Chapter 2.
Functions

Topic: Ignore

The function interface is a set of Application Procedural Interface (API) and Direct Interface (DI) functions that an application can invoke to interact with ACIS. API functions, which combine modeler functionality with application support features such as argument error checking and roll back, are the main interface between applications and ACIS. The DI functions provide access to modeler functionality, but do not provide the additional application support features, and, unlike APIs, are not guaranteed to remain consistent from release to release. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

api_get_active_part

Function: Part Management

Action: Gets the active part.

Prototype:
```
PART* api_get_active_part ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "part/pmhusk/part.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_set_active_part

Function: Part Management

Action: Sets the part active.

Prototype:
```
void api_set_active_part (
    PART* this_part          // selected part
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″part/pmhusk/part.hxx″
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

api_initialize_part_manager

Function: Modeler Control, Part Management

Action: Initializes the part manager library.

Prototype:
```
outcome api_initialize_part_manager ();
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″kernel/kernapi/api/api.hxx″
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

api_part_add_entity

Function: Part Management

Action: Adds an ENTITY to a PART.

Prototype:
```
outcome api_part_add_entity (
    ENTITY* entity,          // entity to be added
    PART* part               // part to which to add
                             // entity
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/part.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/data/entity.hxx″
```

Description: This API adds a specified entity to a specified part. If the entity is already in a different PART, it is first removed from the old part. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_create

Function: History and Roll, Part Management

Action: Creates a new PART.

Prototype:
```
outcome api_part_create (
    unsigned int,            // initial size of entity
                             // table for part
    PART*& part              // returns part
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/part.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″part/pmhusk/api/part_api.hxx″
```

Description: This API creates a new part. It initially allocates enough space to contain the specified size (the number of entities). All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_delete

Function: Part Management

Action: Deletes a PART.

Prototype:
```
outcome api_part_delete (
    PART* part              // part
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”part/pmhusk/part.hxx”
#include ”part/pmhusk/api/part_api.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: This API deletes the specified part. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_delete_all_states

Function: History and Roll, Part Management

Action: Deletes all states.

Prototype:
```
outcome api_part_delete_all_states (
    HISTORY_STREAM* hs      // history stream to
        = NULL              // delete
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API deletes all operations defined using api_part_start_state and api_part_note_state for the given history stream. Use this API when clearing a part in preparation for loading or creating a new part. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_entities

Function: Part Management

Action: Gets a list of entities in a PART.

Prototype:
```
outcome api_part_entities (
    PART* part,              // part from which to get
                             // entities
    entity_filter* filter,   // filter used to select
                             // entities or NULL
    ENTITY_LIST& list        // returns list of
                             // entities found
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "part/pmhusk/part.hxx"
#include "kernel/geomhusk/efilter.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
```

Description: This API returns the list of entities fount in a part that match the specified filter. If filter is NULL, this API returns all entities in the part. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_part_entity_id

Function: Part Management

Action: Gets the entity ID and part for an ENTITY.

Prototype:
```
outcome api_part_entity_id (
    ENTITY* entity,          // entity to identify
    entity_id_t& id,         // entity to identify
    PART*& part              // part containing the
                             // entity
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "part/pmhusk/entityid.hxx"
#include "part/pmhusk/part.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/data/entity.hxx"
```

Description: This API returns the entity ID (id) and the part containing the specified entity. If the entity is not in the part, this API returns the entity ID (id) as 0 and the part as NULL. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_part_get_distribution_mode

Function: History and Roll

Action: Gets history distribution mode which is either TRUE or FALSE.

Prototype:
```
outcome api_part_get_distribution_mode (
    logical& distribute     // TRUE if history is
                            // distributed to part
                            // stream
                            // after or during
                            // note state
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: API gets the distribution mode for history streams. If set to TRUE, bulletins are distributed to the appropriate part's stream during note state.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

api_part_load

Function: Part Management

Action: Loads a file into a PART.

Prototype:
```
outcome api_part_load (
    FILE* fp,                   // file containing
                                // entities to load
    logical text_mode,          // TRUE (text) or
                                // FALSE (binary)
    PART* the_part,             // part in which to
                                // load entities
    logical with_history,       // TRUE to restore
                                // history if it
                                // exists in the file
    ENTITY_LIST& new_entities   // returns list of
                                // entities loaded
                                // into part
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″part/pmhusk/part.hxx″
#include ″baseutil/logical.h″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
```

Description: This API loads the entities defined in an open file fp into the specified part. The file must be open and positioned to the start of the entity data to be read. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_lookup_entity

Function: Part Management

Action: Gets an entity given an ID and a PART.

Prototype:
```
outcome api_part_lookup_entity (
    entity_id_t id,          // entity ID
    PART* part,              // part in which to look
                             // for entity
    ENTITY*& entity          // found entity
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″part/pmhusk/entityid.hxx″
#include ″part/pmhusk/part.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/data/entity.hxx″
```

Description: This API looks up an entity in a part given an entity id. If id does not exist in the part, this API returns NULL. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_part_name_state

Function: History and Roll, Part Management

Action: Names the current state.

Prototype:
```
outcome api_part_name_state (
    const char* name,        // name to give to
                             // current operation
    HISTORY_STREAM* hs       // returns history stream
        = NULL
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/bulletin/bulletin.hxx″
```

Description: This API assigns a name to the recent operation. Call api_part_name_state immediately after api_part_note_state and before opening the next sate with to api_part_start_state. api_part_name_state names the most recent noted state. api_part_name_state can also be called immediately following starting the modeler if it were desired that the “root” state be named. Use the specified name in calls to api_part_roll_to_state to roll to the start of the current operation. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

api_part_note_state

Function: History and Roll, Part Management

Action: Marks the end of a state.

Prototype:

```
outcome api_part_note_state (
    outcome status,          // outcome of operation
    int& depth               // depth of operation
                             // nesting after call
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description:

This API marks the end of an operation. Match calls to api_part_note_state with earlier calls to api_part_start_state. Pairs can be nested to create larger operations. A new delta state is created for the outermost call only. All api_part functions should be thought of as requiring the use of the PART class.

The calls to api_part_start_state and api_part_note_state must be strictly paired regardless of errors. Start state and note state are paired by the use of a static level counter. If the note state were skipped when there was an error, the counter would be off by one and subsequent states would not be noted.

```
int depth;
api_part_start_state(depth);
API_BEGIN

result = api_do_stuff_1(args);
check_outcome(result);     // If result is not ok,
                           // jump to API_END

// Alternate style of using check_outcome
check_outcome(api_do_stuff_2(args));

// Tell the part manager and graphics what happened
record_entity(new top level entity);
update_entity(modified top level entity);

API_END
api_part_note_state(outcome(API_SUCCESS),  depth);
```

If an error occurs, it will be caught by API_END. The api_part_note_state is always called regardless of error. Note that the outcome is checked before recording or updating entities, so the part manager and graphics don't see anything bad.

It is also acceptable to use API_SYS_BEGIN/END or EXCEPTION_BEGIN/TRY/CATCH/END with api_part_start_state in the EXCEPTION_BEGIN block and api_part_note_state in an EXCEPTION_CATCH(TRUE) block.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_part_remove_entity

Function: History and Roll, Part Management

Action: Removes an ENTITY from a part.

Prototype:
```
outcome api_part_remove_entity (
    ENTITY* entity          // entity to be removed
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/api/part_api.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/data/entity.hxx″
```

Description: This API removes an ENTITY from a part. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_roll_n_states

Function: History and Roll, Part Management

Action: Rolls forward or backward a specified number of states.

Prototype:
```
outcome api_part_roll_n_states (
    int n_wanted,           // number of states to
                            // roll
    HISTORY_STREAM* hs,     // history stream to roll
    int& n_actual           // returns actual number
                            // of states rolled
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API rolls a specified number (n_wanted) of states. A negative number rolls to an earlier state; a positive number rolls to a later state. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_roll_to_state

Function: History and Roll, Part Management

Action: Rolls to the start of a named state.

Prototype:
```
outcome api_part_roll_to_state (
    const char* name,       // name of state to which
                            // to roll
    HISTORY_STREAM* hs,     // history stream
    int& n_actual           // number of states
                            // actually rolled
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API rolls to the start of a named operation (name). If multiple operations have the same name, the latest one before the current state is used. If no operations with the given name occur before the current state, the first one after the current state is used. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_save

Function: Part Management

Action: Saves a PART to a file.

Prototype:
```
outcome api_part_save (
    FILE* fp,                 // file in which to save
                              // entities
    logical text_mode,        // TRUE (text) or
                              // FALSE (binary)
    PART* the_part,           // PART containing
                              // entities to save
    logical with_history      // TRUE to save history
        = 0,                  // stream to the file
    logical mainline_only     // TRUE to ignore rolled
        = 0                   // states
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "part/pmhusk/part.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
```

Description: This API saves the entities contained in a PART to an open file (fp). The file must be open and positioned to the location to which the entities are to be written.

If the optional with_history is specified as TRUE, roll back history data will be saved as well. If the optional mainline_only flag is specified as TRUE, only un–rolled states will be saved to the file. All api_part functions should be thought of as requiring the use of the PART class.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Changes model

api_part_set_distribution_mode

Function: History and Roll

Action: Sets history distribution on or off.

Prototype:
```
outcome api_part_set_distribution_mode (
    logical distribute      // TRUE if history is
                            // distributed to part
                            // stream after or during
                            // note state
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: This API sets the distribution mode for history streams. If set to TRUE, bulletins are distributed to the appropriate part's stream during note state.

The distribution mode can only be changed once. Further, once bulletins for a part history are created, the distribution mode cannot be changed. By default, distribution is FALSE.

If your application did not explicitly set the option distributed_history to FALSE (i.e., you either left it at its default value, which was TRUE, or explicitly set it to TRUE), you must now call api_part_set_distribution_mode to get the same behavior in your application.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

api_part_start_state

Function: History and Roll, Part Management

Action: Marks the start of a state.

Prototype:

```
outcome api_part_start_state (
    int& depth              // depth of nesting of
                            // operations after call
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "part/pmhusk/api/part_api.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description:

This API marks the start an operation. Match calls to api_part_start_state with later calls to api_part_note_state. Pairs may be nested to create larger operations. A new delta state is started for the outermost call only. All api_part functions should be thought of as requiring the use of the PART class.

The calls to api_part_start_state and api_part_note_state must be strictly paired regardless of errors. Start state and note state are paired by the use of a static level counter. If the note state were skipped when there was an error, the counter would be off by one and subsequent states would not be noted.

```
int depth;
api_part_start_state(depth);
API_BEGIN

result = api_do_stuff_1(args);
check_outcome(result);     // If result is not ok,
                           // jump to API_END

// Alternate style of using check_outcome
check_outcome(api_do_stuff_2(args));

// Tell the part manager and graphics what happened
record_entity(new top level entity);
update_entity(modified top level entity);

API_END
api_part_note_state(outcome(API_SUCCESS),  depth);
```

If an error occurs, it will be caught by API_END. The api_part_note_state is always called regardless of error. Note that the outcome is checked before recording or updating entities, so the part manager and graphics don't see anything bad.

It is also acceptable to use API_SYS_BEGIN/END or EXCEPTION_BEGIN/TRY/CATCH/END with api_part_start_state in the EXCEPTION_BEGIN block and api_part_note_state in an EXCEPTION_CATCH(TRUE) block.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_set_active_part

Function: Modeler Control, Part Management

Action: Sets the part active.

Prototype:
```
void api_set_active_part (
    PART* this_part          // selected part
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”part/pmhusk/api/part_api.hxx”
#include ”part/pmhusk/part.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: Read–only

api_terminate_part_manager

Function: Modeler Control, Part Management

Action: Terminates the part manager library.

Prototype:
```
outcome api_terminate_part_manager ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”part/pmhusk/api/part_api.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/api/part_api.hxx

Effect: System routine

count_parts

Function: Part Management

Action: Counts and returns the number of parts.

Prototype:
```
int count_parts ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/part.hxx"
```

Description: Counts and returns the number of parts. The next_id also increments.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/part.hxx

Effect: Read–only

find_ID_ATTRIB

Function: Part Management

Action: Finds the ID attribute for an entity.

Prototype:
```
ID_ATTRIB* find_ID_ATTRIB (
    ENTITY* ent             // entity
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/id_attr.hxx"
#include "kernel/kerndata/data/entity.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/id_attr.hxx

Effect: Read–only

find_part

Function: Part Management

Action: Finds a PART given the PART's ID but not its handle.

Prototype:
```
PART* find_part (
    unsigned int id          // part id
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "part/pmhusk/part.hxx"
```

Description: Looks up a PART given the PART's ID but not its handle. If the user knows the PART's handle, use the Part method of part_handle to get the PART instead of this procedure because it is faster and more reliable.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/part.hxx

Effect: Read–only

get_entity_callback_list

Function: Callbacks, Part Management

Action: Gets a global list of entity callbacks.

Prototype:
```
entity_callback_list& get_entity_callback_list ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”part/pmhusk/ent_cb.hxx”
```

Description: Refer to action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/ent_cb.hxx

Effect: Read–only

get_next_part

Function: Part Management

Action: Gets the next PART with an ID that is greater than or equal to the next_id.

Prototype:
```
PART* get_next_part (
    int& index              // part ID
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”part/pmhusk/part.hxx”
```

Description: Refer to action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/part.hxx

Effect: Read–only

get_part

Function: Part Management

Action: Gets the PART for the specified ENTITY.

Prototype:
```
PART* get_part (
    const ENTITY* ent       // entity
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/part.hxx″
#include ″kernel/kerndata/data/entity.hxx″
```

Description: Refer to action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/part.hxx

Effect: Read–only

get_roll_callback_list

Function: History and Roll, Callbacks, Part Management

Action: Gets a list of roll callbacks.

Prototype:
```
roll_callback_list& get_roll_callback_list ();
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″part/pmhusk/roll_cb.hxx″
```

Description: Gets a list of roll callbacks. Use this method to add a new callback to the list.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/roll_cb.hxx

Effect: Read–only

is_ID_ATTRIB

Function: Part Management, SAT Save and Restore

Action: Determines if the entity is an `ID_ATTRIB`.

Prototype:
```
logical is_ID_ATTRIB (
    const ENTITY* e         // entity to test
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kerndata/data/entity.hxx”
#include ”part/pmhusk/id_attr.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/id_attr.hxx

Effect: Read–only

is_UNITS_SCALE

Function: SAT Save and Restore

Action: Determines if the entity is an UNITS_SCALE.

Prototype:
```
logical is_UNITS_SCALE (
    const ENTITY* ent       // entity to test
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kerndata/data/entity.hxx”
#include ”part/pmhusk/units.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: part

Filename: part/part/pmhusk/units.hxx

Effect: Read–only