Chapter 1.
Persistent ID Component

Component: *Persistent ID

The Persistent ID Component (PID), in the pid directory, generates an identifier that is attached to an entity and is retained by the entity from session to session. The identifier is designed to be unique over all sessions of ACIS. The user has great flexibility in controlling the migration of the identifier through ACIS processing. The identifier is attached to an entity by ACIS attributes.

The PID identifier (persistent ID) consists of:

- An 8 character session user identifier
- The time of session start
- The index of the persistent ID within the session
- A copy index for the identifier

Accessing Persistent ID Data

Topic: *Persistent ID

When an ACIS session is initialized, a base is established for the persistent IDs of this session. The user can control the initialization process by replacing the pid_base_init function with the desired functionality. The default action is as follows:

- Set the user identifier from the USER environment variable
- Set the system time as the number of seconds since January 1, 1970
- Set the entity index to 0
- Set the copy number to 1

The persistent ID is attached to an entity by a PID attribute. The PID attribute contains an object of the pid_base class. Only one PID attribute is allowed per entity. The user accesses the persistent ID data of an entity using the following class methods:

get_base_name . . . . . Returns the user name of the entity's identifier.

get_time_val . . . . . Returns the time of the entity's identifier.

get_index . . . . . Returns the index of the entity's identifier.

get_copy_num . . . . . Returns the copy number of the entity's identifier.

On the entity level, the following class methods are available:

remove_pid . . . . . Removes persistent ID from this entity.

set_next_current_pid . . . . . Adds persistent ID to this entity based on the current base for this session, and increments the index.

API Interface

Topic: *Persistent ID

The following APIs are used for PID:

- api_pidset
- api_pidremove
- api_pidget

api_pidset

Topic: Persistent ID

The api_pidset function creates a persistent ID for a given entity, provided one does not already exist. The new identifier is generated from the current session user name and time, and the entity index is incremented by one.

api_pidremove

Topic: Persistent ID

The api_pidremove function removes the persistent ID from a given entity, provided one exists. No attempt is made to reuse this persistent ID for a subsequent entity. For example, refer to the following sequence of function calls:

1. api_pidset(A)

   Entity *A* is given a persistent ID; this ID is associated with index 1.

2. api_pidremove(A)

   Entity *A* has no persistent ID (so, no index).

3. api_pidset(B)

Entity *B* is given a persistent ID with index 2 (the ID associated with index 1 is not reused).

4. api_pidset(A)

   Entity *A* is given a persistent ID with index 3 (the ID associated with index 1 is not reused).

api_pidget

Topic: Persistent ID

The function api_pidget returns the pid_base of the given entity, provided one exists. The data can then be manipulated. For example, the following code fragment gets the pid_base for an entity, then prints the PID data of an entity using the pid_base class methods.

```
const pid_base* P;

api_pidget(Ent, P);

printf(" pid user %s\n", P->get_user());
printf(" pid time %d\n", P->get_time_val());
printf(" pid index %d\n", P->get_index());
printf(" pid copy %d\n", P->get_copy_num());
```

Customizing PID Functions

Topic: *Persistent ID

The user can control certain PID activities by writing a customized version of the function for the corresponding activity, and then linking this version first. PID has been designed so the user can replace the existing functions with their own versions to provide maximum flexibility and control over the behavior of persistent IDs. The functions that a user may want to customize include:

pid_base_init . . . . . Initializes a session.

get_next_current_pid . . . . . Gets the next persistent ID for this session.

pid_attrib_copy . . . . . Copies an entity.

pid_attrib_restore . . . . . Restores an entity.

pid_attrib_save . . . . . Saves an entity.

merge_owner . . . . . Merges an entity.

split_owner . . . . . Splits an entity.

For example, the following customized version of the pid_base_init function is used to initialize the persistent ID. The base of the persistent IDs for this session includes the user field, Testing, instead of the session user time and field zero.

```
#include ”pid/pid.hxx”         //set the fields in the base
void pid_base_init(
    pid_base* session_base)    // basis pid for this session
{

// create_pid_base_name is a method of the pid_base class
create_pid_base_name(”Testing”,  base–>base_name);
session_base–>time_val = 0;
session–index = 0;
session–>copy_num = 1;
}
```

The following are the default actions for the functions:

pid_base_init . . . . . Defines the name by the environment variable, USER. The time is the number of seconds since January 1, 1970. The index is set to 0, and the copy number is set to 1.

get_next_current_pid . . . . . Determines the name and time by the session_base. The index of the session_base is incremented. The copy number is 1.

pid_attrib_copy . . . . . Sets the persistent ID of the copy to be returned from get_next_current_pid.

pid_attrib_restore . . . . . Restores the attribute saved with this entity. No attempt is made to prevent duplication of an entity with the same persistent ID if it is restored twice.

pid_attrib_save . . . . . Saves the attribute attached to this entity.

merge_owner . . . . . No default action.

split_owner . . . . . No default action.