Solaris 10 Resource Manager Developer's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–1975–10
January, 2005

Adobe PostScript™

040929@9495

Contents

Preface

The *Solaris 10 Resource Manager Developer's Guide* describes how to write applications that partition and manage system resources such as processor sets and scheduling class. This book references the programming APIs provided to partition, schedule, and set bounds on the consumption of system resources. Use the programming APIs to make the configuration of resources persistent. This book provides programming examples and a discussion of programming issues to consider when writing an application.

Note – This Solaris™ release supports systems that use the SPARC® and x86 families of processor architectures: UltraSPARC®, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems appear in the *Solaris 10 Hardware Compatibility List* at `http://www.sun.com/bigadmin/hcl`. This document cites any implementation differences between the platform types.

In this document the term "x86" refers to 64-bit and 32-bit systems manufactured using processors compatible with the AMD64 or Intel Xeon/Pentium product families. For supported systems, see the *Solaris 10 Hardware Compatibility List*.

Who Should Use This Book

Application developers and ISVs who write applications that control or monitor system resources. These applications benefit from using, controlling, or tracking system resources for the Solaris operating environment.

Before You Read This Book

For an overview of zones and a discussion of when to use zones, see Chapter 16, “Introduction to Solaris Zones,” in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

For an overview of resource management and a discussion of when to use resource management, see Part I, “Resource Management,” in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

How This Book Is Organized

The *Solaris 10 Resource Manager Developer’s Guide* has the following organization:

Chapter 1 introduces the Solaris 10 Resource Manager.

Chapter 2 describes the projects and tasks facility, provides example code, and discusses programming issues.

Chapter 3 describes the C interface for the extended accounting facility, provides example code, and discusses programming issues.

Chapter 4 describes the perl interface for the extended accounting facility, provides example code, and discusses programming issues.

Chapter 5 describes resource controls, provides example code, and discusses programming issues.

Chapter 6 describes resource pools. Sample code provides examples of how to implement resource pools.

Related Books

For introductory information about the resource manager facility and examples of use its administration commands, see Part I, “Resource Management,” in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print. For a list of documents and how to order them, see "Buy printed documentation" at `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	Command-line placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.

TABLE P–1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. Perform a *patch analysis*. Do *not* save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Resource Management

This chapter discusses the following:

Resource Management

The resource management facility enables you to control how applications use available system resources. You can do the following:

- Allocate computing resources, such as processor time.
- Monitor how these allocations are being used. Adjust these allocations, as necessary.
- Generate extended accounting information for analysis, billing, and capacity planning.

For an overview of resource management, a discussion of when to use its facilities, and a description of how to create a resource management task map, see

- "When to Use Resource Management" in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*
- "Setting Up Resource Management (Task Map)" in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*

Management Applications

Management applications fall into the following categories:

Resource monitoring
: Use extended accounting, `exacct`, to monitor system resource usage for capacity planning.

Resource accounting or billing
: Use `exacct` to process resource usage data by workload for billing.

Resource administration
: Configure resource pools, projects, `rctls`, and FSS.

Managed Applications

Managed applications fall into the following categories:

Resource constrained
: `rctls`, `pools`, or FSS defined by projects.

Resource aware
: Handle a limit that is encountered, such as notification resulting from `rctls` equal to a particular value.

Resource advisory
: Provide hints of resource needs.

Components of the Resource Manager

The components of the Solaris Resource Manager include the:

- Projects and tasks facility. See Chapter 2.
- Extended accounting facility. See Chapter 3 and Chapter 4.
- Resource controls facility. See Chapter 5.
- Resource pools facility. See Chapter 6.

CHAPTER 2

Projects and Tasks

The chapter provides information about projects and tasks.

Overview of Projects and Tasks

A task is a collection of processes that represents a workload component. A project is a collection of tasks that represents an entire workload. At any given time, a process can be a component of only one task and one project.

For an overview of projects and tasks see Chapter 2, "Projects and Tasks (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*. For example commands for administering projects and tasks, see Chapter 3, "Administering Projects and Tasks," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

A user that is a member of more than one project can run processes in multiple projects at the same time. All processes that are started by a process inherit the project of the parent process. When you switch to a new project in a startup script, all child processes run in the new project.

An executing user process has an associated user identity (`uid`), group indentity (`gid`), and project identity (`projid`). Process attributes and abilities are inherited from the user, group, and project identities to form the execution context for a task.

project Structure

The project structure describes the project.

```
struct project {
  char      *pj_name;       /* name of the project */
  projid_t   pj_projid;     /* numerical project id */
  char      *pj_comment;    /* project comment */
  char     **pj_users;      /* vector of pointers to project user names */
  char     **pj_groups;     /* vector of pointers to project group names */
  char      *pj_attr;       /* project attributes */
};
```

The project structure members include:

`*pj_name`
Name of the project.

`pj_projid`
Project ID.

`*pj_comment`
User-supplied project description.

`**pj_users`
Pointers to project user members.

`**pj_groups`
Pointers to project group members.

`*pj_attr`
Project attributes. Use these attributes to set values for resource controls and project.pool.

/etc/project File

The project database is maintained on a system or network either through the /etc/project file or through a network information service, such as NIS or LDAP.

/etc/project contains five standard projects.

system	Used for all system processes and daemons.
user.root	All root processes run in the user.root project.
noproject	Special project for IPQoS.
default	Serves as a catchall for users not matching any other projects.
group.staff	Used for all users in the group staff.

Process Controls

`process.max-port-events`
: `process.max-port-events` specifies the maximum allowable number of events per event port.

`process.min-crypto-sessions`
: When `/dev/crypto` is opened, a fixed-sized session table is allocated. `process.min-crypto-sessions` specifies the number of sessions in this table. The default value is 20. It can only be changed by a privileged process.

`process.add-crypto-sessions`
: When a session table is full, a larger table is allocated. `process.add-crypto-sessions` specifies the number of additional sessions. The default value is 20. Any process can change this value.

`process.max-crypto-sessions`
: `process.max-crypto-sessions` specifies the maximum number of sessions in a session table. The initial session table can be increased up to this maximum value. The default value is 100. This value can only be changed by a privileged process.

`process.crypto-buffer-limit`
: `process.crypto-buffer-limit` limits the number of bytes that can be allocated for `copyin` of user data. The sizes of all the buffers allocated for `copyin` are added together and the result is checked against this resource control. This limit applies to each instance of `/dev/crypto`. The default value for this resource control is 100,000. This value can only be changed by a privileged process.

Project Controls

`project.max-device-locked-memory`
: `project.max-device-locked-memory` specifies the total amount of locked memory allowed.

`project.max-port-ids`
: `project.max-port-ids` specifies the maximum allowable number of event ports.

Project and Task Application Programming Interface

This section discusses the API associated with project creation and project database querying.

Project Creation and Project Database Querying Functions

The following list contains the functions associated with project creation and project database querying. The function name is a link to the corresponding man page.

`setproject`(3PROJECT)
`setprojent`(3PROJECT)
`getdefaultproj`(3PROJECT)
`inproj`(3PROJECT)
`getprojent`(3PROJECT)
`fgetprojent`(3PROJECT)
`getprojbyname`(3PROJECT)
`getprojbyid`(3PROJECT)
`getprojbyname`(3PROJECT)
`endprojent`(3PROJECT)

Code Examples to Access the Project Database

This section provides code examples for accessing project database entries.

Print the First Three Fields of Each Entry of the Project Database

The following example prints the first three fields of each entry of the project database.

The key points of the example include the following:

- `setprojent()` rewinds the project database to start at the beginning. `endprojent()` closes the project database and frees resources.
- Call `getprojent()` with a conservative maximum buffer size that is defined in `project.h`.

```
#include <project.h>

struct project projent;
char buffer[PROJECT_BUFSZ]; /* Use safe buffer size from project.h */
    ...
struct project *pp;
```

```
setprojent();  /* Rewind the project database to start at the beginning */

while (1) {
   pp = getprojent(&projent, buffer, PROJECT_BUFSZ);
      if (pp == NULL)
          break;
    printf("%s:%d:%s\n", pp->pj_name, pp->pj_projid, pp->pj_comment);
        ...
};

endprojent();   /* Close the database and free project resources */
```

Get a Project Database Entry That Matches the Caller's project-id

The following example calls `getprojbyid()` to get a project database entry that matches the caller's project-id. The example then prints the project name and the project ID.

The key point of the example is to call `getprojbyid()` to get an entry from the project database that matches the caller's project-id.

```
#include <project.h>

struct project *pj;
char buffer[PROJECT_BUFSZ]; /* Use safe buffer size from project.h */

main()
{
   projid_t pjid;
   pjid = getprojid();
   pj = getprojbyid(pjid, &projent, buffer, PROJECT_BUFSZ);
   if (pj == NULL) {
       /* fail; */
   }
   printf("My project (name, id) is (%s, %d)\n", pp->pj_name, pp->pj_projid);
}
```

Programming Issues for Project Database Applications

Consider the following issues when writing your application:

- No function exists to explicitly create a new project.

- A user cannot login if no default project for the user can be found in the project database.
- A new task in the user's default project is created when the user logs in.
- Process association with a new project applies the new project's resource controls and pools membership to the process.
- `setproject()` requires privilege. The `newtask -c` command does not. Either can be used to create a task, but only `newtask` can change the project of a running process.
- No parent/child relationship exists between tasks.
- Finalized tasks can be created by using `setproject()` to associate the caller with a new project. Finalized tasks are useful when trying to accurately estimate aggregate resource accounting.
- The reentrant functions, `getprojent()`, `getprojbyname()`, `getprojbyid()`, `getdefaultproj()`, and `inproj()` use buffers supplied by the caller to store returned results. These functions are safe for use in both single-threaded applications and multithreaded applications.
- Reentrant functions require the additional arguments `proj`, `buffer`, and `bufsize`. The `proj` argument must be a pointer to a `project` structure allocated by the caller. On successful completion, these functions return the project entry in this structure. Storage referenced by the `project` structure is allocated from the memory specified by the `buffer` argument. `bufsize` specifies the size in number of bytes.
- If an incorrect buffer size is used, `getprojent()` returns NULL and sets `errno` to ERANGE.

CHAPTER 3

Extended Accounting, C Interface

This chapter describes the C interface for extended accounting:

Overview of Extended Accounting

Project and task should be used to label and separate workloads. Use the extended accounting subsystem to monitor resource consumption by workloads that are running on the system. Extended accounting produces accounting records for the workload tasks and processes.

For an overview of extended accounting and example commands for administering extended accounting, see Chapter 4, "Extended Accounting (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones* and Chapter 5, "Administering Extended Accounting (Tasks)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

Extended Accounting Application Programming Interface

The extended accounting API contains functions that perform:

- `exacct` system calls
- Operations on the `exacct` file

- Operations on `exacct` objects
- Miscellaneous

`exacct` System Calls

The following table lists the system calls to interact with the extended accounting subsystem. The function name is a link to the corresponding man page.

`putacct`(2)
`getacct`(2)
`wracct`(2)

TABLE 3–1 Extended Accounting System Calls

Link to man page	Description
`putacct`(2)	Provides privileged processes the ability to tag accounting records with additional data specific to that process.
`getacct`(2)	Enables privileged processes to request extended accounting buffers from the kernel for currently executing tasks and processes.
`wracct`(2)	Privileged process request to the kernel to write, given its internal state of resource usage, the appropriate data for the specified task or process.

Operations on the `exacct` File

The following table lists the functions that provide access to the `exacct` files. The function name is a link to the corresponding man page.

`ea_open`(3EXACCT)
`ea_close`(3EXACCT)
`ea_get_object`(3EXACCT)
`ea_write_object`(3EXACCT)
`ea_next_object`(3EXACCT)
`ea_previous_object`(3EXACCT)
`ea_get_hostname`(3EXACCT)
`ea_get_creator`(3EXACCT)

Operations on `exacct` Objects

The following table lists the functions that access `exacct` objects. The function name is a link to the corresponding man page.

`ea_set_item`(3EXACCT)
`ea_set_group`(3EXACCT)
`ea_match_object_catalog`(3EXACCT)
`ea_attach_to_object`(3EXACCT)
`ea_attach_to_group`(3EXACCT)
`ea_free_item`(3EXACCT)
`ea_free_object`(3EXACCT)

Miscellaneous Operations

The following table lists the functions associated with miscellaneous operations. The function name is a link to the corresponding man page.

`ea_error`(3EXACCT)
`ea_match_object_catalog`(3EXACCT)

Code Examples for Accessing `exacct` Files

This section provides code examples for accessing `exacct` files.

Figure 3–1 shows the data flow for the `ea_unpack_object()` and `ea_pack_object()` functions.

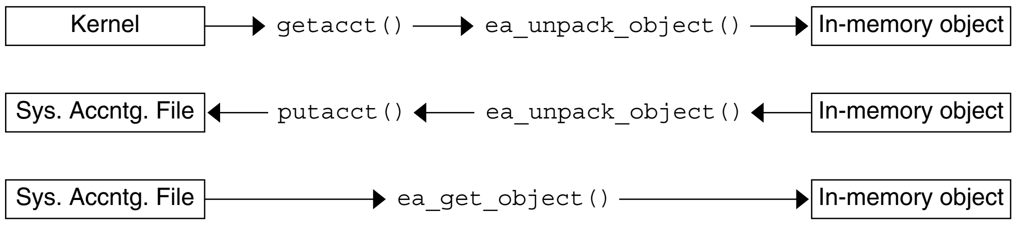


FIGURE 3–1 Data Flow for `ea_unpack_object()` and `ea_pack_object()`

Display the `exacct` Data for a Designated pid

The following example displays a specific pid's `exacct` data snapshot from the kernel.

```
...
  ea_object_t *scratch;
  int unpk_flag = EUP_ALLOC;  /* use the same allocation flag */
                              /* for unpack and free */

  /* Omit return value checking, to keep code samples short */

  bsize = getacct(P_PID, pid, NULL, 0);
  buf = malloc(bsize);

  /* Retrieve exacct object and unpack */
  getacct(P_PID, pid, buf, bsize);
  ea_unpack_object(&scratch, unpk_flag, buf, bsize);

  /* Display the exacct record */
  disp_obj(scratch);
  if (scratch->eo_type == EO_GROUP) {
         disp_group(scratch);
  }
  ea_free_object(scratch, unpk_flag);
        ...
```

Identify Individual Tasks During a Kernel Build

This example evaluates kernel builds and displays a string that describes the portion of the source tree being built by this task make. Display the portion of the source being built to aid in the per-source-directory analysis.

The key points of this example include the following:

- To aggregate the time for a `make`, with possibly many processes, each `make` is fired off as a task. Child `make` processes that arise are different tasks. To aggregate across the make tree, the parent-child task relationship must be identified.
- Add a tag with this information to the task's `exacct` file. Add a current working directory string that describes the portion of the source tree being built by this task make.

  ```
  ea_set_item(&cwd, EXT_STRING | EXC_LOCAL | MY_CWD,
                                  cwdbuf, strlen(cwdbuf));
  ```

```
  ...
/* Omit return value checking and error processing */
/* to keep code sample short */
ptid = gettaskid();    /* Save "parent" task-id */
tid = settaskid(getprojid(), TASK_NORMAL);    /* Create new task *

/* Set data for item objects ptskid and cwd */
ea_set_item(&ptskid, EXT_UINT32 | EXC_LOCAL | MY_PTID, &ptid, 0);
ea_set_item(&cwd, EXT_STRING | EXC_LOCAL | MY_CWD, cwdbuf, strlen(cwdbuf));

/* Set grp object and attach ptskid and cwd to grp */
```

```
ea_set_group(&grp, EXT_GROUP | EXC_LOCAL |  EXD_GROUP_HEADER);
ea_attach_to_group(&grp, &ptskid);
ea_attach_to_group(&grp, &cwd);

/* Pack the object and put it back into the accounting stream */
ea_buflen = ea_pack_object(&grp, ea_buf, sizeof(ea_buf));
putacct(P_TASKID, tid, ea_buf, ea_buflen, EP_EXACCT_OBJECT);

/* Memory management: free memory allocate in ea_set_item */
ea_free_item(&cwd, EUP_ALLOC);  /* free memory allocated in ea_set_item */
  ...
```

Read and Display the Contents of a System `exacct` File

This example shows how to read and display a system `exacct` file for a process or a task.

The key points of this example include the following:

- Call `ea_get_object()` to get the next object in the file. Call `ea_get_object()` in a loop until EOF enables a complete traversal of the `exacct` file.
- `catalog_name()` uses the `catalog_item` structure to convert a Solaris catalog's type id to a meaningful string that describes the content of the object's data. The type id is obtained by masking the lowest 24 bits, or 3 bytes.

  ```
  switch(o->eo_catalog & EXT_TYPE_MASK) {
    case EXT_UINT8:
        printf(" 8: %u", o->eo_item.ei_uint8);
        break;
    case EXT_UINT16:
    ...
  }
  ```

- The upper 4 bits of TYPE_MASK are used to find out the data type to print the object's actual data.
- `disp_group()` takes a pointer to a group object and the number of objects in the group. For each object in the group, `disp_group()` calls `disp_obj()` and recursively calls `disp_group()` if the object is a group object.

```
/* Omit return value checking and error processing */
/* to keep code sample short */
main(int argc, char *argv)
{
  ea_file_t ef;
  ea_object_t scratch;
  char *fname;

  fname = argv[1];
  ea_open(&ef, fname, NULL,  EO_NO_VALID_HDR, O_RDONLY, 0);
```

```
  bzero(&scratch, sizeof (ea_object_t));
  while (ea_get_object(&ef, &scratch)  != -1) {
         disp_obj(&scratch);
         if (scratch.eo_type == EO_GROUP)
             disp_group(&ef, scratch.eo_group.eg_nobjs);
         bzero(&scratch, sizeof (ea_object_t));
  }
  ea_close(&ef);
}

struct catalog_item {   /* convert Solaris catalog's type id */
                        /* to a meaningful string */
     int    type;
     char *name;
 } catalog[] = {
     { EXD_VERSION,     "version\t" },
     ...
     { EXD_PROC_PID,    "  pid\t" },
     ...
   };

 static char *
 catalog_name(int type)
 {
     int i = 0;
     while (catalog[i].type != EXD_NONE) {
         if (catalog[i].type == type)
             return (catalog[i].name);
         else
             i++;
     }
     return ("unknown\t");
 }

 static void disp_obj(ea_object_t *o)
 {
     printf("%s\t", catalog_name(o->eo_catalog & 0xffffff));
     switch(o->eo_catalog & EXT_TYPE_MASK) {
     case EXT_UINT8:
         printf(" 8: %u", o->eo_item.ei_uint8);
         break;
     case EXT_UINT16:
     ...
 }
 static void disp_group(ea_file_t *ef, uint_t nobjs)
 {
     for (i = 0; i < nobjs; i++) {
         ea_get_object(ef, &scratch));
                  disp_obj(&scratch);
         if (scratch.eo_type == EO_GROUP)
                disp_group(ef, scratch.eo_group.eg_nobjs);
        }
 }
```

CHAPTER 4

Extended Accounting Perl Interface

The perl interface provides a perl binding to the extended accounting tasks and projects. The interface allows the accounting files produced by the `exacct` framework to be read by perl scripts. The interface also allows the writing of `exacct` files by perl scripts.

This chapter has the following organization:

Extended Accounting Overview

`exacct` is a new accounting framework for the Solaris operating environment that provides additional functionality to that provided by the traditional SVR4 accounting mechanism. Traditional SVR4 accounting suffers from several drawbacks:

- The data collected by SVR4 accounting cannot be modified.

  The type or quantity of statistics SVR4 accounting gathers cannot be customized for each application. Changes to the data SVR4 accounting collects would break all the existing consumers of the accounting files.
- The SVR4 accounting mechanism is not open.

  Applications cannot embed their own data in the system accounting data stream.
- The SVR4 accounting mechanism has no aggregation facilities.

  The Solaris operating environment writes an individual record for each process that exits. No facilities are provided for grouping sets of accounting records into higher level aggregates.

For an overview of extended accounting, see Chapter 4, "Extended Accounting (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

Perl Interface to `libexacct`(3LIB)

The perl `exacct` framework addresses the deficiencies of SVR4 accounting and provides a configurable, open, and extensible framework for the collection of accounting data.

- The data that is collected can be configured using the `exacct` API.
- Applications can either embed their own data inside the system accounting files, or create and manipulate their own custom accounting files.
- The introduction of two new concepts overcomes the lack of data aggregation facilities in the traditional accounting mechanism. The new concepts are *tasks* and *projects*. Tasks identify a set of processes which are to be considered as a unit of work. Projects allow the processes executed by a set of users to be aggregated into a higher-level entity. See `project`(4) for more details about tasks and projects.

Object Model

The `Sun::Solaris::Exacct` module is the parent of all the classes provided by this library. `libexacct`(3LIB) provides operations on several types of entities: `exacct` format files, `catalog` tags and `exacct` objects. `exacct` objects are subdivided into two types.

- Items

  Single data values
- Groups

  Lists of Items

Design Criteria

The perl extensions to extended accounting provide a perl interface to the underlying libexacct(3LIB) API.

- Full equivalence to C API

  Provide a perl interface that is functionally equivalent to the underlying C API. The goal is to provide a mechanism for accessing exacct files that does not require C coding. All the functionality that is available from C is also available via the perl interface.

- Ease of use

  Data obtained from the underlying C API is presented as perl data types. Perl data types ease access to the data, and remove the need for buffer pack and unpack operations.

- Automated memory management

  The C API requires that the programmer take responsibility for managing memory when accessing `exacct` files. Memory management takes the form of passing the appropriate flags to functions, such as `ea_unpack_object`(3EXACCT), and explicitly allocating buffers to pass to the API. The perl API removes these requirements, as all memory management is performed by the perl library.

- Prevent incorrect use of API

  The `ea_object_t` structure provides the in-memory representation of `exacct` records. The `ea_object_t` structure is a union type, used for manipulating both Group and Item records. As a result, an incorrectly typed structure can be passed to some of the API functions. The addition of a class hierarchy prevents this type of programming error.

Perl Double-typed Scalars

The modules described in this document make extensive use of the perl *double-typed scalar* facility. The double-typed scalar facility allows a scalar value to behave either as an integer or as a string, depending upon context. This behavior is the same as exhibited by the `$!` perl variable (`errno`). The double-typed scalar facility avoids the need to map from an integer value into the corresponding string in order to display a value. The following example illustrates the use of double-typed scalars.

```
# Assume $obj is a Sun::Solaris::Item
my $type = $obj->type();

# prints out "2 EO_ITEM"
printf("%d %s\n", $type, $type);

# Behaves as an integer, $i == 2
my $i = 0 + $type;

# Behaves as a string, $s = "abc EO_ITEM xyx"
my $s = "abc $type xyz";
```

Perl Modules

The various project, task and exacct-related functions have been separated into groups, and each placed in a separate perl module. Each function has the SMI standard `Sun::Solaris::` perl package prefix. The following table summarizes the modules with links to detailed descriptions of each module.

TABLE 4–1 Perl Modules

Module	Description
“`Sun::Solaris::Project` Module” on page 29	Provides functions to access the project manipulation functions: `getprojid`(2), `setproject`(3PROJECT), `project_walk`(3PROJECT), `getprojent`(3PROJECT), `getprojbyname`(3PROJECT), `getprojbyid`(3PROJECT), `getdefaultproj`(3PROJECT), `inproj`(3PROJECT), `getprojidbyname`(3PROJECT), `setprojent`(3PROJECT), `endprojent`(3PROJECT), `fgetprojent`(3PROJECT).
“`Sun::Solaris::Task` Module” on page 31	Provides functions to access the task manipulation functions `settaskid`(2), `gettaskid`(2).
“`Sun::Solaris::Exacct` Module” on page 32	Top-level `exacct` module. Functions in this module access both the `exacct`-related system calls `getacct`(2), `putacct`(2), and `wracct`(2) as well as the `libexacct`(3LIB) library function `ea_error`(3EXACCT). This module contains constants for all the various `exacct` `EO_*`, `EW_*`, `EXR_*`, `P_*`, `TASK_*` macros.
“`Sun::Solaris::Exacct::Catalog` Module” on page 33	Provides object-oriented methods to access the bitfields within an `exacct` catalog tag as well as the `EXC_*`, `EXD_*`, `EXD_*` macros.

TABLE 4–1 Perl Modules *(Continued)*

Module	Description
"Sun::Solaris::Exacct::File Module" on page 35	Provides object-oriented methods to access the libexacct(3LIB) accounting file functions: ea_open(3EXACCT), ea_close(3EXACCT), ea_get_creator(3EXACCT), ea_get_hostname(3EXACCT), ea_next_object(3XACCT), ea_previous_object(3EXACCT), ea_write_object(3EXACCT).
"Sun::Solaris::Exacct::Object Module" on page 37	Provides object-oriented methods to access the individual exacct accounting file object. An exacct object is represented as an opaque reference that is blessed into the appropriate Sun::Solaris::Exacct::Object subclass. This module is further subdivided into the two types of possible object: Item and Group. Methods are also provided to access the ea_match_object_catalog (3EXACCT), ea_attach_to_object (3EXACCT) functions.
"Sun::Solaris::Exacct::Object::Item Module" on page 38	Provides object-oriented methods to access an individual exacct accounting file Item. Objects of this type inherit from Sun::Solaris::Exacct::Object.
"Sun::Solaris::Exacct::Object::Group Module" on page 39	Provides object-oriented methods to access an individual exacct accounting file Group. Objects of this type inherit from Sun::Solaris::Exacct::Object, and provide access to the ea_attach_to_group(3EXACCT) function. The Items contained within the Group are presented as a perl array.
"Sun::Solaris::Exacct::Object::_Array Module" on page 40	Private array type, used as the type of the array within a Sun::Solaris::Exacct::Object::Group.

Sun::Solaris::Project Module

The Sun::Solaris::Project module provides wrappers for the project-related system calls and the libproject(3LIB) library.

`Sun::Solaris::Project` Constants

The `Sun::Solaris::Project` module uses constants from the project-related header files.

MAXPROJID
PROJNAME_MAX
PROJF_PATH
PROJECT_BUFSZ
SETPROJ_ERR_TASK
SETPROJ_ERR_POOL

`Sun::Solaris::Project` Functions, Class Methods, and Object Methods

The perl extensions to the `libexacct`(3LIB) API provide the following functions for projects. The function name is a link to the corresponding man page.

`setproject`(3PROJECT)
`setprojent`(3PROJECT)
`getdefaultproj`(3PROJECT)
`inproj`(3PROJECT)
`getprojent`(3PROJECT)
`fgetprojent`(3PROJECT)
`getprojbyname`(3PROJECT)
`getprojbyid`(3PROJECT)
`getprojbyname`(3PROJECT)
`endprojent`(3PROJECT)

The `Sun::Solaris::Project` module has no class methods.

The `Sun::Solaris::Project` module has no object methods.

`Sun::Solaris::Project` Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants and functions defined in this module.

Tag	Constant or Function
:SYSCALLS	`getprojid()`

Tag	Constant or Function
:LIBCALLS	`setproject()`, `activeprojects()`, `getprojent()`, `setprojent()`, `endprojent()`, `getprojbyname()`, `getprojbyid()`, `getdefaultproj()`, `fgetprojent()`, `inproj()`, `getprojidbyname()`
:CONSTANTS	MAXPROJID_TASK, PROJNAME_MAX, PROJF_PATH, PROJECT_BUFSZ, SETPROJ_ERR, SETPROJ_ERR_POOL
:ALL	:SYSCALLS, :LIBCALLS, :CONSTANTS

`Sun::Solaris::Task` Module

The `Sun::Solaris::Task` module provides wrappers for the `settaskid`(2) and `gettaskid`(2) system calls.

`Sun::Solaris::Task` Constants

The `Sun::Solaris::Task` module uses the following constants.

TASK_NORMAL
TASK_FINAL

`Sun::Solaris::Task` Functions, Class Methods, and Object Methods

The perl extensions to the `libexacct`(3LIB) API provides the following functions for tasks.

`settaskid`(2)
`gettaskid`(2)

The `Sun::Solaris::Task` module has no class methods.

The `Sun::Solaris::Task` module has no object methods.

Sun::Solaris::Task Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants and functions defined in this module.

Tag	Constant or Function
:SYSCALLS	settaskid(),gettaskid()
:CONSTANTS	TASK_NORMAL, TASK_FINAL
:ALL	:SYSCALLS, :CONSTANTS

Sun::Solaris::Exacct Module

The Sun::Solaris::Exacct module provides wrappers for the ea_error(3EXACCT) function, and for all the exacct system calls.

Sun::Solaris::Exacct Constants

The Sun::Solaris::Exacct module provides constants from the various exacct header files. The P_PID, P_TASKID, P_PROJID and all the EW_*, EP_*, EXR_* macros are extracted during the module build process. The macros are extracted from the exacct header files under /usr/include and provided as perl constants. Constants passed to the Sun::Solaris::Exacct functions can either be an integer value such as. EW_FINAL or a string representation of the same variable such as. “EW_FINAL”.

Sun::Solaris::Exacct Functions, Class Methods, and Object Methods

The perl extensions to the libexacct(3LIB) API provide the following functions for the Sun::Solaris::Exacct module. The function name is a link to the corresponding man page.

getacct(2)
putacct(2)
wracct(2)
ea_error(3EXACCT)
ea_error_str
ea_register_catalog
ea_new_file
ea_new_item
ea_new_group

ea_dump_object

Note – `ea_error_str()` is provided as a convenience, so that repeated blocks of code like the following can be avoided:

```
if (ea_error() == EXR_SYSCALL_FAIL) {
        print("error: $!\n");
} else {
        print("error: ", ea_error(), "\n");
}
```

The `Sun::Solaris::Exacct` module has no class methods.

The `Sun::Solaris::Exacct` module has no object methods.

`Sun::Solaris::Exacct` Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants and functions defined in this module.

Tag	Constant or Function
:SYSCALLS	`getacct(),putacct(),wracct()`
:LIBCALLS	`ea_error(),ea_error_str()`
:CONSTANTS	P_PID, P_TASKID, P_PROJID, EW_*, EP_*, EXR_*
:SHORTAND	`ea_register_catalog(),ea_new_catalog(), ea_new_file(),ea_new_item(),ea_new_group(), ea_dump_object()`
:ALL	:SYSCALLS, :LIBCALLS, :CONSTANTS, :SHORTHAND
:EXACCT_CONSTANTS	:CONSTANTS, plus the :CONSTANTS tags for `Sun::Solaris::Catalog`, `Sun::Solaris::File`, `Sun::Solaris::Object`
:EXACCT_ALL	:ALL, plus the :ALL tags for `Sun::Solaris::Catalog`, `Sun::Solaris::File`, `Sun::Solaris::Object`

`Sun::Solaris::Exacct::Catalog` Module

The `Sun::Solaris::Exacct::Catalog` module provides a wrapper around the 32-bit integer used as a catalog tag. The catalog tag is represented as a perl object blessed into the `Sun::Solaris::Exacct::Catalog` class. Methods can be used to manipulate fields in a catalog tag.

`Sun::Solaris::Exacct::Catalog` Constants

All the EXT_*, EXC_*, EXD_* macros are extracted during the module build process from the `/usr/include/sys/exact_catalog.h` file and are provided as constants. Constants passed to the `Sun::Solaris::Exacct::Catalog` methods can either be an integer value, such as EXT_UINT8, or the string representation of the same variable, such as "EXT_UINT8".

`Sun::Solaris::Exacct::Catalog` Functions, Class Methods, and Object Methods

The perl extensions to the `libexacct`(3LIB) API provide the following class methods for`Sun::Solaris::Exacct::Catalog`. `Exacct`(3PERL) and`Exacct::Catalog`(3PERL)

register
new

The perl extensions to the `libexacct`(3LIB) API provide the following object methods for `Sun::Solaris::Exacct::Catalog`.

value
type
catalog
id
type_str
catalog_str
id_str

`Sun::Solaris::Exacct::Catalog` Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants and functions defined in this module.

Tag	Constant or Function
:CONSTANTS	EXT_*, EXC_*, EXD_*.
:ALL	:CONSTANTS

Additionally, any constants defined with the `register()` function can optionally be exported into the caller's package.

`Sun::Solaris::Exacct::File` Module

The `Sun::Solaris::Exacct::File` module provides wrappers for the `exacct` functions that manipulate accounting files. The interface is object-oriented, and allows the creation and reading of `exacct` files. The C library calls that are wrapped by this module are:

`ea_open`(3EXACCT)
`ea_close`(3EXACCT)
`ea_next_object`(3EXACCT)
`ea_previous_object`(3EXACCT)
`ea_write_object`(3EXACCT)
`ea_get_object`(3EXACCT)
`ea_get_creator`(3EXACCT)
`ea_get_hostname`(3EXACCT)

The file read and write methods operate on `Sun::Solaris::Exacct::Object` objects. These methods perform all the necessary memory management, packing, unpacking and structure conversions that are required.

`Sun::Solaris::Exacct::File` Constants

`Sun::Solaris::Exacct::File` provides the EO_HEAD, EO_TAIL, EO_NO_VALID_HDR, EO_POSN_MSK, EO_VALIDATE_MSK constants. Other constants that are needed by the `new()` method are in the standard perl `Fcntl` module. Table 4–2 describes the action of `new()` for various values of `$oflags` and `$aflags`.

`Sun::Solaris::Exacct::File` Functions, Class Methods, and Object Methods

The `Sun::Solaris::Exacct::File` module has no functions.

The perl extensions to the `libexacct`(3LIB) API provide the following class method for`Sun::Solaris::Exacct::File`.

new

The following table describes the `new()` action for combinations of the `$oflags` and `$aflags` parameters.

TABLE 4–2 `$oflags` and `$aflags` Parameters

$oflags	$aflags	Action
O_RDONLY	Absent or EO_HEAD	Open for reading at the start of the file.
O_RDONLY	EO_TAIL	Open for reading at the end of the file.
O_WRONLY	Ignored	File must exist, open for writing at the end of the file.
O_WRONLY | O_CREAT	Ignored	Create file if the file does not exist. Otherwise, truncate, and open for writing.
O_RDWR	Ignored	File must exist, open for reading or writing, at the end of the file.
O_RDWR | O_CREAT	Ignored	Create file if the file does not exist. Otherwise, truncate, and open for reading or writing.

Note – The only valid values for `$oflags` are the combinations of O_RDONLY, O_WRONLY, O_RDWR, O_CREAT. `$aflags` describes the required positioning in the file for O_RDONLY. Either EO_HEAD or EO_TAIL are allowed. If absent, EO_HEAD is assumed.

The perl extensions to the `libexacct`(3LIB) API provide the following object methods for`Sun::Solaris::Exacct::File`.

creator
hostname
next
previous
get
write

Note – Close a `Sun::Solaris::Exacct::File`. There is no explicit `close()` method for a `Sun::Solaris::Exacct::File`. The file is closed when the filehandle object is undefined or reassigned.

Sun::Solaris::Exacct::File Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants that are defined in this module.

Tag	Constant or Function
:CONSTANTS	EO_HEAD, EO_TAIL, EO_NO_VALID_HDR, EO_POSN_MSK, EO_VALIDATE_MSK.
:ALL	:CONSTANTS, Fcntl(:DEFAULT).

Sun::Solaris::Exacct::Object Module

The Sun::Solaris::Exacct::Object module serves as a parent of the two possible types of exacct objects: Items and Groups. An exacct Item is a single data value, an embedded exacct object, or a block of raw data. An example of a single data value is the number of seconds of user CPU time consumed by a process. An exacct Group is an ordered collection of exacct Items such as all of the resource usage values for a particular process or task. If Groups need to be nested within each other, the inner Groups can be stored as embedded exacct objects inside the enclosing Group.

The Sun::Solaris::Exacct::Object module contains methods that are common to both exacct Items and Groups. Note that the attributes of Sun::Solaris::Exacct::Object and all classes derived from it are read-only after initial creation via new(). The attributes made read-only prevents the inadvertent modification of the attributes which could give rise to inconsistent catalog tags and data values. The only exception to the read-only attributes is the array used to store the Items inside a Group object. This array can be modified using the normal perl array operators.

Sun::Solaris::Exacct::Object Constants

Sun::Solaris::Exacct::Object provides the EO_ERROR, EO_NONE, EO_ITEM, EO_GROUP constants.

Sun::Solaris::Exacct::Object Functions, Class Methods, and Object Methods

The Sun::Solaris::Exacct::Object module has no functions.

The perl extensions to the libexacct(3LIB) API provide the following class method forSun::Solaris::Exacct::Object.

dump

The perl extensions to the `libexacct`(3LIB) API provide the following object methods for`Sun::Solaris::Exacct::Object`.

type
catalog
match_catalog
value

`Sun::Solaris::Exacct::Object` Exports

By default, nothing is exported from this module. The following tags can be used to selectively import constants and functions defined in this module.

Tag	Constant or Function
:CONSTANTS	EO_ERROR, EO_NONE, EO_ITEM, EO_GROUP
:ALL	:CONSTANTS

`Sun::Solaris::Exacct::Object::Item` Module

The `Sun::Solaris::Exacct::Object::Item` module is used for `exacct` data Items. An `exacct` data Item is represented as an opaque reference, blessed into the `Sun::Solaris::Exacct::Object::Item` class, which is a subclass of the `Sun::Solaris::Exacct::Object` class. The underlying `exacct` data types are mapped onto perl types as follows.

TABLE 4–3 `exacct` Data Types Mapped to Perl Data Types

exacct type	perl internal type
EXT_UINT8	IV (integer)
EXT_UINT16	IV (integer)
EXT_UINT32	IV (integer)
EXT_UINT64	IV (integer)
EXT_DOUBLE	NV (double)
EXT_STRING	PV (string)

TABLE 4–3 `exacct` Data Types Mapped to Perl Data Types *(Continued)*

exacct type	perl internal type
EXT_EXACCT_OBJECT	`Sun::Solaris::Exacct::Object` subclass
EXT_RAW	PV (string)

`Sun::Solaris::Exacct::Object::Item` Constants

`Sun::Solaris::Exacct::Object::Item` has no constants.

`Sun::Solaris::Exacct::Object::Item` Functions, Class Methods, and Object Methods

`Sun::Solaris::Exacct::Object::Item` has no functions.

`Sun::Solaris::Exacct::Object::Item` inherits all class methods from the `Sun::Solaris::Exacct::Object` base class, plus the `new()` class method.

new

`Sun::Solaris::Exacct::Object::Item` inherits all object methods from the `Sun::Solaris::Exacct::Object` base class.

`Sun::Solaris::Exacct::Object::Item` Exports

`Sun::Solaris::Exacct::Object::Item` has no exports.

`Sun::Solaris::Exacct::Object::Group` Module

The `Sun::Solaris::Exacct::Object::Group` module is used for `exacct` Group objects. An `exacct` Group object is represented as an opaque reference, blessed into the `Sun::Solaris::Exacct::Object::Group` class, which is a subclass of the `Sun::Solaris::Exacct::Object` class. The Items within a Group are stored inside a perl array, and a reference to the array can be accessed via the inherited `value()` method. This means that the individual Items within a Group can be manipulated with the normal perl array syntax and operators. All data elements of the array must be derived from the `Sun::Solaris::Exacct::Object` class. Group objects can also be nested inside each other merely by adding an existing Group as a data Item.

`Sun::Solaris::Exacct::Object::Group` Constants

`Sun::Solaris::Exacct::Object::Group` has no constants.

`Sun::Solaris::Exacct::Object::Group` Functions, Class Methods, and Object Methods

`Sun::Solaris::Exacct::Object::Group` has no functions.

`Sun::Solaris::Exacct::Object::Group` inherits all class methods from the `Sun::Solaris::Exacct::Object` base class, plus the `new()` class method.

new

`Sun::Solaris::Exacct::Object::Group` inherits all object methods from the `Sun::Solaris::Exacct::Object` base class, plus the `new()` class method.

as_hash
as_hashlist

`Sun::Solaris::Exacct::Object::Group` Exports

`Sun::Solaris::Exacct::Object::Group` has no exports.

`Sun::Solaris::Exacct::Object::_Array` Module

The `Sun::Solaris::Exacct::Object::_Array` class is used internally for enforcing type checking of the data Items that are placed in an exacct Group. `Sun::Solaris::Exacct::Object::_Array` should not be created directly by the user.

`Sun::Solaris::Exacct::Object::_Array` Constants

`Sun::Solaris::Exacct::Object::_Array` has no constants.

`Sun::Solaris::Exacct::Object::_Array` Functions, Class Methods, and Object Methods

`Sun::Solaris::Exacct::Object::_Array` has no functions.

`Sun::Solaris::Exacct::Object::_Array` has internal-use class methods.

`Sun::Solaris::Exacct::Object::_Array` uses perl TIEARRAY methods.

`Sun::Solaris::Exacct::Object::_Array` Exports

`Sun::Solaris::Exacct::Object::_Array` has no exports.

Perl Code Examples

This section shows perl code examples for accessing `exacct` files.

Pseudocode Prototype

In typical use the perl `exacct` library reads existing `exacct` files. Use pseudocode to show the relationships of the various perl `exacct` classes. Illustrate in pseudocode the process of opening and scanning an `exacct` file, and processing objects of interest. In the following pseudocode, the 'convenience' functions are used in the interest of clarity.

```
-- Open the exacct file ($f is a Sun::Solaris::Exacct::File)
my $f = ea_new_file(...)

-- While not EOF ($o is a Sun::Solaris::Exacct::Object)
while (my $o = $f->get())

        -- Check to see if object is of interest
        if ($o->type() == &EO_ITEM)
             ...

        -- Retrieve the catalog ($c is a Sun::Solaris::Exacct::Catalog)
        $c = $o->catalog()

        -- Retrieve the value
        $v = $o->value();

        -- $v is a reference to a Sun::Solaris::Exacct::Group for a Group
        if (ref($v))
             ....

        -- $v is perl scalar for Items
        else
```

Recursively `dump` an `exacct` Object

```
sub dump_object
{
    my ($obj, $indent) = @_;
    my $istr = '  ' x $indent;

    #
    # Retrieve the catalog tag.  Because we are doing this in an array
    # context, the catalog tag will be returned as a (type, catalog, id)
    # triplet, where each member of the triplet will behave as an integer
    # or a string, depending on context.  If instead this next line provided
    # a scalar context, e.g.
    #    my $cat  = $obj->catalog()->value();
    # then $cat would be set to the integer value of the catalog tag.
    #
    my @cat = $obj->catalog()->value();

    #
    # If the object is a plain item
    #
    if ($obj->type() == &EO_ITEM) {
          #
          # Note:  The '%s' formats provide s string context, so the
          # components of the catalog tag will be displayed as the
          # symbolic values.  If we changed the '%s' formats to '%d',
          # the numeric value of the components would be displayed.
          #
          printf("%sITEM\n%s  Catalog = %s|%s|%s\n",
              $istr, $istr, @cat);
          $indent++;

          #
          # Retrieve the value of the item.  If the item contains in
          # turn a nested exacct object (i.e. a item or group), then
          # the value method will return a reference to the appropriate
          # sort of perl object (Exacct::Object::Item or
          # Exacct::Object::Group). We could of course figure out that
          # the item contained a nested item or group by examining
          # the catalog tag in @cat and looking for a type of
          # EXT_EXACCT_OBJECT or EXT_GROUP.
          my $val = $obj->value();
          if (ref($val)) {
             # If it is a nested object, recurse to dump it.
             dump_object($val, $indent);
          } else {
             # Otherwise it is just a 'plain' value, so display it.
             printf("%s  Value = %s\n", $istr, $val);
          }

        #
        # Otherwise we know we are dealing with a group.  Groups represent
        # contents as a perl list or array (depending on context), so we
        # can process the contents of the group with a 'foreach' loop, which
        # provides a list context.  In a list context the value method
        # returns the content of the group as a perl list, which is the
        # quickest mechanism, but doesn't allow the group to be modified.
        # If we wanted to modify the contents of the group we could do so
        # like this:
        #    my $grp = $obj->value();   # Returns an array reference
```

```
        #    $grp->[0] = $newitem;
        # but accessing the group elements this way is much slower.
        #
        } else {
                printf("%sGROUP\n%s  Catalog = %s|%s|%s\n",
                    $istr, $istr, @cat);
                $indent++;
                # 'foreach' provides a list context.
                foreach my $val ($obj->value()) {
                        dump_object($val, $indent);
                }
                printf("%sENDGROUP\n", $istr);
        }
}
```

Create a New Group Record and Write to File

```
# Prototype list of catalog tags and values.
my @items = (
   [ &EXT_STRING | &EXC_DEFAULT | &EXD_CREATOR      => "me"          ],
   [ &EXT_UINT32 | &EXC_DEFAULT | &EXD_PROC_PID     => $$            ],
   [ &EXT_UINT32 | &EXC_DEFAULT | &EXD_PROC_UID     => $<            ],
   [ &EXT_UINT32 | &EXC_DEFAULT | &EXD_PROC_GID     => $(            ],
   [ &EXT_STRING | &EXC_DEFAULT | &EXD_PROC_COMMAND => "/bin/stuff" ],
);

# Create a new group catalog object.
my $cat = new_catalog(&EXT_GROUP | &EXC_DEFAULT | &EXD_NONE);

# Create a new Group object and retrieve its data array.
my $group = new_group($cat);
my $ary = $group->value();

# Push the new Items onto the Group array.
foreach my $v (@items) {
        push(@$ary, new_item(new_catalog($v->[0]), $v->[1]));
}

# Nest the group within itself (performs a deep copy).
push(@$ary, $group);

# Dump out the group.
dump_object($group);
```

dump an exacct File

```
#!/usr/perl5/5.6.1/bin/perl

use strict;
use warnings;
use blib;
use Sun::Solaris::Exacct qw(:EXACCT_ALL);

die("Usage is dumpexacct

# Open the exact file and display the header information.
my $ef = ea_new_file($ARGV[0], &O_RDONLY) || die(error_str());
printf("Creator:  %s\n", $ef->creator());
printf("Hostname: %s\n\n", $ef->hostname());

# Dump the file contents
while (my $obj = $ef->get()) {
        ea_dump_object($obj);
}

# Report any errors
if (ea_error() != EXR_OK && ea_error() != EXR_EOF)  {
       printf("\nERROR: %s\n", ea_error_str());
       exit(1);
}
exit(0);
```

Output From `dump` Method

The following example shows the formatted output of the `Sun::Solaris::Exacct::Object->dump()` method.

```
GROUP
  Catalog = EXT_GROUP|EXC_DEFAULT|EXD_GROUP_PROC_PARTIAL
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_PID
    Value = 3
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_UID
    Value = 0
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_GID
    Value = 0
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_PROJID
    Value = 0
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_TASKID
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CPU_USER_SEC
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CPU_USER_NSEC
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CPU_SYS_SEC
    Value = 890
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CPU_SYS_NSEC
    Value = 760000000
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_START_SEC
    Value = 1011869897
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_START_NSEC
    Value = 380771911
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_FINISH_SEC
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_FINISH_NSEC
    Value = 0
  ITEM
    Catalog = EXT_STRING|EXC_DEFAULT|EXD_PROC_COMMAND
    Value = fsflush
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_TTY_MAJOR
    Value = 4294967295
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_TTY_MINOR
    Value = 4294967295
  ITEM
    Catalog = EXT_STRING|EXC_DEFAULT|EXD_PROC_HOSTNAME
    Value = mower
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_FAULTS_MAJOR
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_FAULTS_MINOR
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_MESSAGES_SND
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_MESSAGES_RCV
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_BLOCKS_IN
    Value = 19
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_BLOCKS_OUT
    Value = 40833
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CHARS_RDWR
    Value = 0
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CONTEXT_VOL
```

```
    Value = 129747
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_CONTEXT_INV
    Value = 79
  ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_SIGNALS
    Value = 0
    ITEM
    Catalog = EXT_UINT64|EXC_DEFAULT|EXD_PROC_SYSCALLS
    Value = 0
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_ACCT_FLAGS
    Value = 1
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_ANCPID
    Value = 0
  ITEM
    Catalog = EXT_UINT32|EXC_DEFAULT|EXD_PROC_WAIT_STATUS
    Value = 0
ENDGROUP
```

CHAPTER 5

Resource Controls

This chapter describes resource controls and their properties.

Overview of Resource Controls

Use the extended accounting facility to determine the resource consumption of workloads on your system. After the resource consumption has been determined, use the resource control facility to place bounds on resource usage. Bounds that are placed on resources prevent workloads from over-consuming resources.

For an overview of resource controls and example commands for administering resource controls, see Chapter 6, "Resource Controls (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones* and Chapter 7, "Administering Resource Controls (Tasks)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

The resource control facility adds the following benefits.

- Dynamically set.

  Resource controls can be adjusted while the system is running.
- Containment level granularity.

  Resource controls are arranged in a containment level of project, task, or process. The containment level simplifies the configuration and aligns the collected values closer to the particular project, task, or process.

- Threshold preservation.

  If an attempt is made to set the maximum value less than the actual resource consumption, no change in to the maximum value is made.

Resource Controls Flags and Actions

This section describes flags, actions, and signals associated with resource controls.

`rlimit`, Resource Limit

`rlimit` is process-based. `rlimit` establishes a restricting boundary on the consumption of a variety of system resources by a process. Each process that the process creates inherits from the original process. A resource limit is defined by a pair of values. The values specify the current (soft) limit and the maximum (hard) limit.

A process might irreversibly lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with superuser ID can raise the hard limit. See `setrlimit()` and `getrlimit()`.

The `rlimit` structure contains two members that define the soft limit and hard limit.

```
rlim_t     rlim_cur;       /* current (soft) limit */
rlim_t     rlim_max        /* hard limit */
```

`rctl`, Resource Control

`rctl` extends the process-based limits of `rlimit` by controlling resource consumption by processes, tasks, and projects defined in the project database.

Note – The `rctl` mechanism is preferred to the use of `rlimit` to set resource limits. The only reason to use the `rlimit` facility is when portability is required across UNIX platforms.

Applications fall into the following broad categories depending on how an application deals with resource controls. Based on the action that is taken, resource controls can be further classified. Most report an error and terminate operation. Other resource controls allow applications to resume operation and adapt to the reduced resource usage. A progressive chain of actions at increasing values can be specified for each resource control.

The list of attributes for a resource control consists of a privilege level, a threshold value, and an action that is taken when the threshold is exceeded.

`rctl` Privilege Values

Each threshold value on a resource control must be associated with one of the following privilege levels:

RCPRIV_BASIC
: Privilege level can be modified by the owner of the calling process. RCPRIV_BASIC is associated with a resource's soft limit.

RCPRIV_PRIVILEGED
: Privilege level can be modified only by privileged (superuser) callers. RCPRIV_PRIVILEGED is associated with a resource's hard limit.

RCPRIV_SYSTEM
: Privilege level remains fixed for the duration of the operating system instance.

Figure 5–2 shows the timeline for setting privilege levels for signals that are defined by the `/etc/project` file `process.max-cpu-time` resource control.

Local Actions and Local Flags

The local action and local flags are applied to the current resource control value represented by this resource control block. Local actions and local flags are value-specific. For each threshold value that is placed on a resource control, the following local actions and local flags are available:

RCTL_LOCAL_NOACTION
: No local action is taken when this resource control value is exceeded.

RCTL_LOCAL_SIGNAL
: The specified signal, set by `rctlblk_set_local_action()`, is sent to the process that placed this resource control value in the value sequence.

RCTL_LOCAL_DENY
: When this resource control value is encountered, the request for the resource is denied. Set on all values if RCTL_GLOBAL_DENY_ALWAYS is set for this control. Cleared on all values if RCTL_GLOBAL_DENY_NEVER is set for this control.

RCTL_LOCAL_MAXIMAL
: This resource control value represents a request for the maximum amount of resource for this control. If RCTL_GLOBAL_INFINITE is set for this resource control, RCTL_LOCAL_MAXIMAL indicates an unlimited resource control value that is never exceeded.

Global Actions and Global Flags

Global flags apply to all current resource control values represented by this resource control block. Global actions and global flags are set by `rctladm`(1M). Global actions and global flags cannot be set with `setrctl()`. Global flags apply to all resource controls. For each threshold value that is placed on a resource control, the following global actions and global flags are available:

RCTL_GLOBAL_NOACTION
: No global action is taken when a resource control value is exceeded on this control.

RCTL_GLOBAL_SYSLOG
: A standard message is logged by the `syslog()` facility when any resource control value on a sequence associated with this control is exceeded.

RCTL_GLOBAL_NOBASIC
: No values with the RCPRIV_BASIC privilege are permitted on this control.

RCTL_GLOBAL_LOWERABLE
: Non-privileged callers are able to lower the value of privileged resource control values on this control.

RCTL_GLOBAL_DENY_ALWAYS
: The action that is taken when a control value is exceeded on this control always includes denial of the resource.

RCTL_GLOBAL_DENY_NEVER
: The action that is taken when a control value is exceeded on this control always excludes denial of the resource. The resource is always granted, although other actions can also be taken.

RCTL_GLOBAL_FILE_SIZE
: The valid signals for local actions include the SIGXFSZ signal.

RCTL_GLOBAL_CPU_TIME
: The valid signals for local actions include the SIGXCPU signal.

RCTL_GLOBAL_SIGNAL_NEVER
: No local actions are permitted on this control. The resource is always granted.

RCTL_GLOBAL_INFINITE
: This resource control supports the concept of an unlimited value. Generally, an unlimited value applies only to accumulation-oriented resources, such as CPU time.

RCTL_GLOBAL_UNOBSERVABLE
: Generally, a task or project related resource control does not support observational control values. An RCPRIV_BASIC privileged control value placed on a task or process generates an action only if the value is exceeded by that process.

Resource Control Sets Associated With a Project, Processes and Tasks

The following figure shows the resource control sets associated with tasks, processes and a project.

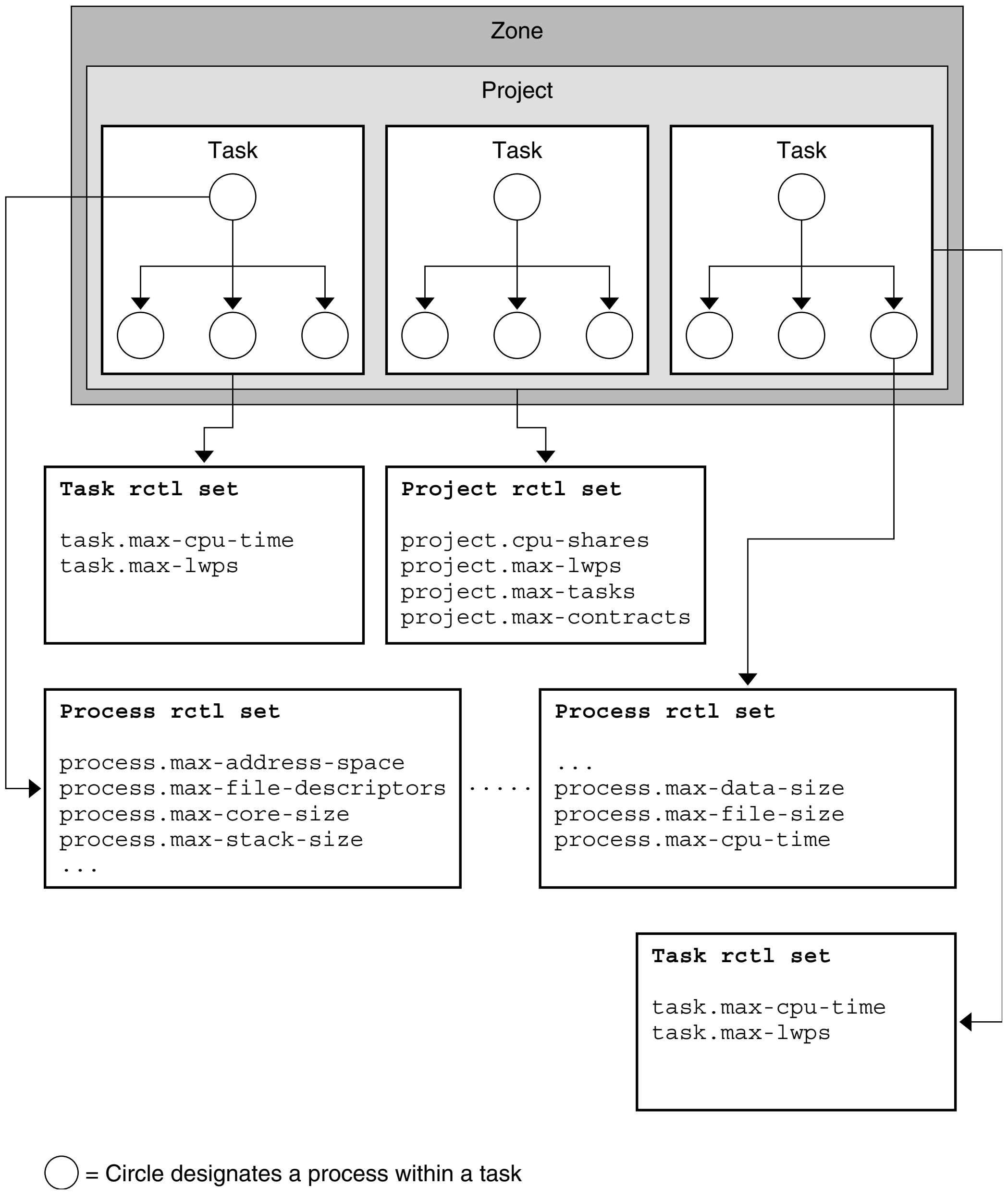


FIGURE 5–1 Resource Control Sets for Task, Project, and Process

More than one resource control can exist on a resource, each resource control at a containment level in the process model. Resource controls can be active on the same resource for both a process and collective task or collective project. In this case, the action for the process takes precedence. For example, action is taken on `process.max-cpu-time` before `task.max-cpu-time` if both controls are encountered simultaneously.

Resource Controls Associated With a Project

Resource controls associated with a project include the following:

`project.cpu-shares`
: The number of CPU shares that are granted to this project for use with the fair share scheduler, `FSS`(7).

`project.max-msg-ids`
: Maximum number of System V message queues allowed for a project.

`project.max-sem-ids`
: Maximum number of System V semaphores allowed for a project.

`project.max-port-ids`
: Maximum allowable number of event ports.

Resource Controls Associated With Tasks

Resource controls associated with tasks include the following:

`task.max-cpu-time`
: Maximum CPU time (seconds) available to this task's processes.

`task.max-lwps`
: Maximum number of LWPs simultaneously available to this task's processes.

Resource Controls Associated With Processes

Resource controls associated with processes include the following:

`process.max-address-space`
: Maximum amount of address space (bytes), as summed over segment sizes, available to this process.

`process.max-core-size`
: Maximum size (bytes) of a core file that is created by this process.

`process.max-cpu-time`
: Maximum CPU time (seconds) available to this process.

`process.max-file-descriptor`
: Maximum file descriptor index that is available to this process.

`process.max-file-size`
: Maximum file offset (bytes) available for writing by this process.

`process.max-msg-messages`
: Maximum number of messages on a message queue. This value is copied from the resource control at `msgget()` time.

`process.max-msg-qbytes`
: Maximum number (bytes) of messages on a message queue. This value is copied from the resource control at `msgget()` time.When you set a new `project.max-msg-qbytes` value, initialization occurs only on the subsequently created values. The new `project.max-msg-qbytes` value does not effect existing values.

`process.max-sem-nsems`
: Maximum number of semaphores allowed for a semaphore set.

`process.max-sem-ops`
: Maximum number of semaphore operations that are allowed for a `semop()` call. This value is copied from the resource control at `msgget()` time.A new `project.max-sem-ops` value only affects the initialization of subsequently created values and has no effect on existing values.

`process.max-port-events`
: Maximum number of events that are allowed per event port.

`process.crpto-buffer-limit`
: Maximum number of bytes that are allocated for copying.

`process.max-crypto-sessions`
: Maximum number of entries in the session table.

`process.add-crypto-sessions`
: Number of entries that are added when enlarging the session table.

`process.min-crypto-sessions`
: Minimum number of entries in the session table.

Signals Used With Resource Controls

For each threshold value that is placed on a resource control, the following restricted set of signals is available:

SIGBART
: Terminate the process.

SIGXRES
: Signal generated by resource control facility when the resource control limit is exceeded.

SIGHUP
: When carrier drops on an open line, the process group that controls the terminal is sent a hangup signal, SIGHUP.

SIGSTOP
: Job control signal. Stop the process. Stop signal not from terminal.

SIGTERM
: Terminate the process. Termination signal sent by software.

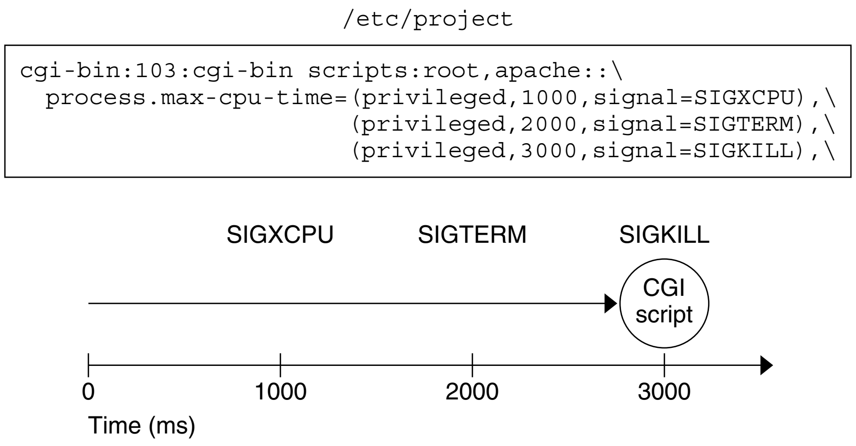
SIGKILL
: Terminate the process. Kill the program.

SIGXFSX
: Terminate the process. File size limit exceeded. Available only to resource controls with the RCTL_GLOBAL_FILE_SIZE property.

SIGXCPU
: Terminate the process. CPU time limit exceeded. Available only to resource controls with the RCTL_GLOBAL_CPUTIME property.

Other signals might be permitted due to global properties of a specific control.

Note – Calls to `setrctl()` with illegal signals fail.



/etc/project

```
cgi-bin:103:cgi-bin scripts:root,apache::\
  process.max-cpu-time=(privileged,1000,signal=SIGXCPU),\
                       (privileged,2000,signal=SIGTERM),\
                       (privileged,3000,signal=SIGKILL),\
```

FIGURE 5–2 Setting Privilege Levels for Signals

Resource Controls Application Programming Interface

The resource controls API contains functions that:

- "Operate on Action-value Pairs of a Resource Control" on page 56
- "Operate on Local Modifiable Values" on page 56
- "Retrieve Local Read-only Values" on page 56
- "Retrieve Global Read-only Actions" on page 57

Operate on Action-value Pairs of a Resource Control

The following list contains the functions that set or get the resource control block. The function name is a link to the corresponding man page.

setrctl
getrctl

Operate on Local Modifiable Values

The following list contains the functions associated with the local, modifiable resource control block. The function name is a link to the corresponding man page.

`rctlblk_set_privilege`(3C)
`rctlblk_get_privilege`(3C)
`rctlblk_set_value`(3C)
`rctlblk_get_value`(3C)
`rctlblk_set_local_action`(3C)
`rctlblk_get_local_action`(3C)
`rctlblk_set_local_flags`(3C)
`rctlblk_get_local_flags`(3C)

Retrieve Local Read-only Values

The following list contains the functions associated with the local, read-only resource control block. The function name is a link to the corresponding man page.

`rctlblk_get_recipient_pid`(3C)

```
rctlblk_get_firing_time(3C)
rctlblk_get_enforced_value(3C)
```

Retrieve Global Read-only Actions

The following list contains the functions associated with the global, read-only resource control block. The function name is a link to the corresponding man page.

```
rctlblk_get_global_action(3C)
rctlblk_get_global_flags(3C)
```

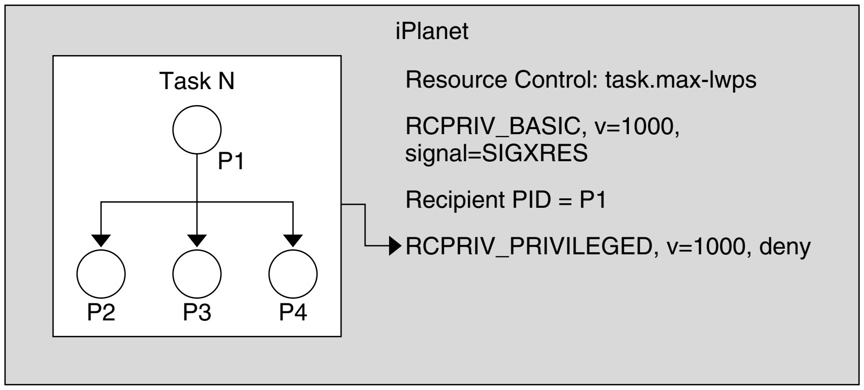
Code Examples of Resource Controls

This section provides code examples of the resource controls interface.

Master Observing Process for Resource Controls

The following example is the master observer process. Figure 5–3 shows the resource controls for the master observing process.

Note – The line break is not valid in an `/etc/project` file. The line break is shown here only to allow the example to display on a printed or displayed page. Each entry in the `/etc/project` file must be on a separate line.



/etc/project

```
...
iPlanet:200:iPlanet Application Server:root::\
              task.max=lwps=(PRIVILEGED,1000,deny)
```

FIGURE 5–3 Master Observing Process

The key points for the example include the following:

- Because the task's limit is privileged, the application cannot change the limit, or specify an action, such as a signal. A master process solves this problem by establishing the same resource control as a basic resource control on the task. The master process uses the same value or a little less on the resource, but with a different action, signal = XRES. The master process creates a thread to wait for this signal.
- The `rctlblk` is opaque. The struct needs to be dynamically allocated.
- Note the blocking of all signals before creating the thread, as required by sigwait(2).
- The thread calls `sigwait`(2) to block for the signal. If `sigwait()` returns the SIGXRES signal, the thread notifies the master process' children, which adapts to reduce the number of LWPs being used. Each child should also be modelled similarly, with a thread in each child, waiting for this signal, and adapting its process' LWP usage appropriately.

```
rctlblk_t *mlwprcb;
sigset_t smask;

/* Omit return value checking/error processing to keep code sample short */
/* First, install a RCPRIV_BASIC, v=1000, signal=SIGXRES rctl */
mlwprcb = calloc(1, rctlblk_size());     /* rctl blocks are opaque: */
       rctlblk_set_value(mlwprcb, 1000);
       rctlblk_set_privilege(mlwprcb, RCPRIV_BASIC);
       rctlblk_set_local_action(mlwprcb, RCTL_LOCAL_SIGNAL, SIGXRES);
       if (setrctl("task.max-lwps", NULL, mlwprcb, RCTL_INSERT) == -1) {
           perror("setrctl");
           exit (1);
       }

/* Now, create the thread which waits for the signal */
        sigemptyset(&smask);
        sigaddset(&smask, SIGXRES);
        thr_sigsetmask(SIG_BLOCK, &smask, NULL);
thr_create(NULL, 0, sigthread, (void *)SIGXRES, THR_DETACHED, NULL));

/* Omit return value checking/error processing to keep code sample short */

void *sigthread(void *a)
{
        int sig = (int)a;
        int rsig;
        sigset_t sset;

        sigemptyset(&sset);
        sigaddset(&sset, sig);

        while (1) {
                 rsig = sigwait(&sset);
          if (rsig == SIGXRES) {
              notify_all_children();
              /* e.g. sigsend(P_PID, child_pid, SIGXRES); */
             }
        }
}
```

List all the Value-action Pairs for a Specific Resource Control

The following example lists all the value-action pairs for a specific resource control, `task.max-lwps`. The key point for the example is that `getrctl`(2) takes two resource control blocks, and returns the resource control block for the RCTL_NEXT flag. To iterate through all resource control blocks, repeatedly swap the resource control block values, as shown here using the `rcb_tmp` `rctl` block.

```
rctlblk_t *rcb1, *rcb2, *rcb_tmp;
    ...
/* Omit return value checking/error processing to keep code sample short */
rcb1 = calloc(1, rctlblk_size()); /* rctl blocks are opaque: */
                               /* "rctlblk_t rcb" does not work */
rcb2 = calloc(1, rctlblk_size());
getrctl("task.max-lwps", NULL, rcb1, RCTL_FIRST);
while (1) {
     print_rctl(rcb1);
     rcb_tmp = rcb2;
     rcb2 = rcb1;
     rcb1 = rcb_tmp;        /* swap rcb1 with rcb2 */
     if (getrctl("task.max-lwps", rcb2,  rcb1, RCTL_NEXT) == -1) {
          if (errno == ENOENT) {
               break;
     } else {
          perror("getrctl");
          exit (1);
     }
     }
}
```

Set project.cpu-shares and Add a New Value

The key points of the example include the following:

- This example is similar to the example shown in "Set pool.comment Property and Add New Property" on page 75.
- Use bcopy(), rather than buffer swapping as in "List all the Value-action Pairs for a Specific Resource Control" on page 59.
- To change the resource control value, call setrctl() with the RCTL_REPLACE flag. The new resource control block is identical to the old resource control block except for the new control value.

  ```
  rctlblk_set_value(blk1, nshares);
  if (setrctl("project.cpu-shares", blk2, blk1, RCTL_REPLACE) != 0)
  ```

The example gets the project's CPU share allocation, project.cpu-shares, and changes its value to *nshares*.

```
/* Omit return value checking/error processing to keep code sample short */
blk1 = malloc(rctlblk_size());
getrctl("project.cpu-shares", NULL, blk1, RCTL_FIRST);
my_shares = rctlblk_get_value(blk1);
printout_my_shares(my_shares);
/* if privileged, do the following to */
/* change project.cpu-shares to "nshares" */
blk1 = malloc(rctlblk_size());
blk2 = malloc(rctlblk_size());
if (getrctl("project.cpu-shares", NULL, blk1, RCTL_FIRST) != 0) {
     perror("getrctl failed");
     exit(1);
}
bcopy(blk1, blk2, rctlblk_size());
rctlblk_set_value(blk1, nshares);
if (setrctl("project.cpu-shares", blk2, blk1, RCTL_REPLACE) != 0) {
     perror("setrctl failed");
     exit(1);
}
```

Set LWP Limit on Resource Control Blocks

In the following example, our application has set a privileged limit of 3000 LWPs that may not be exceeded. In addition, our application has set a basic limit of 2000 LWPs. When this limit is exceeded, a SIGXRES is sent to the application. Upon receiving a SIGXRES, our application might send notification to its child processes that might in turn reduce the number of LWPs the processes use or need.

```
/* Omit return value and error checking */

#include <rctl.h>

rctlblk_t *rcb1, *rcb2;

/*
         * Resource control blocks are opaque
         * and must be explicitly allocated.
         */
rcb1 = calloc(rctlblk_size());

rcb2 = calloc(rctlblk_size());


/* Install an RCPRIV_PRIVILEGED, v=3000: do not allow more than 3000 LWPs */
rctlblk_set_value(rcb1, 3000);
rctlblk_set_privilege(rcb1, RCPRIV_PRIVILEGED);
rctlblk_set_local_action(rcb1, RCTL_LOCAL_DENY);
setrctl("task.max-lwps", NULL, rcb1, RCTL_INSERT);


/* Install an RCPRIV_BASIC, v=2000 to send SIGXRES when LWPs exceeds 2000 */
rctlblk_set_value(rcb2, 2000);
rctlblk_set_privilege(rcb2, RCPRIV_BASIC);
rctlblk_set_local_action(rcb2, RCTL_LOCAL_SIGNAL, SIGXRES);
setrctl("task.max-lwps", NULL, rcb2, RCTL_INSERT);
```

Programming Issues Associated With Resource Controls

Consider the following issues when writing your application:

- The resource control block is opaque. The control block needs to be dynamically allocated.
- If a basic resource control is established on a task or project, the process that establishes this resource control becomes an observer. The action for this resource control block is applied to the observer. However, some resources cannot be observed in this manner.

- If a privileged resource control is set on a task or project, no observer process exists. However, any process that violates the limit becomes the subject of the resource control action.
- Only one action is permitted for each type: global and local.
- Only one basic `rctl` is allowed per process per resource control.

CHAPTER 6

Resource Pools

This chapter describes resource pools and their properties.

Overview of Resource Pools

Resource pools provide a framework for managing processor sets and thread scheduling classes. Resource pools are used for partitioning machine resources. Resource pools enable you to separate workloads so that workload consumption of certain resources does not overlap. The resource reservation helps to achieve predictable performance on systems with mixed workloads.

For an overview of resource pools and example commands for administering resource pools, see Chapter 12, "Dynamic Resource Pools (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones* and Chapter 13, "Administering Dynamic Resource Pools (Tasks)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

A processor set groups the CPUs on a system into a bounded entity, on which a process or processes can run exclusively. Processes cannot extend beyond the processor set, nor can other processes extend into the processor set. A processor set enables tasks of similar characteristics to be grouped together and a hard upper boundary for CPU use to be set.

The resource pool framework allows the definition of a soft processor set with a maximum and minimum CPU count requirement. Additionally, the framework provides a hard-defined scheduling class for that processor set.

A resource pool defines

- Processor set groups
- Scheduling class

Scheduling Class

Scheduling classes provide different CPU access characteristics to threads that are based on algorithmic logic. The scheduling classes include:

- Real-time scheduling class
- Inter-active scheduling class
- Fixed-priority scheduling class
- Time-sharing scheduling class
- Fair-share scheduling class

For an overview of fair share scheduler and example commands for administering the fair share scheduler, see Chapter 8, "Fair Share Scheduler (Overview)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones* and Chapter 9, "Administering the Fair Share Scheduler (Tasks)," in *System Administration Guide: N1 Grid Containers, Resource Management, and Solaris Zones*.

Do not mix scheduling classes in a set of CPUs. If scheduling classes are mixed in a CPU set, system performance might become erratic and unpredictable. Use processor sets to segregate applications by their characteristics. Assign scheduling classes under which the application best performs. For more information about the characteristics of an individual scheduling class, see `priocntl`(1).

For an overview of resource pools and a discussion of when to use pools, see Chapter 6.

Resource Pools

The `libpool` library defines properties that are available to the various entities that are managed within the pools facility. Each property falls into the following categories:

Constraint.
: A constraint defines boundaries of a property. Typical constraints are the maximum and minimum allocations specified in the `libpool` configuration.

Objective.
: An objective changes the resource assignments of the current configuration to generate new candidate configurations that observe the established constraints. (See `poold`(1M).) An objective has the following categories:

Workload dependent.	A workload dependent objective varies according to the conditions imposed by the workload. An example of the workload dependent objective is the `utilization` objective.
Workload independent.	A workload independent objective does not vary according to the conditions imposed by the workload. An example of the workload independent objective is the `cpu locality` objective.

Multiplier.
Text required.

System Properties

`system.allocate-method` (writable string)
Allocation method to use when this configuration is instantiated. See `libpool`(3LIB) for the valid values of `system.allocate-method`.

`system.bind-default` (writable boolean)
If the specified pool is not found in <filename>/etc/project </filename>, bind to pool with the `pool.default` property set to TRUE.

`system.comment` (writable string)
User description of system. `system.comment` is not used by the default pools commands, except when a configuration is initiated by the `poolcfg` utility. In this case, the system puts an informative message in the `system.comment` property for that configuration.

`system.name` (writable string)
User name for the configuration.

`system.version` (read-only integer)
`libpool` version required to manipulate this configuration.

Pool Properties

All pool properties are writable.

`pool.active` (writable boolean)
If TRUE, mark this pool as active.

`pool.comment` (writable string)
User description of pool.

`pool.default` (writable boolean)
If TRUE, mark this pool as the default pool. See the `system.bind-default` property.

`pool.importance` (writable integer)
Relative importance of this pool. Used for possible resource dispute resolution.

`pool.name` (writable string)
User name for pool. `setproject`(3PROJECT) uses `pool.name` as the value for the `project.pool` project attribute in the `project`(4) database.

`pool.scheduler` (writable string)
Scheduler class to which consumers of this pool are bound. This property is optional and if not specified, the scheduler bindings for consumers of this pool are not affected. For more information about the characteristics of an individual scheduling class, see `priocntl`(1). Scheduler classes include:

- RT for real-time scheduler
- TS for time-sharing scheduler
- IA for inter-active scheduler
- FSS for fair share scheduler
- FX for fixed-priority scheduler

Processor Sets Properties

`pset.comment` (writable string)
User description of resource.

`pset.default` (read-only boolean)
Identifies the default processor set.

`pset.load` (read-only unsigned integer)
The load for this processor set. The lowest value is 0. The value increases in a linear fashion with the load on the set, as measured by the number of jobs in the system run queue.

`pset.max` (writable unsigned integer)
Maximum number of CPUs that are permitted in this processor set.

`pset.min` (writable unsigned integer)
Minimum number of CPUs that are permitted in this processor set.

`pset.name` (writable string)
Name for the resource.

`pset.size` (read-only unsigned integer)
Current number of CPUs in this processor set.

`pset.sys_id` (read-only integer)
System-assigned processor set ID.

`pset.type` (read-only string)
Names the resource type. Value for all processor sets is `pset`.

`pset.units` (read-only string)
Identifies the meaning of size-related properties. The value for all processor sets is `population`.

`cpu.comment` (writable string)
User description of cpu.

`cpu.status` (writable integer)
Processor status, on-line, offline, or interrupts disabled.

- `off-line` sets the CPU offline. An off-line processor does not process any LWPs. Usually, an off-line processor is not interruptible by I/O devices in the system. On some processors or under certain conditions, disabling interrupts might not be possible for an off-line processor. Thus, the actual effect for being off-line might vary from machine to machine.

  A processor cannot be taken off-line if any LWPs are bound to the processor. On some architectures, taking certain processors off-line might not be possible. For example, the system depends on some resource provided by the processor.
- `on-line` sets the CPU online. An on-line processor processes LWPs and can be interrupted by I/O devices in the system.
- `powered-off` sets CPU status to powered off.
- `no-intr` disables interrupt processing on the CPU. A no-intr processor continues to process LWPs.

  At least one processor in the system must be able to process LWPs. At least one processor must also be able to be interrupted. An off-line processor is interruptible. An operational system with a single no-intr processor and all other processors off-line contains one or more processors that can be interrupted.
- `spare` marks an off-line CPU as spare.

These strings are defined in `<sys/processor.h>` as the PS_OFFLINE, PS_ONLINE, PS_POWEROFF, PS_NOINTR, and PS_SPARE macros. A CPU may enter an additional *faulted* state, as a result of action taken by the kernel. *faulted* cannot be set by the user.

`cpu.sys_id` (read-only integer)
System-assigned processor ID.

`libpool` Pool Configuration Manipulation Library

The `libpool`(3LIB) pool configuration library defines the interface for reading and writing pools configuration files. The library also defines the interface for committing an existing configuration to becoming the running operating system configuration. The `<pool.h>` header provides type and function declarations for all library services.

The resource pools facility brings together process-bindable resources into a common abstraction that is called a pool. Processor sets and other entities can be configured, grouped, and labelled in a persistent fashion. Workload components can be associated with a subset of a system's total resources. The `libpool`(3LIB) library provides a C language API for accessing the resource pools facility. The `pooladm`(1M), `poolbind`(1M), and `poolcfg`(1M) make the resource pools facility available through command invocations from a shell.

Manipulate `psets`

The following table lists the functions associated with creating or destroying psets and manipulating psets. The function name is a link to the corresponding man page.

`pset_setattr`(2)
`pset_getattr`(2)
`processor_bind`(2)
`pset_create`(2)
`pset_assign`(2)
`pset_bind`(2)
`pset_destroy`(2)`processor_bind`(2)

Resource Pools Application Programming Interface

The resource pools API contains functions that:

- "Create or Destroy Resource Pools" on page 69
- "Associate a Pool with Resources" on page 69
- "Bind Workloads to a Resource Pool" on page 69
- "Iterate or Walk Through a Resource Pool" on page 69
- "Query the Configuration of a Pool" on page 70
- "Query Resource Pool" on page 70
- "Query the Resources of a Pool" on page 70
- "Query the Components of a Pool" on page 70
- "Convert Elements of a Pool" on page 70
- "Operate on Properties of a Pool" on page 71
- "Operate on Resources of a Pool" on page 71
- "Manipulate Configuration-related Information from a Pool" on page 71
- "Operate on Values of a Pool Property" on page 72
- "Retrieve Error-related Information from a Pool" on page 72

- “Operate on the Resource Pool Framework” on page 72

The imported interfaces for libpool for swap sets is identical to the ones defined in this document.

Create or Destroy Resource Pools

The following table lists the functions associated with creating or destroying resource pools. The function name is a link to the corresponding man page.

`pool_create`(3POOL)
`pool_destroy`(3POOL)

Associate a Pool with Resources

The following table lists the functions that associate a pool with resources. The function name is a link to the corresponding man page.

`pool_associate`(3POOL)
`pool_dissociate`(3POOL)

Bind Workloads to a Resource Pool

The following table lists the functions that bind workloads to a resource pool. The function name is a link to the corresponding man page.

`pool_set_binding`(3POOL)
`pool_get_binding`(3POOL)

Iterate or Walk Through a Resource Pool

The following table lists the functions that iterate or walk through a resource pool. The function name is a link to the corresponding man page.

`pool_walk_components`(3POOL)
`pool_walk_pools`(3POOL)
`pool_walk_resources`(3POOL)

Query the Configuration of a Pool

The following table lists the functions that query the configuration of a resource pool. The function name is a link to the corresponding man page.

`pool_query_components`(3POOL)
`pool_query_pools`(3POOL)
`pool_query_resources`(3POOL)
`pool_query_resource_components`(3POOL)

Query Resource Pool

The following table lists the functions that query pools. The function name is a link to the corresponding man page.

`pool_info`(3POOL)
`pool_query_pool_resources`(3POOL)

Query the Resources of a Pool

The following table lists the functions that query the resources of a pool. The function name is a link to the corresponding man page.

`pool_resource_info`(3POOL)
`pool_query_resource_components`(3POOL)

Query the Components of a Pool

The following table lists the functions that query the components of a resource pool. The function name is a link to the corresponding man page.

`pool_component_info`(3POOL)
`pool_get_owning_resource`(3POOL)

Convert Elements of a Pool

Each entity for which a property needs to be read or be written must first be converted to the *element* entity. A read or write of the property can then be done on this element.

The following list contains the functions that convert elements of a pool. The function name is a link to the corresponding man page.

`pool_to_elem`(3POOL)
`pool_resource_to_elem`(3POOL)
`pool_component_to_elem`(3POOL)
`pool_conf_to_elem`(3POOL)

Operate on Properties of a Pool

The following list contains the functions that operate on properties of a pool. The function name is a link to its man page.

`pool_get_property`(3POOL)
`pool_put_property`(3POOL)
`pool_rm_property`(3POOL)
`pool_walk_properties`(3POOL)

Operate on Resources of a Pool

The following list contains the functions that operate on resources of a pool. The function name is a link to the corresponding man page.

`pool_resource_create`(3POOL)
`pool_resource_destroy`(3POOL)
`pool_resource_info`(3POOL)
`pool_resource_transfer`(3POOL)
`pool_resource_xtransfer`(3POOL)

Manipulate Configuration-related Information from a Pool

The following list contains the functions that manipulate configuration-related information from resource pools. The function name is a link to the corresponding man page.

`pool_conf_alloc`(3POOL)
`pool_conf_free`(3POOL)
`pool_conf_open`(3POOL)

pool_conf_close(3POOL)
pool_conf_commit(3POOL)
pool_conf_export(3POOL)
pool_conf_info(3POOL)
pool_conf_location(3POOL)
pool_conf_remove(3POOL)
pool_conf_rollback(3POOL)
pool_conf_status(3POOL)
pool_conf_update(3POOL)
pool_conf_validate(3POOL)

Operate on Values of a Pool Property

The following list contains the functions associated with values of pool properties. The function name is a link to the corresponding man page.

pool_value_alloc(3POOL)
pool_value_free(3POOL)
pool_value_get_bool(3POOL)
pool_value_set_bool(3POOL)
pool_value_get_double(3POOL)
pool_value_set_double(3POOL)
pool_value_get_int64(3POOL)
pool_value_set_int64(3POOL)
pool_value_set_uint64(3POOL)
pool_value_get_uint64(3POOL)
pool_value_get_name(3POOL)
pool_value_set_name(3POOL)
pool_value_get_string(3POOL)
pool_value_set_string(3POOL)
pool_value_get_type(3POOL)
pool_value_set_type

Retrieve Error-related Information from a Pool

The following list contains the functions that retrieve error-related information from a resource pool. The function name is a link to the corresponding man page.

pool_error(3POOL)
pool_strerror(3POOL)

Operate on the Resource Pool Framework

The following list contains the functions associated with the resource pool framework. The function name is a link to the corresponding man page.

pool_dynamic_location(3POOL)
pool_static_location(3POOL)
pool_version(3POOL)

Code Examples of Resource Pool

This section contains code examples of the resource pools interface.

Ascertain the Number of CPUs in the Resource Pool

sysconf(3C) provides information about the number of CPUs on an entire system. The following example provides the granularity of ascertaining the number of CPUs that are defined in a particular application's pools pset.

The key points for this example include the following:

- pvals[] should be a NULL terminated array.
- pool_query_pool_resources() returns a list of all resources that match the pvals array type pset from the application's pool my_pool. Because a pool can have only one instance of the pset resource, each instance is always returned in nelem. reslist[] contains only one element, the pset resource.

```
pool_value_t *pvals[2] = {NULL};  /* pvals[] should be NULL terminated */

/* NOTE: Return value checking/error processing omitted */
/* in all examples for brevity */

conf_loc = pool_dynamic_location();
conf = pool_conf_alloc();
pool_conf_open(conf, conf_loc, PO_RDONLY);
my_pool_name = pool_get_binding(getpid());
my_pool = pool_get_pool(conf, my_pool_name);
pvals[0] = pool_value_alloc();
pvals2[2] = { NULL, NULL };
pool_value_set_name(pvals[0], "type");
pool_value_set_string(pvals[0], "pset");

reslist = pool_query_pool_resources(conf, my_pool, &nelem, pvals);
pool_value_free(pvals[0]);
pool_query_resource_components(conf, reslist[0], &nelem, NULL);
printf("pool %s: %u cpu", my_pool_ name, nelem);
pool_conf_close(conf);
```

List all Resource Pools

The following example lists all resource pools defined in an application's pools `pset`.

The key points of the example include the following:

- Open the dynamic `conf` file read-only, PO_RDONLY. `pool_query_pools()` returns the list of pools in `pl` and the number of pools in `nelem`. For each pool, call `pool_get_property()` to get the `pool.name` property from the element into the `pval` value.
- `pool_get_property()` calls `pool_to_elem()` to convert the `libpool` entity to an opaque value. `pool_value_get_string()` gets the string from the opaque pool value.

```
conf    = pool_conf_alloc();
pool_conf_open(conf, pool_dynamic_location(), PO_RDONLY);
pl = pool_query_pools(conf, &nelem, NULL);
pval = pool_value_alloc();
for (i = 0; i < nelem; i++) {
    pool_get_property(conf, pool_to_elem(conf, pl[i]), "pool.name", pval);
    pool_value_get_string(pval, &fname);
    printf("%s\n", name);
}
pool_value_free(pval);
free(pl);
pool_conf_close(conf);
```

Report Pool Statistics for a Given Pool

The following example reports statistics for the designated pool.

The key points for the example include the following:

- `pool_query_pool_resources()` gets a list of all resources in `rl`. Because the last argument to `pool_query_pool_resources()` is NULL, all resources are returned. For each resource, the `name`, `load` and `size` properties are read, and printed.
- The call to `strdup()` allocates local memory and copies the string returned by `get_string()`. The call to `get_string()` returns a pointer that is freed by the next call to `get_property()`. If the call to `strdup()` is not included, subsequent references to the string(s) could cause the application to fail with a segmentation fault.

```
printf("pool %s\n:" pool_name);
pool = pool_get_pool(conf, pool_name);
rl = pool_query_pool_resources(conf, pool, &nelem, NULL);
for (i = 0; i < nelem; i++) {
  pool_get_property(conf, pool_resource_to_elem(conf, rl[i]), "type", pval);
  pool_value_get_string(pval, &type);
  type = strdup(type);
  snprintf(prop_name, 32, "%s.%s", type, "name");
  pool_get_property(conf, pool_resource_to_elem(conf, rl[i]),
        prop_name, pval);
  pool_value_get_string(val, &res_name);
  res_name = strdup(res_name);
  snprintf(prop_name, 32, "%s.%s", type, "load");
  pool_get_property(conf, pool_resource_to_elem(conf, rl[i]),
        prop_name, pval);
  pool_value_get_uint64(val, &load);
  snprintf(prop_name, 32, "%s.%s", type, "size");
  pool_get_property(conf, pool_resource_to_elem(conf, rl[i]),
        prop_name, pval);
  pool_value_get_uint64(val, &size);
  printf("resource %s: size %llu load %llu\n", res_name, size, load);
  free(type);
  free(res_name);
}
free(rl);
```

Set `pool.comment` Property and Add New Property

The following example sets the `pool.comment` property for the `pset`. The example also creates a new property in `pool.newprop`.

The key point for the example includes the following:

- In the call to `pool_conf_open()`, using PO_RDWR on a static configuration file, requires the caller to be root.
- To commit these changes to the `pset` after running this utility, issue a `pooladm -c` command. To have the utility commit the changes, call `pool_conf_commit()` with a non-zero second argument.

```
pool_set_comment(const char *pool_name, const char *comment)
{
  pool_t *pool;
  pool_elem_t *pool_elem;
  pool_value_t *pval = pool_value_alloc();
  pool_conf_t  *conf = pool_conf_alloc();
  /* NOTE: need to be root to use PO_RDWR on static configuration file */
  pool_conf_open(conf, pool_static_location(), PO_RDWR);
  pool = pool_get_pool(conf,  pool_name);
  pool_value_set_string(pval, comment);
  pool_elem = pool_to_elem(conf, pool);
  pool_put_property(conf, pool_elem, "pool.comment", pval);
  printf("pool %s: pool.comment set to %s\n:" pool_name, comment);
  /* Now, create a new property, customized to installation site */
  pool_value_set_string(pval, "New String Property");
  pool_put_property(conf, pool_elem, "pool.newprop", pval);
  pool_conf_commit(conf, 0); /* NOTE: use 0 to ensure only */
                             /* static file gets updated */
  pool_value_free(pval);
  pool_conf_close(conf);
  pool_conf_free(conf);
  /* NOTE: Use "pooladm -c" later, or pool_conf_commit(conf, 1) */
  /* above for changes to the running system */
}
```

An alternative way of modifying a pool's comment and adding a new pool property is to use `poolcfg`(1M).

```
poolcfg -c 'modify pool pool-name (string pool.comment = "cmt-string")'
poolcfg -c 'modify pool pool-name (string pool.newprop =
                                   "New String Property")'
```

Programming Issues Associated With Resource Pools

Consider the following issues when writing your application.

- Each site can add its own list of properties to the pools configuration.

  Multiple configurations can be maintained in multiple configuration files. The system administrator can commit different files to reflect changes to the resource consumption at different time slots. These time slots can include different times of the day, week, month, or seasons depending on load conditions.
- Resource sets can be shared between pools, but a pool has only one resource set of a given type. So, the `pset_default` can be shared between the default and a particular application's database pools.
- Use `pool_value_*()` interfaces carefully. Keep in mind the memory allocation issues for string pool values. See "Report Pool Statistics for a Given Pool" on page 74.

CHAPTER 7

Configuration Examples

This chapter show example configurations for the `/etc/project` file.

`/etc/project` Project File

The project file is a local source of project information. The project file can be used in conjunction with other project sources, including the NIS maps `project.byname` and `project.bynumber` and the LDAP database project. Programs use the `getprojent`(3PROJECT) routines to access this information.

Define Two Projects

`/etc/project` defines two projects: `database` and `appserver`. The *user* defaults are `user.database` and `user.appserver`. The *admin* default can switch between `user.database` or `user.appserver`.

```
hostname# cat /etc/project
```

```
.
.
.
user.database:2001:Database backend:admin::
user.appserver:2002:Application Server frontend:admin::
.
.
```

Configure Resource Controls

The `/etc/project` file shows the resource controls for the application.

```
hostname# cat /etc/project

.
.
.
development:2003:Developers:::task.ax-lwps=(privileged,10,deny);
process.max-addressspace=(privileged,209715200,deny)
.
.
```

Configure Resource Pools

The `/etc/project` file shows the resource pools for the application.

```
hostname# cat /etc/project

.
.
.
batch:2001:Batch project:::project.pool=batch_pool
process:2002:Process control:::project.pool=process_pool
.
.
.
```

Configure FSS `project.cpu-shares` for a Project

Set up FSS for two projects: *database* and *appserver*. The *database* project has 20 cpu shares. The *appserver* project has 10 cpu shares.

```
hostname# cat /etc/project

.
.
.
user.database:2001:database backend:admin::project.cpu-shares=(privileged,
     20,deny)
user.appserver:2002:Application Server frontend:admin::project.cpu-shares=
     (privileged,10,deny)
.
.
.
```

Note – The line break in the lines that precede "20,deny" and "(privileged," is not valid in an /etc/project file. The line breaks are shown here only to allow the example to display on a printed or displayed page. Each entry in the /etc/project file must be on a single line.

You can assign FSS as the default user-space scheduling class. However, without share assignment, the scheduling class behaves like the timeshare class because all of the threads would exist in one thread group. Shares can be assigned in an ad hoc fashion to running processes and can also be defined as a project attribute.

Configure Five Applications with Different Characteristics

The following example configures five applications with different characteristics.

TABLE 7–1 Target Applications and Characteristics

Application Type and Name	Characteristics
Application server, app_server.	Negative scalability beyond two CPUs. Assign a two-CPU processor set to app_server. Use TS scheduling class.
Database instance, app_db.	Heavily multithreaded. Use FSS scheduling class.
Test and development, development.	Motif based. Hosts untested code execution. Interactive scheduling class ensures user interface responsiveness. Use process.max-address-space to impose memory limitations and minimize the effects of antisocial processing.
Transaction processing engine, tp_engine.	Response time is paramount. Assign a dedicated set of at least two CPUs to ensure response latency is kept to a minimum. Use timeshare scheduling class.
Standalone database instance, geo_db.	Heavily multithreaded. Serves multiple time zones. Use FSS scheduling class.

Note – Consolidate database applications (app.db and geo_db) onto a single processor set of at least four CPUs. Use FSS scheduling class. Application app_db gets 25% of the project.cpu-shares. Application geo_db gets 75% of the project.cpu-shares.

Edit the /etc/project file. Map users to resource pools for the app_server, app_db, development, tp_engine, and geo_db project entries.

```
hostname# cat /etc/project
.
.
.
user.app_server:2001:Production Application Server::
     project.pool=appserver_pool
user.app_db:2002:App Server DB:::project.pool=db_pool,
     project.cpu-shares=(privileged,1,deny)
development:2003:Test and delopment::staff:project.pool=dev.pool,
     process.max-addressspace=(privileged,536870912,deny)
user.tp_engine:Transaction Engine:::project.pool=tp_pool
user.geo_db:EDI DB:::project.pool=db_pool;
     project.cpu-shares=(privileged,3,deny)
```

Note – The line break in the lines that begin with "project.pool" , "project.cpu-shares=", "process.max-addressspace", and "project.cpu-shares=" is not valid in a project file. The line breaks are shown here only to allow the example to display on a printed or displayed page. Each entry must be on one and only one line.

Create the `pool.host` script and add entries for resource pools.

```
hostname# cat pool.host
create system host
create pset dev_pset (unit pset.max = 2)
create pset tp_pset (unit pset.min = 2)
create pset db_pset (unit pset.min = 4; uint pset.max = 6)
create pset app_pset (unit pset.min = 1; uint pset.max = 2)
create pool dev_pool (string pool.scheduler="IA")
create pool appserver_pool (string pool.scheduler="TS")
create pool db_pool (string pool.scheduler="FSS")
create pool tp_pool (string pool.scheduler="TS")
associate pool pool_default (pset pset_default)
associate pool dev_pool (pset dev_pset)
associate pool pool appserver_pool (pset app_pset)
associate pool db_pool (pset db_pset)
associate pool tp_pool (pset tp_pset)
```

Note – The line break in the line that begins with "boolean" is not valid in a `pool.host` file. The line break is shown here only to allow the example to display on a printed or displayed page. Each entry must be on one and only one line.

Run the `pool.host` script and modify the configuration as specified in the `pool.host` file.

```
hostname# poolcfg -f pool.host
```

Read the `pool.host` resource pool configuration file and initialize the resource pools on the system.

```
hostname# pooladm —c
```

Index