GWD-R

Andreas Haas, Sun Microsystems (maintainer)
Roger Brobst, Cadence Design Systems
Andreas Haas, Sun Microsystems
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud*, Sun Microsystems
*co-chairs

Distributed Resource Management
Application API (DRMAA) Working Group

June, 2003

Distributed Resource Management Application API C Bindings v1.0

Abstract

This document describes the Distributed Resource Management Application API (DRMAA) C binding. The document is based on the implementations work of the DRMAA GWD-R document.

Table of Contents

1.1 Overview

This document lists a C language binding for the DRMAA interface. For information related to interface semantics, possible argument values, error conditions etc., consult chapter "3.2 DRMAA API" of the interface specification. The C header file below is complete only with regard to the information needed by a C compiler and linker.

This document contains a few references to changes/additions of the DRMAA 1.0 proposed specification, highlighted in yellow. These changes/additions are currently being reviewed by the DRMAA-WG. If accepted, it is expected that these changes/additions would be merged with other proposed changes into a revision of the DRMAA 1.0 specification.

This C binding may be used with C++ programs through use of the extern "C" { } wrapping technique, which is widely used to import C binding interfaces in C++ programs. An example is listed in the header file.

1.2 The C header file

The header file contains a C function prototype for each interface operation described in the DRMAA interface specification. The function names in this document are always identical with the names from the interface specification.

Function prototypes and opaque data types in the header file that do not have a counterpart in the interface specification are specific to the C language binding. The DRMAA interface makes frequent use of strings, string vectors as input and output arguments. Since C language does not have a “real” string data type, a few additional opaque data types and helper functions are used to handle output string vector arguments with the actual interface calls. To minimize the complexity that was added for the C language binding compared to the language independent specification, traditional C constructs such as "const char *" and "const char *job_ids[]" are used whenever possible. As a result not much has been added compared to the "3.2 DRMAA API" description.


```
#ifndef __DRMAA_H
#define __DRMAA_H

#ifdef  __cplusplus
extern "C" {
#endif

/* ------------------- Constants ------------------- */

/*
 * Agreed buffer length constants
 * these are recommended minimum values
```

```
 */

#define DRMAA_ATTR_BUFFER           1024
#define DRMAA_CONTACT_BUFFER        1024
#define DRMAA_DRM_SYSTEM_BUFFER     1024
#define DRMAA_ERROR_STRING_BUFFER   1024
#define DRMAA_JOBNAME_BUFFER        1024
#define DRMAA_SIGNAL_BUFFER         32

/*
 * Agreed constants
 */
#define DRMAA_TIMEOUT_WAIT_FOREVER  -1
#define DRMAA_TIMEOUT_NO_WAIT       0

#define DRMAA_JOB_IDS_SESSION_ANY   "DRMAA_JOB_IDS_SESSION_ANY"
#define DRMAA_JOB_IDS_SESSION_ALL   "DRMAA_JOB_IDS_SESSION_ALL"

#define DRMAA_SUBMISSION_STATE_ACTIVE "drmaa_active"
#define DRMAA_SUBMISSION_STATE_HOLD   "drmaa_hold"

/*
 * Agreed placeholder names
 */
#define DRMAA_PLACEHOLDER_INCR       "$drmaa_incr_ph$"
#define DRMAA_PLACEHOLDER_HD         "$drmaa_hd_ph$"
#define DRMAA_PLACEHOLDER_WD         "$drmaa_wd_ph$"

/*
 * Agreed names of job template attributes
 */
#define DRMAA_REMOTE_COMMAND         "drmaa_remote_command"
#define DRMAA_JS_STATE               "drmaa_js_state"
#define DRMAA_WD                     "drmaa_wd"
#define DRMAA_JOB_CATEGORY           "drmaa_job_category"
#define DRMAA_NATIVE_SPECIFICATION   "drmaa_native_specification"
#define DRMAA_BLOCK_EMAIL            "drmaa_block_email"
#define DRMAA_START_TIME             "drmaa_start_time"
#define DRMAA_JOB_NAME               "drmaa_job_name"
#define DRMAA_INPUT_PATH             "drmaa_input_path"
#define DRMAA_OUTPUT_PATH            "drmaa_output_path"
#define DRMAA_ERROR_PATH             "drmaa_error_path"
#define DRMAA_JOIN_FILES             "drmaa_join_files"
#define DRMAA_TRANSFER_FILES         "drmaa_transfer_files"
#define DRMAA_DEADLINE_TIME          "drmaa_deadline_time"
#define DRMAA_WCT_HLIMIT             "drmaa_wct_hlimit"
#define DRMAA_WCT_SLIMIT             "drmaa_wct_slimit"
#define DRMAA_DURATION_HLIMIT        "drmaa_durartion_hlimit"
#define DRMAA_DURATION_SLIMIT        "drmaa_durartion_slimit"

/* Agreed names of job template vector attributes */
#define DRMAA_V_ARGV                 "drmaa_v_argv"
#define DRMAA_V_ENV                  "drmaa_v_env"
#define DRMAA_V_EMAIL                "drmaa_v_email"

/*
 * Agreed DRMAA errno values
 *
 * Note: The order in the enum is significant!
 */
enum {
```

```
   /* ------------- these are relevant to all sections --------------
*/
   DRMAA_ERRNO_SUCCESS = 0,
   DRMAA_ERRNO_INTERNAL_ERROR,
   DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
   DRMAA_ERRNO_AUTH_FAILURE,
   DRMAA_ERRNO_INVALID_ARGUMENT,
   DRMAA_ERRNO_NO_ACTIVE_SESSION,
   DRMAA_ERRNO_NO_MEMORY,

   /* -------------- init and exit specific --------------- */
   DRMAA_ERRNO_INVALID_CONTACT_STRING,
   DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR,
   DRMAA_ERRNO_DRMS_INIT_FAILED,
   DRMAA_ERRNO_ALREADY_ACTIVE_SESSION,
   DRMAA_ERRNO_DRMS_EXIT_ERROR,

   /* ---------------- job attributes specific -------------- */
   DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
   DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE,
   DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES,

   /* --------------------- job submission specific -------------- */
   DRMAA_ERRNO_TRY_LATER,
   DRMAA_ERRNO_DENIED_BY_DRM,

   /* ------------------------- job control specific -------------- */
   DRMAA_ERRNO_INVALID_JOB,
   DRMAA_ERRNO_RESUME_INCONSISTENT_STATE,
   DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE,
   DRMAA_ERRNO_HOLD_INCONSISTENT_STATE,
   DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE,
   DRMAA_ERRNO_EXIT_TIMEOUT,

   DRMAA_NO_ERRNO
};

/*
 * Agreed DRMAA job states as returned by drmaa_job_ps()
 */
enum {
 DRMAA_PS_UNDETERMINED          = 0x00,
 DRMAA_PS_QUEUED_ACTIVE         = 0x10,
 DRMAA_PS_SYSTEM_ON_HOLD        = 0x11,
 DRMAA_PS_USER_ON_HOLD          = 0x12,
 DRMAA_PS_USER_SYSTEM_ON_HOLD   = 0x13,
 DRMAA_PS_RUNNING               = 0x20,
 DRMAA_PS_SYSTEM_SUSPENDED      = 0x21,
 DRMAA_PS_USER_SUSPENDED        = 0x22,
 DRMAA_PS_USER_SYSTEM_SUSPENDED = 0x23,
 DRMAA_PS_DONE                  = 0x30,
 DRMAA_PS_FAILED                = 0x40
};

/*
 * Agreed DRMAA actions for drmaa_control()
 */
enum {
 DRMAA_CONTROL_SUSPEND = 0,
 DRMAA_CONTROL_RESUME,
 DRMAA_CONTROL_HOLD,
```

```
 DRMAA_CONTROL_RELEASE,
 DRMAA_CONTROL_TERMINATE
};

/* ------------------- Data types ------------------- */
/*
 * Agreed opaque DRMAA job template type
 * struct drmaa_job_template_s is defined elsewhere
 */
typedef struct drmaa_job_template_s drmaa_job_template_t;

/* ---------- C/C++ language binding specific interfaces -------- */

typedef struct drmaa_attr_names_s drmaa_attr_names_t;
typedef struct drmaa_attr_values_s drmaa_attr_values_t;
typedef struct drmaa_job_ids_s  drmaa_job_ids_t;

/*
 * get next string attribute from string vector
 *
 * returns DRMAA_ERRNO_SUCCESS or DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE
 * if no such exists
 */

int drmaa_get_next_attr_name(drmaa_attr_names_t* values, char *value,
                    int value_len);
int drmaa_get_next_attr_value(drmaa_attr_values_t* values, char *value,
                    int value_len);
int drmaa_get_next_job_id(drmaa_job_ids_t* values, char *value, int
                    value_len);

/*
 * release opaque string vector
 *
 * Opaque string vectors can be used without any constraint
 * until the release function has been called.
 */
void drmaa_release_attr_names( drmaa_attr_names_t* values );
void drmaa_release_attr_values( drmaa_attr_values_t* values );
void drmaa_release_job_ids( drmaa_job_ids_t* values );

/* ------------------- init/exit routines ------------------- */

int drmaa_init(const char *contact, char *error_diagnosis, size_t
error_diag_len);
int drmaa_exit(char *error_diagnosis, size_t error_diag_len);

/* ------------------- job template routines ------------------- */

int drmaa_allocate_job_template(drmaa_job_template_t **jt,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_delete_job_template(drmaa_job_template_t *jt,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_set_attribute(drmaa_job_template_t *jt, const char *name,
      const char *value, char *error_diagnosis, size_t error_diag_len);

int drmaa_get_attribute(drmaa_job_template_t *jt, const char *name,
      char *value, size_t value_len,
      char *error_diagnosis, size_t error_diag_len);
```

```
int drmaa_set_vector_attribute(drmaa_job_template_t *jt,
      const char *name,
      const char *value[], char *error_diagnosis, size_t
error_diag_len);

int drmaa_get_vector_attribute(drmaa_job_template_t *jt,
      const char *name,
      drmaa_attr_values_t **values,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_get_attribute_names( drmaa_attr_names_t **values,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_get_vector_attribute_names(drmaa_attr_names_t **values,
      char *error_diagnosis, size_t error_diag_len);

/* ------------------- job submission routines ------------------- */

int drmaa_run_job(char *job_id, size_t job_id_len,
      drmaa_job_template_t *jt,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_run_bulk_jobs( drmaa_job_ids_t **jobids,
      drmaa_job_template_t *jt,
      int start, int end, int incr,
      char *error_diagnosis, size_t error_diag_len);

/* ------------------- job control routines ------------------- */

int drmaa_control(const char *jobid, int action,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wait(const char *job_id, char *job_id_out,
      size_t job_id_out_len,
      int *stat, signed long timeout, drmaa_attr_values_t **rusage,
      char *error_diagnosis, size_t error_diagnois_len);

int drmaa_wifexited(int *exited, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wexitstatus(int *exit_status, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wifsignaled(int *signaled, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wtermsig(char *signal, size_t signal_len, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wcoredump(int *core_dumped, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_wifaborted(int *aborted, int stat,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_job_ps( const char *job_id, int *remote_ps,
      char *error_diagnosis, size_t error_diag_len);

/* ------------------- auxiliary routines ------------------- */
```

```
const char *drmaa_strerror(int drmaa_errno);

int drmaa_get_contact(char *contact, size_t contact_len,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_version(unsigned int *major, unsigned int *minor,
      char *error_diagnosis, size_t error_diag_len);

int drmaa_get_DRM_system(char *drm_system, size_t drm_system_len,
      char *error_diagnosis, size_t error_diag_len);

#ifdef  __cplusplus
}
#endif

#endif /* __DRMAA_H */
```

1.3 C binding example

The C test program below serves as an example of an application that uses the DRMAA C binding interface. It illustrates submission of both single and bulk remote jobs. After submission drmaa_synchronize() call is used to synchronize the remote jobs execution. The call returns after all the jobs have finished executing. Finally, drmaa_wait() call is used to retrieve and print out the remote jobs execution information.

A full path for the remote command is passed as the first argument to the test program. That value is directly used as “drmaa_remote_command” job template attribute. The C binding example uses value “5” as a first argument to the job template vector attribute “drmaa_v_argv”. Passing “/bin/sleep” as a first argument to the test program will for example cause 32 sleep jobs to be run that sleep for 5 seconds each before finishing execution. Note that we expect to find “/bin/sleep” command on all of the remote nodes.

```
#include <stdio.h>;
#include <unistd.h>;
#include <string.h>;

#include "drmaa.h"

#define JOB_CHUNK 8
#define NBULKS 3

static drmaa_job_template_t *create_job_template(const char *job_path,
              int seconds, int as_bulk_job);

int main(int argc, char *argv[])
{
   char diagnosis[DRMAA_ERROR_STRING_BUFFER];
   const char *all_jobids[NBULKS*JOB_CHUNK + JOB_CHUNK+1];
   char jobid[100];
   int drmaa_errno, i, pos = 0;
   const char *job_path;
```

```
   drmaa_job_template_t *jt;

   if (argc < 2) {
      fprintf(stderr, "usage: example path-to-job\n");
      return 1;
   }
   job_path = argv[1];

   if (drmaa_init(NULL, diagnosis, sizeof(diagnosis)-1) !=
         DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "drmaa_init() failed: %s\n", diagnosis);
      return 1;
   }

   /* submit some bulk jobs */
   if (!(jt = create_job_template(job_path, 5, 1))) {
      fprintf(stderr, "create_job_template() failed\n");
      return 1;
   }
   for (i=0; I  < NBULKS; i++) {
      drmaa_job_ids_t *jobids;
      int j;

      while ((drmaa_errno=drmaa_run_bulk_jobs(&jobids, jt, 1,
JOB_CHUNK,
                     1, diagnosis, sizeof(diagnosis)- 1))
                      ==DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE) {
         fprintf(stderr, "drmaa_run_bulk_jobs() failed - retry: %s\n",
                  diagnosis);
         sleep(1);

      }
      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "drmaa_run_bulk_jobs() failed: %s\n",
                  diagnosis);
         return 1;
      }

      printf("submitted bulk job with jobids:\n");
      for (j=0; j < JOB_CHUNK; j++) {
         drmaa_get_next_job_id(jobids, jobid, sizeof(jobid)-1);
         all_jobids[pos++] = strdup(jobid);
         printf("\t \"%s\"\n", jobid);
      }
      drmaa_release_job_ids(jobids);
   }
   drmaa_delete_job_template(jt, NULL, 0);

   /* submit some sequential jobs */
   if (!(jt = create_job_template(job_path, 5, 0))) {
      fprintf(stderr, "create_sleeper_job_template() failed\n");
      return 1;
   }
   for (i=0; I  JOB_CHUNK; i++) {
      while ((drmaa_errno=drmaa_run_job(jobid, sizeof(jobid)-1, jt,
               diagnosis, sizeof(diagnosis)-1)) ==
               DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE) {
         fprintf(stderr, "drmaa_run_job() failed - retry: %s\n",
                  diagnosis);
         sleep(1);
      }
```

```
      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "drmaa_run_job() failed: %s\n", diagnosis);
         return 1;
      }
      printf("\t \"%s\"\n", jobid);
      all_jobids[pos++] = strdup(jobid);
   }

   /* set string array end mark */
   all_jobids[pos] = NULL;

   drmaa_delete_job_template(jt, NULL, 0);

   /* synchronize with all jobs */
   drmaa_errno = drmaa_synchronize(all_jobids,
                 DRMAA_TIMEOUT_WAIT_FOREVER, 0,
                 diagnosis, sizeof(diagnosis)-1);
   if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "drmaa_synchronize(DRMAA_JOB_IDS_SESSION_ALL,
                    dispose) failed: %s\n", diagnosis);
      return 1;
   }
   printf("synchronized with all jobs\n");

   /* wait all those jobs */
   for (pos=0; pos < NBULKS*JOB_CHUNK + JOB_CHUNK; pos++) {
      int stat;
      int aborted, exited, exit_status, signaled;

      drmaa_errno = drmaa_wait(all_jobids[pos], jobid, sizeof(jobid)-1,
         &stat, DRMAA_TIMEOUT_WAIT_FOREVER, NULL,
          diagnosis, sizeof(diagnosis)-1);

      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "drmaa_wait(%s) failed: %s\n",
                   all_jobids[pos], diagnosis);
         return 1;
      }

      /* report how job finished */
      drmaa_wifaborted(&aborted, stat, NULL, 0);
      if (aborted)
         printf("job \"%s\" never ran\n", all_jobids[pos]);
      else {
         drmaa_wifexited(&exited, stat, NULL, 0);
         if (exited) {
            drmaa_wexitstatus(&exit_status, stat, NULL, 0);
            printf("job \"%s\" finished regularly with exit status
d\n",
                   all_jobids[pos], exit_status);
         } else {
            drmaa_wifsignaled(&signaled, stat, NULL, 0);
            if (signaled) {
               char termsig[DRMAA_SIGNAL_BUFFER+1];
               drmaa_wtermsig(termsig, DRMAA_SIGNAL_BUFFER, stat,
                    NULL, 0);
               printf("job \"%s\" finished due to signal %s\n",
                  all_jobids[pos], termsig);
            } else
               printf("job \"%s\" finished with unclear conditions\n",
                  all_jobids[pos]);
```

```
         }
      }
   }

   if (drmaa_exit(diagnosis, sizeof(diagnosis)-1) !=
DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "drmaa_exit() failed: %s\n", diagnosis);
      return 1;
   }

  return 0;
}

static drmaa_job_template_t *create_job_template(const char *job_path,
        int seconds, int as_bulk_job)
{
   const char *job_argv[2];
   drmaa_job_template_t *jt = NULL;
   char buffer[100];

   if (drmaa_allocate_job_template(&jt, NULL, 0)!=DRMAA_ERRNO_SUCCESS)
      return NULL;

   /* run in users home directory */
   drmaa_set_attribute(jt, DRMAA_WD, DRMAA_PLACEHOLDER_HD, NULL, 0);

   /* the job to be run */
   drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, job_path, NULL, 0);

   /* the job's arguments */
   sprintf(buffer, "%d", seconds);
   job_argv[0] = buffer;
   job_argv[1] = NULL;
   drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, job_argv, NULL, 0);

   /* join output/error file */
   drmaa_set_attribute(jt, DRMAA_JOIN_FILES, "y", NULL, 0);

   /* path for output */
   if (!as_bulk_job)
      drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH,
               DRMAA_PLACEHOLDER_HD"/DRMAA_JOB",  NULL, 0);
   else
      drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH,
               DRMAA_PLACEHOLDER_HD"/DRMAA_JOB."DRMAA_PLACEHOLDER_INCR,
               NULL, 0);

   return jt;
}
```

Security Considerations

Security issues are not discussed in this document. The scheduling scenario described here assumes that security is handled at the point of job authorization/execution on a particular resource.

Author Information

Roger Brobst
rbrobst@cadence.com
Cadence Design Systems, Inc
555 River Oaks Parkway
San Jose, CA 95134

Andreas Haas
andreas.haas@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

Hrabri L. Rajic
hrabri.rajic@intel.com
Intel Americas Inc.
1906 Fox Drive
Champaign, IL 61820

John Tollefsrud
j.t@sun.com
Sun Microsystems
200 Jefferson Drive UMPK29-302
Menlo Park, CA 94025

Intellectual Property Statement

Full Copyright Notice