: [17,24,8]
volumes: [34]
volume_ranges: [[2,4], [120,300]]
```

CAD surfaces, curves, and vertices can be referenced by id in the Morph input file with the surfaces, surface_ranges, curves, curve_ranges, vertices, and vertex_names keywords using the same syntax as above for volumes.

11.1 Assign Names in Cubit

One example of a Morph pre-processing step that is often done in Cubit is to assign names to geometry entities. By assigning a name in Cubit, the geometry entity can be referenced in the Morph input file by name instead of by ID. IDs can change and are thus unreliable. Once a name is assigned to a geometry entity, that entity can be reliably referenced in a Morph input file. For example, if a user wants Morph to build a sideset on a set of surfaces, first use one of the following command in Cubit:

```
surface <ids> name "PressureSurf"    # Adds additional name
surface <ids> rename "PressureSurf"  # Replaces existing name
```

Then in the Morph input file, you can refer to these surfaces in the sideset section as follows:

```
surface_names: ["PressureSurf*"]
```

Note the wildcard was added at the end to capture all surfaces that were assigned names with the above Cubit commands.

12 Geometry Engines

By default, Morph will use the ACIS geometry engine when given a SAT or STEP file. Optionally, Morph will use the SGM geometry engine for STEP or SGM files if the following is specified in the input deck:

```
sgm: true
```

13 Faceted Geometry

This option controls how Morph does geometric evaluations with the CAD engine. If OFF, Morph asks the CAD engine directly for all geometric evaluations. This results in direct analytic and spline evaluations of the CAD BREP surfaces, which can be slow, especially if the model has spline surfaces. If ON, Morph asks the CAD engine for the graphics facets of the model and then does the geometric evaluations on the facets avoiding the expensive geometry evaluation calls on the CAD engine.

The default is ON for ACIS geometry engine and OFF for the SGM geometry engine. You can also manually control the behavior like this:

```
use_faceted_geometry: off
```

14 Geometric Healing

By default, Morph heals the geometry immediately after import for both the ACIS and the SGM geometry engines. Sometimes healing causes more problems than it fixes, and can result in Morph failing to mesh, or causes Morph to build an unexplainably strange mesh, or ignore some volumes completely. If any of these occur, turning OFF healing with the following input file syntax could help:

```
heal_after_acis_import: ON/off
```

Optionally, the user can turn OFF healing for Morph, and the user can do the healing themselves in Cubit before calling Morph.

If Morph is using the SGM geometry engine healing will always be done, and cannot be turned off.

15 Volumes To Mesh

By default, Morph meshes all of the solid volumes in the model, ignoring any sheet bodies. If only a subset of the volumes should be meshed, or if the user wants Morph to mesh a set of non-watertight sheet bodies as if it were a solid volume, use the volumes_to_mesh section in the Morph input file as shown below. For the options below, the CAD file does not need to be changed, rather any geometry entities not specified below will simply be ignored.

15.1 Option 1 - Subset of Solid Volumes

To instruct Morph to only mesh a subset of the solid volumes in the CAD file, use the include and exclude options of volumes_to_mesh. The following example will instruct Morph to mesh all solid geometric volumes with the names "Housing*" except volumes named "Volume2*".

```
volumes_to_mesh:
  - include:
       volume_names: ["Housing*"]
 - exclude:
       volume_names: ["Housing2*"]
```

15.2 Option 2 - Surface Collection Volumes

Morph has the ability to treat a collection of disconnected non-watertight CAD sheet bodies as a single volume, meshing the logical interior as indicated by the normals of the specified surfaces. For example:

```
volumes_to_mesh:
  - surface_collection_volumes:
      - surface_names: ["BoundarySurfaces1*"]
        name: surfColVol1
      - surface_names: ["BoundarySurfCol2*"]
        name: surfColVol2
```

This example instructs Morph to build 2 pseudo volumes as follows:

1. The surfaces named "BoundarySurfaces1*" will form the boundary of the pseudo volume named "surfColVol1".

2. The surfaces named "BoundarySurfaces2*" will form the boundary of the pseudo volume named "surfColVol2".

IMPORTANT NOTE: The specified surfaces must be oriented with their surface normals pointing outward from the logical interior of the space that the specified surfaces bound. Similarly, while Morph can handle small overlaps and gaps between the surfaces specified for a surface_collection_volume, gross overlaps, or completely identical surfaces should be avoided.

With this specified in the Morph input file, these surface_collection_volumes can then be referenced by name anywhere in the Morph input file. For example, they could be referenced in the blocks or geometry_priority_order sections as if they were normal volumes.

If even 1 surface collection volume is specified, all solid bodies will be ignored for that Morph run.

16 Tet 10 higher order elements

By default, Morph outputs tet4 (4-noded tetrahedral) elements. The 'tet10' option specifies that the resulting mesh should have 10-noded tetrahedral elements. The midnodes are projected to the geometry unless the resulting neighbor elements have a normalized inradius quality value less than the value specified with 'tet10_normalized_inradius_straightening_threshold'. Here is an example of using both of these options:

```
tet10: on
tet10_normalized_inradius_straightening_threshold: 0.15
```

17 Blocks

By default, Morph builds 1 tetrahedral element block for every solid volume in the geometry CAD file. This can be changed by adding a blocks section in the Morph input file. Morph can build tetrahedral, triangular, EM quad slot, or beam element blocks. Tetrahedral elements cannot currently be combined with any other block type in a single Morph run. Triangular, EM quad slot, and beam elements can all be generated in a single Morph run, but currently not combined with tetrahedral blocks.

17.1 One Tetrahedral Block Per Volume

Morph's default output is one tetrahedral block per volume to the resulting exodus file. Volumes can be excluded from getting a default output block as shown with the input file syntax shown in the example below, where all volumes except volumes 3 and 4 get default output blocks:

```
blocks:
  - tet_block_per_volume:
        exclude:
            volumes: [3,4]
```

By default, each "block per volume" will be given the name and ID of the corresponding volume. With this default behavior, you can change the name of the blocks by changing the name of the corresponding volumes in the geometry CAD file. The name of each "block_per_volume" can also be prepended with a string as follows:

```
blocks:
  - tet_block_per_volume:
        named_volume_name_prefix: "myBlockPrefix_"
```

These 2 examples can be combined to both specify a prefix and exclude volumes from getting a default block as shown in the example below:

```
blocks:
  - tet_block_per_volume:
        named_volume_name_prefix: "myBlockPrefix_"
        exclude:
            volumes: [3,4]
```

In this example, all volumes except 3 and 4 will get a default block written for them, and their names will be the names of the CAD volume prepended with "myBlockPrefix_"

17.2 Custom Tetrahedral Blocks

Morph will build tetrahedral blocks composed of more than one volume using the Morph input file syntax below:

```
blocks:
  - tet:
        volume_names: [bolt*]
        name: "BoltBlock"  #optional
        id: 17 # optional
```

In this example, a block with ID 17 and name "BoltBlock" will be created from all volumes with names following the naming pattern "bolt*". Both name and id are optional arguments, although one or the other must be specified. If no ID is specified, a default ID will be assigned. If no name is specified, the name "block_X" will be assigned where X is the ID of the block.

Any number of custom tetrahedral block can be specified.

```
blocks:
  - tet:
        volume_names: [bolt*]
        name: "BoltBlock"
  - tet:
        volume_names: [washer*]
        name: "WasherBlock"
  - tet:
        volume_names: [nut*]
        name: "NutBlock"
```

Any solid volume in the CAD file not included in a custom block will be a candidate for a block following the tet_block_per_volume options that can be specified along side any custom tet block specifications.

17.3 Custom Triangle Blocks

Tri3 element blocks can be built on any surface in the CAD file using the Morph input file syntax below:

```
blocks:
  - tri:
        surface_names: [surface9,surface10]
        name: "myTriBlock"
        id: 19
```

The above example will build a tri mesh on surfaces9 and surface10 and put all of the resulting triangles in block 19 with name myTriBlock.

Tri blocks cannot be specified in the same Morph run as tet blocks. However, tri blocks can be specified with em_quad_slot and beam blocks in the same Morph run.

When a tri block is specified, the same overlay grid is generated as if a tetrahedral mesh was being built. The only difference is what is written to the exodus file (i.e. tris vs tets written). This allows tri meshes to be built with the same geometry tolerance concepts as Morph uses for tet meshes.

Morph has some extra abilities with tri blocks to propagate starting from the triangles on the user specified surfaces to eliminate the need to specify so many surfaces. There are 2 keywords to control this propagation as shown below:

```
propagate_skin: OFF/on
propagate_into_gaps_wrt:
    volume_names: [VolumeNamesToGuidePropagation]
```

Both of these are OFF by default.

17.3.1 propagate_skin

Using "propagate_skin" tells Morph to start from the triangles on the specified surfaces for the block and propagate to adjacent triangles also on CAD surfaces AND also in contact with the void space surrounding the object. The void space surrounding the object could be on the exterior of the CAD assembly, or within a cavity/enclosure inside of the assembly. The "propagate_skin" option only propagates to triangles that are on other CAD surfaces. It will not propagate to triangles covering any non-watertight gap in bounding surfaces of a volume (for example a non-closed STL, or a surface_collection_volume)

17.3.2 propagate_into_gaps_wrt

Using "propagate_into_gaps_wrt", along with some specified volumes tells Morph to propagate into triangles covering non-watertight gaps in the specified volumes. The propagate_into_gaps_wrt keyword is used extensively with surface collection volumes that have large gaps between surfaces. For example, a Morph input file might look like:

```
volumes_to_mesh:
  - surface_collection_volumes:
      - surface_names: [ExteriorSurface*]
        name: ExteriorSkinVol
```

```
      - surface_names: [CavitySurface*]
        name: CavityVol
geometry_priority_order:
  - volume_names: [CavityVol]
  - volume_names: [ExteriorSkinVol]
blocks:
  - tri:
        surface_names: [ExteriorSurface*]
        propagate_into_gaps_wrt:
            volume_names: [ExteriorSkinVol]
        name: "ExteriorTriBlock"
  - tri:
        surface_names: [CavitySurface*]
        propagate_into_gaps_wrt:
            volume_names: [CavityVol]
        name: "CavityTriBlock"
```

In this example, 2 surface_collection_volumes are created, one representing the exterior of the CAD assembly, and another representing an interior cavity. Since the cavity is spatially inside of the assembly, the cavity is prioritized highest so it will be cut out of the tets built for the exterior surface_collection_volume. Then 2 tri blocks are created, one for the exterior and one for the cavity, with propagate_into_gaps_wrt specifying each the respective surface_collection_volumes to fill any gaps in the surfaces used define the surface_collection_volumes.

17.4 Custom Quad Slot Blocks

```
blocks:
  - em_quad_slot:
        name: "myBlockName"
        id: 20
        inner:
            curve_names: ["innerCurve*"]
        outer:
            curve_names: ["outerCurve*"]
```

When an em_quad_slot block is specified, Morph will ensure that the curves specified in the inner section is meshed with the exact same number of nodes as the curves specified in the outer section, and then the corresponding nodes are connected to form a set of quads starting on the inner curves and ending on the outer curves. This forms a custom set of quads that can connect the one part of the model with another. For example, this might represent a slot that connects an interior cavity to the exterior of an assembly.

Obviously, em_quad_slot blocks do not result in a general quad mesh on a surface. In fact, no surface is required or specified. The resulting quads simply connect the specified inner curves with the specified outer curves, while ensuring conformity with the tri blocks that may be spatially near the specified curves.

em_quad_slot blocks cannot be specified in the same Morph run as tet blocks. However, em_quad_slot blocks can be specified with tri and beam blocks in the same Morph run.

17.5 Custom Beam Blocks

Morph can also create beam element blocks along specified curves, using the syntax below.

```
blocks:
  - beam:
        curves: [2,7]
        name: "myBeamBlock"
        id: 18
```

In this example, beam elements will be created along curves 2 and 7 and put into block 18 with the name "myBeamBlock".

The number of beams along each curve is decided by the global_element_size and any refinements specified.

The beam block will share nodes with whatever triangles may be spatially near the specified curves or the vertices of the specified curves.

Beam blocks cannot be specified in the same Morph run as tet blocks. However, beam blocks can be specified with tri and em_quad_slot blocks in the same Morph run.

18 Sidesets

Sidesets can be added to a Morph mesh through the Morph input file only. There are no command line options to add sidesets.

Morph adds sidesets to CAD surfaces by including a sidesets section in the Morph input file. Any number of sidesets can be specified. Each sideset must have its own sub-section under the "sidesets" section. Each sideset is specified with a collection of CAD surfaces, an id, and a name:

```
id: <integer> # required
name: <string> # optional
surface_names/surfaces/surface_ranges: # required
wrt: <OWNING|opposing_blocks|all_touching_blocks> # optional
```

The wrt keyword is optional and indicates the sense of the sideset with respect to the adjacent blocks. OWNING will be used if wrt is not specified and will create the sideset with respect to the block of the volume the specified surfaces are topologically connected to. If wrt is specified as opposing_blocks, the sideset will be created with respect to the block opposing the topologically connected volume. If wrt is specified as all_touching_blocks, the sideset will be created with respect to all adjacent blocks.

Below is an example of a sidesets section in a Morph input file that will create 2 sidesets:

```
sidesets:
  - id: 13
    name: pressureSurface
    surface_names: ["PressureLoad*"]
  - id: 17
    name: temperatureBC
    surface_names: ["Heat*Source"]
    wrt: opposing_blocks
```

This will create 2 sidesets. Sideset 13 will be named pressureSurface and contain all surfaces with the naming pattern of "PressureLoad*". Sideset 17 will be named temperatureBC and contain all surfaces with the naming pattern of "Heat*Source"

19 Local Refinement

The mesh built by Morph will be refined through the Morph input file only. There are no command line options to refine the mesh.

Morph refines the resulting mesh if a "refine_at" section is added in the Morph input file. Any number of refinements can be specified. Each refinement must have its own sub-section under the "refine_at" section. With the exception of 2 custom refinements (i.e. surface_curvature and sizes_from_file), each refine_at sub-section must have a location indicator (i.e. where to refine), and a target size desired at that location.

The target size is specified with the "size" keyword as follows:

```
- size: <double>
```

supported location indicators include:

```
- location: [<double-x>, <double-y>, <double-z>]
- bounding_box:
     min: [<double-x>, <double-y>, <double-z>] # min corner
     max: [<double-x>, <double-y>, <double-z>] # max corner
- line_segment:
     start: [<double-x>, <double-y>, <double-z>]
     end: [<double-x>, <double-y>, <double-z>]
- bounding_sphere:
     center: [<double-x>, <double-y>, <double-z>]
     radius: <double>
- volume_names
- volumes
- volume_ranges
- surface_names
```

```
 - surfaces
 - surface_ranges
 - curve_names
 - curves
 - curve_ranges
 - vertex_names
 - vertices
 - vertex_ranges
```

Below is an example of a refine_at section with 3 refinements specified:

```
refine_at:
 - surface_names: ["BCSurface"]
   size: 4.5
 - location: [-3.4, 5.1, -0.02]
   size: 1.25
 - vertex_names: ["PointLoadLocation"]
   size: 5.3e-2
```

19.1 Using Sizes From File

Morph supports an advanced refinement feature to honor sizes specified in an input exodus file. The syntax of this option is:

```
refine_at:
 - sizes_from_file: /path/MeshWithNodalSizeField.exo
   nodal_size_field_name: MyNodalSizeField
```

In this example, the specified exodus file must contain a nodal size field named MyNodalSizeField. Morph will refine the resulting mesh according to the sizes stored in the nodal field.

The sizes_from_file option can be combined with other types of local refinement specifications.

20 Interfaces

By default, Morph meshes everything contiguously, meaning adjacent elements associated to different volumes will share the same nodes. The "interfaces" section of the input file allows you to specify sets of volumes that should be disconnected from other sets of volumes. Disconnecting volumes can be useful when two volumes are physically close to each other but should not actually be touching or when you want the simulation code to compute contact between two volumes.

In the following example, any element in a volume that has a name starting with 'bolt' or 'washer' will be disconnected from the volume named 'frame'. In addition, elements in the volume 'fred' will be disconnected from elements in the volume 'bob':

```
interfaces:
  - disconnect:
        volume_names: [bolt*, washer*]
    from:
        volume_names: [frame]
  - disconnect:
        volume_names: [fred]
    from:
        volume_names: [bob]
```

If you don't want any volumes in the mesh to be meshed contiguously with other volumes, this can be specified with the keyword "disconnect_all" as shown here:

```
interfaces:
  - disconnect_all
```