GWD-R
DRMAA-WG
drmaa-wg@ogf.org

Peter Tröger, Hasso Plattner Institute[1]
Roger Brobst, Cadence Design Systems
Daniel Gruber, Univa
Mariusz Mamoński, PSNC
Andre Merzky, LSU
April 2012

Distributed Resource Management Application API Version 2 (DRMAA) - C Language Binding

Status of This Document

Group Working Draft - Proposed Recommendation (GWD-R)

(See footnote)[1]

Document Change History

Date	*Notes*
April 17th, 2012	Final Draft

Copyright Notice

Trademark

Abstract

This document describes the C language binding for the *Distributed Resource Management Application API Version 2 (DRMAA)*. The intended audience for this specification are DRMAA Version 2 interface implementors.

[1] This is the non-normative annotated version of the specification with line numbers. It includes historical information concerning the content and why features were included or discarded by the working group. It also emphasizes the consequences of some aspects that may not be immediately apparent. This document in only intended for internal working group discussions.

[1] Corresponding author

Notational Conventions

In this document, C language elements and definitions are represented in a `fixed-width` font.

The key words “MUST” “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 [1].

Contents

1 Introduction

The *Distributed Resource Management Application API Version 2 (DRMAA)* specification defines an interface for tightly coupled, but still portable access to the majority of DRM systems. The scope is limited to job submission, job control, reservation management, and retrieval of job and machine monitoring information.

The *DRMAA root specification* [2] describes the abstract API concepts and the behavioral rules of a compliant implementation, while this document standardizes the representation of API concepts in the C programming language.

2 General Design

The mapping of DRMAA IDL constructs to C follows a set of design principles. Implementation-specific extensions of the DRMAA C API described here SHOULD follow these conventions for their own naming and method signatures:

- Namespacing of the DRMAA API, as demanded by by the root specification, is realized with the `drmaa2_` prefix for lower- and upper-case identifiers.
- In identifier naming, "job" is shortened as "j" and "reservation" is shortened as "r" for improved readability.
- The root specification demands a consistent parameter passing strategy for non-scalar values. In DRMAA for C, all such values are passed as call-by-reference parameter.
- Structs and enums are typedef'ed for better readability.
- Struct types get a `_s` suffix on their name. Structures with a non-standardized layout are defined as forward references for the DRMAA implementation. (See footnote)[2]
- Functions with IDL return type `void` have `drmaa2_error` as return type.
- The IDL `boolean` type maps to the `drmaa2_boolean` type.
- The IDL `long` type maps to `long long` in C. One exception is the `exitStatus` variable, which is defined as `int` in order to provide a more natural mapping to operating system interfaces.
- The IDL `string` type maps to `char*` pointer. The allocation of memory for strings returned SHALL be done by the implementation itself. The application frees such memory regions by calling the newly introduced function `drmaa2_string_free`.
- The language binding defines one UNSET macro per utilized C data type (`DRMAA2_UNSET_*`). (See footnote)[3]
- All numerical types are signed, in order to support `-1` as numerical UNSET value.
- Application-created structs should be allocated by the additional support methods (such as `drmaa2_jinfo_create`) to realize the neccessary initialization to UNSET.

[2] This avoids the usage of void* pointers, f.e. with dictionaries and lists.

[3] For UNSET values, the language binding adheres mainly to typical language conventions and not to GLUE as mandated in the root spec.

- All structures have a specific support function for freeing them (`drmaa2_*_free`). (See footnote)[4]
- Both `AbsoluteTime` and `TimeAmount` map directly to `time_t`. RFC 822 support as mandated by the root specification is given by the `%z` formatter for `sprintf`.
- Multiple output parameters are realized by declaring all but one of them as pointer variable. For this reason, the `substate` parameter in `drmaa2_j_get_state` SHALL be interpreted as pointer to a character pointer variable. The DRMAA library creates the buffer and stores the pointer to it as variable value.
- The `const` declarator is used to mark parameters declared as `readonly` in the root specification.
- The two string list types in DRMAA, ordered and unordered, are mapped to one ordered list with the `DRMAA2_STRING_LIST` type.
- The largest possible value for `end_index` in `drmaa2_js_run_bulk_jobs` SHOULD be `sizeof(unsigned long)-1`.
- The `any` member for job sub-state information is defined as `char*`, in order to achieve application portability.

The following structures are only used in result values. For this reason, the according allocation functions are not part of the API:

- `drmaa2_slotinfo`
- `drmaa2_rinfo`
- `drmaa2_notification`
- `drmaa2_queueinfo`
- `drmaa2_version`
- `drmaa2_machineinfo`

The interface membership of a function is mostly expressed by an additional prefix, as show in Table 1.

DRMAA interface	C binding prefix
`DrmaaReflective`	`drmaa2_`
`SessionManager`	`drmaa2_`
`JobSession`	`drmaa2_jsession_`
`ReservationSession`	`drmaa2_rsession_`
`MonitoringSession`	`drmaa2_msession_`
`Reservation`	`drmaa2_r_`
`Job`	`drmaa2_j_`
`JobArray`	`drmaa2_jarray_`
`JobTemplate`	`drmaa2_jtemplate_`
`ReservationTemplate`	`drmaa2_rtemplate_`

Table 1: Mapping of DRMAA interfaces to C method prefix

[4]The deallocation functions are needed to make sure that the allocating entity (the library) also performs the freeing operation. This is needed for cases where the DRMAA library is compiled with a different heap allocator than the DRMAA-based application. It is mainly a problem with Windows-based implementations.

The C binding specifies the function pointer type `drmaa2_callback_t` for a notification callback function, in order to map the IDL `DrmaaCallback` interface. The new constant value `DRMAA2_UNSET_CALLBACK` can be used by the application for the de-registration of callback functions.

2.1 Error Handling

The list of exceptions in the DRMAA root specification is mapped to the new enumeration `drmaa2_error`. The enumeration member `DRMAA2_LASTERROR` is intended to ensure application portability while allowing additional implementation-specific error codes. It MUST always be the enumeration member with the highest value.

The language binding adds two new functions for fetching error number and error message of the last error that occurred: `drmaa2_lasterror` and `drmaa2_lasterror_text`. These functions MUST operate in a thread-safe manner, meaning that both error informations are managed per application thread by the DRMAA implementation.

2.2 Lists and Dictionaries

The C language binding adds generic support functions for the collection data types used by the root specification. The newly defined `drmaa2_lasterror` and `drmaa2_lasterror_text` functions MUST return according error information for these operations.

(See footnote)[5]

Both `drmaa2_list_create` and `drmaa2_dict_create` have an optional parameter `callback`. It allows the application to provide a callback pointer to a collection element cleanup function. This function MUST be called by the implementation once per stored item when the list / dictionary is freed. It MUST be allowed for the application to provide NULL instead of a valid callback pointer.

(See footnote)[6]

The following list operations are defined:

`drmaa2_list_create`**:** Creates a new list instance for the specified type of items. Returns a pointer to the list or NULL on error.

`drmaa2_list_free`**:** Frees the list and the contained members and returns a success indication.

`drmaa2_list_get`**:** Gets the list element at the indicated position. The element index starts at zero. If the index is invalid, the function returns NULL.

`drmaa2_list_add`**:** Adds a new item at the end of the list and returns a success indication. The list MUST contain only a pointer copy, not a deep copy of the provided data structure.

`drmaa2_list_remove`**:** Removes the list element at the indicated position and returns a success indication.

`drmaa2_list_size`**:** Gets the number of elements in the list. If the list is empty, then the function returns 0, which SHALL NOT be treated as an error case.

Similarly, a set of new functions for dictionary handling is introduced:

[5]The definition of list operations in the language binding keeps the application code portable. The original DRMAA error codes are good enough to support them, there is no need for additional ones. DRMAA dictionaries are only used for strings, so we make the dictionary interface less general.

[6]This is again for the heap allocator problem. The entity allocating some memory should also free it.

`drmaa2_dict_create`: Creates a new dictionary instance. Returns a pointer to the dictionary or NULL on error.

`drmaa2_dict_free`: Frees the dictionary and the contained members and returns a success indication.

`drmaa2_dict_list`: Gets all dictionary keys as DRMAA `drmaa2_string_list`. If the dictionary is empty, a valid string list with zero elements SHALL be returned. The application is expected to use `drmaa2_list_free` for freeing the returned data structure.

`drmaa2_dict_has`: Returns an indication if the given key exists in the dictionary. On error, the function SHALL return NULL as result.

`drmaa2_dict_get`: Gets the dictionary value for the specified key. If the key is invalid, the function returns NULL.

`drmaa2_dict_del`: Removes the dictionary entry with the given key and returns a success indication.

`drmaa2_dict_set`: Sets the specified dictionary key to the specified value. Key and value strings MUST be copied as pointer. If the dictionary already has an entry for this name, the value is replaced and the old value is removed.

3 Implementation-specific Extensions

The DRMAA root specification allows the product-specific extension of the DRMAA API in a standardized way.

New methods added to a DRMAA implementation SHOULD follow the conventions from Section 2. Extended `struct` definitions SHOULD use a product-specific prefix for a clear separation of non-portable and portable parts of the API. The extension MUST support the casting of product-specific `struct` pointers to their standard-compliant counterparts (see Listing 1). Any compiler or linking options necessary for this feature MUST be documented accordingly by the DRMAA implementation.

Listing 1: Code example for implementation-specific extension

```
typedef struct
{
     [attributes from drmaa2_jtemplate_s] ...
     int gridengine_specific_attr;
} gridengine_jtemplate_s;
typedef gridengine_jtemplate_s * gridengine_jtemplate;
```

4 Complete Header File

The following text shows the complete C header file for the DRMAAv2 application programming interface. DRMAA-compliant C libraries MUST declare all functions and data structures described here. Implementations MAY add custom parts in adherence to the extensibility principles of this specification and the root specification.

The source file is also available at `http://www.drmaa.org`.

```
#include <time.h>

#ifndef DRMAA2_H
#define DRMAA2_H

typedef enum drmaa2_jstate {
```

```
165   DRMAA2_UNDETERMINED               =  0,
166   DRMAA2_QUEUED                     =  1,
167   DRMAA2_QUEUED_HELD                =  2,
168   DRMAA2_RUNNING                    =  3,
169   DRMAA2_SUSPENDED                  =  4,
170   DRMAA2_REQUEUED                   =  5,
171   DRMAA2_REQUEUED_HELD              =  6,
172   DRMAA2_DONE                       =  7,
173   DRMAA2_FAILED                     =  8
174 } drmaa2_jstate;
175
176 typedef enum drmaa2_os {
177   DRMAA2_OTHER_OS                   =  0,
178   DRMAA2_AIX                        =  1,
179   DRMAA2_BSD                        =  2,
180   DRMAA2_LINUX                      =  3,
181   DRMAA2_HPUX                       =  4,
182   DRMAA2_IRIX                       =  5,
183   DRMAA2_MACOS                      =  6,
184   DRMAA2_SUNOS                      =  7,
185   DRMAA2_TRUE64                     =  8,
186   DRMAA2_UNIXWARE                   =  9,
187   DRMAA2_WIN                        = 10,
188   DRMAA2_WINNT                      = 11
189 } drmaa2_os;
190
191 typedef enum drmaa2_cpu {
192   DRMAA2_OTHER_CPU                  =  0,
193   DRMAA2_ALPHA                      =  1,
194   DRMAA2_ARM                        =  2,
195   DRMAA2_CELL                       =  3,
196   DRMAA2_PARISC                     =  4,
197   DRMAA2_X86                        =  5,
198   DRMAA2_X64                        =  6,
199   DRMAA2_IA64                       =  7,
200   DRMAA2_MIPS                       =  8,
201   DRMAA2_PPC                        =  9,
202   DRMAA2_PPC64                      = 10,
203   DRMAA2_SPARC                      = 11,
204   DRMAA2_SPARC64                    = 12
205 } drmaa2_cpu;
206
207 typedef enum drmaa2_limit {
208   DRMAA2_CORE_FILE_SIZE             =  0,
209   DRMAA2_CPU_TIME                   =  1,
210   DRMAA2_DATA_SEG_SIZE              =  2,
211   DRMAA2_FILE_SIZE                  =  3,
212   DRMAA2_OPEN_FILES                 =  4,
213   DRMAA2_STACK_SIZE                 =  5,
214   DRMAA2_VIRTUAL_MEMORY             =  6,
215   DRMAA2_WALLCLOCK_TIME             =  7
216 } drmaa2_limit;
217
218 typedef enum drmaa2_jtemplate_placeholder {
219   DRMAA2_HOME_DIRECTORY             =  0,
220   DRMAA2_WORKING_DIRECTORY          =  1,
221   DRMAA2_PARAMETRIC_INDEX           =  2
222 } drmaa2_jtemplate_placeholder;
223
224 typedef enum drmaa2_event {
225   DRMAA2_NEW_STATE                  =  0,
226   DRMAA2_MIGRATED                   =  1,
227   DRMAA2_ATTRIBUTE_CHANGE           =  2
228 } drmaa2_event;
229
230 typedef enum {
231   DRMAA2_ADVANCE_RESERVATION        =  0,
232   DRMAA2_RESERVE_SLOTS              =  1,
233   DRMAA2_CALLBACK                   =  2,
234   DRMAA2_BULK_JOBS_MAXPARALLEL      =  3,
235   DRMAA2_JT_EMAIL                   =  4,
```

```
236   DRMAA2_JT_STAGING                  =  5,
237   DRMAA2_JT_DEADLINE                 =  6,
238   DRMAA2_JT_MAXSLOTS                 =  7,
239   DRMAA2_JT_ACCOUNTINGID             =  8,
240   DRMAA2_RT_STARTNOW                 =  9,
241   DRMAA2_RT_DURATION                 = 10,
242   DRMAA2_RT_MACHINEOS                = 11,
243   DRMAA2_RT_MACHINEARCH              = 12
244 } drmaa2_capability;
245
246 typedef enum drmaa2_bool {
247   DRMAA2_FALSE                       =  0,
248   DRMAA2_TRUE                        =  1
249 } drmaa2_bool;
250
251 typedef enum drmaa2_error {
252   DRMAA2_SUCCESS                 =  0,
253   DRMAA2_DENIED_BY_DRMS          =  1,
254   DRMAA2_DRM_COMMUNICATION       =  2,
255   DRMAA2_TRY_LATER               =  3,
256   DRMAA2_SESSION_MANAGEMENT      =  4,
257   DRMAA2_TIMEOUT                 =  5,
258   DRMAA2_INTERNAL                =  6,
259   DRMAA2_INVALID_ARGUMENT        =  7,
260   DRMAA2_INVALID_SESSION         =  8,
261   DRMAA2_INVALID_STATE           =  9,
262   DRMAA2_OUT_OF_RESOURCE         = 10,
263   DRMAA2_UNSUPPORTED_ATTRIBUTE   = 11,
264   DRMAA2_UNSUPPORTED_OPERATION   = 12,
265   DRMAA2_IMPLEMENTATION_SPECIFIC = 13,
266   DRMAA2_LASTERROR               = 14
267 } drmaa2_error;
268
269 drmaa2_error drmaa2_string_free(char*);
270
271 drmaa2_error drmaa2_lasterror(void);
272 char *       drmaa2_lasterror_text(void);
273
274 struct drmaa2_list_s;        /*forward*/
275 typedef struct drmaa2_list_s * drmaa2_list;
276 typedef struct drmaa2_list_s * drmaa2_string_list;
277 typedef struct drmaa2_list_s * drmaa2_j_list;
278 typedef struct drmaa2_list_s * drmaa2_queueinfo_list;
279 typedef struct drmaa2_list_s * drmaa2_machineinfo_list;
280 typedef struct drmaa2_list_s * drmaa2_slotinfo_list;
281 typedef struct drmaa2_list_s * drmaa2_r_list;
282
283 typedef enum drmaa2_listtype {
284   DRMAA2_STRINGLIST,
285   DRMAA2_JOBLIST,
286   DRMAA2_QUEUEINFOLIST,
287   DRMAA2_MACHINEINFOLIST,
288   DRMAA2_SLOTINFOLIST,
289   DRMAA2_RESERVATIONLIST
290 } drmaa2_listtype;
291
292 typedef void (*drmaa2_list_entryfree)(void * value);
293 drmaa2_list  drmaa2_list_create (const drmaa2_listtype t, const drmaa2_list_entryfree callback);
294 drmaa2_error drmaa2_list_free   (      drmaa2_list l);
295 const void * drmaa2_list_get    (      drmaa2_list l, int pos);
296 drmaa2_error drmaa2_list_add    (      drmaa2_list l, const void * value);
297 drmaa2_error drmaa2_list_del    (      drmaa2_list l, int pos);
298 int          drmaa2_list_size   (const drmaa2_list l);
299
300 struct drmaa2_dict_s;        /*forward*/
301 typedef struct drmaa2_dict_s * drmaa2_dict;
302
303 typedef void (*drmaa2_dict_entryfree)(char * value);
304 drmaa2_dict        drmaa2_dict_create (const drmaa2_dict_entryfree callback);
305 drmaa2_error       drmaa2_dict_free   (drmaa2_dict d);
306 drmaa2_string_list drmaa2_dict_list   (const drmaa2_dict d);
```

```
drmaa2_bool        drmaa2_dict_has    (const drmaa2_dict d, const char * key);
const char *       drmaa2_dict_get    (const drmaa2_dict d, const char * key);
drmaa2_error       drmaa2_dict_del    (      drmaa2_dict d, const char * key);
drmaa2_error       drmaa2_dict_set    (      drmaa2_dict d, const char * key, const char * val);

#define  DRMAA2_ZERO_TIME       ((time_t)  0)
#define  DRMAA2_INFINITE_TIME   ((time_t) -1)
#define  DRMAA2_NOW             ((time_t) -2)

#define  DRMAA2_UNSET_BOOL      DRMAA2_FALSE
#define  DRMAA2_UNSET_STRING    NULL
#define  DRMAA2_UNSET_NUM       -1
#define  DRMAA2_UNSET_ENUM      -1
#define  DRMAA2_UNSET_LIST      NULL
#define  DRMAA2_UNSET_DICT      NULL
#define  DRMAA2_UNSET_TIME      ((time_t) -3)
#define  DRMAA2_UNSET_CALLBACK  NULL


typedef struct {
  char *             jobId;
  int                exitStatus;
  char *             terminatingSignal;
  char *             annotation;
  drmaa2_jstate      jobState;
  char *             jobSubState;
  drmaa2_string_list allocatedMachines;
  char *             submissionMachine;
  char *             jobOwner;
  long long          slots;
  char *             queueName;
  time_t             wallclockTime;
  long long          cpuTime;
  time_t             submissionTime;
  time_t             dispatchTime;
  time_t             finishTime;
} drmaa2_jinfo_s;
typedef drmaa2_jinfo_s * drmaa2_jinfo;

drmaa2_jinfo drmaa2_jinfo_create (void);
drmaa2_error drmaa2_jinfo_free   (drmaa2_jinfo ji);

typedef struct {
  char *                machineName;
  long long             slots;
} drmaa2_slotinfo_s;
typedef drmaa2_slotinfo_s * drmaa2_slotinfo;

drmaa2_error drmaa2_slotinfo_free   (drmaa2_slotinfo si);

typedef struct {
  char *               reservationId;
  char *               reservationName;
  time_t               reservedStartTime;
  time_t               reservedEndTime;
  drmaa2_string_list   usersACL;
  long long            reservedSlots;
  drmaa2_slotinfo_list reservedMachines;
} drmaa2_rinfo_s;
typedef drmaa2_rinfo_s * drmaa2_rinfo;

drmaa2_error drmaa2_rinfo_free   (drmaa2_rinfo ri);

typedef struct {
  char *             remoteCommand;
  drmaa2_string_list args;
  drmaa2_bool        submitAsHold;
  drmaa2_bool        rerunnable;
  drmaa2_dict        jobEnvironment;
  char *             workingDirectory;
  char *             jobCategory;
```

```
  drmaa2_string_list email;
  drmaa2_bool        emailOnStarted;
  drmaa2_bool        emailOnTerminated;
  char *             jobName;
  char *             inputPath;
  char *             outputPath;
  char *             errorPath;
  drmaa2_bool        joinFiles;
  char *             reservationId;
  char *             queueName;
  long long          minSlots;
  long long          maxSlots;
  long long          priority;
  drmaa2_string_list candidateMachines;
  long long          minPhysMemory;
  drmaa2_os          machineOS;
  drmaa2_cpu         machineArch;
  time_t             startTime;
  time_t             deadlineTime;
  drmaa2_dict        stageInFiles;
  drmaa2_dict        stageOutFiles;
  drmaa2_dict        resourceLimits;
  char *             accountingId;
} drmaa2_jtemplate_s;
typedef drmaa2_jtemplate_s * drmaa2_jtemplate;

drmaa2_jtemplate drmaa2_jtemplate_create   (void);
drmaa2_error     drmaa2_jtemplate_free     (drmaa2_jtemplate jt);
char*            drmaa2_jtemplate_tostring (drmaa2_jtemplate jt);

typedef struct {
  char *             reservationName;
  time_t             startTime;
  time_t             endTime;
  time_t             duration;
  long long          minSlots;
  long long          maxSlots;
  char *             jobCategory;
  drmaa2_string_list usersACL;
  drmaa2_string_list candidateMachines;
  long long          minPhysMemory;
  drmaa2_os          machineOS;
  drmaa2_cpu         machineArch;
} drmaa2_rtemplate_s;
typedef drmaa2_rtemplate_s * drmaa2_rtemplate;

drmaa2_rtemplate     drmaa2_rtemplate_create (void);
drmaa2_error         drmaa2_rtemplate_free   (drmaa2_rtemplate rt);

typedef struct {
  drmaa2_event   event;
  char *         jobId;
  char *         sessionName;
  drmaa2_jstate  jobState;
} drmaa2_notification_s;
typedef drmaa2_notification_s * drmaa2_notification;

drmaa2_error drmaa2_notification_free   (drmaa2_notification n);

typedef struct {
  char *                       name;
} drmaa2_queueinfo_s;
typedef drmaa2_queueinfo_s * drmaa2_queueinfo;

drmaa2_error drmaa2_queueinfo_free   (drmaa2_queueinfo qi);

typedef struct {
  char *                       major;
  char *                       minor;
} drmaa2_version_s;
typedef drmaa2_version_s * drmaa2_version;
```

```
449
450 drmaa2_error drmaa2_version_free   (drmaa2_version v);
451
452 typedef struct {
453   char *          name;
454   drmaa2_bool     available;
455   long long       sockets;
456   long long       coresPerSocket;
457   long long       threadsPerCore;
458   float           load;
459   long long       physMemory;
460   long long       virtMemory;
461   drmaa2_os       machineOS;
462   drmaa2_version  machineOSVersion;
463   drmaa2_cpu      machineArch;
464 } drmaa2_machineinfo_s;
465 typedef drmaa2_machineinfo_s * drmaa2_machineinfo;
466
467 drmaa2_error drmaa2_machineinfo_free   (drmaa2_machineinfo mi);
468
469 drmaa2_string_list drmaa2_jtemplate_impl_spec     (void);
470 drmaa2_string_list drmaa2_jinfo_impl_spec         (void);
471 drmaa2_string_list drmaa2_rtemplate_impl_spec     (void);
472 drmaa2_string_list drmaa2_rinfo_impl_spec         (void);
473 drmaa2_string_list drmaa2_queueinfo_impl_spec     (void);
474 drmaa2_string_list drmaa2_machineinfo_impl_spec   (void);
475 drmaa2_string_list drmaa2_notification_impl_spec  (void);
476
477 char *       drmaa2_get_instance_value (const void * instance, const char * name);
478 char *       drmaa2_describe_attribute (const void * instance, const char * name);
479 drmaa2_error drmaa2_set_instance_value (      void * instance, const char * name, const char * value);
480
481 typedef void (*drmaa2_callback)(drmaa2_notification * notification);
482
483 struct drmaa2_jsession_s; /*forward*/
484 struct drmaa2_rsession_s; /*forward*/
485 struct drmaa2_msession_s; /*forward*/
486 struct drmaa2_j_s;        /*forward*/
487 struct drmaa2_jarray_s;   /*forward*/
488 struct drmaa2_r_s;        /*forward*/
489
490 typedef struct drmaa2_jsession_s * drmaa2_jsession;
491 typedef struct drmaa2_rsession_s * drmaa2_rsession;
492 typedef struct drmaa2_msession_s * drmaa2_msession;
493 typedef struct drmaa2_j_s        * drmaa2_j;
494 typedef struct drmaa2_jarray_s   * drmaa2_jarray;
495 typedef struct drmaa2_r_s        * drmaa2_r;
496
497 char *         drmaa2_rsession_get_contact          (const drmaa2_rsession rs);
498 char *         drmaa2_rsession_get_session_name     (const drmaa2_rsession rs);
499 drmaa2_r       drmaa2_rsession_get_reservation      (const drmaa2_rsession rs, const char * reservation_id);
500 drmaa2_r       drmaa2_rsession_request_reservation  (const drmaa2_rsession rs, const drmaa2_rtemplate rt);
501 drmaa2_r_list  drmaa2_rsession_get_reservations     (const drmaa2_rsession rs);
502
503 char *            drmaa2_r_get_id                   (const drmaa2_r r);
504 char *            drmaa2_r_get_session_name         (const drmaa2_r r);
505 drmaa2_rtemplate  drmaa2_r_get_reservation_template (const drmaa2_r r);
506 drmaa2_rinfo      drmaa2_r_get_info                 (const drmaa2_r r);
507 drmaa2_error      drmaa2_r_terminate                (drmaa2_r r);
508
509 char *           drmaa2_jarray_get_id            (const drmaa2_jarray ja);
510 drmaa2_j_list    drmaa2_jarray_get_jobs          (const drmaa2_jarray ja);
511 char *           drmaa2_jarray_get_session_name  (const drmaa2_jarray ja);
512 drmaa2_jtemplate drmaa2_jarray_get_job_template  (const drmaa2_jarray ja);
513 drmaa2_error     drmaa2_jarray_suspend           (drmaa2_jarray ja);
514 drmaa2_error     drmaa2_jarray_resume            (drmaa2_jarray ja);
515 drmaa2_error     drmaa2_jarray_hold              (drmaa2_jarray ja);
516 drmaa2_error     drmaa2_jarray_release           (drmaa2_jarray ja);
517 drmaa2_error     drmaa2_jarray_terminate         (drmaa2_jarray ja);
518
519 char *             drmaa2_jsession_get_contact         (const drmaa2_jsession js);
```

```
char *              drmaa2_jsession_get_session_name    (const drmaa2_jsession js);
drmaa2_string_list  drmaa2_jsession_get_job_categories  (const drmaa2_jsession js);
drmaa2_j_list       drmaa2_jsession_get_jobs            (const drmaa2_jsession js,
                                                         const drmaa2_jinfo filter);
drmaa2_jarray       drmaa2_jsession_get_job_array       (const drmaa2_jsession js,
                                                         const char * jobarray_id);
drmaa2_j            drmaa2_jsession_run_job             (const drmaa2_jsession js,
                                                         const drmaa2_jtemplate jt);
drmaa2_jarray       drmaa2_jsession_run_bulk_jobs       (const drmaa2_jsession js,
                                                         const drmaa2_jtemplate jt,
                                                         unsigned long begin_index,
                                                         unsigned long end_index,
                                                         unsigned long step,
                                                         unsigned long max_parallel);
drmaa2_j            drmaa2_jsession_wait_any_started    (const drmaa2_jsession js,
                                                         const drmaa2_j_list l,
                                                         const time_t timeout);
drmaa2_j            drmaa2_jsession_wait_any_terminated (const drmaa2_jsession js,
                                                         const drmaa2_j_list l,
                                                         const time_t timeout);

char *              drmaa2_j_get_id            (const drmaa2_j j);
char *              drmaa2_j_get_session_name  (const drmaa2_j j);
drmaa2_jtemplate    drmaa2_j_get_jt            (const drmaa2_j j);
drmaa2_error        drmaa2_j_suspend           (drmaa2_j j);
drmaa2_error        drmaa2_j_resume            (drmaa2_j j);
drmaa2_error        drmaa2_j_hold              (drmaa2_j j);
drmaa2_error        drmaa2_j_release           (drmaa2_j j);
drmaa2_error        drmaa2_j_terminate         (drmaa2_j j);
drmaa2_jstate       drmaa2_j_get_state         (const drmaa2_j j, char ** substate);
drmaa2_jinfo        drmaa2_j_get_info          (const drmaa2_j j);
drmaa2_j            drmaa2_j_wait_started      (const drmaa2_j j, const time_t timeout);
drmaa2_j            drmaa2_j_wait_terminated   (const drmaa2_j j, const time_t timeout);

drmaa2_r_list             drmaa2_msession_get_all_reservations  (const drmaa2_msession ms);
drmaa2_j_list             drmaa2_msession_get_all_jobs          (const drmaa2_msession ms,
                                                                 const drmaa2_jinfo filter);
drmaa2_queueinfo_list     drmaa2_msession_get_all_queues        (const drmaa2_msession ms,
                                                                 const drmaa2_string_list names);
drmaa2_machineinfo_list   drmaa2_msession_get_all_machines      (const drmaa2_msession ms,
                                                                 const drmaa2_string_list names);

char *              drmaa2_get_drms_name               (void);
drmaa2_version      drmaa2_get_drms_version            (void);
char *              drmaa2_get_drmaa_name              (void);
drmaa2_version      drmaa2_get_drmaa_version           (void);
drmaa2_bool         drmaa2_supports                    (const drmaa2_capability c);
drmaa2_jsession     drmaa2_create_jsession             (const char * session_name, const char * contact);
drmaa2_rsession     drmaa2_create_rsession             (const char * session_name, const char * contact);
drmaa2_jsession     drmaa2_open_jsession               (const char * session_name);
drmaa2_rsession     drmaa2_open_rsession               (const char * session_name);
drmaa2_msession     drmaa2_open_msession               (const char * session_name);
drmaa2_error        drmaa2_close_jsession              (drmaa2_jsession js);
drmaa2_error        drmaa2_close_rsession              (drmaa2_rsession rs);
drmaa2_error        drmaa2_close_msession              (drmaa2_msession ms);
drmaa2_error        drmaa2_destroy_jsession            (const char * session_name);
drmaa2_error        drmaa2_destroy_rsession            (const char * session_name);
drmaa2_string_list  drmaa2_get_jsession_names          (void);
drmaa2_string_list  drmaa2_get_rsession_names          (void);
drmaa2_error        drmaa2_register_event_notification (const drmaa2_callback callback);

#endif
```

5 Security Considerations

The DRMAA root specification [2] describes the behavioral aspects of a standard-compliant implementation. This includes also security aspects.

Software written in C language has well-known security attack vectors, especially with memory handling. Implementors MUST clarify in their documentation which kind of memory management is expected by the application. Implementations MUST also consider the possibility for multi-threaded applications performing re-entrant calls to the library. The root specification clarifies some of the behavioral aspects with this.

6 Contributors

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

Daniel Gruber
Univa GmbH
c/o Rüter und Partner
Prielmayerstr. 3
80335 München, Germany
Email: dgruber@univa.com

Mariusz Mamoński
Poznań Supercomputing and Networking Center
ul. Noskowskiego 10
61-704 Poznań, Poland
Email: mamonski@man.poznan.pl

Andre Merzky
Center for Computation and Technology
Louisiana State University
216 Johnston Hall
70803 Baton Rouge, Louisiana, USA
Email: andre@merzky.net

Peter Tröger (Corresponding Author)
Hasso Plattner Institute at University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
Email: peter@troeger.eu

7 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that

it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

8 Disclaimer

This document and the information contained herein is provided on an "as-is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

9 Full Copyright Notice

10 References

[1] Scott Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997. URL `http://tools.ietf.org/html/rfc2119`.

[2] Peter Tröger, Roger Brobst, Daniel Gruber, Mariusz Mamonski, and Daniel Templeton. Distributed Resource Management Application API Version 2 (DRMAA). `http://www.ogf.org/documents/GFD.194.pdf`, January 2012.