Chapter 4.
Classes

Topic: Ignore

The class interface is a set of C++ classes, including their public and protected data and methods (member functions), that an application can use directly to interact with ACIS. Developers may also derive their own classes from these classes to add application-specific functionality and data. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

ENTITY_IHL

Class: Interactive Hidden Line, SAT Save and Restore

Purpose: Defines the owning organization for all other IHL entity classes.

Derivation: ENTITY_IHL : ENTITY : ACIS_OBJECT : –

SAT Identifier: "ihl"

Filename: ihl/ihl_husk/ihl/en_ihl.hxx

Description: All classes representing permanent objects in the IHL model (except attributes) are derived from the base class ENTITY_IHL, which is derived from the ENTITY class.

The ENTITY_IHL class does not represent any specific object within the modeler, but rather allows consistent performance of operations, such as system debugging, change records (bulletin board), roll back, attributes, and model archiving and communication, which are generic methods of the ENTITY class.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ENTITY_IHL::ENTITY_IHL ();
```

C++ allocation constructor requests memory for this object but does not populate it. The allocation constructor is used primarily by restore. Applications should call this constructor only with the overloaded new operator, because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void ENTITY_IHL::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 ENTITY_IHL::~ENTITY_IHL ();
```

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 ENTITY_IHL(...) then later x–>lose.)

Methods:

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

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

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

Returns TRUE if this can be deep copied.

```
public: void ENTITY_IHL::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*
    ENTITY_IHL::type_name () const;
```

Returns the string “ihl”.

Related Fncs:

is_ENTITY_IHL

IHL_CAMERA

Class: Interactive Hidden Line, SAT Save and Restore

Purpose: Defines a camera viewpoint against which hidden lines are calculated.

Derivation: IHL_CAMERA : ENTITY_IHL : ENTITY : ACIS_OBJECT : –

SAT Identifier: “ihl_camera”

Filename: ihl/ihl_husk/ihl/ihl_cam.hxx

Description: Each camera is defined by its position (eye position) in global model space, its aim (target position) in global model space, and by a perspective flag. The perspective flag is TRUE for a simple perspective projection, where two positions define the view direction and the distance between eye and target. The flag is FALSE for a parallel projection, where the eye and target positions define the view direction, and the distance is not relevant.

The orientation of the camera (i.e., rotation about its own view axis) has no influence on the result of the hidden line calculation.

IHL_CAMERA constructors make a new bulletin entry in the current bulletin board to record the creation of the camera.

Limitations: None

References:

by IHL	ATTRIB_IHL_VW, IHL_STDOUT_MANAGER, ihl_data
BASE	position

Data:

```
protected logical cam_persp;
```
TRUE for perspective projection.

```
protected int cam_persp;
```
TRUE for perspective projection.

```
protected position cam_eyepos, cam_target;
```
Camera eye position and target position.

Constructor:

```
public: IHL_CAMERA::IHL_CAMERA ();
```

C++ allocation constructor requests memory for this object but does not populate it. The allocation constructor is used primarily by restore. Applications should call this constructor only with the overloaded new operator, because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: IHL_CAMERA::IHL_CAMERA (
    const IHL_CAMERA& cam   // camera
    );
```

C++ copy constructor requests memory for this object and populates it with the data from the object supplied as an argument. Applications should call this constructor only with the overloaded new operator, because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: IHL_CAMERA::IHL_CAMERA (
    const SPAposition& eyepos,  // eye position
    const SPAposition& target,  // target position
    logical fPersp          // TRUE for perspective
                            // projection
    );
```

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, because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void IHL_CAMERA::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 IHL_CAMERA::~IHL_CAMERA ();
```

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 IHL_CAMERA(...) then later x–>lose.)

Methods:

```
public: logical operator IHL_CAMERA::!= (
    const IHL_CAMERA& to_chk // camera to check
    ) const;
```

Returns TRUE if either the eye position, target position, or perspective projection flag of the supplied camera differ from those of this camera.

```
public: logical operator IHL_CAMERA::== (
    const IHL_CAMERA& to_chk // camera to check
    ) const;
```

Returns TRUE if the eye position, target position, and perspective projection flag of the supplied camera are all the same as those of this camera.

```
public: virtual void IHL_CAMERA::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: const SPAposition& IHL_CAMERA::eyepos ()
const;
```

Returns the eye position.

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

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

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

Returns TRUE if this can be deep copied.

```
public: virtual int
    IHL_CAMERA::is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: int IHL_CAMERA::perspective () const;
```

Returns the perspective flag.

```
public: logical IHL_CAMERA::perspective () const;
```

Returns the perspective flag.

```
public: void IHL_CAMERA::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 IHL_CAMERA::save (
    ENTITY_LIST&            // ENTITYs to save
    ) const;
```

For internal use only. Refer to the ENTITY class for details. Performs a save operation by calling save_begin, the particular class's save_common, and then save_end.

```
protected: void IHL_CAMERA::save_common (
    ENTITY_LIST&            // ENTITYs to save
    ) const;
```

For internal use only. Refer to the ENTITY class for details. Handles the save operation by writing out the savable data associated with the base class.

```
public: void IHL_CAMERA::set_eyepos (
    const SPAposition& eyepos   // new eye position
    );
```

Sets the eye position.

```
public: void IHL_CAMERA::set_perspective (
    logical fPersp          // new perspective flag,
                            // TRUE for perspective
                            // projection
    );
```

Sets the perspective flag.

```
public: void IHL_CAMERA::set_perspective (
    int fPersp              // new perspective flag,
                            // TRUE for perspective
                            // projection
    );
```

Sets the perspective flag.

```
public: void IHL_CAMERA::set_target (
    const SPAposition& target   // new target
position
    );
```

Sets the target position.

```
public: const SPAposition& IHL_CAMERA::target ()
const;
```

Returns the target position.

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

Returns the string “ihl_camera”.

Related Fncs:

is_IHL_CAMERA

IHL_OUTPUT_MANAGER

Class: Interactive Hidden Line

Purpose: Defines a class to control output from IHL.

Derivation: IHL_OUTPUT_MANAGER : –

SAT Identifier: None

Filename: ihl/ihl_husk/meshmgr/ihloutp.hxx

Description: By deriving a child of this class, the application can control output from IHL. (For example, the IHL_STDOUT_MANAGER class is a child of IHL_OUTPUT_MANAGER, whose methods have been written to define how output is processed in the default case where the application has not yet specified any output manager.)

The application controls what happens to the hidden line output by overriding the announce_2D_segment, announce_3D_segment, and announce_next_object methods.

The application activates a particular output manager by calling api_ihl_set_output_manager and passing it an instance of the IHL_OUTPUT_MANAGER class.

The application specifies the type of hidden line output needed by calling the need_3D_coordinates, need_hidden_segments, and need_interior_segments methods.

Hidden line output is then generated by calling the api_ihl_compute_from_meshes function and the resulting output is handled by the active output manager.

The IHL_OUTPUT_MANAGER class works similarly to the output manager of the faceter.

Limitations: None

References: None

Data:

None

Constructor:

```
public: IHL_OUTPUT_MANAGER::IHL_OUTPUT_MANAGER ();
```

C++ allocation constructor requests memory for this object but does not populate it.

Destructor:

```
public: IHL_OUTPUT_MANAGER::~IHL_OUTPUT_MANAGER ();
```

C++ destructor for IHL_OUTPUT_MANAGER which deallocates memory.

Methods:

```
public: virtual void
    IHL_OUTPUT_MANAGER::add_seg_to_list (
    IHL_SEGMENT*  seg           // pointer to segment
    );
```

Add the segment to the segment list.

```
public: virtual void
    IHL_OUTPUT_MANAGER::announce_2D_segment (
    double* lseg,            // array of 4 doubles
                             // with start/end
                             // 2D–coordinates
    double* l3seg,           // array of 6 doubles
                             // with start/end
                             // 3D–coordinates
    double* lseguv,          // array of 4 doubles
                             // with start/end
                             // uv–coordinates
    double* lsegt,           // array of 2 doubles
                             // with start/end tpars
    void* tag,               // void* from the
                             // corresponding PE_EDGE
    logical fVisible,        // TRUE if segment is
                             // visible
    double tpar,             // tpar from begin to end
                             // of segment(silh point)
    logical onsil,           // if segment on
                             // silhouette edge
    IHL_SEGMENT_JOIN_TYPE    // how to connect to
        join                 // cur_seg
    );
```

Announces to the draw routine that a 2D segment needs to be drawn.

```
public: virtual void
    IHL_OUTPUT_MANAGER::announce_3D_segment (
    double* lseg,            //array of 6 doubles with
                             // start/end
                             // 3D–coordinates
    void* tag,               // void* from the
                             // corresponding PE_EDGE
    logical fVisible         // TRUE if segment is
                             // visible
    );
```

Announces to the draw routine that a 3D segment needs to be drawn.

```
public: virtual void
    IHL_OUTPUT_MANAGER::announce_next_object ();
```

Announces to the draw routine that IHL will compute line segments for the next object in the list of bodies (meshes). It is called once for each body/mesh but it is the responsibility of the derived class to keep track of the bodies. IHL_STDOUT_MANAGER overrides this to attach ATTRIB_IHL_VW attributes to bodies.

```
public: virtual IHL_SEGMENT* IHL_OUTPUT_MANAGER::
    get_cur_seg ();
```

Get the current segment which is newly allocated.

```
public: virtual logical
    IHL_OUTPUT_MANAGER::need_3D_coordinates ();
```

Override this method and return TRUE if you wish IHL to output 3D coordinates in the line segments rather than 2D. The default is FALSE.

```
public: virtual logical
    IHL_OUTPUT_MANAGER::need_hidden_segments ();
```

Override this method and return TRUE if you wish IHL to output visible and nonvisible line segments. The default is FALSE.

```
public: virtual logical
    IHL_OUTPUT_MANAGER::need_interior_segments ();
```

Override this method and return TRUE if you wish IHL to output line segments interior to a face. The default is FALSE.

```
public: virtual logical
    IHL_OUTPUT_MANAGER::need_no_hidden_calc ();
```

Override this method and return TRUE if you wish IHL to not calculate hidden line and only output silhouette segments. The default is FALSE.

```
public: virtual void
    IHL_OUTPUT_MANAGER::set_cur_seg (
    IHL_SEGMENT*  seg       // pointer to segment
    );
```

Set the current segment which is newly allocated

Related Fncs:

None

IHL_SEGMENT

Class: Interactive Hidden Line, SAT Save and Restore

Purpose: Defines a hidden line segment with visibility information.

Derivation: IHL_SEGMENT : ENTITY_IHL : ENTITY : ACIS_OBJECT : –

SAT Identifier: “ihl_segment”

Filename: ihl/ihl_husk/ihl/ihl_seg.hxx

Description: This class specifies a hidden line segment generated by IHL.

The various start and end methods can be called to obtain the start and end points of the segment in 2D or 3D coordinates.

The visible method can be called to determine if the segment is visible.

The on_edge method can be called to determine if the segment lies on a model edge.

The model_ent method can be called to determine the owning model entity.

Limitations: None

References:

by IHL	IHL_EDGE, IHL_STDOUT_MANAGER
KERN	ENTITY

Data:

None

Constructor:

```
public: IHL_SEGMENT::IHL_SEGMENT ();
```

C++ allocation constructor requests memory for this object but does not populate it. The allocation constructor is used primarily by restore. Applications should call this constructor only with the overloaded new operator, because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: IHL_SEGMENT::IHL_SEGMENT (
    IHL_SEGMENT& to_cpy     // IHL_SEGMENT to copy
    );
```

C++ copy constructor requests memory for this object and populates it with the data from the object supplied as an argument. Applications should call this constructor only with the overloaded new operator, because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: IHL_SEGMENT::IHL_SEGMENT (
    double* seg,             // array of 4 doubles
                             // with start/end
                             // coordinates
    double* lseg,            // array of 4 doubles
                             // with start/end
                             // coordinates
    double* lseguv,          // array of 4 doubles
                             // with start/end
                             // uv-coordinates
    double* lsegt,           // array of 2 doubles
                             // with start/end tpars
    ENTITY* ent,             // model entity
    int vis,                 // visibility flag
    double seg_tpar,         // tpar from begin to end
                             // of segment(silh point)
    logical onsil            // if segment on
                             // silhouette edge
    );
```

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, because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: IHL_SEGMENT::IHL_SEGMENT (
    double* seg,            // array of 4 doubles
                            // with start/end
                            // coordinates
    double* lseg,           // array of 4 doubles
                            // with start/end
                            // coordinates
    double* lseguv,         // array of 4 doubles
                            // with start/end
                            // uv-coordinates
    double* lsegt,          // array of 2 doubles
                            // with start/end tpars
    ENTITY* ent,            // model entity
    int vis,                // visibility flag
    double seg_tpar,        // tpar from begin to end
                            // of segment(silh point)
    int onsil               // if segment on
                            // silhouette edge
    );
```

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, because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void IHL_SEGMENT::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 IHL_SEGMENT::~IHL_SEGMENT ();
```

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 IHL_SEGMENT(...) then later x->lose.)

Methods:

```
public: const double* IHL_SEGMENT::coords () const;
```

Returns 2D segment coordinates read only.

```
public: double* IHL_SEGMENT::coords ();
```

Returns 2D segment coordinates for update.

```
public: const double* IHL_SEGMENT::coords3 () const;
```

Returns 3D segment coordinates read only.

```
public: double* IHL_SEGMENT::coords3 ();
```

Returns 3D segment coordinates for update.

```
public: virtual void IHL_SEGMENT::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 IHL_SEGMENT::get_tpar () const;
```

Returns t-parameter of segment which denotes the silhouette point.

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

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

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

Returns TRUE if this can be deep copied.

```
public: virtual int
    IHL_SEGMENT::is_deepcopyable () const;
```

Returns TRUE if this can be deep copied.

```
public: ENTITY* IHL_SEGMENT::model_ent () const;
```

Returns a pointer to the related model entity.

```
public: IHL_SEGMENT* IHL_SEGMENT::next ();
```

Returns a pointer to the next IHL_SEGMENT.

```
public: logical IHL_SEGMENT::on_edge () const;
```

Returns TRUE if the segment is on a model edge.

```
public: int IHL_SEGMENT::on_edge () const;
```

Returns TRUE if the segment is on a model edge.

```
public: logical IHL_SEGMENT::on_sil () const;
```

Returns TRUE if the segment is on a silhouette edge.

```
public: int IHL_SEGMENT::on_sil () const;
```

Returns TRUE if the segment is on a silhouette edge.

```
public: IHL_SEGMENT* IHL_SEGMENT::previous ();
```

Returns a pointer to the previous IHL_SEGMENT.

```
public: void IHL_SEGMENT::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: void IHL_SEGMENT::set_next (
    IHL_SEGMENT* next       // pointer to segment
    );
```

Sets next IHL_SEGMENT pointer.

```
public: void IHL_SEGMENT::set_previous (
    IHL_SEGMENT* prev       // pointer to segment
    );
```

Sets previous IHL_SEGMENT pointer.

```
public: double* IHL_SEGMENT::tpars ();
```

Returns t-parameter edge coordinates for update.

```
public: const double* IHL_SEGMENT::tpars () const;
```

Returns t-parameter edge coordinates. This is read only.

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

Returns the string “ihl_segment”.

```
public: double IHL_SEGMENT::t_end () const;
```

Returns t-parameter edge end point.

```
public: double IHL_SEGMENT::t_start () const;
```

Returns t-parameter edge start point.

```
public: double* IHL_SEGMENT::uvcoords ();
```

Returns segment coordinates for update.

```
public: const double* IHL_SEGMENT::uvcoords () const;
```

Returns segment coordinates read only.

```
public: double IHL_SEGMENT::u_end () const;
```

Returns segment end point.

```
public: double IHL_SEGMENT::u_start () const;
```

Returns segment start point.

```
public: logical IHL_SEGMENT::visible () const;
```

Returns TRUE if the segment is visible.

```
public: int IHL_SEGMENT::visible () const;
```

Returns TRUE if the segment is visible.

```
public: double IHL_SEGMENT::v_end () const;
```

Returns segment end point.

```
public: double IHL_SEGMENT::v_start () const;
```

Returns segment start point.

```
public: double IHL_SEGMENT::x3_end () const;
```

Returns 3D *x*-coordinate of the end point.

```
public: double IHL_SEGMENT::x3_start () const;
```

Returns 3D *x*-coordinate of the start point.

```
public: double IHL_SEGMENT::x_end () const;
```

Returns 2D *x*-coordinate of the end point.

```
public: double IHL_SEGMENT::x_start () const;
```

Returns 2D *x*-coordinate of the start point.

```
public: double IHL_SEGMENT::y3_end () const;
```

Returns 3D *y*-coordinate of the end point.

```
public: double IHL_SEGMENT::y3_start () const;
```

Returns 3D *y*-coordinate of the start point.

```
public: double IHL_SEGMENT::y_end () const;
```

Returns 2D *y*-coordinate of the end point.

```
public: double IHL_SEGMENT::y_start () const;
```

Returns 2D *y*-coordinate of the start point.

```
public: double IHL_SEGMENT::z3_end () const;
```

Returns 3D *z*-coordinate of the end point.

```
public: double IHL_SEGMENT::z3_start () const;
```

Returns 3D *z*-coordinate of the start point.

Related Fncs:

---

is_IHL_SEGMENT