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.

ATTRIB_GENERIC

Class: Attributes, SAT Save and Restore

Purpose: Organization base attribute class for the Generic Attributes Component.

Derivation: ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: "gen"

Filename: ga/ga_husk/attrib/at_gen.hxx

Description: Defines the organization attribute class for the Generic Attributes Component.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_GENERIC::ATTRIB_GENERIC (
    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_GENERIC(...)), because this reserves the memory on the heap, a requirement to support roll back and history management.

Destructor:

```
public: virtual void ATTRIB_GENERIC::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_GENERIC::~ATTRIB_GENERIC ();
```

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_GENERIC(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GENERIC::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_GENERIC::identity (
    int                     // level
        = 0
    ) const;
```

If level is unspecified or 0, returns the type identifier ATTRIB_GENERIC_TYPE. If level is specified, returns ATTRIB_GENERIC_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GENERIC_LEVEL.

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

Returns TRUE if this can be deep copied.

```
public: void ATTRIB_GENERIC::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_GENERIC::type_name () const;
```

Returns the string “gen”.

Related Fncs:

is_ATTRIB_GENERIC

ATTRIB_GEN_ENTITY

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that owns an entity.

Derivation: ATTRIB_GEN_ENTITY : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “entity_attrib”

Filename: ga/ga_husk/attrib/at_ent.hxx

Description: Defines a generic attribute that owns an entity. The owned entity is copied, transformed, and lost along with the attribute’s owner.

Limitations: None

References: KERN ENTITY

Data:

None

Constructor:

```
public: ATTRIB_GEN_ENTITY::ATTRIB_GEN_ENTITY ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_ENTITY), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_ENTITY::ATTRIB_GEN_ENTITY (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    ENTITY* value,           // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_ENTITY::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_GEN_ENTITY::~ATTRIB_GEN_ENTITY ();
```

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_GEN_ENTITY(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_ENTITY::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.

```
protected: virtual void
    ATTRIB_GEN_ENTITY::do_transform (
    SPAtransf const&,        // transformation
    ENTITY_LIST&             // entity list
    );
```

Transforms the entity owned by this attribute in response to the trans_owner method.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_ENTITY::duplicate (
    ENTITY* onto             // entity to copy onto
    ) const;
```

Make a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_ENTITY::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_ENTITY_TYPE. If level is specified, returns ATTRIB_GEN_ENTITY_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_ENTITY_LEVEL.

```
public: logical ATTRIB_GEN_ENTITY::isa (
    int t                    // class type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_ENTITY::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.

read_ptr	Pointer to record in SAT file for owning entity.

```
public: void ATTRIB_GEN_ENTITY::set_value (
    ENTITY* val             // entity value
    );
```

Changes the entity owned by this attribute.

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

Returns the string “entity_attrib”.

```
public: ENTITY* ATTRIB_GEN_ENTITY::value () const;
```

Return the entity owned by this attribute.

Related Fncs:

is_ATTRIB_GEN_ENTITY

ATTRIB_GEN_INTEGER

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains an integer value.

Derivation: ATTRIB_GEN_INTEGER : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “integer_attrib”

Filename: ga/ga_husk/attrib/at_int.hxx

Description: Defines a generic attribute that contains an integer value.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_GEN_INTEGER::ATTRIB_GEN_INTEGER ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_INTEGER), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_INTEGER::ATTRIB_GEN_INTEGER (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    int value,               // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_INTEGER::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_GEN_INTEGER::~ATTRIB_GEN_INTEGER ();
```

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_GEN_INTEGER(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_INTEGER::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.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_INTEGER::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_INTEGER::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_INTEGER_TYPE. If level is specified, returns ATTRIB_GEN_INTEGER_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_INTEGER_LEVEL.

```
public: logical ATTRIB_GEN_INTEGER::isa (
    int t                    // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_INTEGER::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.

read_int	Value of integer stored in SAT file

```
public: void ATTRIB_GEN_INTEGER::set_value (
    int val                  // integer value
    );
```

Changes the integer value contained by this attribute.

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

Returns the string “integer_attrib”.

```
public: int ATTRIB_GEN_INTEGER::value () const;
```

Returns the integer value contained by this attribute.

Related Fncs:

is_ATTRIB_GEN_INTEGER

ATTRIB_GEN_NAME

Class: Attributes, SAT Save and Restore

Purpose: Defines a named attribute for the Generic Attributes Component.

Derivation: ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “name_attrib”

Filename: ga/ga_husk/attrib/at_name.hxx

Description: Defines a named attribute for the Generic Attributes Component.

Limitations: None

References: None

Data:

```
protected char *Name;
```
Name assigned to this attribute.

```
protected copy_action Copy_action;
```
Action to be performed when this attribute’s owner is copied.

```
protected merge_action Merge_action;
```
Action to be performed when this attribute’s owner is merged.

```
protected split_action Split_action;
```
Action to be performed when this attribute’s owner is split.

```
protected trans_action Trans_action;
```
Action to be performed when this attribute’s owner is transformed.

Constructor:

```
public: ATTRIB_GEN_NAME::ATTRIB_GEN_NAME ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_NAME), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_NAME::ATTRIB_GEN_NAME (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_NAME::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_GEN_NAME::~ATTRIB_GEN_NAME ();
```

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_GEN_NAME(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_NAME::copy_owner (
    ENTITY*                  // owning entity
    );
```

Specifies the copy's owner.

```
public: virtual void ATTRIB_GEN_NAME::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.

```
protected: virtual void
    ATTRIB_GEN_NAME::do_transform (
    SPAtransf const&,       // transformation
    ENTITY_LIST&            // entity list
    );
```

Transforms the entity owned by this attribute in response to the trans_owner method.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_NAME::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Returns generic attribute name that the duplicate is copied onto.

```
public: static int ATTRIB_GEN_NAME::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_NAME_TYPE. If level is specified, returns ATTRIB_GEN_NAME_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_NAME_LEVEL.

```
public: virtual logical ATTRIB_GEN_NAME::isa (
    int t                   // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Notifies the ATTRIB_GEN_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: char const* ATTRIB_GEN_NAME::name () const;
```

Get the attribute name.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_NAME::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.

read_enum	Read the enumeration for split_action.
read_enum	Read the enumeration for merge_action
read_enum	Read the enumeration for trans_action.

```
if (restore_version_number >= GA_COPY_ACTION_VERSION)
```

read_enum	Read the enum for the copy action.
read_string	Name of the attribute.

```
public: void ATTRIB_GEN_NAME::set_name (
    char const*             // name
    );
```

Changes the name assigned to this attribute.

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

Notifies the ATTRIB_GEN_NAME 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 void
    ATTRIB_GEN_NAME::trans_owner_list (
    SPAtransf const&,       // transformation
    ENTITY_LIST&            // entity list
    );
```

Notifies the ATTRIB_GEN_NAME that its owner is about to be transformed. The application has the chance to transform 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 transformation occurs.

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

Returns the string “name_attrib”.

Internal Use: full_size

Related Fncs:

is_ATTRIB_GEN_NAME

ATTRIB_GEN_POINTER

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains a reference to an entity.

Derivation: ATTRIB_GEN_POINTER : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “pointer_attrib”

Filename: ga/ga_husk/attrib/at_ptr.hxx

Description: The referenced entity is *not* copied, transformed, or lost along with the attribute’s owner.

Limitations: None

References: KERN ENTITY

Data:

None

Constructor:

```
public: ATTRIB_GEN_POINTER::ATTRIB_GEN_POINTER ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_POINTER), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_POINTER::ATTRIB_GEN_POINTER (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    ENTITY* value,           // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_POINTER::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_GEN_POINTER::~ATTRIB_GEN_POINTER ();
```

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_GEN_POINTER(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_POINTER::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.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_POINTER::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_POINTER::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_POINTER_TYPE. If level is specified, returns ATTRIB_GEN_POINTER_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_POINTER_LEVEL.

```
public: logical ATTRIB_GEN_POINTER::isa (
    int t                   // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_POINTER::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.

read_ptr	Pointer to record in SAT file for referenced entity.

```
public: void ATTRIB_GEN_POINTER::set_value (
    ENTITY* val             // value
    );
```

Changes the entity referenced by this attribute.

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

Returns the string “pointer_attrib”.

```
public: ENTITY* ATTRIB_GEN_POINTER::value () const;
```

Returns the entity referenced by this attribute.

Related Fncs:

is_ATTRIB_GEN_POINTER

ATTRIB_GEN_POSITION

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains a position.

Derivation: ATTRIB_GEN_POSITION : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “position_attrib”

Filename: ga/ga_husk/attrib/at_pos.hxx

Description: Defines a generic attribute that contains a position.

Limitations: None

References: BASE SPAposition

Data:

None

Constructor:

```
public: ATTRIB_GEN_POSITION::ATTRIB_GEN_POSITION ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_POSITION), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_POSITION::ATTRIB_GEN_POSITION (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    SPAposition const& value,   // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_POSITION::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_GEN_POSITION::~ATTRIB_GEN_POSITION ();
```

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_GEN_POSITION(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_POSITION::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.

```
protected: virtual void
    ATTRIB_GEN_POSITION::do_transform (
    SPAtransf const&,       // transformation
    ENTITY_LIST&            // entity list
    );
```

Transforms the entity owned by this attribute in response to the trans_owner method.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_POSITION::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of the attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_POSITION::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_POSITION_TYPE. If level is specified, returns ATTRIB_GEN_POSITION_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_POSITION_LEVEL.

```
public: logical ATTRIB_GEN_POSITION::isa (
    int t                    // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_POSITION::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.

read_position	Position to read from SAT file

```
public: void ATTRIB_GEN_POSITION::set_value (
    SPAposition const& val  // value
    );
```

Changes the position contained by this attribute.

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

Returns the string “position_attrib”.

```
public: SPAposition ATTRIB_GEN_POSITION::value ()
const;
```

Returns the position contained by this attribute.

Related Fncs:

is_ATTRIB_GEN_POSITION

ATTRIB_GEN_REAL

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains a real value.

Derivation: ATTRIB_GEN_REAL : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “real_attrib”

Filename: ga/ga_husk/attrib/at_real.hxx

Description: Defines a generic attribute that contains a real value.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_GEN_REAL::ATTRIB_GEN_REAL ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_REAL), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_REAL::ATTRIB_GEN_REAL (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    double value,            // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_REAL::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_GEN_REAL::~ATTRIB_GEN_REAL ();
```

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_GEN_REAL(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_REAL::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.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_REAL::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_REAL::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_REAL_TYPE. If level is specified, returns ATTRIB_GEN_REAL_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_REAL_LEVEL.

```
public: logical ATTRIB_GEN_REAL::isa (
    int t                   // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_REAL::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.

read_real — Real value to read from SAT file.

```
public: void ATTRIB_GEN_REAL::set_value (
    double val              // value
    );
```

Changes the real value contained by this attribute.

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

Returns the string “real_attrib”.

```
public: double ATTRIB_GEN_REAL::value () const;
```

Returns the real value contained by the attribute.

Related Fncs:

---

is_ATTRIB_GEN_REAL

ATTRIB_GEN_STRING

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains a string value.

Derivation: ATTRIB_GEN_STRING : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “string_attrib”

Filename: ga/ga_husk/attrib/at_str.hxx

Description: Defines a generic attribute that contains a string value.

Limitations: None

References: None

Data:

None

Constructor:

```
public: ATTRIB_GEN_STRING::ATTRIB_GEN_STRING ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_STRING), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_STRING::ATTRIB_GEN_STRING (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    char const* value,       // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_STRING::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_GEN_STRING::~ATTRIB_GEN_STRING ();
```

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_GEN_STRING(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_STRING::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.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_STRING::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_STRING::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_STRING_TYPE. If level is specified, returns ATTRIB_GEN_STRING_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_STRING_LEVEL.

```
public: logical ATTRIB_GEN_STRING::isa (
    int t                   // type to check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_STRING::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.

read_string — Restore string stored in SAT file.

```
public: void ATTRIB_GEN_STRING::set_value (
    char const* val         // value
    );
```

Changes the string value contained by this attribute.

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

Returns the string: “string_attrib”.

```
public: char const*
    ATTRIB_GEN_STRING::value () const;
```

Returns the string value contained by this attribute.

Internal Use: full_size

Related Fncs: is_ATTRIB_GEN_STRING

ATTRIB_GEN_VECTOR

Class: Attributes, SAT Save and Restore

Purpose: Defines a generic attribute that contains a vector.

Derivation: ATTRIB_GEN_VECTOR : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “vector_attrib”

Filename: ga/ga_husk/attrib/at_vec.hxx

Description: Defines a generic attribute that contains a vector.

Limitations: None

References: BASE SPAvector

Data:

None

Constructor:

```
public: ATTRIB_GEN_VECTOR::ATTRIB_GEN_VECTOR ();
```

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 inherited from the ENTITY class (for example, x=new ATTRIB_GEN_VECTOR), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: ATTRIB_GEN_VECTOR::ATTRIB_GEN_VECTOR (
    ENTITY* owner,           // owning entity
    char const* name,        // name
    SPAvector const& value,  // value
    split_action             // split action
        = SplitKeep,
    merge_action             // merge action
        = MergeKeepKept,
    trans_action             // transformation action
        = TransIgnore,
    copy_action              // copy action
        = CopyCopy
    );
```

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

Destructor:

```
public: virtual void ATTRIB_GEN_VECTOR::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_GEN_VECTOR::~ATTRIB_GEN_VECTOR ();
```

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_GEN_VECTOR(...) then later x–>lose.)

Methods:

```
public: virtual void ATTRIB_GEN_VECTOR::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.

```
protected: virtual void
    ATTRIB_GEN_VECTOR::do_transform (
    SPAtransf const&,       // transformation
    ENTITY_LIST&            // entity list
    );
```

Transforms the entity owned by this attribute in response to the trans_owner method.

```
protected: virtual ATTRIB_GEN_NAME*
    ATTRIB_GEN_VECTOR::duplicate (
    ENTITY* onto            // entity to copy onto
    ) const;
```

Makes a copy of this attribute, attached to the given entity.

```
public: static int ATTRIB_GEN_VECTOR::id ();
```

Returns the attribute class identification.

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

If level is unspecified or 0, returns the type identifier ATTRIB_GEN_VECTOR_TYPE. If level is specified, returns ATTRIB_GEN_VECTOR_TYPE for that level of derivation from ENTITY. The level of this class is defined as ATTRIB_GEN_VECTOR_LEVEL.

```
public: logical ATTRIB_GEN_VECTOR::isa (
    int t                   // type check
    ) const;
```

Determines if the attribute class is the specified type.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void ATTRIB_GEN_VECTOR::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.

read_vector	Restore vector stored in SAT file.

```
public: void ATTRIB_GEN_VECTOR::set_value (
    SPAvector const& val    // value
    );
```

Change the vector contained by this attribute.

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

Returns the string “vector_attrib”.

```
public: SPAvector ATTRIB_GEN_VECTOR::value () const;
```

Returns the vector contained by this attribute.

Related Fncs:

is_ATTRIB_GEN_VECTOR

NAMED_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_NAME instead.

Derivation: NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_attribute”

Filename: ga/ga_husk/pmhusk/nm_attr.hxx

Description: This class allows for generically-named attributes. This class is derived from the ATTRIB_ST class. It provides methods and data common to named attributes.

Limitations: None

References: None

Data:

None

Constructor:

```
public: NAMED_ATTRIB::NAMED_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_ATTRIB::NAMED_ATTRIB (
    ENTITY* owner,          // entity owner
    const char* name        // entity name
    );
```

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

Destructor:

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

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 NAMED_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_ATTRIB::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 char* NAMED_ATTRIB::get_name () const;
```

Gets the name of the attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_ATTRIB_TYPE. If level is specified, returns NAMED_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

```
public: logical NAMED_ATTRIB::is_named (
    const char*             // name of attribute
    ) const;
```

Returns TRUE if the attribute has a given name; otherwise, it returns FALSE.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_ATTRIB::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.

read_string	Attribute name string stored in SAT file

```
protected: void NAMED_ATTRIB::set_name (
    const char*             // name
    );
```

Sets the name of the attribute.

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

Returns the string “named_attribute”.

Internal Use: full_size

Related Fncs:

is_NAMED_ATTRIB

NAMED_INT_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_INTEGER instead.

Derivation: NAMED_INT_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_int_attribute”

Filename: ga/ga_husk/pmhusk/nmi_attr.hxx

Description: This class provides named attributes with integer values.

Limitations: None

References: None

Data:

None

Constructor:

```
public: NAMED_INT_ATTRIB::NAMED_INT_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_INT_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_INT_ATTRIB::NAMED_INT_ATTRIB (
    ENTITY* owner,          // entity owner
    const char* name,       // name of attribute
    int value               // integer value
    );
```

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

Destructor:

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

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 NAMED_INT_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_INT_ATTRIB::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 NAMED_INT_ATTRIB::get_value () const;
```

Gets the value of the named integer attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_INT_ATTRIB_TYPE. If level is specified, returns NAMED_INT_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_INT_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_INT_ATTRIB::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.

read_int	Attribute value

```
protected: void NAMED_INT_ATTRIB::set_value (
    int val                 // attribute value
    );
```

Sets the value of the named integer attribute.

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

Returns the string “named_int_attribute”.

Related Fncs:

is_NAMED_INT_ATTRIB

NAMED_LOGICAL_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_INTEGER instead.

Derivation: NAMED_LOGICAL_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_logical_attribute”

Filename: ga/ga_husk/pmhusk/nml_attr.hxx

Description: This class provides named attributes with logical values.

Limitations: None

References: None

Data:

None

Constructor:

```
public:
    NAMED_LOGICAL_ATTRIB::NAMED_LOGICAL_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_LOGICAL_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_LOGICAL_ATTRIB::NAMED_LOGICAL_ATTRIB (
    ENTITY* owner,          // entity owner
    const char* name,       // name of attribute
    logical value           // logical value
    );
```

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

Destructor:

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

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 NAMED_LOGICAL_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void
    NAMED_LOGICAL_ATTRIB::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: logical
    NAMED_LOGICAL_ATTRIB::get_value () const;
```

Gets the value of the named logical attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_LOGICAL_ATTRIB_TYPE. If level is specified, returns NAMED_LOGICAL_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_LOGICAL_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_LOGICAL_ATTRIB::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.

read_int Read in the attribute value

```
protected: void NAMED_LOGICAL_ATTRIB::set_value (
    logical val             // logical value
    );
```

Sets the attribute value.

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

Returns the string “named_logical_attribute”.

Related Fncs:

is_NAMED_LOGICAL_ATTRIB

NAMED_POS_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_POSITION instead.

Derivation: NAMED_POS_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_pos_attribute”

Filename: ga/ga_husk/pmhusk/nmp_attr.hxx

Description: This class provides named attributes with position values.

Limitations: None

References: BASE SPAposition

Data:

None

Constructor:

```
public: NAMED_POS_ATTRIB::NAMED_POS_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_POS_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_POS_ATTRIB::NAMED_POS_ATTRIB (
    ENTITY* owner,           // entity owner
    const char* name,        // name of attribute
    const SPAposition& value // position value
    );
```

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

Destructor:

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

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 NAMED_POS_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_POS_ATTRIB::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: SPAposition
    NAMED_POS_ATTRIB::get_value () const;
```

Gets the value of the named position attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_POS_ATTRIB_TYPE. If level is specified, returns NAMED_POS_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_POS_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_POS_ATTRIB::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.

read_position	Restore the position from the SAT file.

```
protected: void NAMED_POS_ATTRIB::set_value (
    const SPAposition& val  // position
    );
```

Sets the value of the named position attribute.

```
public: virtual void NAMED_POS_ATTRIB::trans_owner (
    SPAtransf const&        // transformation
    );
```

Notifies the NAMED_POS_ATTRIB that its owner is about to be transformed. The application has the chance to transform 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 transformation occurs.

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

Returns the string “named_pos_attribute”.

Related Fncs:

is_NAMED_POS_ATTRIB

NAMED_REAL_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_REAL instead.

Derivation: NAMED_REAL_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_real_attribute”

Filename: ga/ga_husk/pmhusk/nmr_attr.hxx

Description: This class provides named attributes with real values.

Limitations: None

References: None

Data:

None

Constructor:

```
public: NAMED_REAL_ATTRIB::NAMED_REAL_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_REAL_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_REAL_ATTRIB::NAMED_REAL_ATTRIB (
    ENTITY* owner,           // entity owner
    const char* name,        // name of attribute
    double value             // real value
    );
```

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

Destructor:

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

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 NAMED_REAL_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_REAL_ATTRIB::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 NAMED_REAL_ATTRIB::get_value () const;
```

Gets the value of the named real attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_REAL_ATTRIB_TYPE. If level is specified, returns NAMED_REAL_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_REAL_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_REAL_ATTRIB::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.

read_real	Read in the attribute value

```
protected: void NAMED_REAL_ATTRIB::set_value (
    double val              // attribute value
    );
```

Sets the value of the named real attribute.

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

Returns the string “named_real_attribute”.

Related Fncs:

is_NAMED_REAL_ATTRIB

NAMED_STRING_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_STRING instead.

Derivation: NAMED_STRING_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_string_attribute”

Filename: ga/ga_husk/pmhusk/nms_attr.hxx

Description: This class provides named attributes with string values.

Limitations: None

References: None

Data:

None

Constructor:

```
public: NAMED_STRING_ATTRIB::NAMED_STRING_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_STRING_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_STRING_ATTRIB::NAMED_STRING_ATTRIB (
    ENTITY* owner,          // entity owner
    const char* name,       // name of attribute
    const char* value       // string value
    );
```

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

Destructor:

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

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 NAMED_STRING_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_STRING_ATTRIB::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 char*
    NAMED_STRING_ATTRIB::get_value () const;
```

Gets the value for a named string attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_STRING_ATTRIB_TYPE. If level is specified, returns NAMED_STRING_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_STRING_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_STRING_ATTRIB::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.

read_string	Read in the length attribute value

```
protected: void NAMED_STRING_ATTRIB::set_value (
    const char* val         // attribute value
    );
```

Sets the value for a named string attribute.

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

Returns the string “named_string_attribute”.

Internal Use: full_size

Related Fncs:

is_NAMED_STRING_ATTRIB

NAMED_VEC_ATTRIB

Class: Attributes, SAT Save and Restore

Purpose: Obsolete: use ATTRIB_GEN_VECTOR instead.

Derivation: NAMED_VEC_ATTRIB : NAMED_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : ACIS_OBJECT : –

SAT Identifier: “named_vec_attribute”

Filename: ga/ga_husk/pmhusk/nmv_attr.hxx

Description: This class defines and implements functions that provide named attributes with vector values.

Limitations: None

References: BASE SPAvector

Data:

None

Constructor:

```
public: NAMED_VEC_ATTRIB::NAMED_VEC_ATTRIB ();
```

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 inherited from the ENTITY class (for example, x=new NAMED_VEC_ATTRIB), because this reserves the memory on the heap, a requirement to support roll back and history management.

```
public: NAMED_VEC_ATTRIB::NAMED_VEC_ATTRIB (
    ENTITY* owner,           // entity owner
    const char* name,        // name of attribute
    const SPAvector& value   // vector value
    );
```

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

Destructor:

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

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 NAMED_VEC_ATTRIB(...) then later x–>lose.)

Methods:

```
public: virtual void NAMED_VEC_ATTRIB::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: SPAvector NAMED_VEC_ATTRIB::get_value ()
const;
```

Gets the value of a named vector attribute.

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

If level is unspecified or 0, returns the type identifier NAMED_VEC_ATTRIB_TYPE. If level is specified, returns NAMED_VEC_ATTRIB_TYPE for that level of derivation from ENTITY. The level of this class is defined as NAMED_VEC_ATTRIB_LEVEL.

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

Returns TRUE if this can be deep copied.

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

Returns TRUE if this is pattern compatible.

```
public: void NAMED_VEC_ATTRIB::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.

read_vector — Read vector from SAT file.

```
protected: void NAMED_VEC_ATTRIB::set_value (
    const SPAvector& val    // vector value
    );
```

Sets the value of a named vector attribute.

```
public: virtual void NAMED_VEC_ATTRIB::trans_owner (
    SPAtransf const&        // transformation
    );
```

Notifies the NAMED_VEC_ATTRIB that its owner is about to be transformed. The application has the chance to transform 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 transformation occurs.

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

Returns the string “named_vec_attribute”.

Related Fncs:

---

is_NAMED_VEC_ATTRIB