HyperMesh Advanced Training

For technical support, contact us at:

PHONE	(248) 614-2400. Mon – Thurs: 8:00 AM to 7:00 PM (EST). Fri: 8:00 AM to 5:00 PM (EST). Ask for HyperMesh Support
FAX	(248) 614-2410
EMAIL	hmsupport@altair.com
WEB	www.altair.com

FTP Site:

ADDRESS	ftp.altair.com or ftp2.altair.com
LOGIN	ftp
PASSWORD	<your email address>

Altair® HyperMesh® Advanced Training

Comments concerning the training material may be made to documentation@altair.com.

Table of Contents

Preface

Who should attend

This course is designed for students who have attended HyperMesh Basic Training and have a working knowledge of HyperMesh. It covers two topics. The first is advanced geometry clean-up and meshing techniques. The second is macros and macro creation, a tool to improve your productivity using HyperMesh. Even if you have no experience writing any program code, by the end of this session, you will be successful at creating your own macros.

Each section also includes “hands-on” exercises to help you become comfortable with the new techniques presented here.

Manual notations

This manual uses the following notations.

- `courier` for text that you type in
- ***bold italic*** for panel names, button names, and sub-panel names.

Information that is of importance or warning messages will appear in a note box.

This is an example of a note box. Important information appears here.

Macro commands may occasionally be longer than the text line of this document. In those cases, the text wraps to the next line and is slightly indented. When asked to enter such a command, type the entire command ignoring the wrap-around. This will ensure the system can accurately interpret the command you entered.

For more help

Should you desire additional help with material in this course, see the back of the title page of this manual for contact information.

Comments about this manual may be directed to documentation@altair.com.

Section 1: Advanced Geometry Clean-up and Meshing Techniques

Geometry cleanup and automeshing, often cited as the most time consuming aspects of Finite Element Analysis (FEA) modeling, are the most critical aspects in generating a quality mesh that will give accurate analysis results. A methodical approach to geometry cleanup and automeshing activities can save a great deal of time in the analysis process and assist in obtaining dependable results.

Element Quality Criteria

In FEA modeling, element quality greatly effects the accuracy of the analysis results. Many modern FEA solvers have routines to compensate for some measure of poor element quality, but it is not a good practice to rely on these compensations. The FEA modeler must take into consideration element quality, and thereby judge whether the analysis results are meaningful.

The ideal four-node (quad) plate element is a planar square. Two types of errors can result from translating a single node. If one of the nodes is translated in the plane of the remaining nodes, the interior angles change and the edge lengths vary between sides introducing skew and aspect ratio into the element. If one of the nodes is translated out of plane of the others, the result is warpage.

With first order tria elements warpage is not possible, but aspect ratio and skew remain valid measures of element quality. The element checks in HyperMesh test these properties and provide feedback as to the quality of the element.

Keep element quality criteria in mind while observing the geometry of the model. When trying to determine whether or not to cleanup the geometric feature, ask yourself the following questions.

1. **Is this feature important to my analysis?** Often designers model details that are unimportant in FEA. Tapers, fillets, steps and ridges in sheet

metal parts, while required for manufacturing, can often be ignored in analysis.

2. **How will this feature affect my mesh and element quality?** Consider the benefits and consequences of removing the feature versus meshing it. With HyperMesh, often the simplest way to answer this is to mesh with the feature, then mesh without the feature, and compare the results.

3. **Will it be easier to correct the geometry before meshing, or to correct the elements after meshing?** As the geometric features become more detailed and complex, it is often easier and faster to mesh around them and correct the mesh afterwards. With the new element cleanup functionality in HyperMesh, correcting element quality becomes a simple, interactive task.

4. **Do I need to mesh the surfaces, or can this part be modeled using other techniques?** Sometimes the more basic “surfaceless” meshing tools result in a better quality mesh faster and easier than using the surfaces presented. Tools such as spline, ruled mesh, line drag, spin, skin, and element offset can often be used to accurately represent a part without being confined to the given geometry.

The next five example problems demonstrate how to manage these situations.

Exercise 1: Suppressing features adversely affecting element quality

Step 1: Retrieve the model `taper.hm40`

In this exercise, use the automesher to create a surface mesh both before and after suppressing unnecessary feature lines. Use the ***cleanup*** sub-panel within the ***automesh*** panel to avoid extra panel navigation.

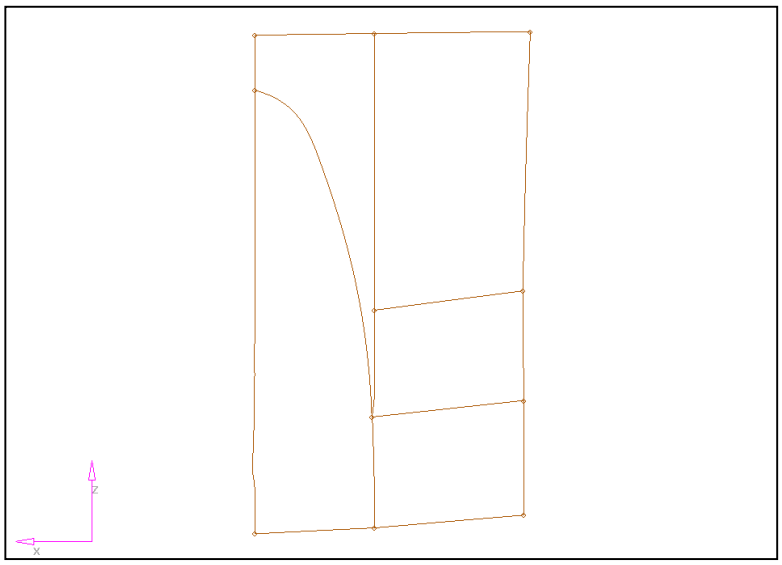
1. Go to the ***files*** panel from any page.

2. Select the ***hm file*** sub-panel by clicking the corresponding radio button.

3. Click in the text field next to ***file =*** and select `taper.hm40` using the file browser, or type in the file name.

4. Click ***retrieve*** to read the file into HyperMesh.

This file contains five surfaces. The surface with a curved edge forms a tail that tapers into a narrow point. This feature, as well as the other trim lines forming the boundaries between the individual surfaces, will result in a less than ideal mesh on these surfaces.



Step 2: Create a baseline mesh

1. From the ***2D*** page, select the ***automesh*** panel.

2. Select the ***create mesh*** sub-panel by clicking the corresponding radio button.

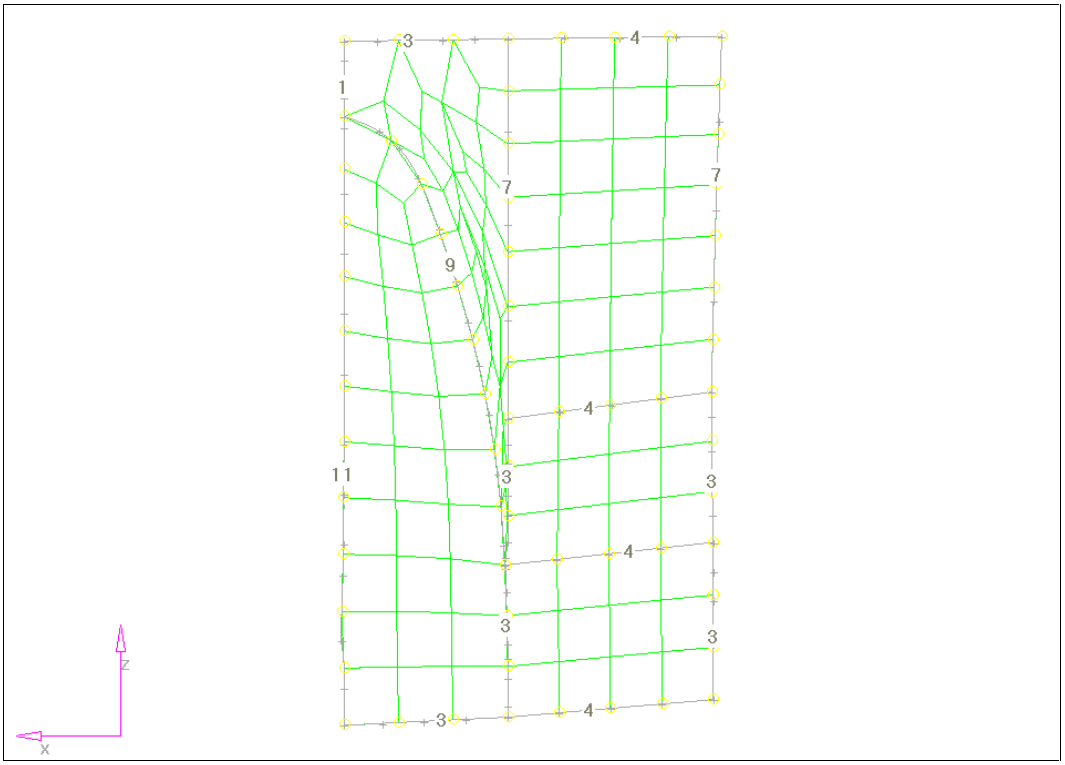
3. Click on the green ***surfs*** button and select ***all*** from the pop-up menu.

4. Click in the text field next to ***elem size =*** and enter `25.0`

5. Click ***mesh*** to enter the ***secondary automesh*** menu.

6. Click ***mesh*** to display the preliminary mesh.

Notice how the curved line results in severely distorted elements. Since the curved line is unimportant to our analysis, we can suppress it and improve the element quality.



Step 3: Creating an improved mesh

1. Click ***abort*** to discard the existing mesh.

2. While still in the main ***automesh*** panel, select the ***cleanup*** sub-panel by clicking the corresponding radio button.

3. Click the green ***line*** button under ***toggle:*** to make that the active entity selector.

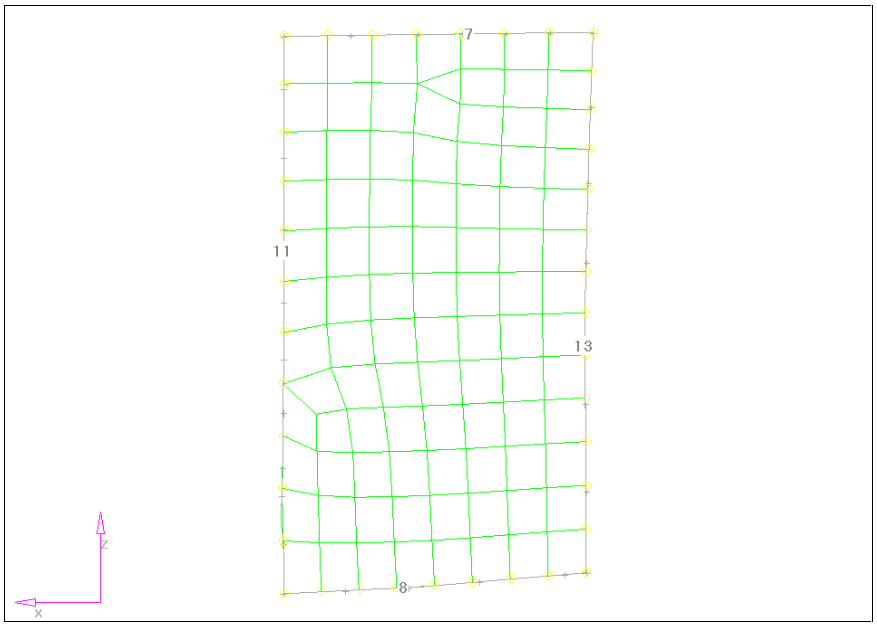
4. Click each of the interior green surface edges to suppress those features.

5. Return to the ***create mesh*** sub-panel.

6. Click ***mesh*** to enter the ***secondary automesh*** menu.

7. Click ***mesh*** to display the preliminary mesh.

Note the improved mesh quality. Since the baseline meshing operation retains the original mesh densities, the remaining problem areas result from transitions due to the different mesh densities on corresponding sides of the rectangular surface. To fix this, manually adjust these densities or re-calculate the edge densities.



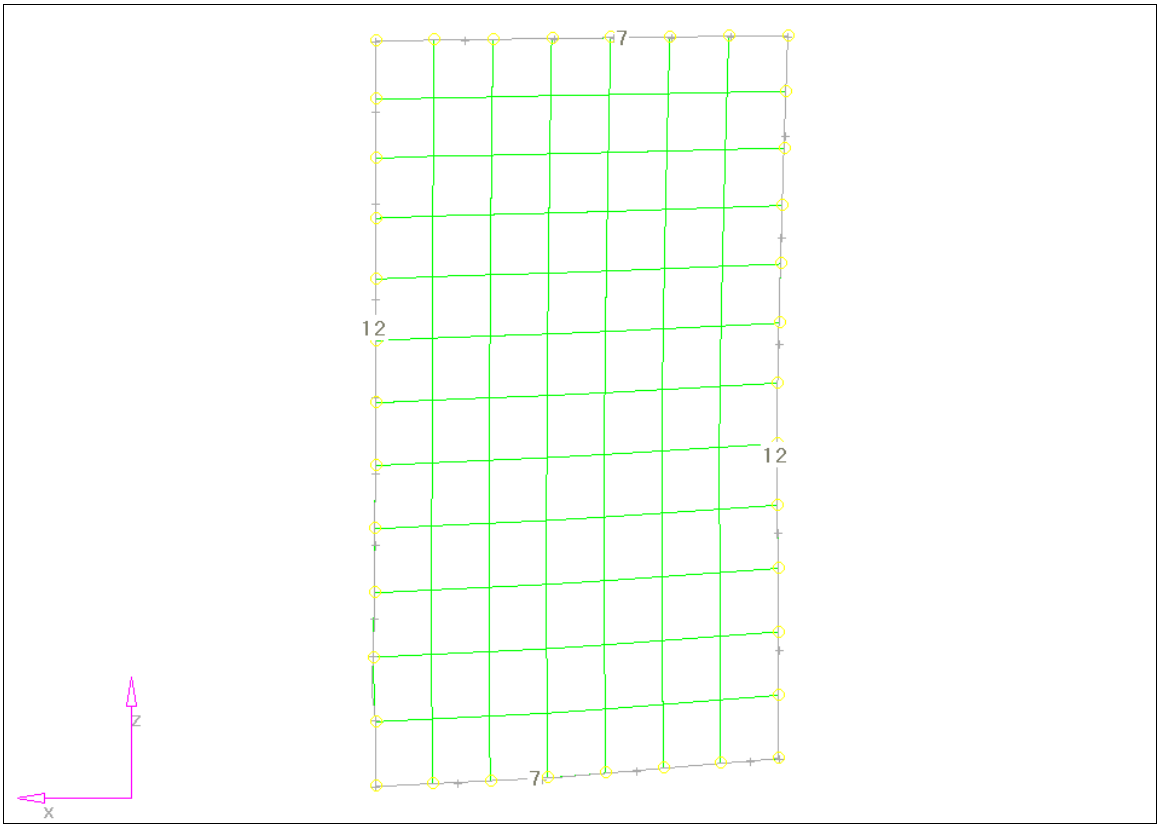
Step 4: Adjusting element densities

1. Click the green ***recalc all*** button to re-compute the edge densities based on the desired 25 mm element size.

2. Click ***mesh*** to re-generate the mesh.

3. Click ***return*** to accept the mesh and go back to the main ***automesh*** panel.

   Note the improved consistency in the resulting mesh.



Summary

In this exercise, we used the automesher to point out the problems caused by unnecessary feature lines. By suppressing the lines, the resulting mesh on this surface greatly improved.

Exercise 2: Adjusting fixed points to correct surface edge definitions

A common problem when working with imported geometry is mismatched surface vertices. This occurs when neighboring surfaces do not share the same vertex at a boundary. Often these points are so close together that they appear from normal viewing distances as a single point.

An easy way to identify these areas is to use the interactive meshing mode of the automesher, with the element densities displayed. Pay careful attention to all surface edges with an element density of one. When this occurs in a corner or where three (or more) surface edges come together, it almost always is the result of mismatched vertices.

In this exercise, use the automesher to identify mismatched surface vertices. Once identified, use the ***cleanup*** panel within the ***automesh*** panel to replace multiple fixed points with a single point.

Step 1: Retrieve the model `fixed_points.hm40`

1. Go to the ***files*** panel from any page.

2. Select the ***hm file*** sub-panel by clicking the corresponding radio button.

3. Click in the field next to ***file =*** and select `fixed_points.hm40` using the file browser, or type in the name.

4. Click ***retrieve*** to read the file into HyperMesh.

This file contains a single surface. Near the lower right side, where the curved surface meets the vertical, there is a short surface edge that can be eliminated by adjusting fixed points.

Step 2: Create a baseline mesh

1. On the ***2D*** page, go to the ***automesh*** panel.

2. Select the ***create mesh*** sub-panel by clicking the corresponding radio button.

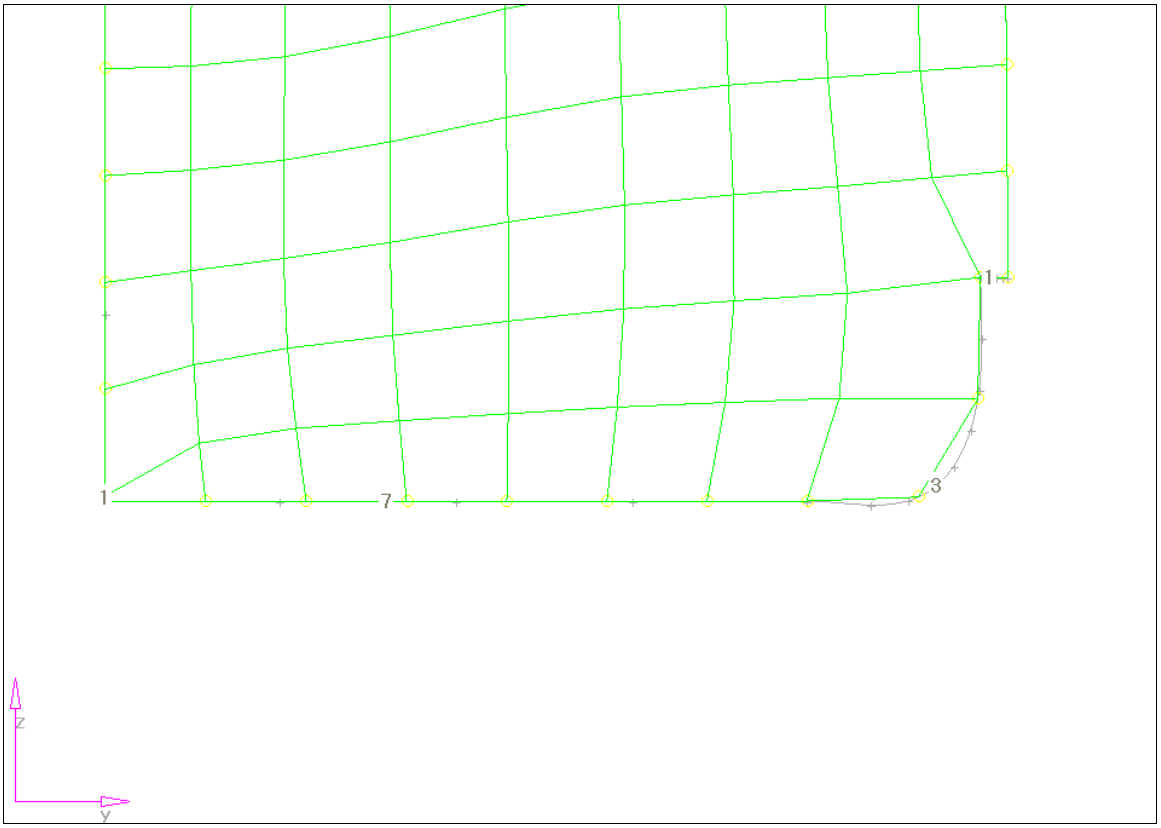
3. Click on the green ***surfs*** button and select ***all*** from the pop-up menu.

4. Click in the field next to ***elem size =*** and enter `25.0`

5. Click ***mesh*** to enter the ***secondary automesh*** menu.

6. Click ***mesh*** to display the preliminary mesh.

Notice the distortion in the mesh due to the short edge mentioned previously. Notice also the distortion at the lower left corner of the surface, and the indicated element density of one. Zoom into this area for a closer look.



With a closer view, what appears at first to be a tria element is actually a quad with a very short side. To correct this problem in both corners, use the ***replace fixed points*** function.

Step 3: Correct the mismatched vertices

1. Click ***abort*** to discard the existing mesh.

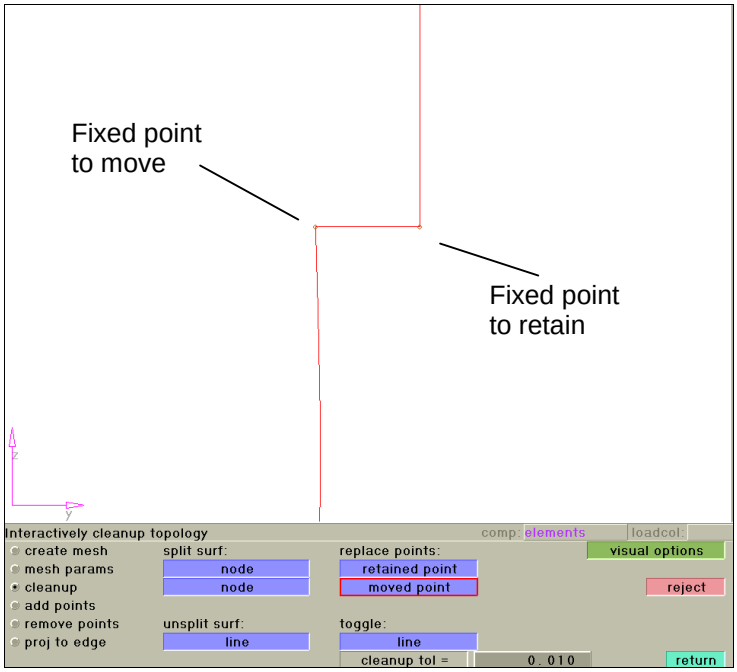
2. While still in the main ***automesh*** panel, select the ***cleanup*** sub-panel by clicking the corresponding radio button.

3. Click the green ***retained point*** button under ***replace points:*** to make this the active function.

4. Zoom in on the lower right area of the surface, near the top of the curved line.

5. Click the fixed point on the right, at the bottom of the vertical line, to select it as the point to retain.

   Notice once you select the retained point, the active selector automatically changes to ***moved point***.



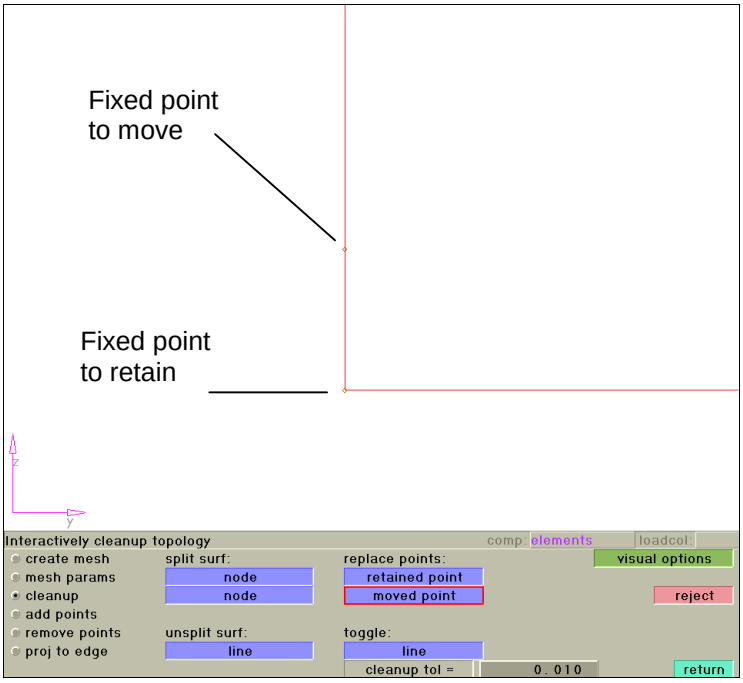
6. Click the point on the left, at the top of the curved line.

   The points automatically combine at the bottom of the vertical line.

7. Click **f** on the blue permanent menu (alternately, hit the **F** key on the keyboard) to fit the model to the screen.

8. Zoom in on the lower left corner.

9. Select the lower fixed point as the retained point, and the upper point as the one to move.



10. Click **f** on the blue permanent menu (alternately, hit the **F** key on the keyboard) to fit the model to the screen.

Step 4: Create an improved mesh

Now that the problems at the vertices are corrected, create the mesh.

1. Return to the ***create mesh*** sub-panel by selecting the corresponding radio button.

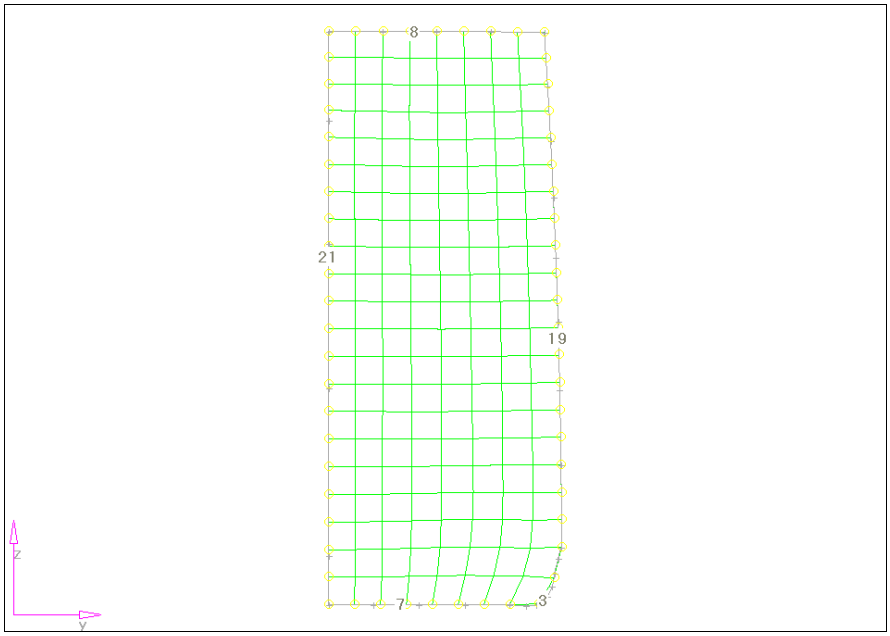
2. Select the surface.

3. Click ***mesh.***

4. Click ***mesh*** again to create the mesh.

5. Click ***return*** to accept the mesh.

   Note the regular features of the meshed area.



Summary

In this exercise, we used the automesher to identify two meshing problems caused by mismatched surface vertices. Once identified, the problems were corrected using the ***replace fixed points*** function resulting in a better quality mesh.

Exercise 3: Meshing around a troublesome feature

In this exercise, use the automesher to identify troublesome geometric features. Once identified, use alternate meshing techniques to obtain a good quality mesh throughout the component.

Step 1: Create a baseline mesh

1. Retrieve the model file `crease.hm40` using the ***files*** panel.

The crease running horizontally across the model causes some problems with the mesh on this part. This is a required feature, so it cannot be ignored.

A baseline mesh will point out the problem areas.

2. On the ***2D*** page, go to the ***automesh*** panel.

3. Select the ***create mesh*** sub-panel by clicking the corresponding radio button.

4. Click on the green ***surfs*** button and select ***all*** from the pop-up menu.

5. Click in the field next to ***elem size =*** and enter `25.0`

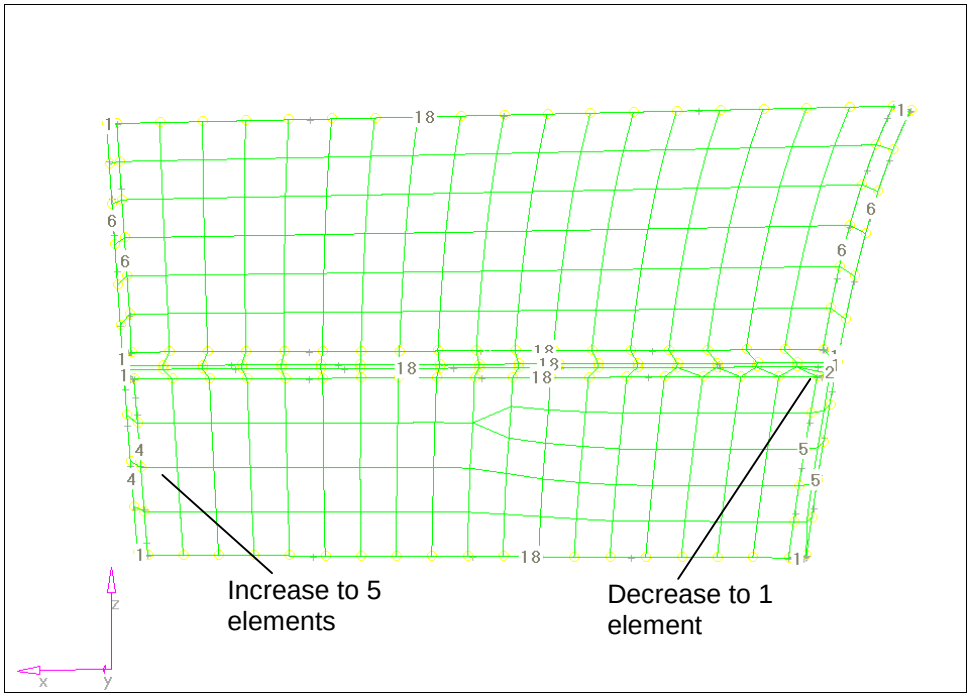
6. Click ***mesh*** to enter the ***secondary automesh*** menu.

7. Click ***mesh*** to display the preliminary mesh.

Notice the transition element in the lower half. Also, if you rotate the model slightly, another transition occurs in the vertical surface on the right side of the lower half of the model. You can easily eliminate them by adjusting element densities.

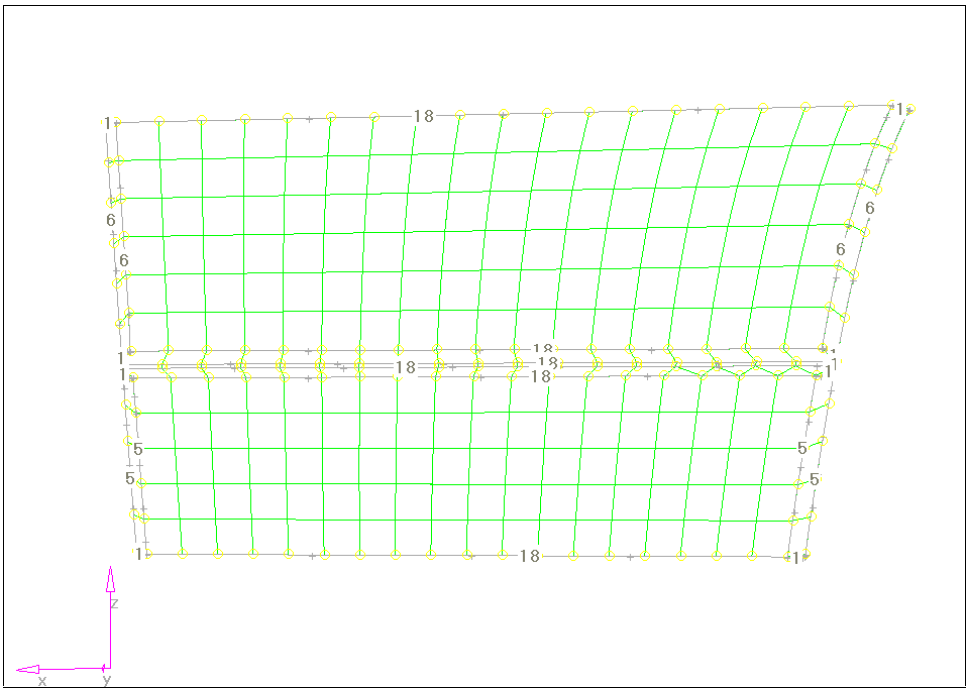
8. On the ***density*** sub-panel, make sure the ***adjust edge*** selector is active (it is highlighted with a blue halo). If not, click on the green button to select it.

9. The numbers on the left side of the lower half of the part indicate an element density of four. Click once on the numbers to increment these edges to five elements.



10. On the right side of the model, right click once on the edge density of two. This will decrease the element density on this edge to one.

11. Click ***mesh*** to regenerate the mesh.



Notice the transition elements are eliminated. This is because the edge densities for corresponding surface edges match.

While the mesh in general appears consistent, further inspection on the right side of the crease indicates a grouping of many tria elements. Also, the narrow surface in the center of the crease has resulted in elements with a very high aspect ratio.

The geometry of the crease caused the grouping of tria elements on the right side of the crease. Because of the curvature at the edges of the part, the crease is considerably longer in the center than at the top edges. This causes the nodes to be mismatched along the crease, and leads to skewed elements.

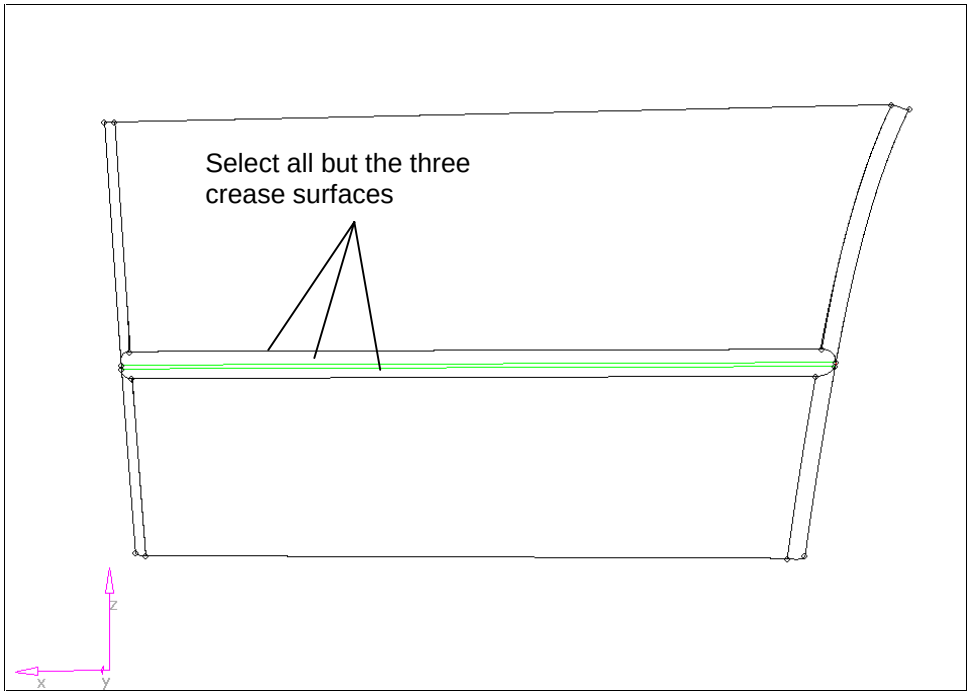
To correct this situation, you could edit the geometry to trim the ends of the crease, then deal with the central area and the ends separately. Alternately, you could mesh around the crease and use a different method to create the crease elements. This is the approach we will use here.

Step 2: Abort the mesh and create a new mesh around the crease

1. Click ***abort*** to discard the baseline mesh and return to the main ***automesh*** panel.

2. Click ***reset*** button under ***surfs*** to reset the surface selection.

3. Pick the individual surfaces above and below the crease surfaces by clicking them one at a time. Select a total of six surfaces.



4. Click ***mesh*** to enter the ***secondary automesh*** panel.

5. Click ***mesh*** again to display the preliminary mesh.

6. Click ***return*** to accept this mesh and go back to the main ***automesh*** panel.

7. Click ***return*** to go back to the main menu.

Step 3: Meshing the ends of the crease

If you were to mesh the crease now, the narrow surface at the bottom would be ignored. To prepare for correcting connectivity across this gap, you need to perform some hand editing of the mesh to build in the transition elements from the edges to the crease. First, stitch the elements across the ends of the crease together.

1. Restore the ***left_end_crease*** view using the ***view*** panel on the blue permanent menu.

2. Press the ***F3*** key to jump to the ***replace nodes*** panel.

3. Check the ***at mid-point*** option.

4. Click the nodes on each side of the narrow surface. After you select the second node, the gap closes and the elements are connected across the narrow surface.

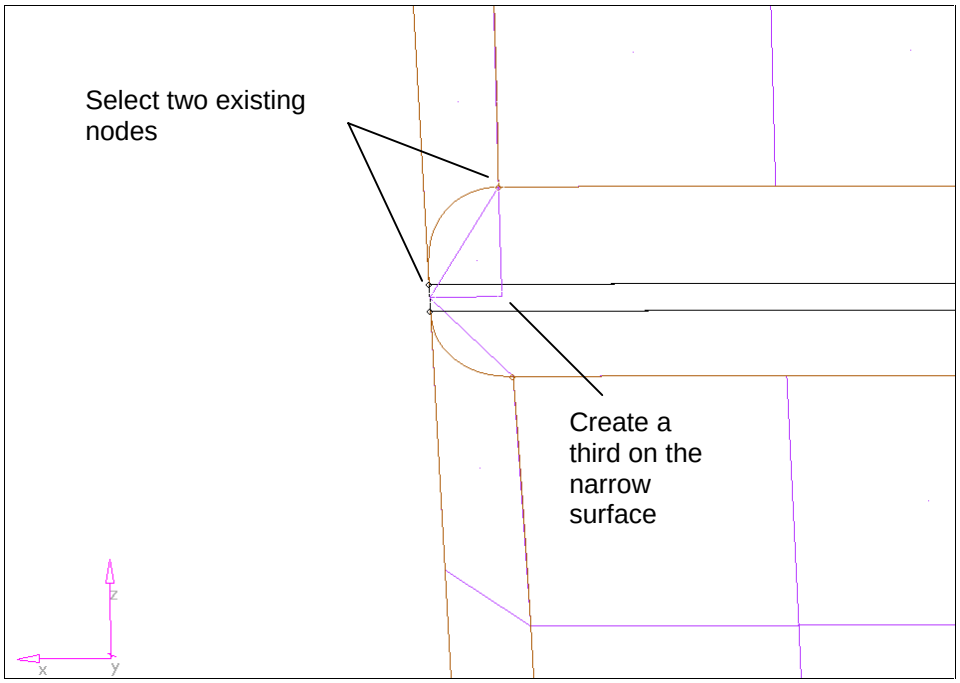
Now, we need to build some tria elements to form the transition into the elements of the crease.

5. Press the ***F6*** key to jump to the ***edit element*** panel.

6. Select the ***create*** sub-panel.

7. Select the radio button next to ***tria***.

8. Select the two nodes as indicated in the following diagram.



To create the tria element we need create one more node, at the center of the narrow surface on the bottom of the crease.

9. Click and hold the left mouse button. After two seconds, the cursor changes from a crosshair to a square box.

   Keep holding the mouse button down while moving the mouse until the narrow surface is selected.

10. Click on the highlighted surface at the position indicated in the illustration.

11. Select the corresponding three nodes to create a tria element on the other side of the crease.

To fix the right end of the crease, we will repeat the same steps.

12. Restore the ***right_end_crease*** view from the ***view*** panel.

13. Stitch the elements together using the ***replace*** function.

14. Create the two tria elements using the ***edit element*** panel.

Step 4: Create the remainder of the elements for the crease

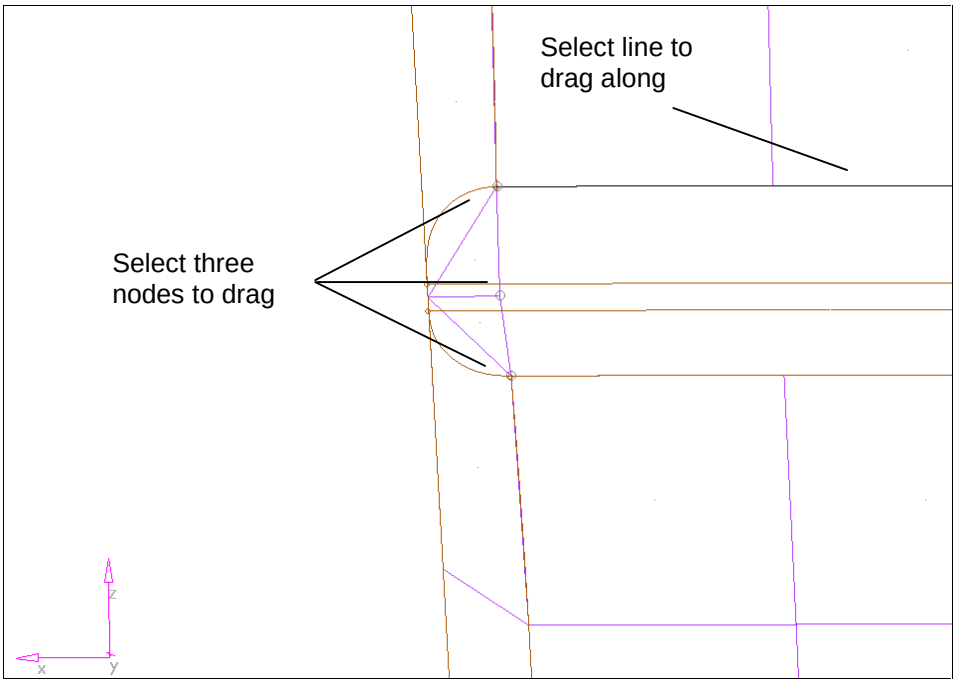
1. On the ***2-D*** page, go to the ***Line Drag*** panel.

2. Restore the ***left_end_crease*** view.

3. Select the ***drag geoms*** sub-panel.

4. Select ***node list*** as the desired geometry type to drag.

5. Select the three nodes as indicated below.



6. Click the green ***line list*** button next to ***along:***.

7. Select the top or bottom edge of the crease. If you are in standard graphics mode, you may need to zoom out.

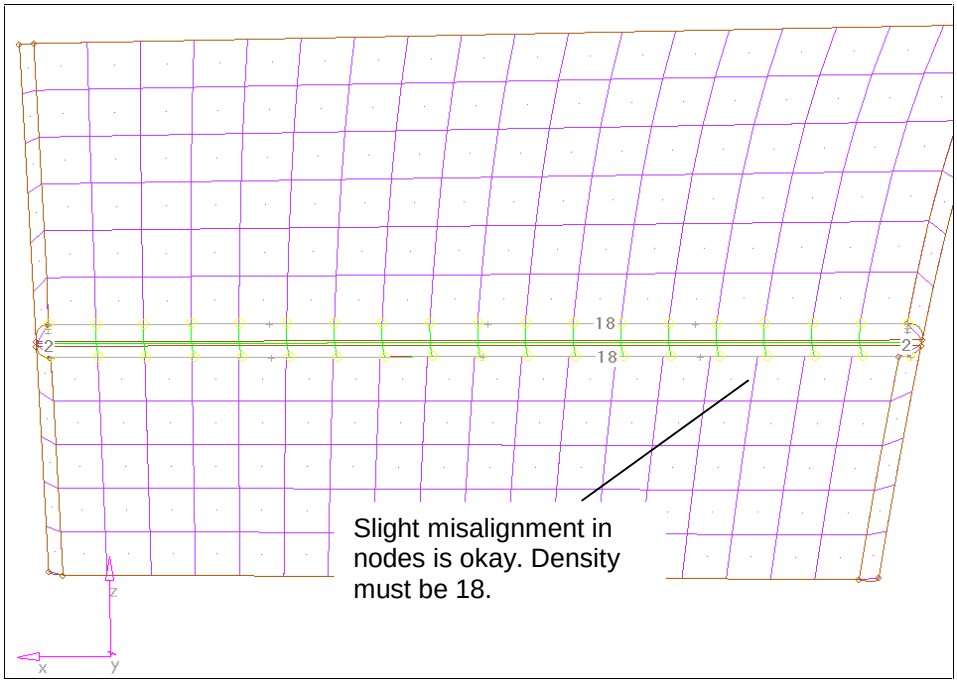
8. Click ***drag***.

The nodes indicated need to be adjusted as close as possible to the existing nodes so they can be combined in the next step. The number of elements along the crease needs to be exactly 18, the same as the adjacent surface edges. The position of the nodes will be off a bit because the top and bottom edges of the crease are unequal in length.

9. If necessary, adjust the density of the elements along the crease to 18 by clicking the displayed density number. If you adjusted the density, then click ***mesh*** to refresh the preliminary element display.

10. Once the number of elements is 18, click ***return*** to go back to the ***line drag*** page.

11. Click ***return*** to go back to the main menu.



Step 5: Use the equivalence function to correct the mismatched nodes

1. On the ***tools*** page, go to the ***edges*** sub-panel.

2. With ***comps*** as the active selector, pick any element in the model.

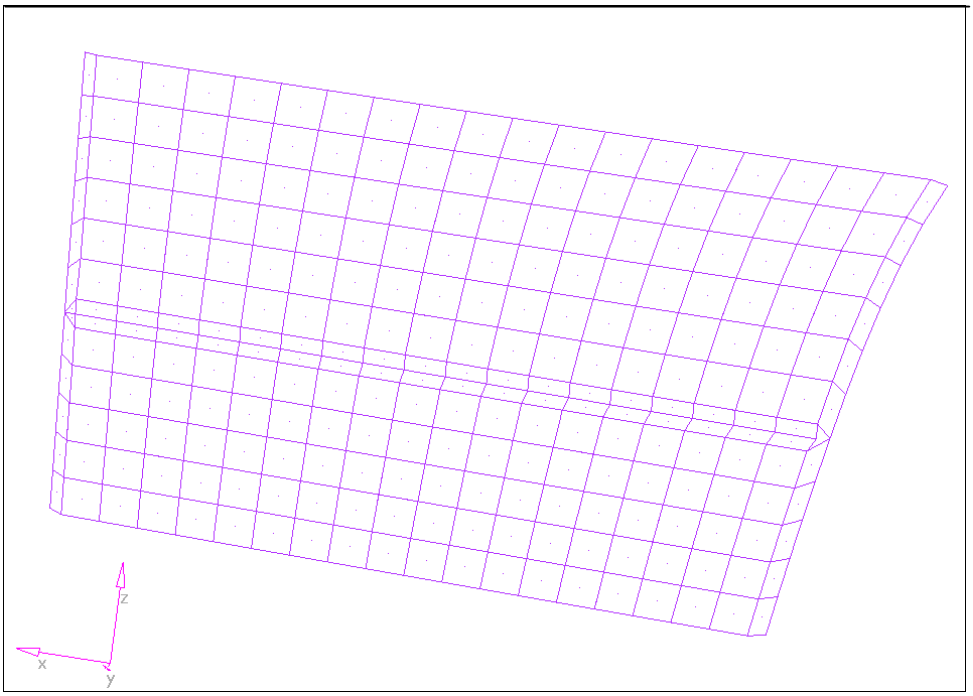
3. Click in the number field next to ***tolerance=*** and enter `6.0`.

4. Click ***preview equiv*** to identify the nodes to be equivalenced.

   The identified nodes should outline the crease elements created by the line drag operation. Adjust the tolerance up or down until exactly 40 nodes are identified.

5. Click ***equivalence*** to automatically replace the nodes and stitch the elements together.

6. Click ***return*** to go back to the main menu.



Summary

In this exercise, we used the automesher and identified a troublesome geometric feature and the meshing problems it caused. Our solution was to ignore the troublesome surfaces of the feature, and mesh the area using one of the surfaceless meshing tools in HyperMesh. Since some of the elements were created using surfaces and others were not, we can not guarantee connectivity between them. Therefore, some element and node editing was required.

Working with Defeaturing and Geometry Cleanup Tools

The defeaturing tools in HyperMesh are located on the ***defeature*** panel on the ***Geom*** page. Five sub-panels access functions that remove trim lines from surfaces, identify and remove pinholes, identify and remove surface fillets and identify and remove edge fillets.

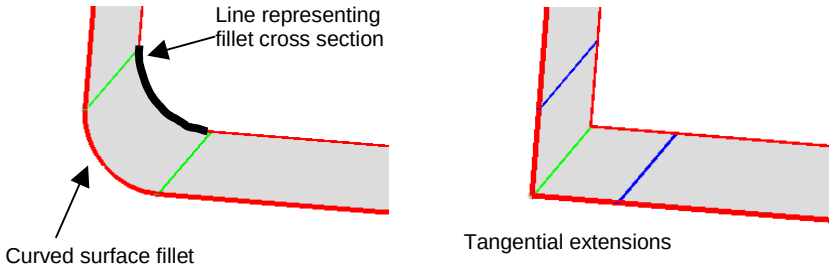
The ***trim lines*** sub-panel removes surface trimming operations. Two modes are available, removing interior trim lines and removing all trim lines. Removing all trim lines reverts a surface to its initial definition. Depending on the method used to create the surface, the results of this operation may vary. Removing interior trim lines will remove selected surface edges entirely contained within the boundary of the parent surface.

The ***pinholes*** sub-panel identifies and removes holes within the surface. To be considered a hole, the internal surface edge must be continuous and close upon itself. In topology mode, it will appear as a free, red edge. The approximate diameter of the hole (or longest distance across in the case of non-round holes) is required for identification. Once candidate holes are identified, a confirmation step allows individual holes to be un-selected.

A fixed point is created at the center of the hole once they are removed. This can be used to create weld or beam elements representing a bolted or riveted connection.

The ***remove interior trim lines*** function can be used to remove holes without creating the residual fixed point. Alternately, the fixed points can easily be suppressed before meshing.

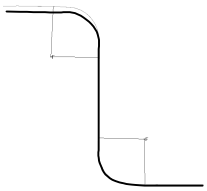
The ***surf fillets*** sub-panel identifies and removes surface fillets. When fillets are removed, they are replaced with a hard corner formed by a tangential extension of the fillet surface



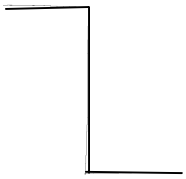
The surfaces can be identified by selecting a representative fillet cross section or by searching selected surfaces for fillets that fall between a range of radii.

Once surface fillets are identified, a secondary menu allows the fine-tuning of the selection criteria.

The ***edge fillets*** sub-panel identifies and removes edge fillets. When fillets are removed, the curved edge is replaced with a hard corner. Edge fillets must be located on free surface edges to qualify for removal.



Curved edge fillets



Hard corners

The ***trim-intersect*** sub-panel also removes edge fillets, however, the points of tangency are specified by you.

Exercise 4: Preparing the bracket

In this exercise, prepare a simple bracket for incorporation into a larger model part. Use the tools on the ***defeature*** panel to remove holes, edge fillets and surface fillets, then export the modified model as IGES data. Later, it will be imported and become a component in a larger part.

Step 1: Importing the original bracket model

1. Go to the ***files/import*** sub-panel.

2. Select the iges import translator.

3. In the filename field, enter `bracket.iges` or use the browser to find the file.

4. Click ***import***.

HyperMesh allows you to pre-select an entity by pressing down and holding the left mouse button and dragging the cursor over an entity. HyperMesh highlights it in white. If this is the entity you want, release the mouse button and the selection is made. This feature improves the precision of entity selection.

Since imported component collectors are gray, it is difficult to differentiate them from pre-selected entities. We recommend you change the imported component's color to make it easier for you to see pre-selected entities.

The following steps take you through the color change process.

Step 2: Change the color of the geometry

1. Go to the ***colors*** panel (on any page except ***Tool***).

2. If the entity selector is not set to ***comps***, change it using the switch.

3. Click in the ***comps*** button and check ***lvl8*** as the component you want to change.

4. Click ***select*** to place that component in the selection mark.

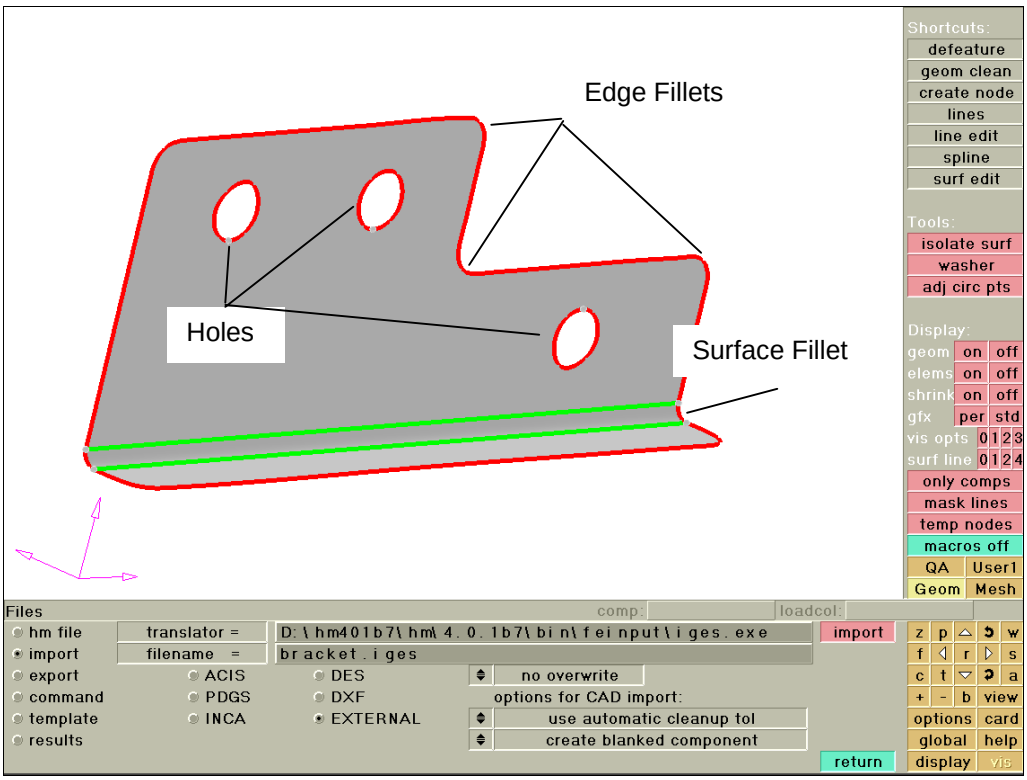
5. Click on the ***color*** button and select a color from the pop-up color palette.

6. Click ***set color*** to change the color of the ***lvl8*** component.

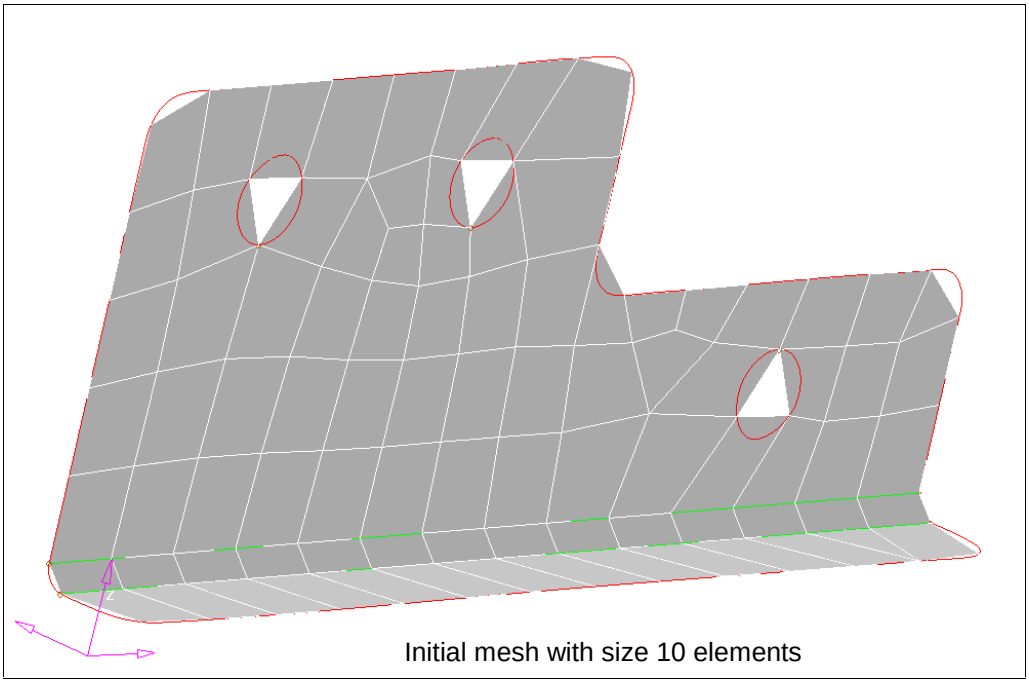
7. Click ***return*** to go back to the main menu.

☞	Starting with HyperMesh version 4.0, the color palette is mapped in the `hm.cfg` configuration file that is read when HyperMesh starts. This file is in the `hm/bin` directory of your installation. You can edit it using any standard text editor. The RGB values for each of the 16 colors in the standard color palette are set using the *setcolor command.

The file contains three surfaces , with several holes, edges fillets and surface fillets. The scale of the part, along with these features would result in a very poor mesh.



Notice the element distortions around the holes as well as at the rounded edge fillets. Removing these features will greatly improve the mesh quality.



Initial mesh with size 10 elements

Step 3: Removing holes

1. On the ***Geom*** page, select the ***defeature*** panel.

2. Select the ***pinholes*** sub-panel.

3. With the ***surfs*** selector active, pick the surface containing the three holes.

4. In the number field next to ***diameter <***, enter `10.0`.

5. Click ***find***.

   At this point, all three holes are identified with a white **xP**. Should you want to retain any of these holes, they must be deselected by right clicking the **P**. In this problem, we will remove them all.

6. Click ***delete*** to remove the holes.

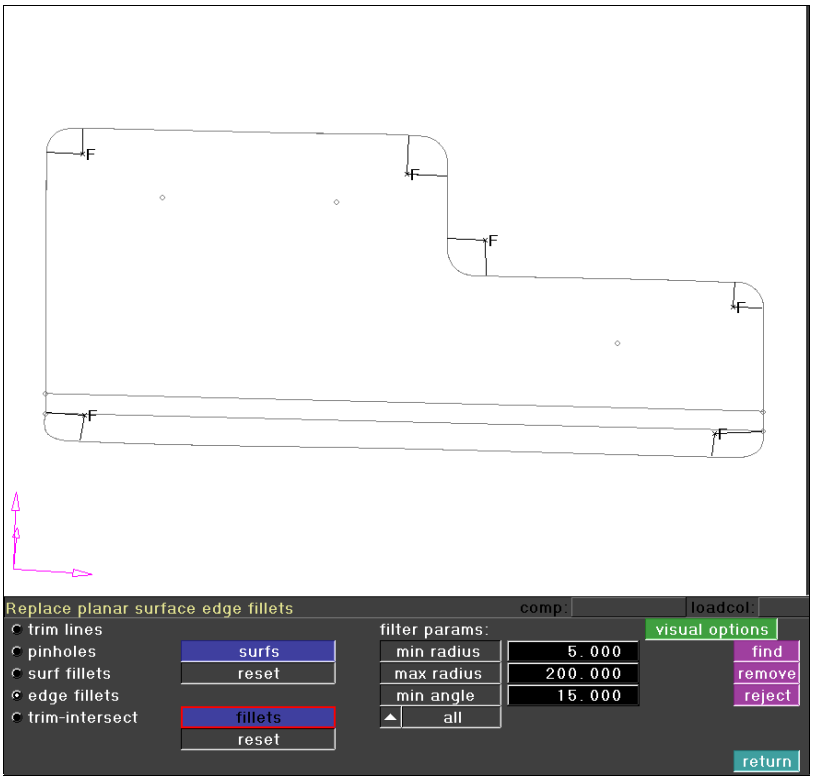
Fixed points appear at the center and the holes are removed. The fixed points force nodes to be located at these points when the surface is meshed. This facilitates the creation of a weld or bar element to represent a fastener at the correct location. If they are not required, the fixed points can easily be suppressed during the meshing process.

Step 4: Removing edge fillets

1. Go to the ***edge fillets*** sub-panel.

2. Click on the ***surfs*** selector and pick ***all*** from the pop-up menu.

3. Click ***find*** to identify the edge fillets. A total of six fillets should be identified.



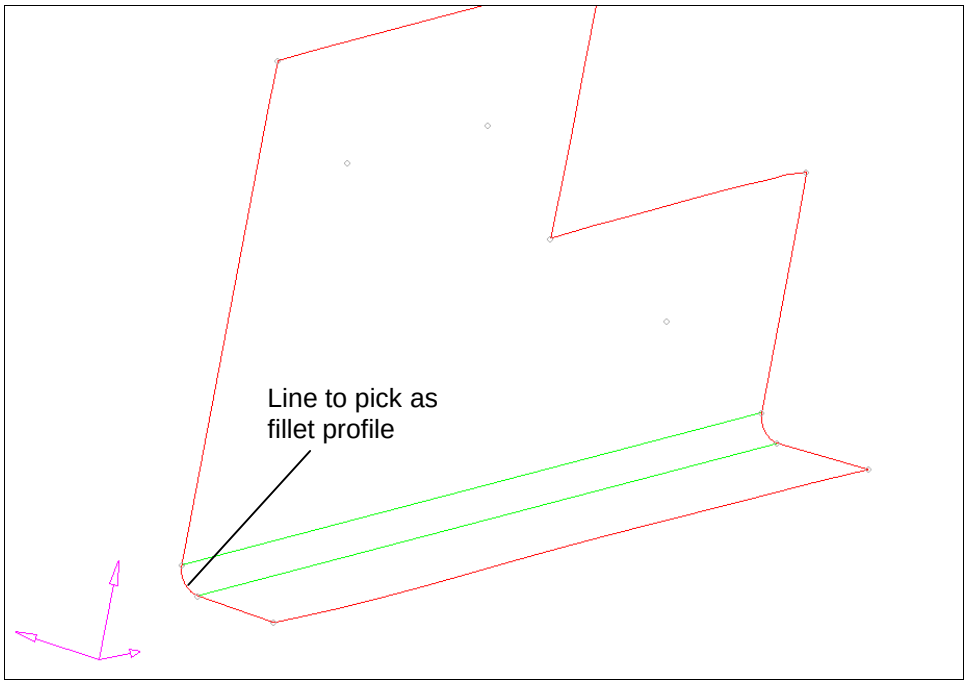
Like the pinhole operation, the edge fillets are pre-selected as they are identified. If any of the fillets are not to be removed, they should be deselected with a right mouse click.

4. Click ***remove***.

Step 5: Removing the surface fillet

1. Go to the ***surf fillets*** sub-panel.

2. Pick one of the lines representing the cross section of the fillet.



3. Click ***find fillets*** to identify the fillet.

4. Click ***remove*** to replace the curved surface fillet with a hard corner.

5. Click ***return*** to go back to the main menu.

Step 6: Creating a midline

In preparation for spot welding the components together, we need to create a midline on the lower flange.

1. Go to the ***lines/create midline*** sub-panel.

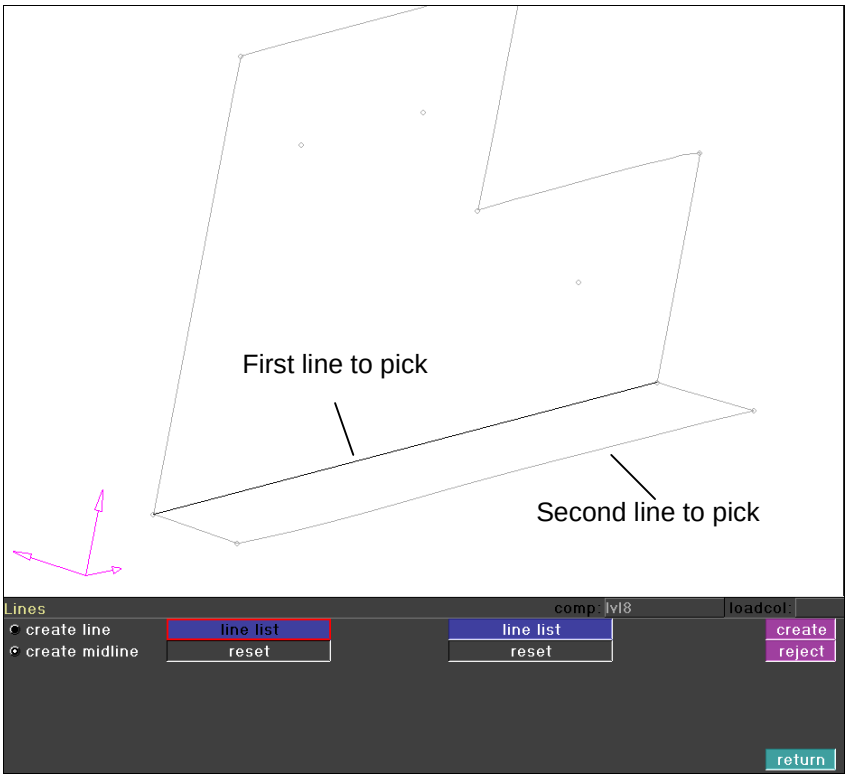
2. Pick one of the lines on the lower flange of the bracket.

3. Click on the green ***line list*** box on the right side of the menu to make it active.

4. Pick the other line on the lower flange of the bracket.

5. Click ***create***.

6. Click ***return*** to go back to the main menu.



Step 7: Exporting the model

Now that we have completed defeaturing the bracket model, we need to export it as IGES data so it can be brought into the rest of the assembly.

1. Go to the ***files/export*** sub-panel.

2. Click the radio button next to ***IGES***.

3. Enter `modified_bracket.iges` in the ***filename*** field.

4. Click ***write.***

Step 8: Clearing the HyperMesh database

For the time being, we are done with the bracket model. The next step is to bring in the base component. Before doing so, the HyperMesh database for this session needs to be cleared.

1. Enter the ***delete*** panel by hitting the ***F2*** key.

2. Click ***delete model*** to clear the database.

3. Answer ***yes*** to the confirmation.

4. Click ***return*** to go back to the main menu.

Summary

In this exercise, we prepared a bracket for meshing using the tools on the ***defeature*** panel. The ***pinholes function*** identified and removed interior holes. The ***edge fillet*** function turned problematic curved edge fillets into hard corners that are much easier to mesh. The ***surface fillet*** function created tangential extensions in lieu of curved surface fillets. We created a midline on the flange in preparation for locating spotwelds. Finally, we exported the part as IGES data so it could be combined with another part to form an assembly.

Exercise 5: Preparing the base component

In this exercise, bring in a new component and defeature it. Then, import the `modified_bracket` file and spot weld the two components together.

Step1: Import the Part

1. Import the file `base1.iges`.

2. Change the color of the **lvl6** component.

Step 2: Removing edge fillets

1. Click ***return*** to go back to the main menu.

2. Go to the ***defeature/edge fillets*** sub-panel.

3. Select all of the surfaces.

4. Click ***find***.

5. Click ***remove*** to replace the edge fillets with hard corners.

Step 3: Removing surface fillets

1. Got to the ***surf fillets*** sub-panel.

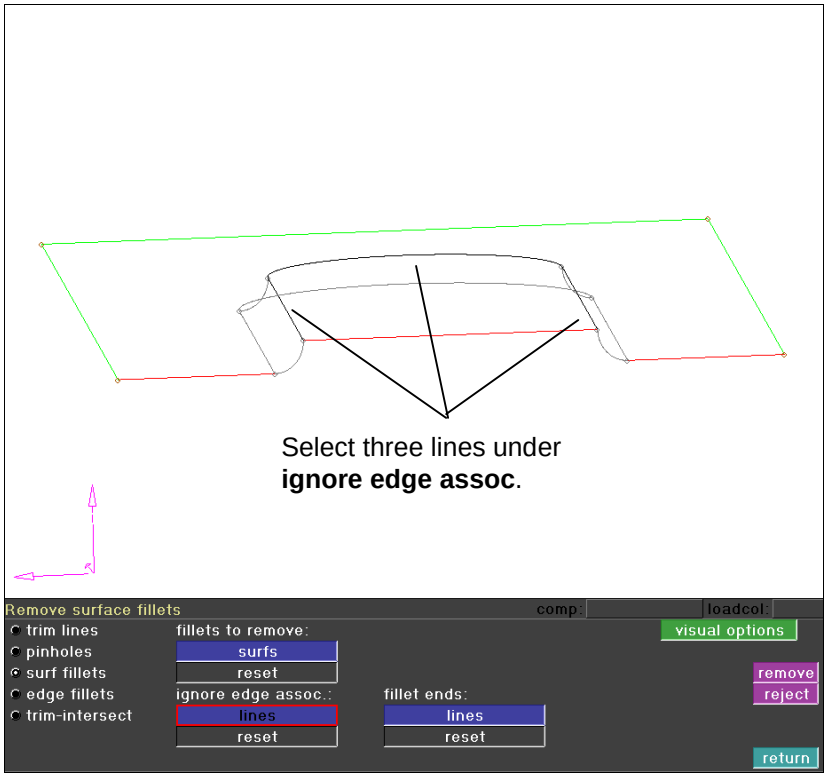
2. Select ***all*** for ***surfaces to search***.

3. Enter `2.0` for the ***min radius*** under ***fillet params:*** .

4. Click ***find*** to identify the fillets.

5. Click on the green ***lines*** button under ***ignore edge assoc.:*** .

6. Select the inner lines around the top relief, as shown below.



These lines must be identified under ***ignore edge associativity*** to indicate that the surface adjacent to the fillet should not be considered when calculating the tangent.

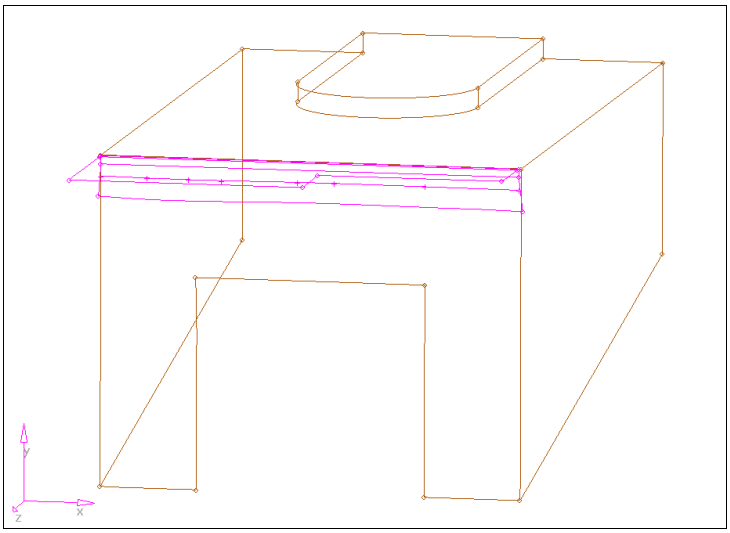
7. Click ***remove*** to replace the fillets with hard edges.

At this point, we are done defeaturing the base model. Before continuing, save your work as an `.hm` binary file using the ***files/hm file*** sub-panel.

Step 4: Combining the two components

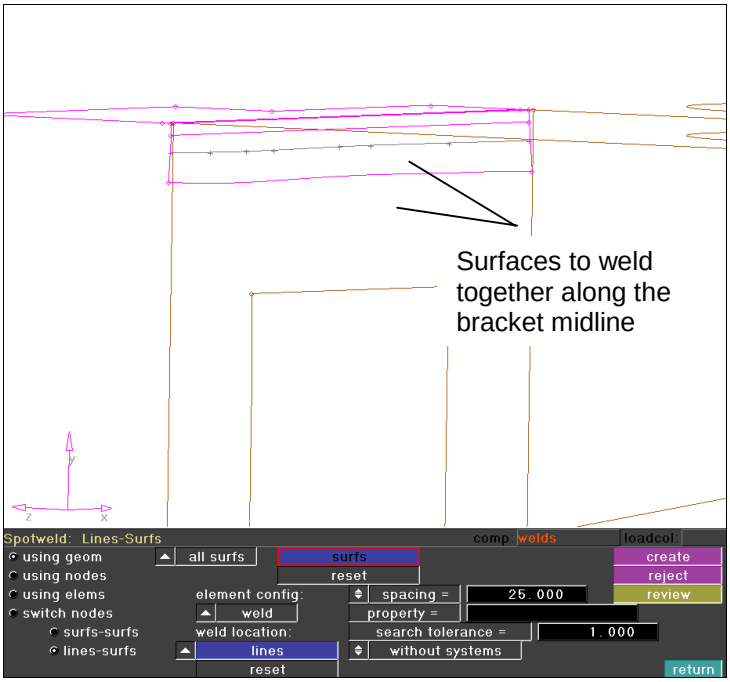
1. Import the `modified_bracket.iges` file.

2. Change the color of the ***lvl5*** component.



3. Create a component collector named `welds` to organize the spotwelds.

4. On the ***1-D*** page, select the ***spotweld*** panel.



The ***spotweld*** panel allows the creation of weld elements joining components before a mesh is generated. We have a midline (previously created on the bracket) that will locate the welds.

5. Go to the ***Geom/lines-surfs*** sub-panel.

6. Select the bracket flange and the face of the base component as the surfaces to weld together.

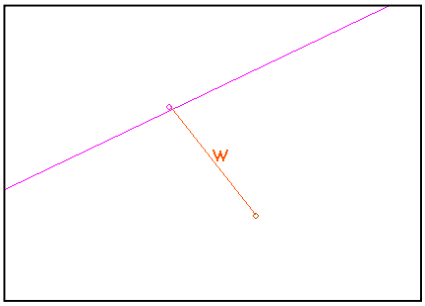
7. Click the green ***lines*** button under ***weld location*** and select the midline on the bracket flange.

8. Enter `25.0` in the filed next to ***spacing=.***

9. Enter `1.0` in the field for ***search tolerance =.***

10. Click the toggle next to ***build systems*** to switch to ***without systems***.

11. Click ***create*** to make the welds.



Six weld elements are created. Fixed points are created on both surfaces at the weld locations. The welds are located along the line every 25 mm. Fixed points are created on the surfaces at the location of the weld elements. When the surfaces are meshed using the automesher, nodes will be forced to these locations, ensuring that the plate mesh is connected to the weld elements.

Weld elements will only be created if the surfaces are within the search tolerance at the weld location.

12. Save your work as a HyperMesh binary file.

Summary

In this exercise, we revisited the edge fillet and surface fillet tools and used them on another component. The surface fillets we removed were comprised of several individual surfaces, including three way connections. Next, we imported the bracket prepared in Exercise 4 and spot welded it to the base component, forming an assembly.

Exercise 6: Meshing the parts

In this exercise, mesh the combined assembly. Use the mixed element type to improve the mesh quality.

Step 1: Meshing the bracket

1. Go to the ***display*** panel on the blue permanent menu.

2. Switch the toggle on the right side of the panel from ***elems*** to ***geoms***.

3. Right click the checkbox corresponding to the **lvl6** collector.

4. Go to the ***automesh/cleanup*** sub-panel.

5. Click the ***toggle*** button, then select the two edges parallel to the corner edge.

   These two edges indicate the previous location of the surface fillet. During the defeaturing operations, these edges were suppressed. However, the edge topology (free/shared/suppressed) is a function of HyperMesh, and that information is lost when the geometry is written using the IGES format.

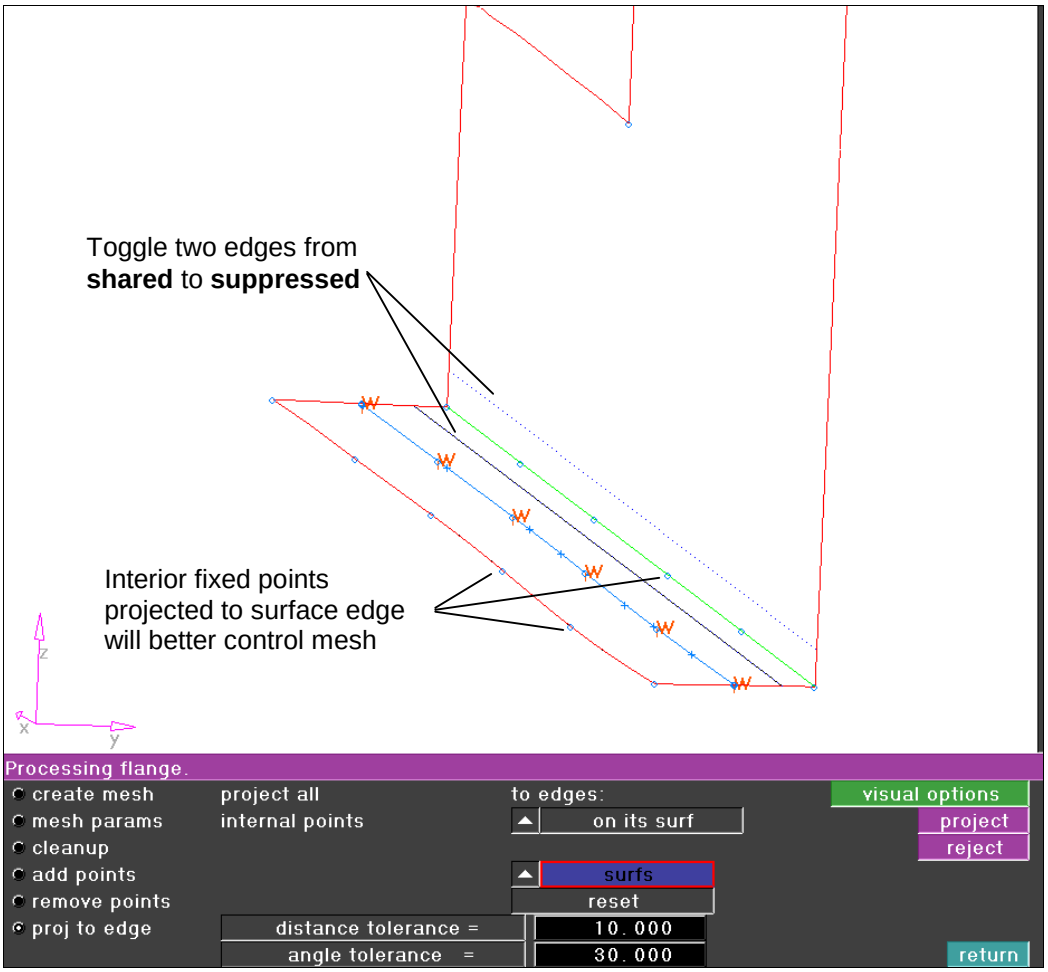
6. Go to the **proj to edge** sub-panel.

7. Click the switch under ***to edges:*** and select ***on its surf*** from the pop-up menu.

8. Click on the green ***surfs*** button and select the flange surface.

9. Click ***project*** to create additional fixed points on the surface edge corresponding to the weld points.

10. Go to the ***remove points*** sub-panel, and suppress the extraneous fixed points at the location of the first and last welds.



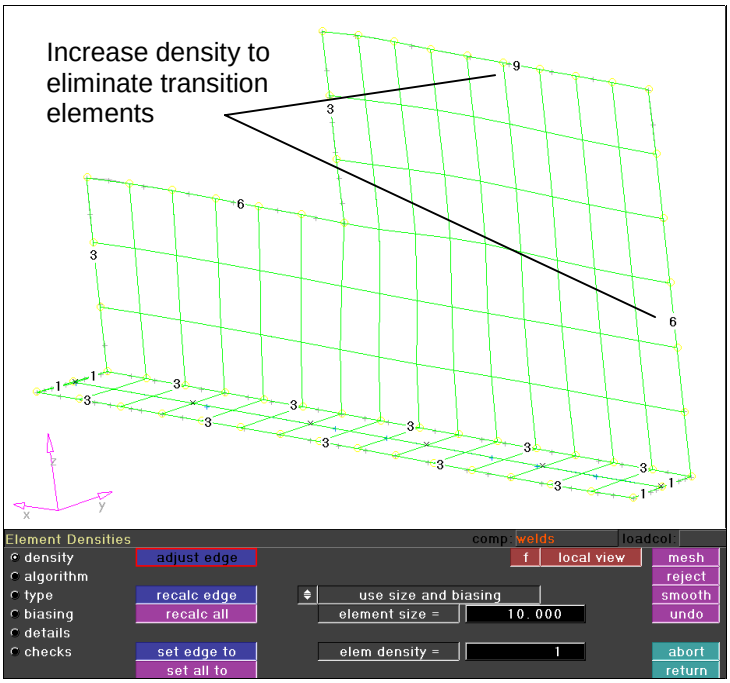
11. Go to the ***create mesh*** sub-panel.

12. Pick both of the bracket surfaces.

13. Switch the toggle at the bottom of the panel from ***elements to current component*** to ***elements to surface's component***.

14. Click ***mesh***.

15. Increase the element density on the two edges as shown in the illustration until the transition elements are eliminated.



16. Click ***return*** to accept the mesh and go back to the ***automesh*** panel.

Step 2: Meshing the base1 component

1. Go to the ***display*** panel, and turn the geometry display of the ***lvl5*** component off, and the ***lvl6*** component back on.

2. Using the ***automesh/proj to edge*** sub-panel, project the fixed points from the weld elements to the surface edges.

3. On each end of the weld line, use the ***remove points*** sub-panel to suppress the extraneous fixed point at the location of the first and last welds on the line.

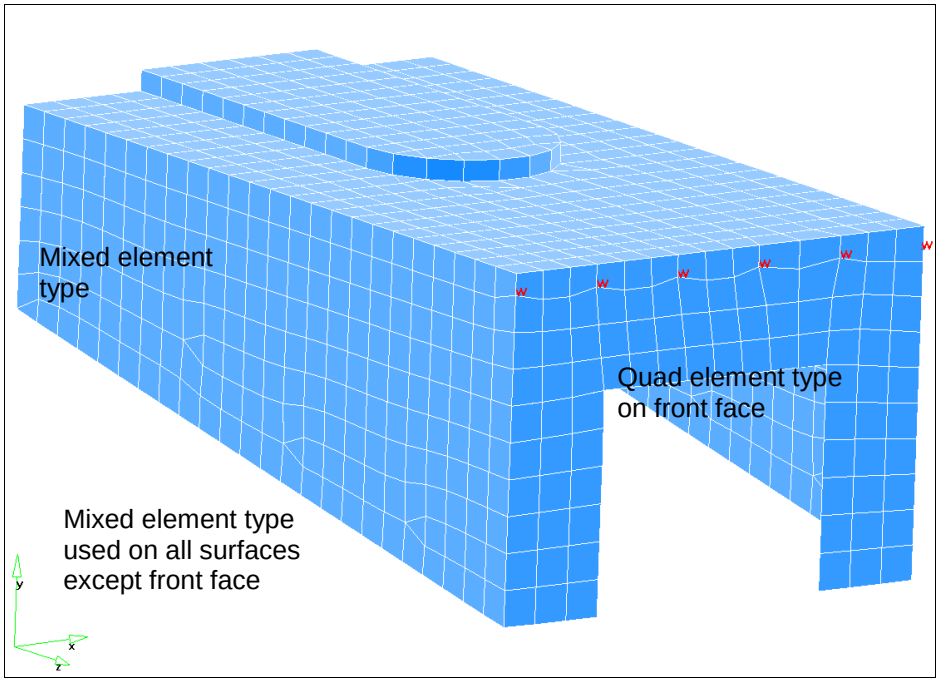
4. Go back to the ***create mesh*** sub-panel and select ***displayed*** as the surfaces to mesh, then create the initial mesh.

Several transition elements appear in the mesh due to the shape and size of the geometry. The mixed element type handles these transitions by introducing a tria element instead of distorting a quad. This will usually result in a better mesh quality.

5. On the ***type*** sub-panel, switch the ***element type:*** to ***mixed***. Click the green ***set all*** button to update the element types for all of the surfaces, then click ***mesh*** to re-generate the initial mesh.

   The mesh improved on all surfaces, except the front face where the bracket attaches.

6. Switch this surface back to the quad element type and re-generate the mesh. The finished mesh should look like this.



Step 3: Correcting element connectivity

1. Use the Macro menu to turn the display of the geometry ***off*** and the elements ***on***.

   This is the equivalent of changing the display of the model using the ***display*** panel.

The fixed points on the surfaces corresponding to the last weld element were suppressed to simplify the meshing operations. Equivalencing nodes will not correct this because the distance between the components is less than the distance between the nodes that need to be equivalenced. The replace node function easily corrects the connectivity.

2. Zoom in to the weld in the most positive X location.

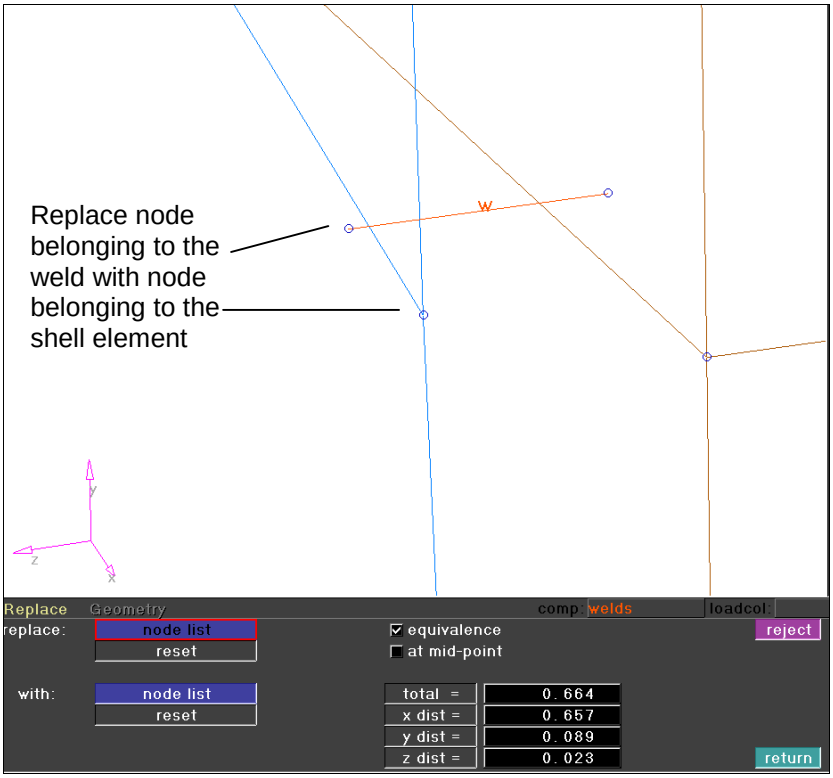
3. On the ***1-D, 2-D*** or ***3-D*** page, select the ***replace*** function.

4. Uncheck the ***at mid-point*** option.

5. Select one of the nodes on the weld, then the corresponding node on the plate element.

6. Do the same for the other end of the weld element, then repeat this at the opposite side of the backet, if required.

   The nodes are combined and the elements are connected.



Summary

In this exercise, we meshed the two previously defeatured components. To better manage the meshing process, each component was meshed individually. The mesh quality greatly improved on a majority of the surfaces using the mixed element type. We switched the surface on the front of the base component back to the quad mapping to better handle the transitions around the spotweld locations. Finally, the welds were redefined to better match the nodes on the surface mesh.

Section 2: HyperMesh Macros

Overview

Implementation of the ***Macro*** menu began with the HyperWorks 4.0 release of HyperMesh. You can create a customized user interface consisting of a combination of controls including buttons (which may have user-defined macros associated with them), button groups (function like radio buttons), and text.

HyperMesh macros allow you to combine several steps into a single mouse click. Routines you use often, such as saving the model file or repetitive tasks that you apply to your models can be converted into macros. Ultimately, the appropriate use of macros will save time.

This section introduces the basic concepts of the HyperMesh macro functionality and procedures for creating and implementing macros.

The following topics are included:

- What is a HyperMesh Macro
- Default Macro menu
- Associated Files (filenames and functions)
- HyperMesh ***options*** panel
- New HyperMesh Macro Commands
- Creating Basic HyperMesh Macros
  - Save HyperMesh file macro
  - Reverse video JPEG macro
- HyperMesh Tcl commands
- Creating HyperMesh macros using Tcl
  - Modified save HyperMesh file macro
  - Modified bad Jacobian check macro

What is a HyperMesh Macro

A HyperMesh macro is similar to a user-defined script or executable that can be used to automate a HyperMesh process or execute a series of steps semi-automatically. Macros are interpreted. This means each command completes in the order in which it appears in the macro. The HyperMesh macro language is an extension of the HyperMesh command language. You can write simple macros that will run off of a HyperMesh command file, embed additional logic, and allow, by adding Tcl/Tk scripting commands, user interaction.

The HyperMesh Macro language allows you to customize the macro interface by creating controls in the form of buttons and button groups. For each control you specify its characteristics. You specify

- the macro page it displays on
- its label
- its location and size
- its help message
- the macro it calls, with any optional arguments

Macros therefore have two parts, the definition of the button that activates the macro and the instructions the macro performs.

While macros offer a great deal of flexibility, remember that once a macro is executed, there is no way to cancel the execution or reject the results. In addition, a macro may not call itself.

The Macro Menu

The Macro menu, as shipped with HyperMesh, displays on the right side of the graphics region when HyperMesh starts. The macro menu divides into four areas - pages, display options, shortcut buttons, and tool buttons.

The Macro menu may be turned on and off from the ***options*** panel ***menu config*** sub-panel.

A brief description of each page and its respective macros follows.

Page

The ***Page*** selection buttons are at the bottom of the Macro menu. The four default pages are ***Geom***, ***Mesh***, ***QA***, and ***User1***. Each page contains different shortcuts and tool macros. Click the button for the macro page you desire.

Geom (macro page 1) contains macros for geometry functionality.

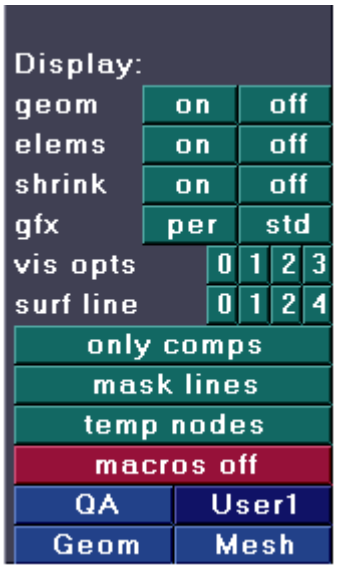
Mesh (macro page 2) contains macros used for meshing operations.

QA (macro page 3) handles element quality assurance functions.

User1 (macro page 4) contains space for user-defined shortcuts and tools.

Display

The ***Display*** buttons are located above the page selection buttons and are available on each macro page. They allow you to modify the graphics display in several different ways:



geom Turns off or on all of the geometry in the model.

elems Turns off or on all of the elements in the model.

shrink Reduces the size of the elements in the model by 20%.

gfx Displays the model either in performance or standard graphics mode.

vis opts Selects the topology visualization mode for displaying the model. Four modes are available:

- ***0*** Standard mode, the mode most commonly used.
- ***1*** Component color, the model is always displayed with the edges the same color as the associated component, even in the ***automesh*** panel.
- ***2*** Topology mode, the surface edges are displayed according to connectivity, as in the ***Geom cleanup*** panel.

3 Shaded mode, allows you to view the model in shaded mode regardless of which panel you are currently using.

surf line Places surface lines on your model. You can place one, two, four, or no lines on each surface.

only comps Turns off every type of collector except component collectors.

mask lines Masks all of the lines in the model that are displayed.

temp nodes Removes all of the displayed temporary nodes.

macros off Turns off the Macro menu.

Shortcuts

The ***Shortcuts*** buttons open the panel labeled on the button. They operate in the same manner as function and shift keys. The selection of shortcut buttons varies from page to page.

Tool

Each macro page has its own grouping of tools. ***Tool*** buttons allow you to perform functions quickly that would normally take several steps. Behind each tool button is a default macro.

On the **Geom** page there are three tools:

isolate surf Allows you to isolate either an inner or an outer surface layer from a 3-D model. This macro works only on the surfaces attached to the selected surface. The other layer and thickness are then placed in a temp directory and masked.

washer Scale a circular line 1.5 times and then trim that new line into the surface. This will allow for a better mesh quality around circular holes.

adj circ pts Places three additional fixed points on an inner line, and then projects those points to a concentric line. This allows you to create a higher quality mesh.

On the ***QA*** page there are twelve tools to help you cleanup a pre-existing mesh quickly. There are eight tools to isolate any elements that fail certain element check criteria. The macro displays only those elements that fail. The values can be changed in the `hm.mac` file but are preset to:

Length	5.0
Jacobian	0.5
Warpage	20.0
Aspect Ratio	5.0
Max Angle Quad	150.0
Max Angle Tria	140.0
Min Angle Quad	20.0
Min Angle Tria	10.0

You can then use the next four macros to quickly modify any elements that fail the element checks.

Find attached Finds all of the elements attached to the displayed elements.

Find between Finds the nodes that are shared between two components.

Remesh Allows you to remesh the selected elements plus one, two, or three attached layers of elements. The remesh uses the current size, does not break connectivity, and uses the mixed element type.

Smooth Allows you to apply the smoothing algorithm to the selected elements plus one, two, or three attached layers of elements.

On the ***Mesh*** page there are seven shortcuts and four macros.

elems by surf	Deletes elements associated to a selected surface.
split warped	Processes the entire model and splits all quad elements with a warpage greater than 20 into trias along the diagonal of the quad.
Remesh	Remeshes the selected elements plus one, two, or three attached layers of elements. The remesh uses the current size, does not break connectivity, and uses the mixed element type.
Smooth	Applies the smoothing algorithm to the selected elements plus one, two, or three attached layers of elements.

Files Associated with HyperMesh Macros

The macros have files associated with them. You will find it helpful to understand the functions of these files while working with and creating macros.

hm.cfg

The display of the Macro menu is controlled by a command in the HyperMesh Configuration file, **`hm.cfg`**. By default, the Macro menu displays when HyperMesh starts. If you want to change the default and remove the Macro menu from the screen display, delete the asterix (*) before the `*enablemacromenu()` command.

hm.mac

This file defines the ***Macro*** menu. Its commands control the display and available operations of the Macro menu.

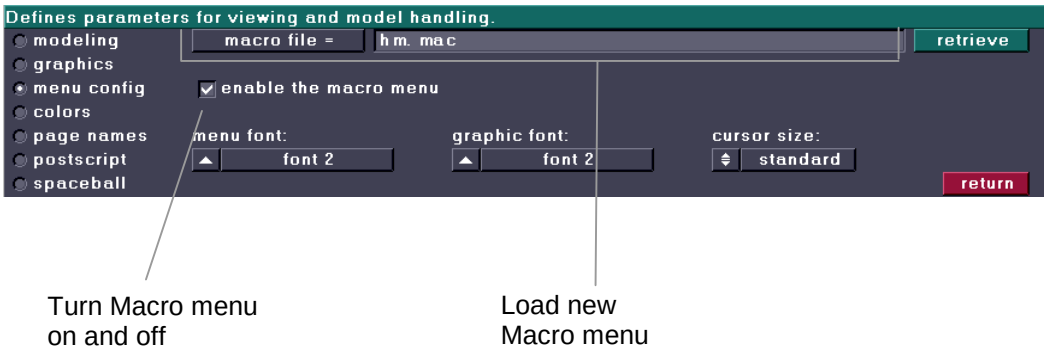
If **`hm.mac`** exists in the current HOME (UNIX only) or in the application's base directory when HyperMesh starts, it automatically defines the attributes and contents of the ***Macro*** menu.

mac.cmf

The **`mac.cmf`** file holds the commands of the last macro executed.

HyperMesh *options* Panel

The HyperMesh ***Options*** panel contains the ***menu config*** sub-panel. This sub-panel allows you to turn the Macro menu on and off as well as load your own Macro menu file into HyperMesh.



HyperMesh Macro Commands

HyperMesh macro commands are located in the default `hm.mac` file. When you open this file, you will see that it defines the macro pages in order by page. To change a macro on a given page, find that page in the file and the macro name that you wish to modify.

Attributes of the macro page that you can change include:

- which macro page displays the menu
- the text displayed on the control
- the location and size of the menu buttons
- the help string displayed on the menu bar
- the macro called, with optional arguments to pass.

Macros consist of valid command file or templex commands, and are enclosed by the **beginmacro()* and **endmacro()* commands. Macros may accept variable arguments, passed to them from a control, by using the arguments $1, $2, etc. to

specify where the arguments should be substituted. The following is the skeleton format of a macro.

```
*beginmacro()
    macro command statements go here
*endmacro()
```

To activate the macro from HyperMesh, you must create a button on the Macro menu that invokes the macro. Use the `*createbutton()` command to define the button and its characteristics. The syntax for this command is:

```
*createbutton(page, name, row, column, width, COLOR,
  helpString, macroName [ , arg1 … ])
```

Where:

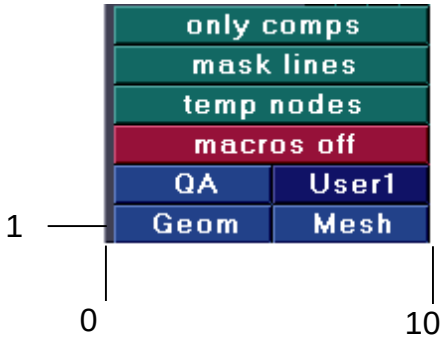
`page` Indicates the page number on which the button is to appear (values 1 through n; initially there are 4).

`name` The text to display on the button. Enclose the text with quotes (“).

`row` The row in which to place the button (value 1- n)..

`column` The column where the button starts (values 0 - 10).

`width` The width of the button (max 10).



`COLOR` The color of the button. The available button colors are: RED, BLUE, GREEN, CYAN, BUTTON, and MAGENTA. The color name must appear in capital letters.

`helpString` The string to be displayed in the menu bar when the button is selected. Enclose text of string in quotes (“).

`macroName` The name of the macro to call when the button is selected. Enclose text of string in quotes (“).

`arg1…` Optional arguments to pass to the macro. You may have as many arguments as your computer’s memory will allow.

Exercise 1: Creating a button

This is the first step in the creation of a macro. You create and position the button that activates your macro. For this class, we create everything to appear on the ***User1*** page (macro page 4).

1. Copy the ***hm.mac*** file from the ***../hm/bin*** directory into the working directory, the directory from where HyperMesh starts. Next change the permissions of the local ***hm.mac*** file to `writeable`.

2. Open the ***hm.mac*** file from the working directory in a text editor and go to the bottom of the file. This takes you to page 4, the ***User1*** page.

3. Add the command:

```
*createbutton(4,"Test",20,5,5,MAGENTA,"This is a test
  button for experience","macroSample")
```

   To refresh your memory, here is the format for the `*createbutton` command and its arguments:

```
*createbutton(page, name, row, column, width, COLOR,
  helpString, macroName [ , arg1 … ])
```

The end of a command is a hard return (press the ENTER key). In your word processing package, type each command until you get to the end of it, then press ENTER. If you get word-wrap, that is fine, as the macro processor will recognize only the hard return.

4. Save the modified ***hm.mac*** file.

5. Load your session of HyperMesh from the working directory. This will allow the new session to use the modified ***hm.mac*** file.

6. Press the ***User1*** button on the Macro menu. You will see the button, ***Test***, we defined in Step 3. Compare this button to our definition. It is magenta in color, begins in the middle of the row, and only goes half way across the Macro menu.

There is no functionality behind the button at this time because the macro has not been written. All that has happened is that the button to activate the macro now appears on the ***User1*** page.

This is a partial list of commands available to you for use in the construction of your macros. See Appendix A for the command arguments.

Command Name	Description
*appendmark()	Add additional entities to a mark.
*beginmacro(*name)*	Start the definition of a Macro menu macro.
*callmacro()	Call a macro from within another macro.
*createbutton()	Add a button to the Macro menu.
*createbuttongroup()	Add a button group to the Macro menu.
*createlistpanel()	Allow you to interactively select a list of entities.
*createmarklast()	Place the entities from the last operation into a mark.
*createmarkpanel()	Allow you to interactively select entities.
*createtext()	Add text to the Macro menu.
*endmacro()	End the definition of a macro started with the *beginmacro* command.
*enterpanel()	Enter the passed HyperMesh panel.
*includemacrofile()	Include the contents of the passed macro file.
insertbutton()*	Insert a button into the Macro menu.
*inserttext()	Insert text into the Macro menu.
*setactivegroup()	Set the default selection for a button group.
*setactivepage()	Set the current displayed Macro menu page.
*setbuttongroupactivecolor()	Set the color for the Macro menu button groups current selection indicator.

Process for Creating Basic HyperMesh Macros

To assist in your creation of HyperMesh macros, we suggest the following process.

1. Define the task.

2. Run through the process in HyperMesh.

3. Add macro button commands.

4. Extract the commands from the ***command.cmf*** file.

5. Add them to the **hm.mac** file.

6. Modify necessary HyperMesh commands.

7. Add macro wrapper commands.

8. Load the modified **hm.mac** file into HyperMesh using ***options*** panel.

Exercise 2: A macro to save the model

The first step in creating a macro is to know the process you want to automate. In this exercise, we want to create a one-button macro to automate saving the current HyperMesh model to a file named `temp.hm`. The actions necessary to complete this task are

- Leave the current panel.
- Enter the ***files*** panel.
- Enter the ***hm file*** sub-panel.
- Use the ***file=*** file browser to locate a directory and filename.
- Click ***save***.

This macro automates these actions.

Step 1: Run through the process in HyperMesh

We now execute the full process within HyperMesh. Every command issued in HyperMesh appears, in the order executed, in the ***command.cmf*** file. To see only the commands you are interested in placing in your macro, delete the current ***command.cmf*** file, then execute the commands necessary to complete the process.

1. Locate the ***command.cmf*** file. The file is in the specified working directory, which is the directory you started HyperMesh in, or in the ***`/altair/hm/4.0/bin/`*** directory.

2. Delete the ***command.cmf*** file. (As soon as you begin working in HyperMesh all executed commands are written to the ***command.cmf*** file. If the file already exists, the commands are appended to the file.)

3. From any page in HyperMesh, select the ***files*** panel.

4. In the ***file =*** field, enter the filename `temp.hm` and select ***save***.

Step 2: Edit the `hm.mac` file

Create an activation button in the Macro menu to execute the macro. Our button will be placed on page 4 (***User1***) of the Macro menu as that page is dedicated to user-defined macros. We will be using page 4 to hold our macro buttons throughout this training. Also please refer to Appendix A for the arguments that the *createbutton command requires.

1. Open the ***hm.mac*** file.

2. At the bottom of the file, change the `*createbutton` command created in the previous exercise to look like this.

```
*createbutton(4,"file save",20,0,10,GREEN,"Save the
   file","macroSave")
```

Step 3: Extract the commands from the *command.cmf* file

This is the command that writes the model file.

1. Open the ***command.cmf*** file using any text editor.

2. Locate the `*writefile()` command at or near the end of the ***command.cmf*** file. You should see the following.

   ```
   *writefile("temp.hm",0)
   ```

   Select and copy this line.

3. On the line following the `*createbutton()` command in the ***hm.mac*** file, paste the `*writefile()` command copied from the ***command.cmf*** file.

Step 4: Create the macro

1. Insert the following commands following the `*createbutton` command, enclosing the commands from the command file between the `*beginmacro()` and `*endmacro()` lines.

macroName

```
*beginmacro(macroSave)
   *writefile("temp.hm",0)
   *answer(yes)
*endmacro()
```

The macro name connects the button with the macro via the **`macroName`** field in the createbutton command.

The command `*answer(yes)` suppresses the overwrite prompt in the event `temp.hm` already exists.

☞	All commands can be found in the Commands Section of the HyperMesh on-line help. Argument definitions and examples are included.

2. Save the `hm.mac` file.

Step 5: Load the new `hm.mac` file into HyperMesh

1. Return to the HyperMesh session, go to the ***options*** panel and select the ***menu config*** sub-panel.

2. Click on the ***macro file =*** button and locate the new `hm.mac` file and click ***retrieve*** to load the updated Macro menu.

☞	Anytime a change is made to the Macro menu it must be retrieved through this panel to view or test the changes.

3. Press ***return***.

4. Click on the ***User1*** button on the Macro menu.

   A new button should be on the menu. The new button should be titled ***"file save"***.

5. Click this button to automatically save your file.

   The file is saved to the directory specified in the macro. In this case no directory was specified so HyperMesh saves the file to the start-up or working directory. It will also have the naming convention used in the macro.

Summary

In this exercise, we created a macro to simplify the process of saving a HyperMesh file. First we defined the process and executed it in HyperMesh. Next, we created a button in the macro file, then created the command file commands, including the appropriate heading and footer and added them to the `hm.mac` file. Once we loaded the macro file into HyperMesh, we tested it.

Exercise 3: A macro to create a reverse video JPEG

In this exercise, we create a macro that will automate the following into one button click.

- Set the background color to white
- Set the mesh line color to black
- Generate a JPEG image file
- Return the colors back to default

Step 1: Run through the process in HyperMesh

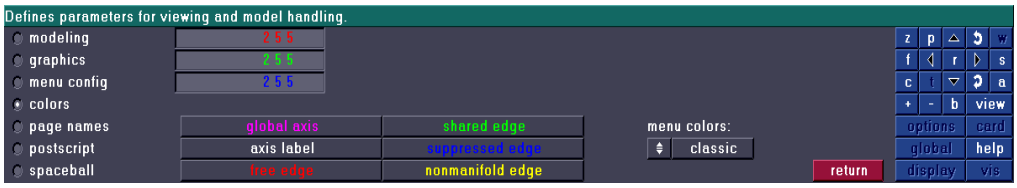
1. Start HyperMesh and load the `turbine_engine.hm` file.

2. Set the graphics mode to ***performance*** graphics by clicking on the ***per*** button next to the ***gfx*** label in the Macro menu.

3. Enter the ***options*** panel.

4. Select the ***colors*** sub-panel.

5. Set each of the three color values to `255`. This turns the background white.



6. Enter the ***vis*** panel using the blue permanent menu.

7. Set the ***mesh color*** to `color 2`.

8. Click ***return*** to leave the panel.

9. Press and hold down the ***<CTRL>*** key and then click the ***<F6>*** key. This captures the screen image in a JPEG image file that will be saved in either the working directory or the `/altair/hm/4.0/bin/` directory.

Step 2: Edit the `hm.mac` file

1. Open the `hm.mac` file from the working directory in a text editor and go to the bottom of the file.

2. Insert this command to create the button.

```
*createbutton(4,"JPEG",19,0,10,GREEN,"Create a JPEG w/
   reverse video",macroJpeg)
```

Step 3: Extract the commands from the command file

1. Open the new ***command.cmf*** file in a text editor.

2. Find the three commands near the bottom which were used to change the background color, the mesh line color and create the JPEG. They are:

```
*setbackgroundcolor(255,255,255)
*setmeshlinecolor(2)
*jpegfile()
```

3. Highlight and copy these commands.

Step 4: Create the macro

1. Paste the copied commands at the end of the `hm.mac` file.

2. Change the `2` in the `*setmeshlinecolor()` command to `20`.

   HyperMesh interprets color 20 as black.

3. Surround the commands extracted from the `command.cmf` file with the `*beginmacro()` and `*endmacro()` commands as follows:

```
*beginmacro(macroJpeg)
   *setbackgroundcolor(255,255,255)
   *setmeshlinecolor(20)
   *jpegfile()
*endmacro()
```

4. Save the `hm.mac` file.

Every macro written needs a unique button for activation. To avoid overlapping, each button must have a unique row number on each page.

Step 5: Load the new *hm.mac* file into HyperMesh

1. Return to the HyperMesh session and enter the ***options*** panel.

2. Select the ***menu config*** sub-panel.

3. Click on the ***macro file =*** button.

4. Locate the new `hm.mac` file.

5. Click ***retrieve*** to load your new macro file.

6. Press ***return*** to exit this panel.

Step 6: Use the JPEG macro

1. Select the ***User 1*** button from the Macro menu.

2. The new button named ***JPEG*** is visible on the Macro menu.

3. Select this button.

A JPEG file is created. Check the message on the header bar to verify the JPEG file's creation. The message also shows the name and location of the JPEG file. The background color remains white and the mesh line color is black. You can add two commands to the macro to change the colors back to the default to avoid repeating the procedure.

Step 7: Edit JPEG macro to return the colors to default after the JPEG file is created

1. Open the `hm.mac` file and go to the end of the file.

2. After the `*jpegfile()` command and before the `*endmacro()` command, insert these commands.

```
*setbackgroundcolor(0,0,0)
*setmeshlinecolor(0)
```

   The three numbers correlate to the red, green and blue levels; 255 is the maximum and 0 is no color.

3. Save the file and reload the macro file using the ***options*** panel.

4. Create the new JPEG and open the file up to see what was created.

A process that takes 10 or more button clicks is now automated into one button click.

Summary

In this exercise, we created a macro to automate a process that normally requires several button clicks on different pages. First, we defined the process and executed it in HyperMesh. Next, we edited the macro file to create a new button. The commands written to the ***command.cmf*** file were extracted into the macro file, and the macro was created. After loading and testing the macro, we made a modification to reset the original background and mesh line colors.

Creating HyperMesh Macros Using Tcl

Also included with the HyperWorks 4.0 release of HyperMesh is Tcl/Tk v8.2.3. Tcl (Tool Control Language) is an interpreted scripting language. Tk (Tool Kit) contains widgets that can be incorporated into Tcl to build graphical user interfaces. Tcl/Tk allows you to develop custom scripted applications, including user-defined panels, and the ability to add logic to HyperMesh command files.

Virtually all of the functionality of HyperMesh's command file replay system is available through Tcl. Additional commands allow you to extract information from the HyperMesh database, as in the entity id numbers on a mark, a list of assemblies, component names, elements per component, nodes per element, node values, and so forth. There is also a command to allow you to access HyperMesh using the template system.

These commands can be used within Tcl/Tk to retrieve information from the HyperMesh database. You are allowed to add other variables to the macro and have their values stored for future use. See Appendix B for arguments.

Command	Usage

hm_answernext	Used to force an answer for the following star command, as in
	
	`*answernext "yes"`
	
	`*deletemodel`

hm_complist	Return a list of component ids for the current model.
hm_elemlist	Return a list of element ids for the passed component id.
hm_errormessage	Display an error message on the HyperMesh menu bar.
hm_getentityvalue	Return a value by using the HyperMesh template system interface.
hm_getfilename	Returns a user input filename, by posting the file selection panel in the HyperMesh menu area.
hm_getfloat	Returns a user input floating point value, by posting a panel in the HyperMesh menu area.

Command	Usage
hm_getint	Returns a user input integer value, by posting a panel in the HyperMesh menu area.
hm_getmark	Return a list of ids for the passed entity type and mark mask.
hm_getstring	Returns a user input string, by posting a panel in the HyperMesh menu area.
hm_markclear	Clear the mark for the passed entity type and mark mask.
hm_nodelist	Return a list of node ids for the passed element id.
hm_nodevalue	Return a list containing the passed node ids XYZ values.
hm_usermessage	Display a message on the HyperMesh menu bar.

Syntax of Macros vs Tcl/Tk

Compare the macro and its Tcl script equivalent. Note particularly the lack of punctuation in Tcl. Parentheses, commas, and quotes will be misinterpreted by Tcl if you use them. Otherwise the macro statement in Tcl appears very much like it does in the macro process we discussed above.

Macro in `hm.mac` file	Tcl script

`*beginmacro("macroJpeg")`
`*setbackgroundcolor(255,255,255)`
`*setmeshlinecolor(20)`
`*jpegfile()`
`*setbackgroundcolor(0,0,0)`
`*setmeshlinecolor(0)`
`*endmacro()`
	`Not used in script`
	`*setbackgroundcolor 255 255 255`
	`*setmeshlinecolor 20`
	`*jpegfile`
	`*setbackgroundcolor 0 0 0`
	`*setmeshlinecolor`
	`not used in script`

Creating Macros using Tcl/Tk

You follow the same macro creation process discussed previously. Create a button to activate the macro, then create the macro using Tcl/Tk commands.

Tcl/Tk Help

The HyperWorks installation includes an on-line Tcl/Tk reference guide. To initiate the help manual, execute the file ***tcl82.hlp*** from the directory `/Altair/tcl/tcl8.2.3/doc/`

Tcl/Tk links:

http://dev.scriptics.com/software/tcltk/
http://www.tclfaq.wservice.com/tcl-faq/
http://www.tcltk.com/main.html

The Tcl/Tk news group: comp.lang.tcl

Recommended books:

Tcl/Tk in 24 hours

Tcl/Tk Programmer's Reference Guide

Exercise 4: Create a macro to invoke a Tcl script

1. Edit the `hm.mac` file and go to the bottom of the file.

2. Create a button in the Macro menu to call the Tcl script for the next macro.

```
*createbutton(4,"file sv2",17,0,10,GREEN,"Save the
   file","macroFile")
```

   The Tcl script name, in this case `macroFile`, is stored within the button command.

3. Next create the macro.

```
*beginmacro(macroFile)
   *evaltclscript("save.tcl",0)
   *answer(yes)
*endmacro()
```

4. Save the `hm.mac` file

The command to call a Tcl script is `evaltclscript("scriptname",0)`. Any other commands that you want to execute can be added before or after this command and they will be executed in the order given. Notice in this case that the `*answer(yes)` command is given within the macro to suppress the confirmation prompt. Please see Appendix B for further information.

Exercise 5: Create a macro to save a file to a user-specified directory and filename

The macros we previously created are very powerful in their automation of repetitive tasks. However, there are limitations as to what they can do. For example, in the Save File macro, if we wanted to save the model to another filename, we must edit the `hm.mac` file. By integrating Tcl/Tk with the macro, you can increase its flexibility by interactively changing the filename.

This exercise creates an interactive macro that automates saving the current HyperMesh model to a user-specified directory and filename.

Step 1: Setup and create the Tcl script

1. Edit the `hm.mac` file.

2. Go to the bottom of the file, to the **`macroSave`** macro.

3. Open a text editor session (in vi name the file `save.tcl`).

4. Copy the **`macroSave`** macro into an empty session of the text editor.

5. Save the new file in the working directory as `save.tcl`

	Tcl macros can be saved in either the working directory or the `../bin/scripts` directory. This is true for Windows and UNIX workstations. However with a UNIX workstation, two rules must be followed. First, set the environment variable TCL_INCLUDE as follows: TCL_INCLUDE = “path to TCL scripts”. Second, the `hm.tcl` file, which can be located in the `../bin/scripts` directory must be in the working directory or the directory that TCL_INCLUDE points to.

6. Remove the `*beginmacro(macroSave )` line and the `*endmacro()` lines from the new macro.

7. Replace all of the parenthesis, quotes, and commas [`( ) ,”`] with spaces.

```
*writefile temp.hm 0
```

	Tcl recognizes the ASCII characters “( ) or , “and would interpret them incorrectly. Therefore remove them from the macro command lines when you put them into a Tcl script.

8. Set a variable called `filename` with the command `set filename`.

9. Assign the value to the variable `filename` using the `hm_getfilename` command. This will go at the top of the file.

10. The final macro will look like this.

```
set filename [ hm_getfilename “select or enter a
  filename” ]
*writefile $filename 0
```

`$filename` passes the value in the variable previously set in the command.

11. Save this file as `save.tcl` in the working directory.

Step 2: Load the new `hm.mac` file into HyperMesh and use the macro

1. Go back into the HyperMesh session and retrieve the `hm.mac` file using the ***options*** panel and ***menu config*** sub-panel.

2. Go to the ***User1*** page of the Macro menu. You should see a new button labeled ***file sv2***.

3. Select this button.

   A window pops up asking for a file name. You can either type in a filename or double click the file button to change directories or retrieve a previously used name.

4. Enter a name for the file and select return. The file is saved to your working directory with the chosen name.

Summary

In this exercise, we introduced powerful new functionality into the macros using Tcl. We modified the macro created in Exercise 2 to prompt for the name of the input file. Using the appropriate variable substitutions, the new filename was used in the command to save the file.

Exercise 6: A macro that checks the model for elements having a Jacobian below a user specified value

Create a macro that will check the Jacobian of the displayed elements. This macro is similar to the Jacobian macro from the default Macro menu, but is interactive by prompting for the Jacobian check value.

Step 1: Setup and create the macro

1. Load the `Jacobian_check.hm` file.

2. Open the `hm.mac` file.

3. Create a button to start the macro.

```
*createbutton(4,”Jacobian2”,16,0,10,GREEN,”Check
     Elements for Jacobian”,”macroJacob2”,”jacob.tcl”)
```

4. Search the `hm.mac` file for `macroElementJacobian`.

5. Copy the entire macro under the button and change the name in the *`beginmacro()`* command from `macroElementJacobian` to `macroJacob2`.

```
*beginmacro(macroJacob2)
   *createmark(elements,1) displayed
   *elementtestJacobian(elements,1,$1,2,2,0,"2D Element
     Jacobian")
   *displaycollectorwithfilter(components,"none","",1,0
     )
   *findmark(elements,2,0,0,undefined,0,2)
   *plot()
*endmacro()
```

6. Change `*createmark(elements,1)` to `*createmarkpanel(elements,1,”Select elements”)`.

7. After the new `*createmarkpanel` command, add the command to evaluate a Tcl script.

```
*evaltclscript($1,0)
```

8. Add the command `*clearmark (elements,1)` before the `*plot()` command.

9. Cut the `*elementtestJacobian` command from the `hm.mac` file and paste it into a new session of the text editor.

10. The macro in the `hm.mac` file should look like this.

```
*beginmacro(macroJacob2)
  *createmarkpanel(elements,1,”Select elements”)
  *evaltclscript($1,0)
  *displaycollectorwithfilter(components,"none","",1,0)
  *findmark(elements,2,0,0,undefined,0,2)
  *clearmark (elements,1)
  *plot()
*endmacro()
```

11. Save the `hm.mac` file.

Step 2: Create the Tcl script

Edit the `jacob.tcl` file in the following ways.

1. Replace all the parentheses and commas from the lines with spaces.

```
*elementtestjacobian elements 1 $1 2 2 0 "2D Element
  Jacobian"
```

2. Set a variable called `jacobval` with the command `set jacobval`, and then assign the value of the variable filename using the `hm_getfloat` command. This goes at the top of the file. Please refer to Appendix A for the required arguments for this command.

```
set jacobval [ hm_getfloat “Element Checking”
  “Jacobian <” ]
```

3. Substitute the `$jacobval` variable for the Jacobian test threshold (`$1`), then save the file as `jacob.tcl` in the working directory. The finished Tcl script looks like this.

```
set jacobval [ hm_getfloat "Element Checking"
   "Jacobian <" ]

*elementtestjacobian elements 1 $jacobval 2 2 0 "2D
   Element Jacobian"
```

Step 3: Load the new hm.mac file into HyperMesh and use the macro

1. Retrieve the `hm.mac` file in the ***options/menu config*** panel.

   This can be done before the Tcl file is written because HyperMesh only sees the command to read the script. Therefore the scripts can be modified over and over again with no need to reload the `hm.mac` file.

2. Enter the ***User1*** page through the Macro menu.

3. There will be a new button titled ***Jacobian2*****.** Select this button.

4. Select the elements ***by displayed*** and click the middle mouse button.

5. A new panel will appear asking for a value to be entered.

6. Enter `.5` in the field.

7. Click ***return.***

   Only the elements that fall below the minimum Jacobian criteria value display.

Summary

In this exercise, we modified an existing macro that uses the Jacobian test to check element quality to make it more powerful and flexible. We used Tcl used to introduce a user prompt, and that input became the threshold value in the Jacobian check.

Appendix A: HyperMesh Macro Menu Commands

This appendix presents a sample macro file plus additional commands available.

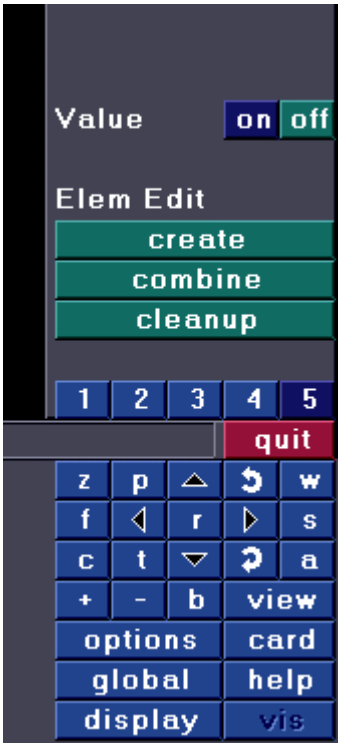
Macro File Example

This is a portion of the hm.mac file.

This section of the macro file defines the five buttons that appear above the ***quit*** button. Each button in turn calls a macro page of its own. The last command of this section declares page 5 to be the active page, and thus its controls appear above the five macro page buttons.

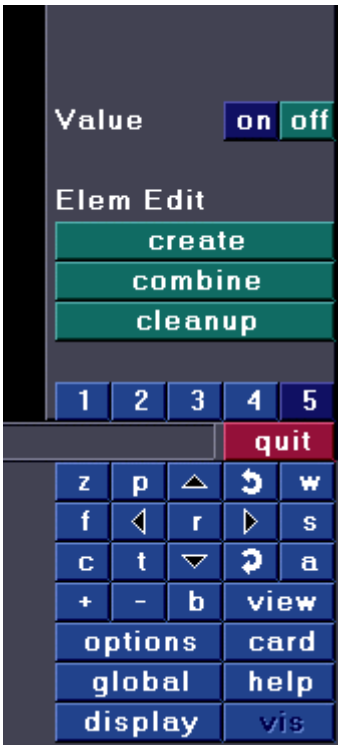
The creation of the controls for macro page 5 and their functionality appears on the next page.

```
#
# Permanant buttons, page 0 will always be
displayed
#
*createbuttongroup(0, 0, "1", 1, 0, 2, CYAN,
   "Page 1, Fixed Points", "SetActivePage",
   1)
*createbuttongroup(0, 0, "2", 1, 2, 2, CYAN,
   "Page 2, Edges", "SetActivePage", 2)
*createbuttongroup(0, 0, "3", 1, 4, 2, CYAN,
   "Page 3, Surfaces", "SetActivePage", 3)
*createbuttongroup(0, 0, "4", 1, 6, 2, CYAN,
   "Page 4, Mesh", "SetActivePage", 4)
*createbuttongroup(0, 0, "5", 1, 8, 2, CYAN,
   "Page 5, Element Editing", "SetActivePage",
   5)
*setbuttongroupactivecolor(BLUE)
*setactivegroup(0, 0, 5)
```



```
#
# Page 5
#
*createtext(5, "Value", 8, 0)
*createbuttongroup(5, 1, "on",  8, 6, 2,
   GREEN, "Enable the value", "Set", 1)
*createbuttongroup(5, 1, "off", 8, 8, 2,
   GREEN, "Disable the value", "Set", 0)
*createtext(5, "Elem Edit", 6, 0)
*createbutton(5, "create",  5, 0, 10, GREEN,
   "Create an element", "ReadBumper",
   "demos\bumper.hm")
*createbutton(5, "combine", 4, 0, 10, GREEN,
   "Combine elements", "Combine")
*createbutton(5, "cleanup", 3, 0, 10, GREEN,
   "Enter the Cad Cleanup panel",
   "EnterPanel", "geom  cleanup")
*setactivepage(5)

#
# Macros
#
*beginmacro("SetActivePage")
    *setactivepage($1)
*endmacro()
```




```
*beginmacro("ReadBumper")
    *readfile($1)
    *viewset( 0.155815, -0.837776, 0.523310, 0.000000,
              0.770903, -0.228101, -0.594708,0.000000,
              0.617600, 0.496086, 0.610303, 0.000000,
              -314.335103, 1025.278697, 371.660590, 1.000000,
              -104.720170, 134.274912, 749.720170, 672.065084)
    *window(3,396.256943,168.126312,640.688167,389.118473)
    *callmacro("EnterPanel") "geom cleanup"
*endmacro()

*beginmacro("EnterPanel")
    *enterpanel($1)
*endmacro()

*enablemacromenu()
```

Command List

The following pages contain an alphabetical list of available commands.

*appendmark()

Description Add additional entities to a mark.

Syntax `*appendmark(entityType, markId)`

`entityType` The type of the entity to be added to the mark.

`markId` The id(s) of the entity to be added to the mark.

Comments A list of the entity IDs and/or the collector names must follow the `*appendmark()` command. The list must include the IDs and/or the collector names of the entities, separated by a space or comma.

The `*appendmark()` command may also be followed by one of the options below:

```
all
displayed
retrieve
reverse
"by comps" [ id or name ]
"by id" id
"by assems" [ id or name ]
"by group" [ id or name ]
"by config" [ id or name ]
"by sets" [ id or name ]
"by surface" id
"by adjacent"
"by attached"
"by outputblock" [ id or name ]
"on plane" x y x i j k tol plane touching
    plane:    0 = plane, 1 = normal of plane
    touching: 0 = entities touching, 1 = entities on
```

Example Place elements 1 and 35 on mark 1, and then add all elements of config type 104 to it.

```
*createmark(elements,1) 1 35
*appendmark(elements,1) "by config" 104
```

*beginmacro()

Description Start the definition of a Macro menu macro.

Syntax `*beginmacro(name)`

`name` The name of the macro to create.

Comments A macro may contain any valid command file command.

The ***createbutton*** and ***createbuttongroup*** commands allow you to specify the macro to call when the button is selected. When optional arguments are specified for the button, you may use the variables $1, $2, etc to specify where the arguments should be substituted. When a macro expects variable arguments, and not enough arguments are supplied in the controls definition, an error message displays when the macro executes.

Example Define a macro that will read in the passed filename (`$1`), and set the view.

```
*createbutton(5,"input",5,0,10,GREEN,"Input","ReadFile","de
  mos\bumper.hm")
*beginmacro("ReadFile")
*readfile($1)
*viewset( 0.155815, -0.837776, 0.523310, 0.000000,
   0.770903, -0.228101, -0.594708, 0.000000,
   0.617600, 0.496086, 0.610303, 0.000000,
   -314.335103, 1025.278697, 371.660590, 1.000000,
   -104.720170, 134.274912, 749.720170, 672.065084)
*window(3,396.256943,168.126312,640.688167,389.118473)
*endmacro()
```

*callmacro()

Description Call a macro from within another macro.

Syntax `*callmacro(name) [ arg1, arg2, … argn ]`

`name` The name of the macro to call.

`arg1` Optional arguments to pass to the macro

Comments When no arguments are supplied, the arguments that were passed into the macro where the `callmacro` command was made will be passed to the named macro.

Example Define a macro that will read in the passed filename ($1), and set the view.

```
*beginmacro("ReadFile")
*readfile($1)
*viewset(0.155815, -0.837776, 0.523310, 0.000000,
         0.770903, -0.228101, -0.594708, 0.000000,
         0.617600, 0.496086, 0.610303, 0.000000,
         -314.335103, 1025.278697, 371.660590, 1.000000,
         -104.720170, 134.274912, 749.720170, 672.065084)

*window(3,396.256943,168.126312,640.688167,389.118473)
*callmacro("EnterPerfMode")
*endmacro()
```

*createbutton()

Description Add a button to the Macro menu.

Syntax

```
*createbutton(page, name, row, column, width,
   COLOR, helpString, macroName [ , arg1 … ])
```

`page`	The page number that the button is to appear on.
`name`	The text to display on the button.
`row`	The row to place the button on.
`column`	The column to place the button at (1 - 10)
`width`	The width of the button (max 10)
`COLOR`	The color of the button.
`helpString`	The string to be displayed in the menu bar when the button is depressed.
`macroName`	The name of the macro to call when the button is selected.
`arg1…`	Optional arguments to pass to the macro

Comments Specifying `page 0` causes the control to appear on all pages.

Rows in the Macro menu start with row one being just above the menu bar, and progresses upwards. Specifying a row of 0 causes the button to be placed on the next available row. Specifying a row as a negative value causes the button to be placed above the next available row, plus the whole value of the row argument. For example, if you passed row as –2, then the button will be placed 2 rows above the next free row. One exception to this is that if because of the column placement, the button would fit down into the previous row, and a text item was not on the row, it will fall down into it. This allows you to place multiple buttons on the previous row, as in a set of state buttons.

An error message displays if the control will cover an existing control.

If a macro was specified in a control, and the macro does not exist, an error message displays when the control is selected.

Button colors may be specified as one of the following:

BUTTON	GREEN	YELLOW
RED	CYAN	
BLUE (default)	MAGENTA	

Example The following statement creates a button on page 5 of the Macro menu.

```
*createbutton(5,"input",5,0,10,GREEN,"Create an
   element","ReadBumper")
```

The following statements create text and three buttons on 1 row above the next available row, with a text string above them, on macro page 4.

```
*createbutton(4, "UL", -1, 1, 3, BLUE, "light source upper
   left","SetLightSource", -1.0, 1.0, 1.0)

*createbutton(4, "UC",  0, 4, 3, BLUE, "light source upper
   center","SetLightSource",  0.0, 1.0, 1.0)

*createbutton(4, "UR",  0, 7, 3, BLUE, "light source upper
   right","SetLightSource",  1.0, 1.0, 1.0)

*createtext(4, "Light Source:", 0, 0);
```

*createbuttongroup()

Description Add a button group to the Macro menu.

Syntax

```
*createbuttongroup(page, group, name, row,
column, width, color, helpString, macroName [ ,
arg1 … ])
```

`page`	The page number that the button appears on.
`group`	The group id for the buttons.
`name`	The text to display on the button.
`row`	The row to place the button on.
`column`	The column to place the button at (1 - 10).
`width`	The width of the button (max 10).
`color`	The color of the button.
`helpString`	The string to be displayed in the menu bar when the button is depressed.
`macroName`	The name of the macro to call when the button is selected.
`arg1 …`	Optional arguments to pass to the macro.

Comments Specifying page 0 causes the control to appear on all pages.

Rows in the Macro menu start with row one, being just above the menu bar, and progresses upwards. Also, refer to the `createbutton` command for information on button placement on the next available row.

An error message displays if the control covers an existing control.

If a macro was specified in a control and the macro does not exist, an error message displays when the control is selected.

Button colors may be specified as one of the following:

BUTTON	GREEN	YELLOW
RED	CYAN	
BLUE (default)	MAGENTA	

Example The following creates a button group on page 3 of the Macro menu that allows you to toggle some setting.

```
*createtext(3,"Display",5,0)
*createbuttongroup(3,1,"yes",5,6,2,GREEN,"Yes","ToggleDisp
   lay",1)
*createbuttongroup(3,1,"no",5,8,2,GREEN,"No","ToggleDispla
   y",0)
```

*createlistpanel()

Description Allows you to interactively select a list of entities.

Syntax `*createlistpanel(entityType, markId, message)`

`entityType` The type of the entity to be added to the list.

`markId` The id(s) of the entity to be added to the list. (1 or 2)

`message` The message to display in the menu bar.

Comments Nodes, surfaces, and elements are the only entities which are currently supported on the `*createlistpanel` command.

Example Place the selected nodes on list 1.

```
*createlistpanel(nodes,1,"Select the ordered nodes")
```

*createmarklast()

Description Place the entities from the last operation into a mark.

Syntax `*createmarklast(entityType, markId)`

`entityType` The type of the entity to be added to the mark.

`markId` The id(s) of the entity to be added to the mark.

Example The following macro allows you to select elements, and smooth them, along with their two adjacent elements.

```
*beginmacro(macroReMeshElements2)
   *createmarkpanel(elements,1,"select an element")
   *appendmark(elements,1) "by adjacent"
   *appendmark(elements,1) "by adjacent"
   *surfacemode(1)
   *defaultremeshelems(1,24,2,2,1,1,1,1,3.42021977e-300,
      2.62637721e-308,3.3877743e-300,3.42021977e-300,0,30)
   *createmarklast(elements,1)
   *marksmoothelements(1,1,2,3)
   *clearmark(elements,1)
   *plot()
*endmacro()
```

*createmarkpanel()

Description Allows you to interactively select entities.

Syntax `*createmarkpanel(entityType, markId, message)`

`entityType`	The type of the entity to be added to the mark.
`markId`	The id(s) of the entity to be added to the mark. (1 or 2)
message	The message to display in the menu bar.

Example Place the selected elements on mark 1.

```
*createmarkpanel(elements,1,"Select the elements")
```

*createtext()

Description Add text to the Macro menu.

Syntax `*createtext(page, name, row, column)`

`page` The page number that the text is to appear on.

`name` The text to display.

`row` The row to place the text on.

`column` The column to place the text at (1 - 10)

Comments Specifying page 0 causes the text to appear on all pages.

Rows in the Macro menu start with row one being just above the menu bar, and progresses upwards. Also, refer to the `createbutton` command page for information on text placement on the next available row.

An error message displays if the control covers an existing control.

Example The following will create text on row 5 of page 3, starting at column 0 of the Macro menu.

```
*createtext(3, "Display", 5, 0)
```

*endmacro()

Description Ends the definition of a macro started with the `*beginmacro()` command.

Syntax `*endmacro()`

*enterpanel()

Description Enter the passed HyperMesh panel.

Syntax `*enterpanel(panelName)`

`panelName` The name of a HyperMesh panel to pass.

Comments When you use this command in a macro, it must be the last call made, other than mouse commands. All additional commands are skipped, and an error message displays.

`panelName` may be one of the following.

ruled	translate	reflect
lines	organize	edit element
distance	delete	project
replace	edges	rotate
create nodes	hidden line	optimization
numbers	color	count
contour	scale	detach
files	normals	check elems
systems	forces	length
mask	tetramesh	drag
solid map	velocities	accels
constraints	moments	pressures
welds	linear solid	safety
line edit	section cut	circles
springs	rigids	bars
rods	renumber	line drag
joints	temp nodes	mass calc
deformed	permute	summary
load types	titles	align node
features	legend edit	elem types
super elems	reparam	sensor
faces	vector plot	temperatures
flux	position	elem offset
interfaces	edit attrib	config selec
rename	xy plots	axes
fd blocks	data curves	transient
find	build menu	collectors
order change	tangents	fillets
assemblies	replay	solid mesh
masses	config edit	gaps
User Select	linear 1d	entity sets

intersect	cut planes	split
dependency	remap	card edit
elem cleanup	planes	spheres
cones	torus	spline
convert	smooth	geom cleanup
debug	defeature	automesh
skin	spin	title edit
surface edit	node edit	surf lines
plane	apply result	cntl cards
rigid walls	line mesh	reorder
output block	optimization	topography
objective	Shape DV	load steps
rbe3	hidden line	contour
animate	rigid body	vector plot
ellipsoid	id offsets	HyperBeam
beam xsect	composites	vectors
temp_topo	equations	line mesh
penetration	solver	fatigue
dummy	seatbelt	topology
spotweld	airbag	Size DV

Example `*enterpanel("geom cleanup")`

*includemacrofile()

Description Include the contents of the passed macro file.

Syntax `*includemacrofile(filename)`

`filename` The name of the macro file to include.

Comments This command allows you to specify the name of a macro file to be read in. When this command is used along with the new empty row arguments to the `createbutton` and `createtext` commands, it allows you to create macros that may be plugged into the Macro menu.

*nextmacrofile()

Description Read in the next macro file from the history stack.

Syntax `*nextmacrofile()`

Comments The Macro menu maintains a stack of the names of the macro files sourced in from the HyperMesh ***menu config*** panel. This command, along with the `prevmacrofile` command, allows you to move up and down the macro filename stack.

Example See the `*pushmacrofile()` command example.

*prevmacrofile()

Description Read in the previous macro file from the history stack.

Syntax `*prevmacrofile()`

Comments The Macro menu maintains a stack of the names of the macro files sourced in from the HyperMesh ***menu config*** panel. This command, along with the *`nextmacrofile`() command, allows you to move up and down the macro filename stack.

Example See the `*pushmacrofile()` command example.

*pushmacrofile()

Description Push a filename onto the macro file history stack.

Syntax `*pushmacrofile(filename)`

`filename` The filename to push onto the Macro menu history stack.

Comments The Macro menu maintains a stack of the names of the macro files sourced in from the HyperMesh ***menu config*** panel. This command allows you to push filenames onto the history stack, and by using the `*prevmacrofile()` and `*nextmacrofile()` commands, move up and down the macro filename stack.

Example The following creates the `next` and `prev` buttons, and push two additional macro filenames onto the history stack.

```
*insertbutton(0,"prev",1,0,5,YELLOW,"prev macro
    file","prevMacroFile")
*insertbutton(0,"next",1,5,5,YELLOW, "next macro
    file","nextMacroFile")
*beginmacro("prevMacroFile")
    *prevmacrofile()
*endmacro()
*beginmacro("nextMacroFile")
    *nextmacrofile()
*endmacro()
*pushmacrofile("metalform.mac")
*pushmacrofile("optimize.mac")
```

*setactivegroup()

Description Sets the default selection for a button group.

Syntax `*setactivegroup(page, group, index)`

`page` The page number that the button group appears on.

`group` The group id for the buttons.

`index` The button number in the group (1 – n)

Example Set the third button in button group 0, on page 0 as active.

```
*setactivegroup(0, 0, 3)
```

*setactivepage()

Description Sets the current displayed Macro menu page.

Syntax

```
*setactivepage(page)
```

`page` The page number that the button group appears on. The default = 1.

Example Display the third page of the Macro menu.

```
*setactivepage(3)
```

*setbuttongroupactivecolor()

Description Sets the color for the Macro menu button groups' current selection indicator.

Syntax `*setbuttongroupactivecolor(color)`

`color` The buttons' color (see below)

Comments Button colors may be specified as one of the following:

```
BUTTON
RED
BLUE ( Default )
GREEN
CYAN
MAGENTA
YELLOW (HM 5.0)
```

Appendix B: HyperMesh Tcl/Tk Interface

Overview

Tcl (Tool Command Language) has a simple and programmable syntax and can be either used as a standalone application or embedded in application programs.

Tk (Tool Kit) is a graphical user interface toolkit that makes it possible to create powerful GUIs quickly.

The Tcl/Tk v8.2.3 scripting language is embedded into the HyperWorks 4.0 release of HyperMesh to provide users the ability to develop custom scripted applications, including user-defined panels, and the ability to add logic to HyperMesh macro functions.

As an example of this, the following lines of code will ask you to select a node, and if any were selected, print out each nodes id and XYZ values.

```
*createmarkpanel nodes 1 "select node(s)"
set nodeList [ hm_getmark nodes 1 ];
if { ! [ Null nodeList ] } {
      foreach nodeId $nodeList {
            set nodeX [ hm_getentityvalue NODES $nodeId "x" 0 ];
            set nodeY [ hm_getentityvalue NODES $nodeId "y" 0 ];
            set nodeZ [ hm_getentityvalue NODES $nodeId "z" 0 ];
            tk_messageBox -message "Node at id: $nodeId  is $nodeX
$nodeY $nodeZ";
      }
}
*clearmark nodes 1
```

Executing Tcl/Tk Scripts

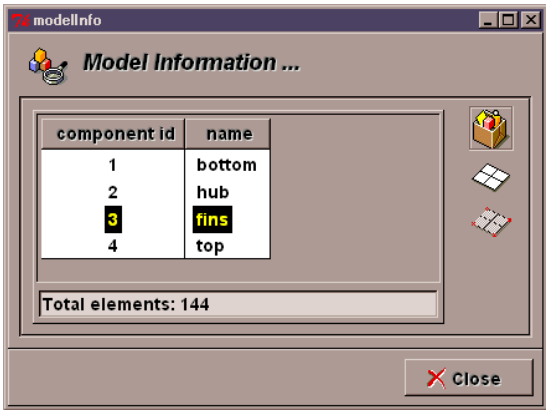
Tcl/Tk commands may be accessed either by creating a Macro menu button to call the Tcl/Tk script, or by running a command file. The ***Model Information*** panel shown below, may be accessed by adding the following lines to the `hm.mac` file to create a button on the ***User1*** page (4). Tcl/Tk script files may be

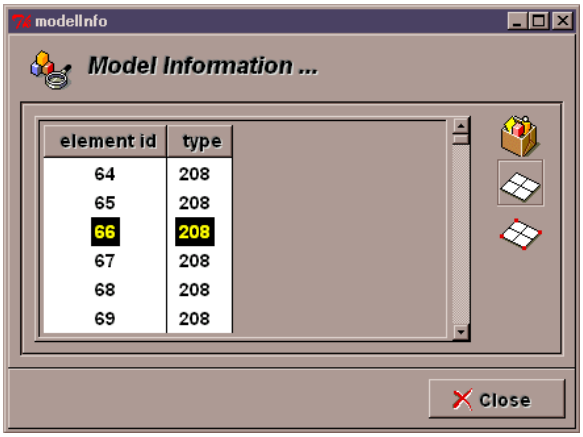
stored in a number of locations. The default scripts directory is`{altair home dir}/hm/{hmver}/scripts`. However, you may also set the environment variable *TCL_INCLUDE* to include a list of pathnames to search for your scripts prior to looking in the default location.

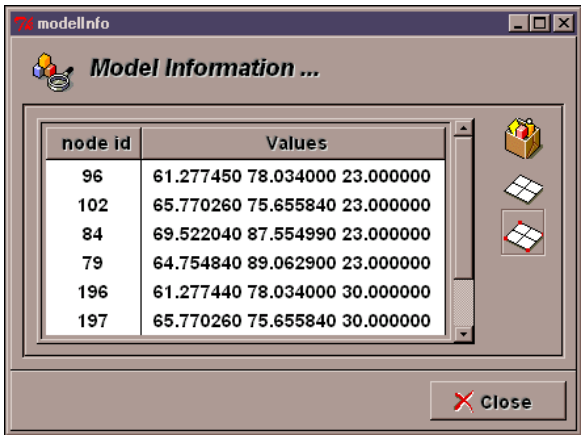
```
*createbutton(4,"model nfo",7,0,10,GREEN,"Display
    model info","EvalTcl","controls.tcl")
*beginmacro("EvalTcl")
     *evaltclscript($1,0)
*endmacro()
```

GUI Development

Through Tcl/Tk you may also generate panels to allow you to gather information before processing. The panels displayed below are used to browse the HyperMesh database for components, elements, and nodal information. Selecting the component name will display the number of elements for that component in the message area. By either double clicking on a component name, or selecting the component name and clicking the ***elements*** icon, a list of elements for the selected component to be displayed. You may continue this process to also generate a list of the nodes for the selected element.






modelInfo
Model Information ...
node id
Values
96
61.277450 78.034000 23.000000
102
65.770260 75.655840 23.000000
84
69.522040 87.554990 23.000000
79
64.754840 89.062900 23.000000
196
61.277440 78.034000 30.000000
197
65.770260 75.655840 30.000000
Close

Additional HyperMesh Commands

The following table lists the additional HyperMesh commands that allow Tcl/Tk users to access the HyperMesh database. Requests for additional functionality should be directed to Altair Support.

hm_answernext

Description To force a yes/no answer to the next HyperMesh command.

Syntax hm_answernext *value*

where

value "yes" or "no".

Example Force HyperMesh to clear the database before reading in a model.

```
set filePath [ hm_getfilename "Select the filename to
   input" ]

if { ! [ Null filePath ] } {
      if { [ file exists $filePath ] } {
         hm_answernext "yes"
         *deletemodel
         *readfile "$filePath"
      } else {
         hm_errormessage "The entered filename does not
   exist!" 2
      }
}
```

Notes This command works in the same manor as the *answer() command in HyperMesh, which is not available in the Tcl implementation, but it must be be issued before the command you want to force the answer for.

hm_complist

Description Get a list of component names or ids for the current model.

Syntax **hm_complist** *listType*

where

listType is either *name* or *id*. Determines the type of list to generate.

Example Find an id for the named component in *compName*.

```
set compNames [ hm_complist name ];
set compIds   [ hm_complist id ];

foreach name $compNames {
set loc [ lsearch -exact $listNames $compName ];

if { $name != -1 } {
return [ lindex $compIds $loc ];
}
}

return -1;
```

hm_elemlist

Description Get a list of element ids or types for the passed component.

Syntax hm_elemlist *listType compName*

where

listType Type of list to generate. *id* or *type*

compName The name of the component containing the elements.

Example Generate a list of elements ids of type *elemType* for a named component.

```
set elemIds   [ hm_elemlist id $compName ];
set elemTypes [ hm_elemlist type $compName ];

set numIds [ llength $elemIds ];

set elemList {};

for { set i 0 } { $i < $numIds } { incr i } {
   set curType [ lindex $elemTypes $i ];

   if { $curType == $elemType } {
   lappend elemList [ lindex $elemIds $i ];
   }
}

return $elemList;
```

hm_errormessage

Description Display an error message in the HyperMesh menu message area.

Syntax hm_errormessage *message [waitTime]*

where

message The message to display in the HyperMesh menu message area.

waitTime The optional time to wait after displaying the message.

hm_getentityvalue

Description Get information for an entity using the HyperMesh template interface.

Syntax hm_getentityvalue *entityType id valueString outputFlag*

where

entityType The type of the entity.

Id The id of the entity.

DataName Template system data name.

OutputFlag 0 - return value is floating point number, 1 - return value is a string.

Example Select and display a nodes XYZ value.

```
*clearmark nodes 1
*createmarkpanel nodes 1

set nodeIds [ hm_getmark nodes 1 ]

if { ! [ Null nodeIds ] } {
    set nodeId [ lindex $nodeIds 0 ]

    set xVal [ hm_getentityvalue NODES $nodeId "x" 0 ]
    set yVal [ hm_getentityvalue NODES $nodeId "y" 0 ]
    set zVal [ hm_getentityvalue NODES $nodeId "z" 0 ]

    tk_messageBox -message "Node $nodeId = $xVal $yVal
$zVal"
}
```

Comments Please refer to the HyperMesh on-line documentation for the available template *datanames*.

When using template system variable names for the *dataname*, make sure that you place a back slash in front of the dollar sign. For example, get the thickness of a component.

```
set compT [ hm_getentityvalue COMPONENTS $compId
    "\$PSHELL_T" 0 ]
```

hm_getfilename

Description Get a filename from the user using a HyperMesh file panel.

Syntax hm_getfilename *message*

where

message The message to display in the HyperMesh menu message area.

Example Get a filename from the user and input it.

```
set filePath [ hm_getfilename "Select the filename to
input" ]

if { ! [ Null filePath ] } {
     if { [ file exists $filePath ] } {
        *readfile "$filePath"
     } else {
        hm_errormessage "The entered filename does not
exist!" 2
     }
}
```

hm_getfloat

Description Get a floating point value from the user using a HyperMesh panel.

Syntax hm_getfloat *caption message*

where

caption The string to display to the left of the edit buffer.

Message The message to display in the HyperMesh menu message area.

Example Translate selected components by a user entered value.

```
*createmarkpanel comps 1

set compIds [ hm_getmark comps 1 ]

if { ! [ Null compIds ] } {
    set xDist [ hm_getfloat "X Dist" "Enter distance
to move in X" ]

    if { ! [ Null xDist ] } {
        *createvector 1 1.0 0.0 0.0
        *translatemark components 1 1 $xDist
    }
}
```

hm_getint

Description Get an integer value from the user using a HyperMesh panel.

Syntax hm_getint *caption message*

where

caption The string to display to the left of the edit buffer.

Message The message to display in the HyperMesh menu message area.

hm_getmark

Description Get the ids for the passed entity type on the passed mark mask.

Syntax hm_getmark *entityType markMask*

where

entityType The type of the entity.

markMask The mark mask (1 or 2)

Example Select and delete elements.

```
hm_markclear elements 1

*createmarkpanel elements 1 "select elements to
delete"

set elemList [ hm_getmark elements 1 ];

if { ! [ Null elemList ] } {
    *deletemark elements 1
}
```

hm_getstring

Description Get a text string from the user using a HyperMesh panel.

Syntax hm_getstring *caption message*

where

caption The string to display to the left of the edit buffer.

message The message to display in the HyperMesh menu message area.

Example Delete a named component.

```
set compName [ hm_getstring "Component" "Enter the
     component name" ]

if { ! [ Null compName ] } {
    if { [ lsearch -exact [ hm_complist name ]
     $compName ] != -1 } {
         *createmark comps 1 "$compName"
         *deletemark comps 1
    } else {
         hm_errormessage "The component '$compName'
            does not exist!" 2
    }
} else {
    hm_errormessage "No component entered!" 2
}
```

hm_markclear

Description Clear the ids for entity type from the passed mark mask.

Syntax hm_markclear *entityType markMask*

where

entityType The type of the entity.

MarkMask The mark mask (1 or 2)

Example Select and delete elements.

```
hm_markclear elements 1

*createmarkpanel elements 1 "select elements to
   delete"

set elemList [ hm_getmark elements 1 ];

if { ! [ Null elemList ] } {

    eval *createmark elements 1 $elemList;
    *deletemark elements 1
}
```

hm_nodelist

Description Get a list of node ids for the passed element.

Syntax hm_nodelist *elemName*

where

elemName The name of the element containing the nodes.

Example Generate a list of node values for an element.

```
set nodeIds   [ hm_nodelist $elemId ];

set numIds [ llength $nodeIds ];

set nodeValList {};

foreach id $nodeIds {
   lappend nodeValList [ hm_nodevalue $id ];
}

return $elemList;
```

hm_nodevalue

Description Get the XYZ values for the passed node id.

Syntax hm_nodevalue *nodeId*

where

nodeId The id of the node.

Example Generate a list of node values for an element.

```
set nodeIds   [ hm_nodelist $elemId ];

set numIds [ llength $nodeIds ];

set nodeValList {};

foreach id $nodeIds {
   lappend nodeValList [ hm_nodevalue $id ];
}

return $elemList;
```

hm_redraw

Description Redraw the HyperMesh window.

Syntax **hm_redraw**

hm_usermessage

Description Display a message in the HyperMesh menu message area.

Syntax hm_usermessage *message [waitTime]*

where

message The message to display in the HyperMesh menu message area.

waitTime The optional time to wait after displaying the message.

Example

Description The following code will cause any surface vertices marked with points within a specified tolerance, to be coincidental.

```
*createmarkpanel points 1 "select the points to
  replace"

set ptIds [ hm_getmark points 1 ]

set ptCnt 0;

if { [ llength $ptIds ] != 0 } {

    set tol [ hm_getfloat "Tolerance:" "Enter the
  point tolerance" ]

    foreach id $ptIds {

        if { [ hm_entityinfo type POINTS $id ] ==
  "vertex" } {

            set x [ hm_getentityvalue POINTS $id "x" 0
  ];
            set y [ hm_getentityvalue POINTS $id "y" 0
  ];
            set z [ hm_getentityvalue POINTS $id "z" 0
  ];

            if { $ptCnt == 0 } {
                set px $x;
                set py $y;
                set pz $z;

                set pId $id
            } else {

                set dx [ expr $x - $px ];
                set dy [ expr $y - $py ];
                set dz [ expr $z - $pz ];

                set dist [ expr sqrt(($dx * $dx ) + (
  $dy * $dy ) + ( $dz * $dz )) ];
```

```
                    if { $dist <= $tol } {
                        *clearmark points 1;
                        *createmark points 1 $id;
                        *verticescombine $pId 1;
                    } else {
                        set pId $id;

                        set px $x;
                        set py $y;
                        set pz $z;
                    }
                }

                incr ptCnt;
            }
        }
    }

*clearmark points 1
```