Distributed Resource Management Application API Version 2.2
( DRMAA )
C Language Binding

Status of This Document
OGF Proposed Recommendation (GFD-R-P.230)

Obsoleses
This document obsoletes GFD-R-P.198 [2].

Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 26th, 2012</td>
<td>Submission to OGF Editor</td>
</tr>
<tr>
<td>September 4th, 2012</td>
<td>Updates from public comment period</td>
</tr>
<tr>
<td>November 4th, 2012</td>
<td>Publication as GFD-R-P.198</td>
</tr>
<tr>
<td>Juli 15th, 2015</td>
<td>Document revision, see Annex A</td>
</tr>
<tr>
<td>February 12th, 2016</td>
<td>Submission of 2015 revision to OGF Editor</td>
</tr>
<tr>
<td>November 4th, 2016</td>
<td>Publication as GFD-R-P.230</td>
</tr>
</tbody>
</table>

Copyright Notice

Trademark
All company, product or service names referenced in this document are used for identification purposes only and may be trademarks of their respective owners.

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 implementors.
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].

---

1 Corresponding authors

drmaa-wg@ogf.org
# Contents

1 Introduction ................................................................. 4  
2 General Design ............................................................... 4  
  2.1 Error Handling ......................................................... 6  
  2.2 Resource Limits ...................................................... 6  
  2.3 Lists and Dictionaries ................................................ 6  
3 Memory Management .......................................................... 7  
4 Implementation-specific Extensions ....................................... 9  
5 Complete Header File ...................................................... 9  
6 Security Considerations .................................................... 16  
7 Contributors ................................................................. 17  
8 Intellectual Property Statement ........................................... 17  
9 Disclaimer ................................................................. 18  
10 Full Copyright Notice ..................................................... 18  
11 References ............................................................... 18  
A Differences to GFD-R-P.198 ............................................. 19
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 [3] 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 root specification concepts to C follows a set of design principles. Implementation-specific extensions of the DRMAA C API SHOULD follow these conventions:

- Namespacing of the DRMAA API, as demanded 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. All such values are passed as call-by-reference parameter in the C binding.
- Structs and enums are typedef’ed for better readability.
- The root specification defines a consistent parameter passing strategy for non-scalar values. All such values are passed as call-by-reference parameter in the C binding.
- Functions with IDL return type void have drmaa2_error as return type.
- The IDL boolean type maps to the drmaa2 bool 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 existing operating system interfaces.
- The IDL string type is mapped in two different ways. Attributes and parameters with string values typically created by the implementation are mapped to the drmaa2_string type. The application frees such memory by calling the newly introduced function drmaa2_string_free. All other string parameters are mapped to the const char * type. Implementations MUST accept calls to drmaa2_string_free for all string pointers, regardless of their type.
- The language binding defines one UNSET macro per utilized C data type (DRMAA2_UNSET_ ).
- The language binding defines separate UNSET members for each enumeration, to avoid C compiler complains when using a common UNSET value for enumerations. Their values MUST all equal to DRMAA2_UNSET_ENUM, so that both variants can be used.
- All numerical types are signed, in order to support -1 as numerical UNSET value.
- 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 strftime.
• 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 string variable created by the DRMAA library.

• 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 `any` member for job sub-state information is defined as `drmaa2_string` to achieve application portability.

Application-created structs should be allocated by the additional support methods (such as `drmaa2:jinfo_create`) to realize the necessary initialization to `UNSET`. This SHOULD be properly documented by the implementation.

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

- `drmaa2_string`
- `drmaa2_slotinfo`
- `drmaa2_rinfo`
- `drmaa2_notification`
- `drmaa2_queueinfo`
- `drmaa2_version`
- `drmaa2_machineinfo`

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

<table>
<thead>
<tr>
<th>DRMAA interface</th>
<th>C binding prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrmaaReflective</td>
<td><code>drmaa2_</code></td>
</tr>
<tr>
<td>SessionManager</td>
<td><code>drmaa2_</code></td>
</tr>
<tr>
<td>JobSession</td>
<td><code>drmaa2_jsession_</code></td>
</tr>
<tr>
<td>ReservationSession</td>
<td><code>drmaa2_rsession_</code></td>
</tr>
<tr>
<td>MonitoringSession</td>
<td><code>drmaa2_msession_</code></td>
</tr>
<tr>
<td>Reservation</td>
<td><code>drmaa2_r_</code></td>
</tr>
<tr>
<td>Job</td>
<td><code>drmaa2_j_</code></td>
</tr>
<tr>
<td>JobArray</td>
<td><code>drmaa2_jarray_</code></td>
</tr>
<tr>
<td>JobTemplate</td>
<td><code>drmaa2_jtemplate_</code></td>
</tr>
<tr>
<td>ReservationTemplate</td>
<td><code>drmaa2_rtemplate_</code></td>
</tr>
</tbody>
</table>

Table 1: Mapping of DRMAA interface name to C method prefix

The C binding specifies the function pointer type `drmaa2_callback` for a notification callback function. This represents the `DrmaaCallback` interface from the root specification. The new constant value `DRMAA2_UNSET CALLBACK` can be used by the application for the de-registration of callback functions.

`drmaa-wg@ogf.org`
2.1 Error Handling

The list of exceptions in the DRMAA root specification is mapped to the new enumeration `drma2_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: `drma2_lasterror` and `drma2_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 Resource Limits

The DRMAA2 root specification demands the definition of a set of string constants, declared in the header file:

```
extern const char *const DRMAA2_CORE_FILE_SIZE;
extern const char *const DRMAA2_CPU_TIME;
extern const char *const DRMAA2_DATA_SIZE;
extern const char *const DRMAA2_FILE_SIZE;
extern const char *const DRMAA2_OPEN_FILES;
extern const char *const DRMAA2_STACK_SIZE;
extern const char *const DRMAA2_VIRTUAL_MEMORY;
extern const char *const DRMAA2_WALLCLOCK_TIME;
```

The implementation part MUST initialize these variables as follows:

```
const char *const DRMAA2_CORE_FILE_SIZE = "CORE_FILE_SIZE";
const char *const DRMAA2_CPU_TIME = "DRMAA2_CPU_TIME";
const char *const DRMAA2_DATA_SIZE = "DRMAA2_DATA_SIZE";
const char *const DRMAA2_FILE_SIZE = "DRMAA2_FILE_SIZE";
const char *const DRMAA2_OPEN_FILES = "DRMAA2_OPEN_FILES";
const char *const DRMAA2_STACK_SIZE = "DRMAA2_STACK_SIZE";
const char *const DRMAA2_VIRTUAL_MEMORY = "DRMAA2_VIRTUAL_MEMORY";
const char *const DRMAA2_WALLCLOCK_TIME = "DRMAA2_WALLCLOCK_TIME";
```

2.3 Lists and Dictionaries

The C language binding adds generic support functions for the collection data types used by the root specification. The newly defined `drma2_lasterror` and `drma2_lasterror_text` functions MUST return according error information for these operations.

Both `drma2_list_create` and `drma2_dict_create` have an optional parameter `callback`. It allows the application or the implementation to store a callback pointer to an element cleanup function. It MUST be allowed for the application to provide `DRMAA2_UNSET_CALLBACK` instead of a valid callback pointer. The implementation MUST provide a default callback implementation for all list and dictionary types. This can be used by both the application and the implementation itself.

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. If a callback function was provided on list creation, it SHALL be called once per list item.

**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 the provided pointer, not a deep copy of the provided data structure.

**drmaa2_list_del**: Removes the list element at the indicated position and returns a success indication. If a callback function was provided on list creation, it SHALL be called before this function returns.

**drmaa2_list_size**: Returns 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:

**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. If a callback function was provided on dictionary creation, it SHALL be called once per dictionary entry.

**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 a boolean indication if the given key exists in the dictionary. On error, the function SHALL return FALSE.

**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. If a callback function was provided on dictionary creation, it SHALL be called before this function returns.

**drmaa2_dict_set**: Sets the specified dictionary key to the specified value. Key and value strings MUST be stored as the provided character pointers. If the dictionary already has an entry for this name, the value is replaced and the old value is removed. If a callback was provided on dictionary creation, it SHALL be called with a NULL pointer for the key and the pointer of the previous value.

### 3 Memory Management

The majority of data structures returned by an implementation is newly created on the heap. All those structures need to be freed by a call to the according counterpart function (`drmaa2_*_free`) by the application. This should be clearly indicated in the end-user documentation, otherwise memory leaks may occur.

The following functions are expected to return only pointers to existing data, which demands no subsequent freeing of the returned data:

- `drmaa2_dict_get`
• drmaa2_list_get
• drmaa2_jsession_wait_any_started
• drmaa2_jsession_wait_any_terminated

The following functions, when successfully executed, return newly allocated data. Their results must either be free directly, or indirectly be freeing the surrounding wrapper structure.

Implementations MAY register their matching default callback for the returned data structure:

• drmaa2_get_drms_name
• drmaa2_get_drms_version
• drmaa2_describe_attribute
• drmaa2_dict_create
• drmaa2_dict_list
• drmaa2_get_instance_value
• drmaa2.get_jsession_names
• drmaa2.get_rsession_names
• drmaa2_jarray_get_id
• drmaa2_jarray_get_jobs
• drmaa2_jarray_get_jtemplate
• drmaa2_jarray_get_session_name
• drmaa2_j_get_id
• drmaa2_j_get_info
• drmaa2.j_get_get_jt
• drmaa2_j_get_session_name
• drmaa2_jinfo_create
• drmaa2_jinfo_impl_spec
• drmaa2_jsession_get_contact
• drmaa2_jsession_get_jarray
• drmaa2_jsession_get_job_categories
• drmaa2_jsession_get_jobs
• drmaa2_jsession_get_session_name
• drmaa2.jsession_run_bulk_jobs
• drmaa2.jsession_run_job
• drmaa2.jtemplate_create
4 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.

New attributes SHOULD use a product-specific prefix for a clear separation of non-portable and portable parts of the API. The struct definitions in `drmaa2.h` SHALL remain unmodified in all cases. Therefore, these attributes are expected to only be accessible through `drmaa2_get_instance_value` and `drmaa2_set_instance_value`. Implementations can store their specific additional attributes behind the standardized `implementationSpecific` pointer in `drmaa2_jinfo_s`, `drmaa2_rinfo_s`, `drmaa2_slotinfo_s`, `drmaa2_jtemplate_s`, `drmaa2_rtemplate_s`, `drmaa2_notification_s`, `drmaa2_queueinfo_s`, `drmaa2_version_s`, and `drmaa2_machineinfo_s`.

5 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](http://www.drmaa.org).
#ifndef DRMAA2_H
#define DRMAA2_H
#include <time.h>
extern const char *const DRMAA2_CORE_FILE_SIZE;
extern const char *const DRMAA2_CPU_TIME;
extern const char *const DRMAA2_DATA_SIZE;
extern const char *const DRMAA2_FILE_SIZE;
extern const char *const DRMAA2_OPEN_FILES;
extern const char *const DRMAA2_STACK_SIZE;
extern const char *const DRMAA2_VIRTUAL_MEMORY;
extern const char *const DRMAA2_WALLCLOCK_TIME;

typedef enum drmaa2_jstate {
    DRMAA2_UNSET_JSTATE = -1,
    DRMAA2_UNDETERMINED = 0,
    DRMAA2_QUEUED = 1,
    DRMAA2_QUEUED_HELD = 2,
    DRMAA2_RUNNING = 3,
    DRMAA2_SUSPENDED = 4,
    DRMAA2_REQUEUED = 5,
    DRMAA2_REQUEUED_HELD = 6,
    DRMAA2_DONE = 7,
    DRMAA2_FAILED = 8
} drmaa2_jstate;

typedef enum drmaa2_os {
    DRMAA2_UNSET_OS = -1,
    DRMAA2_OTHER_OS = 0,
    DRMAA2_AIX = 1,
    DRMAA2_BSD = 2,
    DRMAA2_LINUX = 3,
    DRMAA2_HPUX = 4,
    DRMAA2_IRIX = 5,
    DRMAA2_MACOS = 6,
    DRMAA2_SUNOS = 7,
    DRMAA2_TRU64 = 8,
    DRMAA2_UNIXWARE = 9,
    DRMAA2_WIN = 10,
    DRMAA2_WINNT = 11
} drmaa2_os;

typedef enum drmaa2_cpu {
    DRMAA2_UNSET_CPU = -1,
    DRMAA2_OTHER_CPU = 0,
    DRMAA2_ALPHA = 1,
    DRMAA2_ARM = 2,
    DRMAA2_ARM64 = 3,
    DRMAA2_CELL = 4,
    DRMAA2_PARISC = 5,
    DRMAA2_PARISC64 = 6,
    DRMAA2_X86 = 7,
    DRMAA2_X64 = 8,
    DRMAA2_IA64 = 9,
    DRMAA2_MIPS = 10,
    DRMAA2_MIPS64 = 11,
    DRMAA2_PPC = 12,
    DRMAA2_PPC64 = 13,
    DRMAA2_SPARC = 14,
    DRMAA2_SPARC64 = 15,
    DRMAA2_PPC64LE = 16
} drmaa2_cpu;

typedef enum drmaa2_event {
    DRMAA2_UNSET_EVENT = -1,
    DRMAA2_NEW_STATE = 0,
    DRMAA2_MIGRATED = 1,
    DRMAA2_ATTRIBUTE_CHANGE = 2
} drmaa2_event;
typedef enum drma2_capability {
    DRMA2_unset_capability = -1,
    DRMA2_advance_reservation = 0,
    DRMA2_reserve_slots = 1,
    DRMA2_callback = 2,
    DRMA2_bulk_jobs_maxparallel = 3,
    DRMA2 JT_EMAIL = 4,
    DRMA2 JT_STAGING = 5,
    DRMA2 JT_DEADLINE = 6,
    DRMA2 JT_MAXSLOTS = 7,
    DRMA2 JT_ACCOUNTINGID = 8,
    DRMA2 RT_STARTNOW = 9,
    DRMA2 RT_DURATION = 10,
    DRMA2 RT_MACHINEOS = 11,
    DRMA2 RT_MACHINEARCH = 12
} drma2_capability;

typedef enum drma2_bool {
    DRMA2_FALSE = 0,
    DRMA2_TRUE = 1
} drma2_bool;

typedef enum drma2_error {
    DRMA2_unset_error = -1,
    DRMA2_SUCCESS = 0,
    DRMA2_DENIED_BY_DRMS = 1,
    DRMA2 DRM_COMMUNICATION = 2,
    DRMA2_TRY_LATER = 3,
    DRMA2_SESSION_MANAGEMENT = 4,
    DRMA2_TIMEOUT = 5,
    DRMA2_INVALID_ARGUMENT = 7,
    DRMA2 INVALID_SESSION = 8,
    DRMA2_INVALID_STATE = 9,
    DRMA2_OUT_OF_RESOURCE = 10,
    DRMA2_UNSUPPORTED_ATTRIBUTE = 11,
    DRMA2_UNSUPPORTED_OPERATION = 12,
    DRMA2_IMPLEMENTATION_SPECIFIC = 13,
    DRMA2_LASTERROR = 14
} drma2_error;

typedef char * drma2_string;
void drma2_string_free (drma2_string *);

void drma2_lasterror (void);

void drma2_lasterror_text (void);

struct drma2_list_s;
/* forward */
typedef struct drma2_list_s * drma2_list;
typedef struct drma2_list_s * drma2_string_list;
typedef struct drma2_list_s * drma2_j_list;
typedef struct drma2_list_s * drma2_queueinfo_list;
typedef struct drma2_list_s * drma2_machineinfo_list;
typedef struct drma2_list_s * drma2_slotinfo_list;
typedef struct drma2_list_s * drma2 r_list;

typedef enum drma2_listtype {
    DRMA2_unset_listtype = -1,
    DRMA2_STRINGLIST = 0,
    DRMA2_JOBLIST = 1,
    DRMA2_QUEUEINFO_LIST = 2,
    DRMA2_MACHINEINFO_LIST = 3,
    DRMA2_SLOTINFO_LIST = 4,
    DRMA2 RESERVATION_LIST = 5
} drma2_listtype;

typedef void (*drma2_list_entryfree)(void **value);
void drma2_string_list_default_callback (void **value);
void drma2_j_list_default_callback (void **value);
void drma2_queueinfo_list_default_callback (void **value);
void drma2_machineinfo_list_default_callback (void **value);
#define DRMAA2_NULL_STRING NULL
#define DRMAA2_NULL_NUM -1
#define DRMAA2_NULL_ENUM -1
#define DRMAA2_NULL_LIST NULL
#define DRMAA2_NULL_DICT NULL
#define DRMAA2_NOW ((time_t) -2)
#define DRMAA2_HOME_DIR "$DRMAA2_HOME_DIR$
#define DRMAA2_INDEX "$DRMAA2_INDEX$
#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
#define DRMAA2_UNSET_JINFO NULL
#define DRMAA2_UNSET_VERSION NULL

typedef struct {
    drmaa2_string jobId;
    drmaa2_string jobName;
    int exitStatus;
    drmaa2_string terminatingSignal;
    drmaa2_string annotation;
    drmaa2_jstate jobState;
    drmaa2_string jobSubState;
    drmaa2_slotinfo_list allocatedMachines;
    drmaa2_string submissionMachine;
    drmaa2_string jobOwner;
    long long slots;
    drmaa2_string queueName;
    time_t wallclockTime;
    long long cpuTime;
    time_t submissionTime;
    time_t dispatchTime;
    time_t finishTime;
    void * implementationSpecific;
} drmaa2_jinfo_s;
typedef drmaa2_jinfo_s * drmaa2_jinfo;

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

typedef struct {
    drmaa2_string machineName;
    long long slots;
    void * implementationSpecific;
}
typedef drmaa2_slotinfo_s * drmaa2_slotinfo;
void drmaa2_slotinfo_free (drmaa2_slotinfo * si);

typedef struct {
    drmaa2_string reservationId;
    drmaa2_string reservationName;
    time_t reservedStartTime;
    time_t reservedEndTime;
    drmaa2_string_list usersACL;
    long long reservedSlots;
    drmaa2_slotinfo_list reservedMachines;
    void * implementationSpecific;
} drmaa2_rinfo_s;

typedef drmaa2_rinfo_s * drmaa2_rinfo;
void drmaa2_rinfo_free (drmaa2_rinfo * ri);

typedef struct {
    drmaa2_string remoteCommand;
    drmaa2_string_list args;
    drmaa2_bool submitAsHold;
    drmaa2_bool rerunnable;
    drmaa2_dict jobEnvironment;
    drmaa2_string workingDirectory;
    drmaa2_string_list email;
    drmaa2_bool emailOnStarted;
    drmaa2_bool emailOnTerminated;
    drmaa2_string jobName;
    drmaa2_string inputPath;
    drmaa2_string outputPath;
    drmaa2_string errorPath;
    drmaa2_bool joinFiles;
    drmaa2_string reservationId;
    drmaa2_string 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;
    drmaa2_string accountingId;
    void * implementationSpecific;
} drmaa2_jtemplate_s;

typedef drmaa2_jtemplate_s * drmaa2_jtemplate;
drmaa2_jtemplate drmaa2_jtemplate_create (void);
void drmaa2_jtemplate_free (drmaa2_jtemplate * jt);

typedef struct {
    drmaa2_string reservationName;
    time_t startTime;
    time_t endTime;
    time_t duration;
    long long minSlots;
    long long maxSlots;
    drmaa2_string jobCategory;
    drmaa2_string_list usersACL;
    drmaa2_string_list candidateMachines;
    long long minPhysMemory;
    drmaa2_os machineACL;
    drmaa2_cpu machineArch;
    void * implementationSpecific;
}
typedef drmaa2_rtemplate_s * drmaa2_rtemplate;

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

typedef struct {
    drmaa2_event event;
    drmaa2_string jobId;
    drmaa2_string sessionName;
    drmaa2_jstate jobState;
    void * implementationSpecific;
} drmaa2_notification_s;

typedef drmaa2_notification_s * drmaa2_notification;

void drmaa2_notification_free ( drmaa2_notification * n);

typedef struct {
    drmaa2_string name;
    void * implementationSpecific;
} drmaa2_queueinfo_s;

typedef drmaa2_queueinfo_s * drmaa2_queueinfo;

void drmaa2_queueinfo_free ( drmaa2_queueinfo * qi);

typedef struct {
    drmaa2_string major;
    drmaa2_string minor;
    void * implementationSpecific;
} drmaa2_version_s;

typedef drmaa2_version_s * drmaa2_version;

void drmaa2_version_free ( drmaa2_version * v);

typedef struct {
    drmaa2_string name;
    drmaa2_bool available;
    long long sockets;
    long long coresPerSocket;
    long long threadsPerCore;
    float load;
    long long physMemory;
    long long virtMemory;
    drmaa2_os machineOS;
    drmaa2_version machineOSVersion;
    drmaa2_cpu machineArch;
    void * implementationSpecific;
} drmaa2_machineinfo_s;

typedef drmaa2_machineinfo_s * drmaa2_machineinfo;

void drmaa2_machineinfo_free ( drmaa2_machineinfo * mi);

drmaa2_string_list drmaa2_jtemplate_impl_spec ( void);

drmaa2_string_list drmaa2_jinfo_impl_spec ( void);

drmaa2_string_list drmaa2_rtemplate_impl_spec ( void);

drmaa2_string_list drmaa2_rinfo_impl_spec ( void);

drmaa2_string_list drmaa2_queueinfo_impl_spec ( void);

drmaa2_string_list drmaa2_machineinfo_impl_spec ( void);

drmaa2_string_list drmaa2_get_instance_value ( const void * instance, const char * name);

drmaa2_string drmaa2_describe_attribute ( const void * instance, const char * name);

drmaa2_error drmaa2_set_instance_value ( void * instance, const char * name, const char * value);

typedef void (* drmaa2_callback)(drmaa2_notification * notification);

struct drmaa2_jsession_s; /* forward*/
struct drmaa2_rsession_s; /* forward*/
struct drmaa2_msession_s; /* forward*/
struct drmaa2_j_s; /* forward*/
struct drmaa2_jarray_s; /* forward*/
struct drmaa2_r_s;  /* forward */

typedef struct drmaa2_jsession_s * drmaa2_jsession;
typedef struct drmaa2_rsession_s * drmaa2_rsession;
typedef struct drmaa2_msession_s * drmaa2_msession;
typedef struct drmaa2_j_s * drmaa2_j;
typedef struct drmaa2_jarray_s * drmaa2_jarray;
typedef struct drmaa2_r_s * drmaa2_r;

void drmaa2_jsession_free (drmaa2_jsession * js);
void drmaa2_rsession_free (drmaa2_rsession * rs);
void drmaa2_msession_free (drmaa2_msession * ms);
void drmaa2_j_free (drmaa2_j * j);
void drmaa2_jarray_free (drmaa2_jarray * ja);
void drmaa2_r_free (drmaa2_r * r);

drmaa2_string drmaa2_rsession_get_contact (const drmaa2_rsession rs);
drmaa2_string drmaa2_rsession_get_session_name (const drmaa2_rsession rs);
drmaa2_r drmaa2_rsession_get_reservation (const drmaa2_rsession rs, const drmaa2_string reservationId);
drmaa2_r_list drmaa2_rsession_get_reservations (const drmaa2_rsession rs);
drmaa2_string drmaa2_r_get_id (const drmaa2_r r);
drmaa2_string drmaa2_r_get_session_name (const drmaa2_r r);
drmaa2_rtemplate drmaa2_r_get_reservation_template (const drmaa2_r r);
drmaa2_rinfo drmaa2_r_get_info (const drmaa2_r r);
drmaa2_error drmaa2_r_terminate (drmaa2_r r);
drmaa2_string drmaa2_jarray_get_id (const drmaa2_jarray ja);
drmaa2_j_list drmaa2_jarray_get_jobs (const drmaa2_jarray ja);
drmaa2_string drmaa2_jarray_get_session_name (const drmaa2_jarray ja);
drmaa2_jtemplate drmaa2_jarray_get_jtemplate (const drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_suspend (drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_resume (drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_hold (drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_release (drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_terminate (drmaa2_jarray ja);
drmaa2_error drmaa2_jarray_reap (drmaa2_jarray ja);

drmaa2_string drmaa2_jsession_get_contact (const drmaa2_jsession js);
drmaa2_string drmaa2_jsession_get_session_name (const drmaa2_jsession js);
drmaa2_j_list drmaa2_jsession_get_job_categories (const drmaa2_jsession js);
drmaa2_jtemplate drmaa2_jsession_get_jtemplate (const drmaa2_jsession js, const drmaa2_jinfo filter);
drmaa2_jarray drmaa2_jsession_get_job_array (const drmaa2_jsession js, const drmaa2_string jobarrayId);
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, const long long begin_index, const long long end_index, const long long step, const long 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);

drmaa2_string drmaa2_j_get_id (const drmaa2_j j);
drmaa2_string drmaa2_j_get_session_name (const drmaa2_j j);
drmaa2_jtemplate drmaa2_j_get_jtemplate (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_error drmaa2_j_reap (drmaa2_j j);
drmaa2_jstate drmaa2_j_get_state (const drmaa2_j j, drmaa2_string * substate);

drmaa-wg@ogf.org 15
6 Security Considerations

The DRMAA root specification [3] 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 these scenarios.
7 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)
TU Chemnitz
Reichenhainer Straße 70
09126 Chemnitz, Germany
Email: peter@troeger.eu

Special thanks go to Stefan Klauck (Hasso Plattner Institute) for the DRMAA C binding reference implementation and the debugging of the implementation-related language binding issues.

8 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.

9 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.

10 Full Copyright Notice


This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included as references to the derived portions on all such copies and derivative works. The published OGF document from which such works are derived, however, may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing new or updated OGF documents in conformance with the procedures defined in the OGF Document Process, or as required to translate it into languages other than English. OGF, with the approval of its board, may remove this restriction for inclusion of OGF document content for the purpose of producing standards in cooperation with other international standards bodies.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

11 References


A Differences to GFD-R-P.198

The following changes were applied in the July 2015 revision of this document:

Issue #115, #104:
Section 3 was introduced to describe the rules of memory management from the application point of view.

Issue #59:
The numeric parameters of `drmaa2_jsession_run_bulk_jobs` are now `const`. The `pos` parameter of `drmaa2_list_del` is now `const`. The `pos` parameter of `drmaa2_list_get` is now `const`. This modification is backward-compatible.

Issue #57:
`drmaa2_limit` needed to be removed, since (numeric) enumeration members cannot become keys in dictionaries. Instead, a set of according constants was introduced, as explained in the newly introduced Section 2.2. The text also explains how these constants must be initialized by the implementation code. This modification is not backward-compatible, since the original version was not implementable.

Issue #116:
`drmaa2_dict_default_callback` and a set of default list callbacks were introduced. This modification is backward-compatible.

Issue #160, #162:
All `struct` definitions got an additional `implementationSpecific` member. It is intended to store the implementation-specific attributes, as now explained in Section 4. This modification is not backward-compatible, since the `drmaa2.h` header file changes. All relevant implementations are already updated.

Issue #255:
All enumerations got an additional `UNSET` value. The motivation is now explained in 1. `drmaa2_listtype` was completed with numeric defaults, similar to the other enumerations. This modification is backward-compatible for the known implementations and their compilers.

Issue #113:
`drmaa2_get_all_machines()` returns a list of `drmaa2_machineinfo_s` instances, which contains a pointer to a `drmaa2_version_s` struct. This demanded the addition of `DRMA2_UNSET_VERSION`. This modification is backward-compatible.

Issue #165:
The `allocatedMachines` attribute in `drmaa2_jinfo_s` has now the type `drmaa2_slotinfo_list`, in accordance to the root specification.

Issue #114:
For consistency reasons, the following renaming took place:

- `drmaa2_jarray_get_job_template` becomes `drmaa2_jarray_get_jtemplate`
- `drmaa2_j_get_jt` becomes `drmaa2_j_get_jtemplate`
This modification is not backward-compatible, since the `drma2.h` header file changes. All relevant implementations are already updated.

Issue #63:
All numerical types are signed, in order to support -1 as numerical UNSET value. For this reason, the numerical parameters of `drmaa2_jsession_run_bulk_jobs` were changed to `long long`. This modification is not backward-compatible, since the `drma2.h` header file changes. All relevant implementations are already updated.

Issue #163:
The two new functions `drmaa2_j_reap` and `drmaa2_jarray_reap` were introduced, in accordance to the July 2015 errata of the root specification.

Issue #102:
The July 2015 root specification errata adds `jobName` to the `JobInfo` structure, which is now also reflected in `drmaa2_jinfo_s`. This modification is not backward-compatible, since the `drma2.h` header file changes. All relevant implementations are already updated.

Issue #287:
A new element for the PowerPC 64bit little-endian architecture (PPC64LE) was added to the `drmaa2_cpu` enumeration. This modification is not backward-compatible, since the `drma2.h` header file changes. All relevant implementations are already updated.