Chapter 5.
Classes ATTRIB_HH Ea thru Zz

Topic: Ignore

ATTRIB_HH_ENT

Class: Healing, SAT Save and Restore

Purpose: Base HEAL individual entity–level attribute class.

Derivation: ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “individual_entity_attribute”

Filename: heal/healhusk/attrib/at_enty.hxx

Description: ATTRIB_HH_ENT is the base individual entity–level attribute class from which other HEAL individual entity–level attribute classes are derived. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: None

Data:

```
protected VOID_LIST* m_log_list;
```

Logs changes made to each entity during the healing process.

Constructor:

```
public: ATTRIB_HH_ENT::ATTRIB_HH_ENT (
    ENTITY*                 // owning entity
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void ATTRIB_HH_ENT::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT::~ATTRIB_HH_ENT ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_HH_ENT::append_to_log (
    char*                   // string to add
    );
```

Add strings to the log list.

```
public: virtual void ATTRIB_HH_ENT::debug_ent (
    FILE*                   // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: virtual void ATTRIB_HH_ENT::draw_in_col (
    int col                 // specified color
    );
```

Sketch the owner entity in the specified color.

```
public: virtual VOID_LIST*
    ATTRIB_HH_ENT::get_log_list ();
```

Returns the log list.

```
public: BODY* ATTRIB_HH_ENT::get_owner_body ();
```

Returns the pointer to the owning body.

```
public: virtual int ATTRIB_HH_ENT::identity (
    int                     // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT::is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT::pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT::restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

None	no data is saved

```
public: virtual const char*
    ATTRIB_HH_ENT::type_name () const;
```

Returns the string “individual_entity_attribute”.

Internal Use: draw

Related Fncs:

is_ATTRIB_HH_ENT

ATTRIB_HH_ENT_GEOMBUILD_BASE

Class: Healing, SAT Save and Restore

Purpose: Base HEAL individual entity–level attribute class for the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_entity_geombuild”

Filename: heal/healhusk/attrib/entgmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_BASE is the base geometry building individual entity–level attribute class from which other HEAL individual entity–level attribute classes used in the geometry building phase are derived. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: None

Data:

`protected int m_bad;`
Flag indicating whether the new geometry (if it exists; otherwise, this applies to the old geometry) is bad.

`protected HH_COMPUTED_FLAG m_computed;`
Flag indicating whether or not new geometry has been computed for the owning entity.

`protected int m_good_incoming;`
Whether the geometry is good in the beginning.

`protected logical m_unused;`
Flag indicating whether this attribute has been used for storing geometry.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_BASE::
    ATTRIB_HH_ENT_GEOMBUILD_BASE (
    ENTITY* e                // owning entity
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_BASE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_BASE::
    ~ATTRIB_HH_ENT_GEOMBUILD_BASE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_BASE(...) then later x–>lose.)

Methods:

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_BASE::adv_check ();
```

Virtual function that must be implemented by classes derived from this one for performing advanced checks. For this class, always returns UNSET.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::got_used ();
```

Sets the value of the m_unused flag, which indicates whether this attribute has been used for storing geometry.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_BASE::identity (
    int                      // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_BASE_LEVEL .

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::is_bad ();
```

Returns the value of the m_bad flag.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::is_computed ();
```

Returns the value of the m_computed flag.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual int
ATTRIB_HH_ENT_GEOMBUILD_BASE::is_good_incoming ();
```

Checks the geometry and returns
1 if good incoming geometry
0 if bad incoming geometry
–1 if not determined.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::
    is_healing_required ();
```

Returns a flag indicating whether or not healing is required.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::
    is_marked_for_force_compute ();
```

Returns a flag indicating whether or not this instance is marked for a force compute.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::
    mark_for_force_compute ();
```

This instance is marked for a force compute.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::pattern_compatible
() const;
```

Returns TRUE if this is pattern compatible.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::reset();
```

Resets the value of the m_bad flag (to –1).

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

No data — This class does not save any data

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::set_bad (
    logical val             // new flag value
    );
```

Sets the value of the m_bad flag, which indicates whether the geometry (new if present; otherwise, old) is bad.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_BASE::set_computed (
    logical val             // new flag value
    );
```

Sets the value of the m_computed flag.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_BASE::type_name () const;
```

Returns the string “attrib_entity_geombuild”.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_BASE::unused () const;
```

Gets the value of the m_unused flag, which indicates whether this attribute has been used for storing geometry.

Internal Use: does_not_deviate

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_BASE

ATTRIB_HH_ENT_GEOMBUILD_COEDGE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to coedges in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_COEDGE : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_coedge_geombuild”

Filename: heal/healhusk/attrib/cegmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_COEDGE is the individual entity–level attribute class attached to coedges during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: KERN CURVE, PCURVE

Data:

`protected hh_coedge_details m_coedge_details;`
Structure containing details about the coedge.

`protected int m_coedge_details_updated;`
Flag indicating whether or not the coedge details have been updated.

`protected double m_deviation_from_face;`
Maximum deviation from the face.

`protected int m_geom_is_bad;`
Flag indicating whether or not the pcurve is bad.

`protected CURVE* m_old_edge_curve;`
Stores the geometry of the old edge (this is useful in case the old edge gets deleted during stitching).

`protected REVBIT m_old_face_sense;`
Stores the sense of the face when the old coedge geometry was stored.

`protected PCURVE* m_old_geom;`
Stores the old coedge geometry.

`protected REVBIT m_old_sense;`
Stores the old face sense.

`protected double off_face;`
Maximum distance off the face.

`protected int on_face;`
Indicator of whether or not the coedge is on the face. A value of 1 means the coedge is not on the face; 0 means the coedge is on the face.

```
protected int partner;
```
Indicator of a partner coedge. A value of 1 means there is no partner coedge; 0 means there is a partner.

```
protected int p_curve;
```
Indicates status of the associated pcurve.

0 No pcurve
1 Pcurve present
2 No pcurve present but required (spline surface)
3 Missing defining geometry
4 Direction is inconsistent with coedge
5 Distance from the edge is greater than SPAresfit

```
protected double p_curve_max_dist;
```
Maximum distance of the pcurve from the edge.

```
protected logical save_sw;
```
For future use.

```
protected int within_domain;
```
Indicator of whether or not the coedge parameters are within the edge range. A value of 0 means they are within range.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE (
    COEDGE* e               // owning coedge
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_COEDGE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    ~ATTRIB_HH_ENT_GEOMBUILD_COEDGE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_COEDGE(...) then later x–>lose.)

Methods:

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::adv_check ();
```

Performs advanced check of coedge. Checks for pcurve status, partner coedge, whether the coedge is within the range of the edge, and to see if the coedge is on the face.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::deviation (
    logical& is_bad              // flag indicating
        = * (logical* )NULL_REF, // bad geometry
    logical maximum              // flag to return
        = TRUE                   // after 1st problem
    );
```

Returns the maximum deviation of the pcurve from the underlying surface. If maximum is passed as FALSE, then the method returns immediately after the first instance of inaccurate coedge geometry is encountered.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    does_not_deviate ();
```

Returns TRUE if the pcurve deviation is more than SPAresabs.

```
public: hh_coedge_details
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_coedge_details ();
```

Returns a structure containing the parametric details and other information about the coedge.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_off_face () const;
```

Gets the value of the maximum distance off the face.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_on_face () const;
```

Gets the value of “on the face” indicator, on_face. A value of 1 means the coedge is not on the face; 0 means the coedge is on the face.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_partner () const;
```

Gets the value of partner indicator, partner. A value of 1 means there is no partner coedge; 0 means there is a partner.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_p_curve () const;
```

Gets the value of pcurve status indicator, p_curve.

0 No pcurve
1 Pcurve present
2 No pcurve present but required (spline surface)
3 Missing defining geometry
4 Direction is inconsistent with coedge
5 Distance from the edge is greater than SPAresfit

```
public: double ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_p_curve_max_dist () const;
```

Gets the value of maximum distance of the pcurve from the edge.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_quality () const;
```

Returns the coedge quality based on values set in the attribute. Possible values are

0 Good quality. The coedge is on the face, *and* the coedge has a partner, *and* the pcurve status (p_curve) is less than 2.

1 Poor quality. The coedge is not on the face, *or* the coedge has no partner, *or* the pcurve status (p_curve) is greater than or equal to 2.

Returns UNSET if neither of these conditions is met.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    get_within_domain () const;
```

Returns the value of the within domain indicator (within_domain). A value of 0 means the coedge parameters are within the edge range.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::identity (
    int                       // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_COEDGE_LEVEL .

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::is_bad ();
```

Returns TRUE if either the pcurve deviation is more than SPAresabs or if the parent edge geometry is bad. In both cases, the coedge geometry may be required to be computed and healed.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::merge_owner  (
    ENTITY* ent,            // given entity
    logical                 // deleting owner
    );
```

Notifies the ATTRIB_HH_ENT_GEOMBUILD_COEDGE class that its owning ENTITY is about to be merged with the given entity. The application has the chance to delete or otherwise modify the attribute. After the merge, this owner will be deleted if the deleting owner logical is TRUE, otherwise it will be retained and the other entity will be deleted. The default action is to do nothing. This function is supplied by the application whenever it defines a new attribute, and is called when a merge occurs.

```
public: PCURVE* ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    new_geometry () const;
```

Returns the new geometry (pcurve) associated with the coedge, if the coedge has been fixed.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    new_sense () const;
```

Get the new sense of the coedge. This returns the old sense if no new sense has been computed.

```
public: CURVE* ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    old_edge_geometry () const;
```

Returns the old edge curve.

```
public: PCURVE* ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    old_geometry ();
```

Returns the old geometry (pcurve) associated with the coedge, if the coedge has not been fixed.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    old_sense () const;
```

Returns the old sense of the coedge.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::reset ();
```

Resets the attribute.

– Sets the maximum distance from the coedge to the edge to 0.0.
– Sets the maximum distance from the face surface to 0.0.
– Sets the “on face,” within domain, partner, and pcurve status indicators to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    reset_coedge_details ();
```

Resets the coedge details so that they are recalculated the next time they are requested.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    set_coedge_details (
    hh_coedge_details*      // input structure
    );
```

Stores the input coedge details structure in the m_coedge_details data member.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_geometry (
    PCURVE* p                // new pcurve
    );
```

Sets the pcurve, pPCU, associated with the owning coedge to the given pcurve.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_off_face (
    double q                 // new value
    );
```

Sets the value of the maximum distance off the face.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_on_face (
    int q                    // new value
    );
```

Sets the value of “on the face” indicator, on_face. A value of 1 means the coedge is not on the face; 0 means the coedge is on the face.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_partner (
    int q                    // new value
    );
```

Sets the value of partner indicator, partner. A value of 1 means there is no partner coedge; 0 means there is a partner.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_p_curve (
    int q                    // new value
    );
```

Sets the value of pcurve status indicator, p_curve.

0 No pcurve
1 Pcurve present
2 No pcurve present but required (spline surface)
3 Missing defining geometry
4 Direction is inconsistent with coedge
5 Distance from the edge is greater than SPAresfit

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    set_p_curve_max_dist (
    double q                // new distance
    );
```

Sets the value of maximum distance of the pcurve from the edge.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::set_sense (
    REVBIT pS               // new sense
    );
```

Set a new sense for the coedge. This function will also reset all the geometry related info stored in the attributes.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    set_within_domain (
    int q                   // new value
    );
```

Sets the value of the within domain indicator (within_domain). A value of 0 means the coedge parameters are within the edge range.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::split_owner (
    ENTITY* ent             // new entity
    );
```

Notifies the ATTRIB_HH_ENT_GEOMBUILD_COEDGE that its owner is about to be split into two parts. The application has the chance to duplicate or otherwise modify the attribute. The default action is to do nothing. This function is supplied by the application whenever it defines a new attribute, and is called when a split occurs.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    type_name () const;
```

Returns the string “attrib_hh_coedge_geombuild”.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_COEDGE::
    update_coedge_details ();
```

Updates the coedge details structure (m_coedge_details) with new information and sets the m_coedge_details_updated flag.

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_COEDGE

ATTRIB_HH_ENT_GEOMBUILD_CURVE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to curves in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_CURVE : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_curve_geombuild”

Filename: heal/healhusk/attrib/curgmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_CURVE is the individual entity–level attribute class attached to curves during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: None

Data:

```
protected int approx_fit;
```

Indicator of whether this is an exact or approximate fit curve. A value of 0 means exact fit; 1 means approximate fit.

```
protected int closure;
```
Indicator of whether this is a closed curve. A value of 0 means not closed; 1 means closed.

```
protected int continuity;
```
Indicator of whether this is a continuous curve. A value of 0 means continuos; 1 means discontinuous.

```
protected int degeneracy;
```
Indicator of whether this is a degenerate curve. A value of 0 means not degenerate; 1 means degenerate.

```
protected int selfint;
```
Indicator of whether this is a self–intersecting curve. A value of 0 means not self–intersecting; 1 means self–intersecting.

```
protected logical save_sw;
```
For future use.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    ATTRIB_HH_ENT_GEOMBUILD_CURVE (
    CURVE* e                // owning curve
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_CURVE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    ~ATTRIB_HH_ENT_GEOMBUILD_CURVE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_CURVE(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::adv_check ();
```

Performs advanced checks on the owning curve. Checks for fit, closure, continuity, and self–intersection.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    get_approx_fit () const;
```

Gets the value of the indicator of whether this is an exact or approximate fit curve. A value of 0 means exact fit; 1 means approximate fit.

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::get_closure ()
const;
```

Gets the value of the indicator of whether this is a closed curve. A value of 0 means not closed; 1 means closed.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    get_continuity () const;
```

Gets the value of the indicator of whether this is a continuous curve. A value of 0 means continuos; 1 means discontinuous.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    get_degeneracy () const;
```

Gets the value of the indicator of whether this is a degenerate curve. A value of 0 means not degenerate; 1 means degenerate.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    get_quality () const;
```

Returns the curve quality based on values set in the attribute. Possible values are

0 Good quality. The curve is continuous, *and* the curve is not degenerate, *and* the curve does not self–intersect, *and* the curve is an exact fit.

1 Poor quality. The curve is not continuous, *or* the curve is degenerate, *or* the curve self–intersects, *or* the curve is an approximate fit.

Returns UNSET if neither of these conditions is met.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    get_selfint () const;
```

Gets the value of the indicator of whether this is a self–intersecting curve. A value of 0 means not self–intersecting; 1 means self–intersecting.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::identity (
    int                       // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_CURVE_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_CURVE::reset ();
```

Resets the attribute. Sets the continuity, degeneracy, self–intersection, closure, and fit indicators to UNSET.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::set_approx_fit (
    int q                   // new indicator value
    );
```

Sets the value of the indicator of whether this is an exact or approximate fit curve. A value of 0 means exact fit; 1 means approximate fit.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::set_closure (
    int c                   // new indicator value
    );
```

Sets the value of the indicator of whether this is a closed curve. A value of 0 means not closed; 1 means closed.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::set_continuity (
    int q                   // new indicator value
    );
```

Sets the value of the indicator of whether this is a continuous curve. A value of 0 means continuos; 1 means discontinuous.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::set_degeneracy (
    int q                    // new indicator value
    );
```

Sets the value of the indicator of whether this is a degenerate curve. A value of 0 means not degenerate; 1 means degenerate.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::set_selfint (
    int q                    // new indicator value
    );
```

Sets the value of the indicator of whether this is a self–intersecting curve. A value of 0 means not self–intersecting; 1 means self–intersecting.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_CURVE::
    type_name () const;
```

Returns the string “attrib_hh_curve_geombuild”.

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_CURVE

ATTRIB_HH_ENT_GEOMBUILD_EDGE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to edges in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_EDGE : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_edge_geombuild”

Filename: heal/healhusk/attrib/edgmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_EDGE is the individual entity–level attribute class attached to edges during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: KERN CURVE

Data:

`protected int curve_quality;`
Indicator of quality of underlying curve geometry. A value of 0 means good quality. (Refer to ATTRIB_HH_ENT_GEOMBUILD_CURVE.)

`protected edge_data_struct edge_data;`
Data of faces linked with the edge.

`protected double edge_mov_tol;`
Local edge movement tolerance.

`protected double length;`
Edge length.

`protected double m_deviation_from_faces;`
Maximum deviation from faces.

`protected double m_deviation_from_vertices;`
Maximum deviation from vertices.

`protected int m_does_not_deviate;`
Flag indicating whether the edge deviates from the face.

`protected logical m_tangential_stringent;`
Flag indicating whether it is a tangential junction.

`protected double m_geombuild_tol;`
Local edge tolerance for geometry building.

`protected CURVE* m_old_geom;`
Stores the old edge geometry.

`protected REVBIT m_old_sense;`
Stores the old edge sense.

`protected logical m_tangential;`
Flag indicating whether this is a tangential junction.

```
protected int m_vertices_do_not_deviate;
```
Flag indicating whether or not the end vertices deviate from the edge curve.

```
protected logical save_sw;
```
For future use.

```
protected int vexity;
```
Indicator of convexity. Possible values are UNKNOWN, CONCAVE, CONVEX, TANGENT, TCONCAVE (tangent and concave), TCONVEX (tangent and convex), or CTC (global and local convexity are a mixture of convex and concave or tangent/convex and concave or tangent/concave and convex). These constants are defined in the header file for this class.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    ATTRIB_HH_ENT_GEOMBUILD_EDGE (
    EDGE* e                  // owning edge
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_EDGE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    ~ATTRIB_HH_ENT_GEOMBUILD_EDGE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_EDGE(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::adv_check ();
```

Performs advanced checks on the edge. Checks the curve, the edge length, and the convexity.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::deviation (
    logical maximum          // flag whether to return
        = TRUE               // max or 1st deviation
    );
```

Gets the deviation of the curve from the underlying surface. If the flag maximum is TRUE, then the maximum deviation is returned; if this flag is FALSE, then the first deviation encountered that is greater than SPAresabs is returned.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    deviation_from_vertices (
    logical maximum          // flag whether to return
        = TRUE               // max or 1st deviation
    );
```

Gets the deviations of the two end vertices from the edge. If the flag maximum is TRUE, then the maximum deviation is returned; if this flag is FALSE, then the first deviation encountered that is greater than SPAresabs.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    does_not_deviate ();
```

Returns TRUE if the curve deviation from the faces is less than SPAresabs.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    do_intersect ();
```

Intersect surfaces and compute the intersection curve.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    edge_movemnt_tol ();
```

Get edge tolerance for its movement.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::geombuild_tol ();
```

Gets the edge tolerance for geometry building (based on the stitch gap).

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    get_coedge_details (
    hh_coedge_details&,      // details for 1st coedge
    hh_coedge_details&       // details for 2nd coedge
    );
```

Gets the parametric details and other miscellaneous information related to the two underlying coedges. This method is invalid for open edges.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    get_curve_quality () const;
```

Gets the indicator of underlying curve quality.

```
public: const edge_data_struct
ATTRIB_HH_ENT_GEOMBUILD_EDGE::get_edge_data ();
```

Gets the edge_data struct which contains two angle values:
min_angle – minimum tangent angle (based on local edge size)
max_angle – maximum tangent angle (based on local edge size)

```
public: double ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    get_length () const;
```

Gets the length of the edge.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    get_quality () const;
```

Returns the edge quality based on values set in the attribute. Possible values are

0 Good quality. The underlying curve quality is good (0), *and* the edge length is greater than the minimum edge length.

1 Poor quality. The underlying curve quality is poor (1), *or* the edge length is less than the minimum edge length (but greater than zero).

Returns UNSET if neither of these conditions is met.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_EDGE::
get_specific_mov_tol (
    ENTITY_LIST             // list
    );
```

Returns the edge tolerance if the input edge was excluded from its neighbors.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    get_vexity () const;
```

Returns the value of the convexity indicator.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::identity (
    int                     // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_EDGE_LEVEL .

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    is_analytic_edge_G1_healed ();
```

Returns TRUE if the two surfaces are G1 continuous at the edge.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    is_analytic_junction ();
```

Determines if the edge forms a junction of two analytic surfaces.

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::is_bad ();
```

Returns TRUE if either the curve deviation from the face is more than SPAresabs, or if the end vertices do not lie on the surrounding faces, or if any of the end vertices do not lie on the edge.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    is_edge_G1_healed ();
```

Returns TRUE if the two surfaces are G1 continuous at the edge.

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::is_isospline ();
```

Returns a flag indicating whether the isospline edge is within the domain of the isospline solver. Returns TRUE if the adjacent faces form an isospline junction (i.e., *uv* boundary/*uv* boundary complete range).

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::is_tangential (
    logical stringent       // whether it is a
        = FALSE             // tangential junction
    );
```

Returns TRUE if the edge forms a tangential junction (within a specific tolerance).

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::is_uv_nonuv ();
```

Returns TRUE if it is a *uv*/non–*uv* junction.

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::is_uv_uv ();
```

Returns TRUE if it is a *uv*/*uv* junction.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    is_uv_uv_boun_boun ();
```

Returns TRUE if it is a *uv* boundary/*uv* boundary junction.

```
public: double
ATTRIB_HH_ENT_GEOMBUILD_EDGE::max_spline_tang_tol ()
const;
```

Gets the largest allowed difference (tolerance) for the tangent angles between two spline faces which will be still considered tangent.

```
public: virtual void
ATTRIB_HH_ENT_GEOMBUILD_EDGE::merge_owner (
    ENTITY* ent,            // given entity
    logical                 // deleting owner
    );
```

Notifies the ATTRIB_HH_ENT_GEOMBUILD_EDGE that its owning ENTITY is about to be merged with given entity. The application has the chance to delete or otherwise modify the attribute. After the merge, this owner will be deleted if the logical deleting owner is TRUE, otherwise it will be retained and other entity will be deleted. The default action is to do nothing. This function is supplied by the application whenever it defines a new attribute, and is called when a merge occurs.

```
public: double
ATTRIB_HH_ENT_GEOMBUILD_EDGE::min_spline_tang_tol ()
const;
```

Gets the smallest allowed difference for the tangent angles between two spline faces which will be still considered tangent.

```
public: CURVE* ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    new_geometry () const;
```

Returns the latest geometry calculated for the edge. Returns the original geometry if no new geometry has been calculated.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    new_sense () const;
```

Returns the latest edge sense calculated. Returns the original sense if no new sense has been calculated.

```
public: CURVE* ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    old_geometry () const;
```

Returns the original geometry associated with the edge. Even after fixing the new geometry, this method continues to return the old geometry.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    old_sense () const;
```

Returns the original edge sense. Even after fixing the new geometry, this method continues to return the old sense.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::reset ();
```

Resets the attribute.

– Sets the edge length to –100.0.
– Sets the curve quality and convexity indicators to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    reset_box_and_param_range ();
```

Resets the parameter range of the owner edge and also its bounding box.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    reset_dependents ();
```

Resets the attribute on the owning edge's dependent entities (such as its coedges, vertices, etc.).

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    reset_geombuild_tol ();
```

Resets the geometry building tolerance so that it is recalculated the next time it is requested.

```
public: void
ATTRIB_HH_ENT_GEOMBUILD_EDGE::reset_tangency_details
();
```

Initializes the tangency information for this edge, specifying whether the edge is a tangent edge or not.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::restore_common  ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    set_analytic_edge_intersect_log_details (
    char*                    // logged data
        = NULL
    );
```

Function for log list addition for edge geometry computation using intersection.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    set_analytic_edge_project_log_details (
    char*                    // logged data
        = NULL
    );
```

Function for log list addition for edge geometry computation using projection.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_bad (
    logical val             // new value
    );
```

Mark the edge as bad.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_curve_quality (
    int q                   // new value
    );
```

Sets the indicator of underlying curve quality.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_geometry (
    CURVE* pC               // new curve
    );
```

Sets the geometry in the attribute to the given curve.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    set_iso_spline_edge_log_details (
    char*                   // log data
        = NULL
    );
```

Function for log list addition for iso_spline junction.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_length (
    double l                // new length
    );
```

Sets the length of the edge.

```
public: void
ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_max_angle (
    double                  // new max_angle
    );
```

Sets the edge_data max_angle to the input value.

```
public: void
ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_min_angle (
    double                  // new min_angle
    );
```

Sets the edge_data min_angle to the input value.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_sense (
    REVBIT pS               // new sense
    );
```

Sets the edge sense.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    set_stitch_split_edge_log_details (
    char*                   // logged data
        = NULL
    );
```

Function for log list addition for stitch_split.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::set_vexity (
    int v                   // new value
    );
```

Sets the value of the convexity indicator.

```
public: virtual void
ATTRIB_HH_ENT_GEOMBUILD_EDGE::split_owner (
    ENTITY* ent             // new entity
    );
```

Notifies the ATTRIB_HH_ENT_GEOMBUILD_EDGE that its owner is about to be split into two parts. The application has the chance to duplicate or otherwise modify the attribute. The default action is to do nothing. This function is supplied by the application whenever it defines a new attribute, and is called when a split occurs.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    sprint_analytic_edge_intersect_log_details (
    char*                    // log data
        = NULL,
    char*                    // log data
        = NULL
    );
```

Returns the log list addition for edge geometry computation using intersection.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    sprint_analytic_edge_project_log_details (
    char*                    // log data
        = NULL,
    char*                    // log data
        = NULL
    );
```

Returns the log list addition for edge geometry computation using projection.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    sprint_iso_spline_edge_log_details (
    char*                    // temp string
        = NULL,
    char*                    // details
        = NULL
    );
```

Returns the log list addition for iso_spline junction.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    sprint_stitch_split_edge_log_details (
    char*                    // temp string
        = NULL,
    char*                    // details
        = NULL
    );
```

Returns the log list addition for stitch_split.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::tang_tol () const;
```

Returns the tolerance at which the tangency is calculated (m_tang_tol).

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    type_name () const;
```

Returns the string “attrib_hh_edge_geombuild”.

```
public: logical
ATTRIB_HH_ENT_GEOMBUILD_EDGE::update_edge_data ();
```

Updates the min_angle and max_angle.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_EDGE::
    vertices_do_not_deviate ();
```

Returns TRUE if none of the end vertices deviate from the curve.

Internal Use: do_sharp, draw

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_EDGE

ATTRIB_HH_ENT_GEOMBUILD_FACE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to faces in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_FACE : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_face_geombuild”

Filename: heal/healhusk/attrib/fagmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_FACE is the individual entity–level attribute class attached to faces during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: KERN SURFACE

Data:

```
protected double area;
```
Area of face.

```
protected double m_face_box_size;
```
Face box size

```
protected double narrow;
```
Face width.

```
protected int face_area;
```
Indicator of face area status. A value of 1 means the area is less than the minimum area or is negative.

```
protected double face_mov_tol;
```
Local face movement tolerance.

```
protected int face_narrow;
```
Indicator of face width status. A value of 1 means the width is less than the minimum width.

```
protected int lolo_inter;
```
Indicator of loop/loop intersections. A value of 0 means there are no loop/loop intersections; 1 means there are such intersections.

```
protected int loop_inter;
```
Indicator of loop self–intersections. A value of 0 means there are no loop self–intersections; 1 means there are such intersections.

```
protected int loops;
```
Indicator of quality of loops. A value of 0 means good quality. (Refer to ATTRIB_HH_ENT_GEOMBUILD_LOOP.)

```
protected int m_discontinuous;
```
Flag indicating whether or not the surface is discontinuous.

```
protected SURFACE* m_old_geom;
```
Stores the old face geometry.

```
protected REVBIT m_old_sense;
```
Stores the old face sense.

```
protected logical save_sw;
```
For future use.

```
protected int surface_quality;
```
Indicator of quality of underlying surface geometry. A value of 0 means good quality. (Refer to ATTRIB_HH_ENT_GEOMBUILD_SURFACE.)

```
protected logical surf_extended;
```
True if surface has been extended; otherwise FALSE.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_FACE::
    ATTRIB_HH_ENT_GEOMBUILD_FACE (
    FACE* e                  // owning edge
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_FACE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_FACE::
    ~ATTRIB_HH_ENT_GEOMBUILD_FACE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_FACE(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_FACE::adv_check ();
```

Performs advanced checks on the face. Checks for loop/loop intersections, loop self–intersections, loop quality, surface quality, face width status, and face area status.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: double
ATTRIB_HH_ENT_GEOMBUILD_FACE::face_box_size ();
```

Returns the approximate box size of the face.

```
public: double
ATTRIB_HH_ENT_GEOMBUILD_FACE::geombuild_tol () const;
```

Get the face tolerance for geombuild (the maximum of all the edge tolerances).

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_FACE::get_area () const;
```

Gets the face area.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_blend_radius ();
```

Gets the radius of the given blend.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_face_area () const;
```

Gets the face area status indicator.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_face_mov_tol ();
```

Get the tolerance for movement of the face.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_face_narrow () const;
```

Gets the face width status indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_lolo_inter () const;
```

Gets the loop/loop intersection indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_loops () const;
```

Gets the loop quality indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_loop_inter () const;
```

Gets the loop self–intersection indicator.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_narrow () const;
```

Gets the face width.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_quality () const;
```

Returns the face quality based on values set in the attribute. Possible values are

0 Good quality. There are no loop/loop intersections, *and* there are no loop self–intersections, *and* the underlying surface quality is good (0), *and* the loop quality is good (0), *and* the face width is greater than the minimum, *and* the face area is greater than the minimum.

1 Poor quality. There are loop/loop intersections, *or* there are loop self–intersections, *or* the underlying surface quality is poor (1), *or* the loop quality is poor (1), *or* the face width is less than the minimum, *or* the face area is less than the minimum.

Returns UNSET if neither of these conditions is met.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_surface_quality () const;
```

Gets the indicator of the underlying surface quality.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_FACE::
    get_surf_extended () const;
```

Returns TRUE if the surface is extended.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_FACE::identity (
    int                       // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_FACE_LEVEL.

```
public: logical
    ATTRIB_HH_ENT_GEOMBUILD_FACE::is_blend ();
```

Determines whether or not the face has a blend attribute.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_FACE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_FACE::
    is_discontinuous ();
```

Returns TRUE if the surface is C1/G1 discontinuous at multiple knots.

```
public: SURFACE* ATTRIB_HH_ENT_GEOMBUILD_FACE::
    new_geometry () const;
```

Returns the latest geometry calculated for the face. Returns the original geometry if no new geometry has been calculated.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_FACE::
    new_sense () const;
```

Returns the latest face sense calculated. Returns the original sense if no new sense has been calculated.

```
public: SURFACE* ATTRIB_HH_ENT_GEOMBUILD_FACE::
    old_geometry () const;
```

Returns the original geometry associated with the face. Even after fixing the new geometry, this method continues to return the old geometry.

```
public: REVBIT ATTRIB_HH_ENT_GEOMBUILD_FACE::
    old_sense () const;
```

Returns the original face sense. Even after fixing the new geometry, this method continues to return the old sense.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_FACE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::reset ();
```

Resets the attribute.

– Sets the face width to 0.0.
– Sets the face area to 0.0.
– Sets the various indicators (loop/loop intersection, loop self–intersection, loop quality, surface quality, face area status, and face width status) to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    reset_coedge_details ();
```

Resets the coedge details of all the coedges of the face.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    reset_dependents ();
```

Resets the attribute on the face's dependent entities (such as its edges, coedges, vertices, etc.).

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    reset_geombuild_tol ();
```

Resets the local tolerances of all the edges and vertices around the face.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::set_area (
    double d                // new area value
    );
```

Sets the face area.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_blend_radius (
    double                  // new blend radius
    );
```

Sets the radius for the given blend.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_face_area (
    int q                    // new area indicator
    );
```

Sets the face area status indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_face_narrow (
    int q                    // new width indicator
    );
```

Sets the face width status indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_geometry (
    SURFACE* pS              // new surface
    );
```

Sets the geometry in the attribute to the given surface.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::
    set_iso_spline_log_details (
    int knots_add_inu,       // u knots
    int knots_add_inv,       // v knots
    int old_nv,              // old nv
    int old_nu,              // old nu
    int new_nv,              // new nv
    int new_nu               // new nu
    );
```

Logs surface knot insertion details. If the FACES was healed in the iso–spline solver, this function adds information to the ATTRIB_HH_ENT_GEOMBUILD_FACE attribute associated with the FACE. The information marks the faces as being healed using the isospline solver.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_lolo_inter (
    int q                    // new indicator
    );
```

Sets the loop/loop intersection indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_loops (
    int q                    // new quality indicator
    );
```

Sets the loop quality indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_loop_inter (
    int q                    // new self–int indicator
    );
```

Sets the loop self–intersection indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_narrow (
    double d                 // new width value
    );
```

Sets the face width.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::set_sense (
    REVBIT pS                // new sense
    );
```

Sets the face sense.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    set_surface_quality (
    int q                    // new value
    );
```

Sets the indicator of the underlying surface quality.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_FACE::
    set_surf_extended (
    logical a               // new value
    );
```

Sets surface extension on or off.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_FACE::
    sprint_iso_spline_log_details (
    char*,                  // string
    int knots_add_inu,      // u knots
    int knots_add_inv,      // v knots
    int old_nv,             // old nv
    int old_nu,             // old nu
    int new_nv,             // new nv
    int new_nu              // new nu
    );
```

Returns the surface knot insertion details from the log. This function prints out the information stored in the iso_spline_log_details if the information exists in the ATTRIB_HH_ENT_GEOMBUILD_FACE attribute associated with the given FACE.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_FACE::
    type_name () const;
```

Returns the string “attrib_hh_face_geombuild”.

Internal Use: do_sharp, set_analytic_log_details, set_gen_spline_log_details, set_geombuild_log_details, sprint_analytic_log_details, sprint_gen_spline_log_details, sprint_geombuild_log_details

Related Fncs: is_ATTRIB_HH_ENT_GEOMBUILD_FACE

ATTRIB_HH_ENT_GEOMBUILD_LOOP

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to loops in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_LOOP :
ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT :
ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_loop_geombuild”

Filename: heal/healhusk/attrib/lpgmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_LOOP is the individual entity–level attribute class attached to loops during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: None

Data:

`protected double max_gap;`
Maximum coedge gap.

`protected int closure;`
Indicator of loop closure. A value of 0 means the loop is closed; 1 means the loop is not closed.

`protected int gaps;`
Indicator of loop gaps between coedges. A value of 0 means there are no gaps; 1 means there are gaps between coedges.

`protected int on_face;`
Indicator of whether the loop is on the face. A value of 0 means the loop is on the face; 1 means the loop is not on the face.

`protected int oriented;`
Indicator of loop orientation quality. A value of 0 means the loop has good orientation; 1 means the loop has bad orientation.

`protected logical save_sw;`
For future use.

`protected int selfint;`
Indicator of loop self–intersections. A value of 0 means there are no loop self–intersections; 1 means there are such intersections.

`protected int within_domain;`
Indicator of loop coedge parameters. A value of 0 means the coedge parameters are correct; 1 means the coedge parameters are not correct.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    ATTRIB_HH_ENT_GEOMBUILD_LOOP (
    LOOP* e                  // owning loop
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_LOOP(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    ~ATTRIB_HH_ENT_GEOMBUILD_LOOP ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_LOOP(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::adv_check ();
```

Performs advanced checks on the owning loop. Checks loop closure, loop orientation, coedge gaps, loop self–intersections, whether the loop is on the face, and whether the loop is within domain.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_closure () const;
```

Gets the value of the loop closure indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_gaps () const;
```

Gets the value of the coedge gaps indicator.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_max_gap () const;
```

Gets the maximum gap.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_on_face () const;
```

Gets the value of the ”on face” indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_oriented () const;
```

Gets the value of the loop orientation indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_quality () const;
```

Returns the loop quality based on values set in the attribute. Possible values are

0 Good quality. Loop is closed, *and* the orientation is good, *and* there are no coedge gaps, *and* there are no self–intersections, *and* the loop is on the face, *and* the loop is within the domain.

1 Poor quality. Loop is not closed, *or* the orientation is bad, *or* there are coedge gaps, *or* there are self–intersections, *or* the loop is not on the face, *or* the loop is not within the domain.

Returns UNSET if neither of these conditions is met.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_selfint () const;
```

Gets the value of the loop self–intersection indicator.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    get_within_domain () const;
```

Gets the value of the within domain indicator.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::identity (
    int                       // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_LOOP_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_LOOP::reset ();
```

Resets the attribute.

– Sets the maximum coedge gap to 0.0.
– Sets the closure, orientation, coedge gaps,self–intersections, “on face”, and within domain indicators to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_closure (
    logical q               // new indicator
    );
```

Sets the value of the loop closure indicator.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_gaps (
    logical q               // new indicator
    );
```

Sets the value of the coedge gaps indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_max_gap (
    double q                // new gap value
    );
```

Sets the maximum gap.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_on_face (
    logical q               // new indicator
    );
```

Sets the value of the ”on face” indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_oriented (
    logical q               // new indicator
    );
```

Sets the value of the loop orientation indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_selfint (
    logical q               // new indicator
    );
```

Sets the value of the loop self–intersection indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::set_within_domain (
    logical q               // new indicator
    );
```

Sets the value of the within domain indicator.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_LOOP::
    type_name () const;
```

Returns the string “attrib_hh_loop_geombuild”.

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_LOOP

ATTRIB_HH_ENT_GEOMBUILD_SURFACE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to surfaces in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_SURFACE : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_surface_geombuild”

Filename: heal/healhusk/attrib/surgmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_SURFACE is the individual entity–level attribute class attached to surfaces during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: None

Data:

```
protected int approx_fit;
```
Indicator of whether this is an exact or approximate fit surface. A value of 0 means exact fit; 1 means approximate fit.

```
protected int closure;
```
Indicator of whether this is a closed surface. A value of 0 means not closed; 1 means closed.

```
protected int continuity;
```
Indicator of whether this is a continuous surface. A value of 0 means continuos; 1 means discontinuous.

```
protected int degeneracy;
```
Indicator of whether this is a degenerate surface. A value of 0 means not degenerate; 1 means degenerate.

```
protected logical save_sw;
```
For future use.

```
protected int selfint;
```
Indicator of whether this is a self–intersecting surface. A value of 0 means not self–intersecting; 1 means self–intersecting.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE (
    SURFACE* e              // owning surface
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_SURFACE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    ~ATTRIB_HH_ENT_GEOMBUILD_SURFACE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_SURFACE(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::adv_check ();
```

Performs advanced checks on the owning surface. Checks for fit, closure, continuity, and self–intersection.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_approx_fit () const;
```

Gets the value of the indicator of whether this is an exact or approximate fit surface.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_closure () const;
```

Gets the value of the indicator of whether this is a closed surface.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_continuity () const;
```

Gets the value of the indicator of whether this is a continuous surface.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_degeneracy () const;
```

Gets the value of the indicator of whether this is a degenerate surface.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_quality () const;
```

Returns the surface quality based on values set in the attribute. Possible values are

0 Good quality. The surface is continuous, *and* the surface is not degenerate, *and* the surface does not self–intersect.

1 Poor quality. The surface is not continuous, *or* the surface is degenerate, *or* the surface self–intersects.

Returns UNSET if neither of these conditions is met.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    get_selfint () const;
```

Gets the value of the indicator of whether this is a self–intersecting surface.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::identity (
    int                     // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_SURFACE_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::reset ();
```

Resets the attribute. Sets the continuity, degeneracy, self–intersection, closure, and fit indicators to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::set_approx_fit (
    int q                    // new indicator
    );
```

Sets the value of the approximate fit indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::set_closure (
    int q                    // new indicator
    );
```

Sets the value of the closure indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::set_continuity (
    int q                    // new indicator
    );
```

Sets the value of the continuity indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::set_degeneracy (
    int q                    // new indicator
    );
```

Sets the value of the degeneracy indicator.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::set_selfint (
    int q                    // new indicator
    );
```

Sets the value of the self–intersection indicator.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_SURFACE::
    type_name () const;
```

Returns the string “attrib_hh_surface_geombuild”.

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_SURFACE

ATTRIB_HH_ENT_GEOMBUILD_VERTEX

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to vertices in the geometry building phase.

Derivation: ATTRIB_HH_ENT_GEOMBUILD_VERTEX : ATTRIB_HH_ENT_GEOMBUILD_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “attrib_hh_vertex_geombuild”

Filename: heal/healhusk/attrib/vegmbld.hxx

Description: ATTRIB_HH_ENT_GEOMBUILD_VERTEX is the individual entity–level attribute class attached to vertices during the geometry building phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase or subphase are managed by the aggregate attribute for that phase/subphase.

Limitations: None

References: KERN APOINT

Data:

`protected double edge_dist;`
Maximum distance from the edge endpoints to vertex.

`protected int edges_meet;`
Indicator of whether associated edges are within SPAresabs of each other. A value of 0 means yes; 1 means no.

`protected double face_dist;`
Maximum distance from the vertex to the adjoining face surfaces.

`protected double meet_dist;`
Maximum distance between edges at vertex.

`protected double m_deviation_from_edges;`
Maximum deviation from the edges.

`protected double m_deviation_from_faces;`
Maximum deviation from the faces.

`protected int m_does_not_deviate_from_edges;`
Flag indicating whether or not the vertex deviates from the underlying edges.

`protected int m_does_not_deviate_from_faces;`
Flag indicating whether or not the vertex deviates from the underlying faces.

```
protected double m_geombuild_tol;
```
Local vertex tolerance for geometry building.

```
protected APOINT* m_old_geom;
```
Stores the old vertex geometry.

```
protected int on_edges;
```
Indicator of whether vertex is within SPAresabs of endpoints of associated edges. A value of 0 means yes; 1 means no.

```
protected int on_faces;
```
Indicator of whether vertex is within SPAresabs of associated faces. A value of 0 means yes; 1 means no.

```
protected logical save_sw;
```
For future use.

Constructor:

```
public: ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX (
    VERTEX* e               // owning vertex
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_GEOMBUILD_VERTEX(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    ~ATTRIB_HH_ENT_GEOMBUILD_VERTEX ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_GEOMBUILD_VERTEX(...) then later x–>lose.)

Methods:

```
public: int
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::adv_check ();
```

Performs advanced checks on the vertex. Checks if vertex is on edges, maximum distance from the edges, if edges meet, maximum distance between edges, if vertex lies on faces, and maximum distance from the faces.

```
public: virtual void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::deviation (
    logical maximum          // flag indicating to
        = TRUE               // get max deviation
    );
```

Gets the deviations of the vertex from the underlying surfaces and edges. If the flag maximum is TRUE, then the maximum deviation is returned. If this flag is FALSE, then the first deviation encountered that is greater than SPAresabs is returned.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    deviation_from_edges (
    logical maximum          // flag indicating to
        = TRUE               // get max deviation
    );
```

Gets the deviations of the vertex from the edges. If the flag maximum is TRUE, then the maximum deviation is returned. If this flag is FALSE, then the first deviation encountered that is greater than SPAresabs is returned.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    deviation_from_faces (
    logical maximum          // flag indicating to
        = TRUE               // get max deviation
    );
```

Gets the deviations of the vertex from the faces. If the flag maximum is TRUE, then the maximum deviation is returned. If this flag is FALSE, then the first deviation encountered that is greater than SPAresabs is returned.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    does_not_deviate ();
```

Returns TRUE if the vertex deviation from the faces and edges is less than SPAresabs.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    does_not_deviate_from_edges ();
```

Returns TRUE if the vertex deviation from the edges is less than SPAresabs.

```
public: logical ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    does_not_deviate_from_faces ();
```

Returns TRUE if the vertex deviation from the faces is less than SPAresabs.

```
public: double
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::geombuild_tol ();
```

Gets the vertex tolerance for geometry building (the maximum of the vertex deviation and all the edge tolerances).

```
public: int ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_edges_meet () const;
```

Gets the indicator of whether or not edges meet.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_edge_dist () const;
```

Gets the maximum distance from the edges.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_face_dist () const;
```

Gets the maximum distance from the faces.

```
public: double ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_meet_dist () const;
```

Gets the maximum distance between the edges.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_on_edges () const;
```

Gets the indicator of whether or not the vertex is on the edges.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_on_faces () const;
```

Gets the indicator of whether or not the vertex is on the faces.

```
public: int ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    get_quality () const;
```

Returns the vertex quality based on values set in the attribute. Possible values are

0 Good quality. The vertex is on the edges, *and* the edges meet and the vertex is on the faces.

1 Poor quality. The vertex is not on the edges, *or* the edges do not meet, or the vertex is not on the faces.

Returns UNSET if neither of these conditions is met.

```
public: virtual int
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::identity (
    int                       // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_GEOMBUILD_VERTEX_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::is_bad ();
```

Returns TRUE if either the vertex deviation from the faces and edges is more than SPAresabs or if any edges around the vertex are bad.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: APOINT* ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    new_geometry () const;
```

Returns the latest geometry calculated for the vertex. Returns the original geometry if no new geometry has been calculated.

```
public: APOINT* ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    old_geometry () const;
```

Returns the original geometry associated with the vertex. Even after fixing the new geometry, this method continues to return the old geometry.

```
public: virtual logical
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::reset ();
```

Resets the attribute.

– Sets the maximum distance to edges, maximum distance to faces, and maximum distance between edges to 0.0.
– Sets the on edge, edges meet, and on faces indicators to UNSET.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    reset_dependents ();
```

Resets the attribute on the vertex's dependent entities (such as its edges, coedges).

```
public: void ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    reset_geombuild_tol ();
```

Resets the geometry building tolerance so that it is recalculated the next time it is requested.

```
public: void ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(save_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_edges_meet  (
    int q                   // new indicator
    );
```

Gets the indicator of whether or not edges meet.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_edge_dist  (
    double q                // new distance
    );
```

Gets the maximum distance from the edges.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_face_dist  (
    double q                // new distance
    );
```

Gets the maximum distance from the faces.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_geometry (
    APOINT* pA              // new point
    );
```

Sets the geometry in the attribute to the given point.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_meet_dist (
    double q                // new distance
    );
```

Gets the maximum distance between the edges.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_on_edges (
    int q                   // new indicator
    );
```

Gets the indicator of whether or not the vertex is on the edges.

```
public: void
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::set_on_faces (
    int q                   // new indicator
    );
```

Gets the indicator of whether or not the vertex is on the faces.

```
public: virtual const char*
    ATTRIB_HH_ENT_GEOMBUILD_VERTEX::
    type_name () const;
```

Returns the string “attrib_hh_vertex_geombuild”.

Internal Use: draw

Related Fncs:

is_ATTRIB_HH_ENT_GEOMBUILD_VERTEX

ATTRIB_HH_ENT_SIMPLIFY_BASE

Class: Healing, SAT Save and Restore

Purpose: Base HEAL individual entity–level attribute class for the geometry simplification phase.

Derivation: ATTRIB_HH_ENT_SIMPLIFY_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “simgeom_base_entity_attribute”

Filename: heal/healhusk/attrib/entsimbs.hxx

Description: ATTRIB_HH_ENT_SIMPLIFY_BASE is the base geometry simplification individual entity–level attribute class from which other HEAL individual entity–level attribute classes used in the geometry simplification phase are derived. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase are managed by the aggregate attribute for that phase.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_HH_ENT_SIMPLIFY_BASE::
    ATTRIB_HH_ENT_SIMPLIFY_BASE (
    ENTITY*                  // owning entity
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_SIMPLIFY_BASE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_SIMPLIFY_BASE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_SIMPLIFY_BASE::
    ~ATTRIB_HH_ENT_SIMPLIFY_BASE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_SIMPLIFY_BASE(...) then later x–>lose.)

Methods:

```
public: virtual void
    ATTRIB_HH_ENT_SIMPLIFY_BASE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: virtual int
    ATTRIB_HH_ENT_SIMPLIFY_BASE::identity (
    int                      // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_SIMPLIFY_BASE_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_SIMPLIFY_BASE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT_SIMPLIFY_BASE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_BASE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

No data	This class does not save any data

```
public: virtual const char*
    ATTRIB_HH_ENT_SIMPLIFY_BASE::
    type_name () const;
```

Returns the string “simgeom_base_entity_attribute”.

Related Fncs:

is_ATTRIB_HH_ENT_SIMPLIFY_BASE

ATTRIB_HH_ENT_SIMPLIFY_FACE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to faces in the geometry simplification phase.

Derivation: ATTRIB_HH_ENT_SIMPLIFY_FACE : ATTRIB_HH_ENT_SIMPLIFY_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “individual_simgeom_attribute”

Filename: heal/healhusk/attrib/entsimg.hxx

Description: ATTRIB_HH_ENT_SIMPLIFY_FACE is the individual entity–level attribute class attached to faces during the geometry simplification phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase are managed by the aggregate attribute for that phase.

Limitations: None

References: KERN SURFACE

Data:

```
protected SURFACE* m_old_geom;
```
Old geometry of the face.

```
protected SURFACE* m_surf;
```
Simplified surface.

```
protected double m_tol;
```
Tolerance at which tolerance is simplified.

```
protected logical save_sw;
```
For future use.

Constructor:

```
public: ATTRIB_HH_ENT_SIMPLIFY_FACE::
    ATTRIB_HH_ENT_SIMPLIFY_FACE (
    FACE*                   // owning face
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_SIMPLIFY_FACE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_SIMPLIFY_FACE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: void
    ATTRIB_HH_ENT_SIMPLIFY_FACE::lose_surf ();
```

Loses the simplified surface if it is not NULL.

```
protected: virtual ATTRIB_HH_ENT_SIMPLIFY_FACE::
    ~ATTRIB_HH_ENT_SIMPLIFY_FACE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_SIMPLIFY_FACE(...) then later x–>lose.)

Methods:

```
public: virtual void
    ATTRIB_HH_ENT_SIMPLIFY_FACE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: int ATTRIB_HH_ENT_SIMPLIFY_FACE::
    expected_surf_identity ();
```

Gets the identity of the expected surface.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_FACE::fix ();
```

Sets the surface geometry of the owner face to the simplified geometry. Applies (fixes) all the changes for the geometry simplification phase that are stored in this individual attribute to the body. The old geometry is then stored in the attribute.

```
public: CONE* ATTRIB_HH_ENT_SIMPLIFY_FACE::
    force_simplify_to_cone ();
```

Simplify the spline to a cone.

```
public: CONE* ATTRIB_HH_ENT_SIMPLIFY_FACE::
    force_simplify_to_cylinder ();
```

Simplify the spline to a cylinder.

```
public: PLANE* ATTRIB_HH_ENT_SIMPLIFY_FACE::
    force_simplify_to_plane ();
```

Simplify the spline to a plane.

```
public: SPHERE* ATTRIB_HH_ENT_SIMPLIFY_FACE::
    force_simplify_to_sphere ();
```

Simplify the spline to a sphere.

```
public: TORUS* ATTRIB_HH_ENT_SIMPLIFY_FACE::
    force_simplify_to_torus ();
```

Simplify the spline to a torus.

```
public: SURFACE*
    ATTRIB_HH_ENT_SIMPLIFY_FACE::get_surf () const;
```

Gets the simplified surface.

```
public: virtual int
    ATTRIB_HH_ENT_SIMPLIFY_FACE::identity (
    int                     // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_SIMPLIFY_FACE_LEVEL.

```
public: logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::is_cone_type ();
```

Returns whether the surface type is a cone.

```
public: logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::is_cylinder_type ();
```

Returns whether the surface type is a cylinder.

```
public: virtual logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::is_plane_type ();
```

Returns whether the surface type is a plane.

```
public: logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::is_sphere_type ();
```

Returns whether the surface type is a sphere.

```
public: logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::is_torus_type ();
```

Returns whether the surface type is a torus.

```
public: SURFACE*
    ATTRIB_HH_ENT_SIMPLIFY_FACE::old_surf () const;
```

Returns the old surface.

```
public: virtual logical
    ATTRIB_HH_ENT_SIMPLIFY_FACE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_FACE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void
    ATTRIB_HH_ENT_SIMPLIFY_FACE::set_log_details ();
```

Set logging details.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_FACE::set_surf (
    SURFACE*                // simplified surface
    );
```

Sets the new simplified surface. The old surface is lost.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_FACE::set_tol (
    double                  // new tolerance
    );
```

Sets the simplification tolerance used for calculating the simplified surface.

```
public: SURFACE*
    ATTRIB_HH_ENT_SIMPLIFY_FACE::simplify ();
```

Analyzes the face at a given tolerance. This is called by the aggregate analyze function, but it could also be called directly from a user interface with a user–defined tolerance.

```
public: void ATTRIB_HH_ENT_SIMPLIFY_FACE::
    sprint_log_details (
    char*                   // surface data
    );
```

Returns the surface data from the log.

```
public: int
    ATTRIB_HH_ENT_SIMPLIFY_FACE::surf_identity ();
```

Gets the surface identity of the owner surface.

```
public: double
    ATTRIB_HH_ENT_SIMPLIFY_FACE::tol () const;
```

Gets the simplification tolerance used for calculating the simplified surface.

```
public: virtual const char*
    ATTRIB_HH_ENT_SIMPLIFY_FACE::
    type_name () const;
```

Returns the string “individual_simgeom_attribute”.

Related Fncs:

is_ATTRIB_HH_ENT_SIMPLIFY_FACE

ATTRIB_HH_ENT_STITCH_BASE

Class: Healing, SAT Save and Restore

Purpose: Base HEAL individual entity–level attribute class for the stitching phase.

Derivation: ATTRIB_HH_ENT_STITCH_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “stitch_base_entity_attribute”

Filename: heal/healhusk/attrib/entstcbs.hxx

Description: ATTRIB_HH_ENT_STITCH_BASE is the base stitching individual entity–level attribute class from which other HEAL individual entity–level attribute classes used in the stitching phase are derived. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase are managed by the aggregate attribute for that phase.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_HH_ENT_STITCH_BASE::
    ATTRIB_HH_ENT_STITCH_BASE (
    ENTITY*                  // owning entity
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_STITCH_BASE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_STITCH_BASE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_STITCH_BASE::
    ~ATTRIB_HH_ENT_STITCH_BASE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_STITCH_BASE(...) then later x–>lose.)

Methods:

```
public: virtual void
    ATTRIB_HH_ENT_STITCH_BASE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: virtual int
    ATTRIB_HH_ENT_STITCH_BASE::identity (
    int                      // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_STITCH_BASE_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_STITCH_BASE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: virtual logical
    ATTRIB_HH_ENT_STITCH_BASE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_STITCH_BASE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

No data	This class does not save any data

```
public: virtual const char*
    ATTRIB_HH_ENT_STITCH_BASE::
    type_name () const;
```

Returns the string “stitch_base_entity_attribute”.

Related Fncs:

is_ATTRIB_HH_ENT_STITCH_BASE

ATTRIB_HH_ENT_STITCH_EDGE

Class: Healing, SAT Save and Restore

Purpose: Individual entity–level healing attribute class attached to edges in the stitching phase.

Derivation: ATTRIB_HH_ENT_STITCH_EDGE : ATTRIB_HH_ENT_STITCH_BASE : ATTRIB_HH_ENT : ATTRIB_HH : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “individual_stitch_attribute”

Filename: heal/healhusk/attrib/entstch.hxx

Description: ATTRIB_HH_ENT_STITCH_EDGE is the individual entity–level attribute class attached to edges during the stitching phase. Individual entity–level attributes are attached to the individual entities of body being healed to store entity–specific information about each phase or subphase of the healing process. The individual entity–level attributes for each phase are managed by the aggregate attribute for that phase.

Limitations: None

References:

BASE	SPAposition
KERN	EDGE

Data:

```
protected SPAposition end;
```
Original end position.

```
protected EDGE* gap_ref_edge;
```
Edge whose gap size is stored.

```
protected HH_GAP_SIZE_CACHE m_gap_size_cache;
```
For internal use only.

```
protected HH_PT_PERP_CACHE m_pt_perp_cache;
```
For internal use only.

```
protected double m_box_tol;
```
Set the box tolerance of the face to the value passed.

```
protected double m_gap;
```
Gap size between the paired edges.

```
protected double m_len;
```
Original length.

```
protected EDGE* m_partner_edge;
```
Paired edge for stitching together.

```
protected logical save_sw;
```
For future use.

```
protected SPAposition start;
```
Original start position.

Constructor:

```
public: ATTRIB_HH_ENT_STITCH_EDGE::
    ATTRIB_HH_ENT_STITCH_EDGE (
    EDGE*                    // owning edge
        = NULL
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments. Applications should call this constructor only with the overloaded new operator inherited from the ENTITY class (for example, x=new ATTRIB_HH_ENT_STITCH_EDGE(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void
    ATTRIB_HH_ENT_STITCH_EDGE::lose ();
```

Posts a delete bulletin to the bulletin board indicating the instance is no longer used in the active model. The lose methods for attached attributes are also called.

```
protected: virtual ATTRIB_HH_ENT_STITCH_EDGE::
    ~ATTRIB_HH_ENT_STITCH_EDGE ();
```

This C++ destructor should never be called directly. Instead, applications should use the overloaded lose method inherited from the ENTITY class, because this supports history management. (For example, x=new ATTRIB_HH_ENT_STITCH_EDGE(...) then later x–>lose.)

Methods:

```
public: logical ATTRIB_HH_ENT_STITCH_EDGE::analyze (
    EDGE*,                   // edge to test pairing
    double tol,              // tolerance to test at
    double& gap              // gap between edges
    );
```

Analyzes the owning edge and another specified edge to see if they can be paired at the specified tolerance. This method returns the gap size between the two edges.

```
public: double ATTRIB_HH_ENT_STITCH_EDGE::box_tol ();
```

Returns the box_tol, the tolerance depending on the box size.

```
public: virtual void
    ATTRIB_HH_ENT_STITCH_EDGE::debug_ent (
    FILE*                    // file pointer
    ) const;
```

Prints the type and address of this object, roll back pointer, attributes, and any unknown subtype information to the specified file. Refer to the ENTITY class for more details.

```
public: double
    ATTRIB_HH_ENT_STITCH_EDGE::gap_size ();
```

Gets the gap size between the owner edge and the paired edge. This returns a valid size only if the owner edge is paired.

```
public: SPAposition
    ATTRIB_HH_ENT_STITCH_EDGE::get_end ();
```

Returns the original end position.

```
public: double ATTRIB_HH_ENT_STITCH_EDGE::get_len ();
```

Returns the original length.

```
public: SPAposition
    ATTRIB_HH_ENT_STITCH_EDGE::get_start ();
```

Returns the original start position.

```
public: virtual int
    ATTRIB_HH_ENT_STITCH_EDGE::identity (
    int                      // derivation level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_HH_TYPE. If level is specified, returns <class>_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_HH_ENT_SIMPLIFY_FACE_LEVEL.

```
public: virtual logical
    ATTRIB_HH_ENT_STITCH_EDGE::
    is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: logical
    ATTRIB_HH_ENT_STITCH_EDGE::is_paired ();
```

Returns TRUE if the owner edge is either shared or has has been paired up with another edge.

```
public: virtual void
ATTRIB_HH_ENT_STITCH_EDGE::merge_owner (
    ENTITY* ent,            // given entity
    logical                 // deleting owner
    );
```

Notifies the class_name that its owning ENTITY is about to be merged with given entity. The application has the chance to delete or otherwise modify the attribute. After the merge, this owner will be deleted if the logical deleting owner is TRUE, otherwise it will be retained and other entity will be deleted. The default action is to do nothing. This function is supplied by the application whenever it defines a new attribute, and is called when a merge occurs.

```
public: logical ATTRIB_HH_ENT_STITCH_EDGE::pair (
    EDGE*,                  // edge to pair with
    double tol              // tolerance to pair at
    );
```

Pairs the owning edge with another edge. This performs an analysis of whether the edges can be paired at the given tolerance, and returns FALSE if unsuccessful.

```
public: EDGE*
    ATTRIB_HH_ENT_STITCH_EDGE::partner_edge () const;
```

Gets the partner edge.

```
public: virtual logical
    ATTRIB_HH_ENT_STITCH_EDGE::
    pattern_compatible () const;
```

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::
    restore_common ();
```

The RESTORE_DEF macro expands to the restore_common method, which is used in reading information from a SAT file. This method is never called directly. It is called by a higher hierarchical function if an item in the SAT file is determined to be of this class type. An instance of this class will already have been created through the allocation constructor. This method then populates the class instance with the appropriate data from the SAT file.

```
if(restore_version_number >= TOL_MODELING_VERSION)
    read_logical                    This is the save_sw data item.
```

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::set_box_tol (
    double b_tol            // box tolerance
    );
```

Sets the box_tol, the tolerance depending on the box size.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::set_end (
    const SPAposition& pos  // new end position
    );
```

Sets the end position.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::set_gap (
    double                  // new gap
    );
```

Sets the gap.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::set_len (
    double len              // new length
    );
```

Sets the length.

```
public: void
    ATTRIB_HH_ENT_STITCH_EDGE::set_partner_edge (
    EDGE*                   // partner edge
    );
```

Sets the partner edge of the owner as the given edge. This method just sets the partner and does no analysis. Therefore, analysis must be done before using this method.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::set_start (
    const SPAposition& pos  // new start position
    );
```

Sets the start position.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::
    set_stitch_edge_log_details (
    double*                 // log data
        = NULL,
    double*                 // log data
        = NULL
    );
```

Logs edge stitch details.

```
public: virtual void
    ATTRIB_HH_ENT_STITCH_EDGE::split_owner (
    ENTITY* ent             // owning entity
    );
```

Splits the attribute when the edge is split.

```
public: void ATTRIB_HH_ENT_STITCH_EDGE::
    sprint_stitch_edge_log_details (
    char*                   // log data
        = NULL,
    double*                 // log data
        = NULL,
    double*                 // log data
        = NULL
    );
```

Returns edge stitch details from the log.

```
public: virtual const char*
    ATTRIB_HH_ENT_STITCH_EDGE::
    type_name () const;
```

Returns the string “individual_stitch_attribute”.

```
public: logical
    ATTRIB_HH_ENT_STITCH_EDGE::unshared ();
```

Determines if the owner edge is unshared and returns an appropriate flag.

Internal Use: add_gap_size_entry_to_cache, analyze_using_cache, get_gap_from_cache, get_pt_perp_from_cache, reset_cache, reset_cache_for_nearby_edges

Related Fncs:

---

is_ATTRIB_HH_ENT_STITCH_EDGE