

RDC

Generated by Doxygen 1.8.5

Mon Dec 14 2020 11:02:10



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	rdc_device_attributes_t Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.2	rdc_field_group_info_t Struct Reference . . . . .	5
3.2.1	Detailed Description . . . . .	5
3.2.2	Field Documentation . . . . .	6
3.2.2.1	field_ids . . . . .	6
3.3	rdc_field_value Struct Reference . . . . .	6
3.3.1	Detailed Description . . . . .	6
3.3.2	Field Documentation . . . . .	6
3.3.2.1	value . . . . .	6
3.4	rdc_gpu_usage_info_t Struct Reference . . . . .	6
3.4.1	Detailed Description . . . . .	7
3.5	rdc_group_info_t Struct Reference . . . . .	7
3.5.1	Detailed Description . . . . .	8
3.5.2	Field Documentation . . . . .	8
3.5.2.1	entity_ids . . . . .	8
3.6	rdc_job_group_info_t Struct Reference . . . . .	8
3.6.1	Detailed Description . . . . .	8
3.7	rdc_job_info_t Struct Reference . . . . .	8
3.7.1	Detailed Description . . . . .	9
3.7.2	Field Documentation . . . . .	9
3.7.2.1	summary . . . . .	9
3.8	rdc_stats_summary_t Struct Reference . . . . .	9
3.8.1	Detailed Description . . . . .	9

<b>4</b>	<b>File Documentation</b>	<b>11</b>
4.1	rdc.h File Reference	11
4.1.1	Detailed Description	14
4.1.2	Typedef Documentation	14
4.1.2.1	rdc_handle_t	14
4.1.3	Enumeration Type Documentation	14
4.1.3.1	rdc_status_t	14
4.1.3.2	rdc_group_type_t	15
4.1.3.3	rdc_field_t	15
4.1.4	Function Documentation	16
4.1.4.1	rdc_init	16
4.1.4.2	rdc_shutdown	16
4.1.4.3	rdc_start_embedded	16
4.1.4.4	rdc_stop_embedded	17
4.1.4.5	rdc_connect	17
4.1.4.6	rdc_disconnect	17
4.1.4.7	rdc_job_start_stats	18
4.1.4.8	rdc_job_get_stats	18
4.1.4.9	rdc_job_stop_stats	18
4.1.4.10	rdc_job_remove	19
4.1.4.11	rdc_job_remove_all	19
4.1.4.12	rdc_field_update_all	19
4.1.4.13	rdc_device_get_all	19
4.1.4.14	rdc_device_get_attributes	20
4.1.4.15	rdc_group_gpu_create	20
4.1.4.16	rdc_group_gpu_add	20
4.1.4.17	rdc_group_gpu_get_info	21
4.1.4.18	rdc_group_get_all_ids	21
4.1.4.19	rdc_group_gpu_destroy	21
4.1.4.20	rdc_group_field_create	22
4.1.4.21	rdc_group_field_get_info	22
4.1.4.22	rdc_group_field_get_all_ids	22
4.1.4.23	rdc_group_field_destroy	23
4.1.4.24	rdc_field_watch	23
4.1.4.25	rdc_field_get_latest_value	23
4.1.4.26	rdc_field_get_value_since	24
4.1.4.27	rdc_field_unwatch	24
4.1.4.28	rdc_status_string	24
4.1.4.29	field_id_string	25
4.1.4.30	get_field_id_from_name	25





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">rdc_device_attributes_t</a>	Represents attributes corresponding to a device . . . . .	5
<a href="#">rdc_field_group_info_t</a>	The structure to store the field group info . . . . .	5
<a href="#">rdc_field_value</a>	The structure to store the field value . . . . .	6
<a href="#">rdc_gpu_usage_info_t</a>	The structure to hold the GPU usage information . . . . .	6
<a href="#">rdc_group_info_t</a>	The structure to store the group info . . . . .	7
<a href="#">rdc_job_group_info_t</a>	The structure to store the job info . . . . .	8
<a href="#">rdc_job_info_t</a>	The structure to hold the job stats . . . . .	8
<a href="#">rdc_stats_summary_t</a>	The structure to store summary of data . . . . .	9





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

[rdc.h](#)

The rocm\_rdc library api is new, and therefore subject to change either at the ABI or API level. Instead of marking every function prototype as "unstable", we are instead saying the API is unstable (i.e., changes are possible) while the major version remains 0. This means that if the API/ABI changes, we will not increment the major version to 1. Once the ABI stabilizes, we will increment the major version to 1, and thereafter increment it on all ABI breaks . . . . .

11



## Chapter 3

# Data Structure Documentation

### 3.1 rdc\_device\_attributes\_t Struct Reference

Represents attributes corresponding to a device.

```
#include <rdc.h>
```

#### Data Fields

- char [device\\_name](#) [RDC\_MAX\_STR\_LENGTH]  
*Name of the device.*

#### 3.1.1 Detailed Description

Represents attributes corresponding to a device.

The documentation for this struct was generated from the following file:

- [rdc.h](#)

### 3.2 rdc\_field\_group\_info\_t Struct Reference

The structure to store the field group info.

```
#include <rdc.h>
```

#### Data Fields

- uint32\_t [count](#)  
*count of fields in the group*
- char [group\\_name](#) [RDC\_MAX\_STR\_LENGTH]  
*field group name*
- [rdc\\_field\\_t field\\_ids](#) [RDC\_MAX\_FIELD\_IDS\_PER\_FIELD\_GROUP]

#### 3.2.1 Detailed Description

The structure to store the field group info.

### 3.2.2 Field Documentation

#### 3.2.2.1 `rdc_field_t rdc_field_group_info_t::field_ids[RDC_MAX_FIELD_IDS_PER_FIELD_GROUP]`

The list of fields in the group

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.3 `rdc_field_value` Struct Reference

The structure to store the field value.

```
#include <rdc.h>
```

### Data Fields

- [rdc\\_field\\_t field\\_id](#)  
*The field id of the value.*
- `int status`  
*RDC\_ST\_OK or error status.*
- `uint64_t ts`  
*Timestamp in usec since 1970.*
- [rdc\\_field\\_type\\_t type](#)  
*The field type.*
- `union {`  
    `int64_t i_int`  
    `double dbl`  
    `char str [RDC_MAX_STR_LENGTH]`  
} [value](#)

### 3.3.1 Detailed Description

The structure to store the field value.

### 3.3.2 Field Documentation

#### 3.3.2.1 `union { ... } rdc_field_value::value`

Value of the field. Value type depends on the field type.

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.4 `rdc_gpu_usage_info_t` Struct Reference

The structure to hold the GPU usage information.

```
#include <rdc.h>
```

## Data Fields

- uint32\_t [gpu\\_id](#)  
*GPU\_ID\_INVALID for summary information.*
- uint64\_t [start\\_time](#)  
*The time to start the watching.*
- uint64\_t [end\\_time](#)  
*The time to stop the watching.*
- uint64\_t [energy\\_consumed](#)  
*GPU Energy consumed.*
- uint64\_t [ecc\\_correct](#)  
*Correctable errors.*
- uint64\_t [ecc\\_uncorrect](#)  
*Uncorrectable errors.*
- [rdc\\_stats\\_summary\\_t](#) [pcie\\_tx](#)  
*Bytes sent over PCIe stats.*
- [rdc\\_stats\\_summary\\_t](#) [pcie\\_rx](#)  
*Bytes received over PCIe stats.*
- [rdc\\_stats\\_summary\\_t](#) [power\\_usage](#)  
*GPU Power usage stats.*
- [rdc\\_stats\\_summary\\_t](#) [gpu\\_clock](#)  
*GPU Clock speed stats.*
- [rdc\\_stats\\_summary\\_t](#) [memory\\_clock](#)  
*Mem. Clock speed stats.*
- [rdc\\_stats\\_summary\\_t](#) [gpu\\_utilization](#)  
*GPU Utilization stats.*
- [rdc\\_stats\\_summary\\_t](#) [gpu\\_temperature](#)  
*GPU temperature stats.*
- uint64\_t [max\\_gpu\\_memory\\_used](#)  
*Maximum GPU memory used.*
- [rdc\\_stats\\_summary\\_t](#) [memory\\_utilization](#)  
*Memory Utilization statistics.*

### 3.4.1 Detailed Description

The structure to hold the GPU usage information.

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.5 rdc\_group\_info\_t Struct Reference

The structure to store the group info.

```
#include <rdc.h>
```

## Data Fields

- unsigned int [count](#)  
*count of GPUs in the group*
- char [group\\_name](#) [[RDC\\_MAX\\_STR\\_LENGTH](#)]  
*group name*
- uint32\_t [entity\\_ids](#) [[RDC\\_GROUP\\_MAX\\_ENTITIES](#)]

### 3.5.1 Detailed Description

The structure to store the group info.

### 3.5.2 Field Documentation

#### 3.5.2.1 uint32\_t rdc\_group\_info\_t::entity\_ids[RDC\_GROUP\_MAX\_ENTITIES]

The list of entities in the group

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.6 rdc\_job\_group\_info\_t Struct Reference

The structure to store the job info.

```
#include <rdc.h>
```

## Data Fields

- char [job\\_id](#) [[RDC\\_MAX\\_STR\\_LENGTH](#)]  
*job id*
- [rdc\\_gpu\\_group\\_t](#) [group\\_id](#)  
*group name*
- uint64\_t [start\\_time](#)  
*job start time*
- uint64\_t [stop\\_time](#)  
*job stop time*

### 3.6.1 Detailed Description

The structure to store the job info.

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.7 rdc\_job\_info\_t Struct Reference

The structure to hold the job stats.

```
#include <rdc.h>
```

## Data Fields

- uint32\_t [num\\_gpus](#)  
*Number of GPUs used by job.*
- [rdc\\_gpu\\_usage\\_info\\_t](#) [summary](#)
- [rdc\\_gpu\\_usage\\_info\\_t](#) [gpus](#) [16]  
*Job usage summary statistics by GPU.*

### 3.7.1 Detailed Description

The structure to hold the job stats.

### 3.7.2 Field Documentation

#### 3.7.2.1 [rdc\\_gpu\\_usage\\_info\\_t](#) [rdc\\_job\\_info\\_t::summary](#)

Job usage summary statistics (overall)

The documentation for this struct was generated from the following file:

- [rdc.h](#)

## 3.8 rdc\_stats\_summary\_t Struct Reference

The structure to store summary of data.

```
#include <rdc.h>
```

## Data Fields

- uint64\_t [max\\_value](#)  
*Maximum value measured.*
- uint64\_t [min\\_value](#)  
*Minimum value measured.*
- uint64\_t [average](#)  
*Average value measured.*
- double [standard\\_deviation](#)  
*The standard deviation.*

### 3.8.1 Detailed Description

The structure to store summary of data.

The documentation for this struct was generated from the following file:

- [rdc.h](#)





## Chapter 4

# File Documentation

### 4.1 rdc.h File Reference

The rocm\_rdc library api is new, and therefore subject to change either at the ABI or API level. Instead of marking every function prototype as "unstable", we are instead saying the API is unstable (i.e., changes are possible) while the major version remains 0. This means that if the API/ABI changes, we will not increment the major version to 1. Once the ABI stabilizes, we will increment the major version to 1, and thereafter increment it on all ABI breaks.

```
#include <stdint.h>
```

#### Data Structures

- struct [rdc\\_device\\_attributes\\_t](#)  
*Represents attributes corresponding to a device.*
- struct [rdc\\_group\\_info\\_t](#)  
*The structure to store the group info.*
- struct [rdc\\_stats\\_summary\\_t](#)  
*The structure to store summary of data.*
- struct [rdc\\_gpu\\_usage\\_info\\_t](#)  
*The structure to hold the GPU usage information.*
- struct [rdc\\_job\\_info\\_t](#)  
*The structure to hold the job stats.*
- struct [rdc\\_field\\_value](#)  
*The structure to store the field value.*
- struct [rdc\\_field\\_group\\_info\\_t](#)  
*The structure to store the field group info.*
- struct [rdc\\_job\\_group\\_info\\_t](#)  
*The structure to store the job info.*

#### Macros

- #define [GPU\\_ID\\_INVALID](#) -1  
*ID used to represent an invalid GPU.*
- #define [RDC\\_GROUP\\_ALL\\_GPUS](#) -1000  
*Used to specify all GPUs.*
- #define [RDC\\_JOB\\_STATS\\_FIELDS](#) -1000  
*Used to specify all stats fields.*

- `#define RDC_MAX_STR_LENGTH 256`  
*The max rdc field string length.*
- `#define RDC_GROUP_MAX_ENTITIES 64`  
*The max entities in a group.*
- `#define RDC_MAX_NUM_DEVICES 16`  
*Max number of GPUs supported by RDC.*
- `#define RDC_MAX_FIELD_IDS_PER_FIELD_GROUP 128`  
*The max fields in a field group.*
- `#define RDC_MAX_NUM_GROUPS 64`  
*The max number of groups.*
- `#define RDC_MAX_NUM_FIELD_GROUPS 64`  
*The max number of the field groups.*

## Typedefs

- `typedef void * rdc_handle_t`  
*handlers used in various rdc calls*
- `typedef uint32_t rdc_gpu_group_t`  
*GPU Group ID type.*
- `typedef uint32_t rdc_field_grp_t`  
*Field group ID type.*

## Enumerations

- `enum rdc_status_t {`  
`RDC_ST_OK = 0, RDC_ST_NOT_SUPPORTED, RDC_ST_MSI_ERROR, RDC_ST_FAIL_LOAD_MODULE,`  
`RDC_ST_INVALID_HANDLER, RDC_ST_BAD_PARAMETER, RDC_ST_NOT_FOUND, RDC_ST_CONFLICT,`  
`RDC_ST_CLIENT_ERROR, RDC_ST_ALREADY_EXIST, RDC_ST_MAX_LIMIT, RDC_ST_INSUFF_RESOURCES,`  
`RDC_ST_FILE_ERROR, RDC_ST_NO_DATA, RDC_ST_PERM_ERROR, RDC_ST_UNKNOWN_ERROR`  
`= 0xFFFFFFFF }`  
*Error codes returned by rocm\_rdc\_lib functions.*
- `enum rdc_operation_mode_t { RDC_OPERATION_MODE_AUTO = 0, RDC_OPERATION_MODE_MANUAL }`  
*rdc operation mode rdc can run in auto mode where background threads will collect metrics. When run in manual mode, the user needs to periodically call rdc\_field\_update\_all for data collection.*
- `enum rdc_group_type_t { RDC_GROUP_DEFAULT = 0, RDC_GROUP_EMPTY }`  
*type of GPU group*
- `enum rdc_field_type_t { INTEGER = 0, DOUBLE, STRING, BLOB }`  
*the type stored in the filed value*
- `enum rdc_field_t {`  
`RDC_FI_INVALID = 0, RDC_FI_GPU_COUNT = 1, RDC_FI_DEV_NAME, RDC_FI_GPU_CLOCK = 100,`  
`RDC_FI_MEM_CLOCK, RDC_FI_MEMORY_TEMP = 200, RDC_FI_GPU_TEMP, RDC_FI_POWER_USAGE = 300,`  
`RDC_FI_PCIE_TX = 400, RDC_FI_PCIE_RX, RDC_FI_GPU_UTIL = 500, RDC_FI_GPU_MEMORY_USAGE,`  
`RDC_FI_GPU_MEMORY_TOTAL, RDC_FI_ECC_CORRECT_TOTAL = 600, RDC_FI_ECC_UNCORRECT_TOTAL,`  
`RDC_EVNT_XGMI_0_NOP_TX = 1000,`  
`RDC_EVNT_XGMI_0_REQ_TX, RDC_EVNT_XGMI_0_RESP_TX, RDC_EVNT_XGMI_0_BEATS_TX, RDC_EVNT_XGMI_1_NOP_TX,`  
`RDC_EVNT_XGMI_1_REQ_TX, RDC_EVNT_XGMI_1_RESP_TX, RDC_EVNT_XGMI_1_BEATS_TX, RDC_EVNT_XGMI_0_THRPUT = 1500,`  
`RDC_EVNT_XGMI_1_THRPUT }`

## Functions

- [rdc\\_status\\_t rdc\\_init](#) (uint64\_t init\_flags)  
*Initialize ROCm RDC.*
- [rdc\\_status\\_t rdc\\_shutdown](#) ()  
*Shutdown ROCm RDC.*
- [rdc\\_status\\_t rdc\\_start\\_embedded](#) (rdc\_operation\_mode\_t op\_mode, rdc\_handle\_t \*p\_rdc\_handle)  
*Start embedded RDC agent within this process.*
- [rdc\\_status\\_t rdc\\_stop\\_embedded](#) (rdc\_handle\_t p\_rdc\_handle)  
*Stop embedded RDC agent.*
- [rdc\\_status\\_t rdc\\_connect](#) (const char \*ipAndPort, rdc\_handle\_t \*p\_rdc\_handle, const char \*root\_ca, const char \*client\_cert, const char \*client\_key)  
*Connect to rdcd daemon.*
- [rdc\\_status\\_t rdc\\_disconnect](#) (rdc\_handle\_t p\_rdc\_handle)  
*Disconnect from rdcd daemon.*
- [rdc\\_status\\_t rdc\\_job\\_start\\_stats](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_gpu\_group\_t group\_id, const char job\_id[64], uint64\_t update\_freq)  
*Request the RDC to watch the job stats.*
- [rdc\\_status\\_t rdc\\_job\\_get\\_stats](#) (rdc\_handle\_t p\_rdc\_handle, const char job\_id[64], rdc\_job\_info\_t \*p\_job\_info)  
*Get the stats of the job using the job id.*
- [rdc\\_status\\_t rdc\\_job\\_stop\\_stats](#) (rdc\_handle\_t p\_rdc\_handle, const char job\_id[64])  
*Request RDC to stop watching the stats of the job.*
- [rdc\\_status\\_t rdc\\_job\\_remove](#) (rdc\_handle\_t p\_rdc\_handle, const char job\_id[64])  
*Request RDC to stop tracking the job given by job\_id.*
- [rdc\\_status\\_t rdc\\_job\\_remove\\_all](#) (rdc\_handle\_t p\_rdc\_handle)  
*Request RDC to stop tracking all the jobs.*
- [rdc\\_status\\_t rdc\\_field\\_update\\_all](#) (rdc\_handle\_t p\_rdc\_handle, uint32\_t wait\_for\_update)  
*Request RDC to update all fields to be watched.*
- [rdc\\_status\\_t rdc\\_device\\_get\\_all](#) (rdc\_handle\_t p\_rdc\_handle, uint32\_t gpu\_index\_list[RDC\_MAX\_NUM\_DEVICES], uint32\_t \*count)  
*Get indexes corresponding to all the devices on the system.*
- [rdc\\_status\\_t rdc\\_device\\_get\\_attributes](#) (rdc\_handle\_t p\_rdc\_handle, uint32\_t gpu\_index, rdc\_device\_attributes\_t \*p\_rdc\_attr)  
*Gets device attributes corresponding to the gpu\_index.*
- [rdc\\_status\\_t rdc\\_group\\_gpu\\_create](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_group\_type\_t type, const char \*group\_name, rdc\_gpu\_group\_t \*p\_rdc\_group\_id)  
*Create a group contains multiple GPUs.*
- [rdc\\_status\\_t rdc\\_group\\_gpu\\_add](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_gpu\_group\_t group\_id, uint32\_t gpu\_index)  
*Add a GPU to the group.*
- [rdc\\_status\\_t rdc\\_group\\_gpu\\_get\\_info](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_gpu\_group\_t p\_rdc\_group\_id, rdc\_group\_info\_t \*p\_rdc\_group\_info)  
*Get information about a GPU group.*
- [rdc\\_status\\_t rdc\\_group\\_get\\_all\\_ids](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_gpu\_group\_t group\_id\_list[], uint32\_t \*count)  
*Used to get information about all GPU groups in the system.*
- [rdc\\_status\\_t rdc\\_group\\_gpu\\_destroy](#) (rdc\_handle\_t p\_rdc\_handle, rdc\_gpu\_group\_t p\_rdc\_group\_id)  
*Destroy GPU group represented by p\_rdc\_group\_id.*
- [rdc\\_status\\_t rdc\\_group\\_field\\_create](#) (rdc\_handle\_t p\_rdc\_handle, uint32\_t num\_field\_ids, rdc\_field\_t \*field\_ids, const char \*field\_group\_name, rdc\_field\_grp\_t \*rdc\_field\_group\_id)  
*create a group of fields*

- `rdc_status_t rdc_group_field_get_info` (`rdc_handle_t` p\_rdc\_handle, `rdc_field_grp_t` rdc\_field\_group\_id, `rdc_field_group_info_t` \*field\_group\_info)  
*Get information about a field group.*
- `rdc_status_t rdc_group_field_get_all_ids` (`rdc_handle_t` p\_rdc\_handle, `rdc_field_grp_t` field\_group\_id\_list[], `uint32_t` \*count)  
*Used to get information about all field groups in the system.*
- `rdc_status_t rdc_group_field_destroy` (`rdc_handle_t` p\_rdc\_handle, `rdc_field_grp_t` rdc\_field\_group\_id)  
*Destroy field group represented by rdc\_field\_group\_id.*
- `rdc_status_t rdc_field_watch` (`rdc_handle_t` p\_rdc\_handle, `rdc_gpu_group_t` group\_id, `rdc_field_grp_t` field\_group\_id, `uint64_t` update\_freq, double max\_keep\_age, `uint32_t` max\_keep\_samples)  
*Request the RDC start recording updates for a given field collection.*
- `rdc_status_t rdc_field_get_latest_value` (`rdc_handle_t` p\_rdc\_handle, `uint32_t` gpu\_index, `rdc_field_t` field, `rdc_field_value` \*value)  
*Request a latest cached field of a GPU.*
- `rdc_status_t rdc_field_get_value_since` (`rdc_handle_t` p\_rdc\_handle, `uint32_t` gpu\_index, `rdc_field_t` field, `uint64_t` since\_time\_stamp, `uint64_t` \*next\_since\_time\_stamp, `rdc_field_value` \*value)  
*Request a history cached field of a GPU.*
- `rdc_status_t rdc_field_unwatch` (`rdc_handle_t` p\_rdc\_handle, `rdc_gpu_group_t` group\_id, `rdc_field_grp_t` field\_group\_id)  
*Stop record updates for a given field collection.*
- `const char * rdc_status_string` (`rdc_status_t` status)  
*Get a description of a provided RDC error status.*
- `const char * field_id_string` (`rdc_field_t` field\_id)  
*Get the name of a field.*
- `rdc_field_t get_field_id_from_name` (`const char` \*name)  
*Get the field id from name.*

#### 4.1.1 Detailed Description

The rocm\_rdc library api is new, and therefore subject to change either at the ABI or API level. Instead of marking every function prototype as "unstable", we are instead saying the API is unstable (i.e., changes are possible) while the major version remains 0. This means that if the API/ABI changes, we will not increment the major version to 1. Once the ABI stabilizes, we will increment the major version to 1, and thereafter increment it on all ABI breaks. Main header file for the ROCm RDC library. All required function, structure, enum, etc. definitions should be defined in this file.

#### 4.1.2 Typedef Documentation

##### 4.1.2.1 typedef void\* rdc\_handle\_t

handlers used in various rdc calls

Handle used for an RDC session

#### 4.1.3 Enumeration Type Documentation

##### 4.1.3.1 enum rdc\_status\_t

Error codes returned by rocm\_rdc\_lib functions.

Enumerator

**RDC\_ST\_OK** Success.

**RDC\_ST\_NOT\_SUPPORTED** Not supported feature.

**RDC\_ST\_MSI\_ERROR** The MSI library error.

**RDC\_ST\_FAIL\_LOAD\_MODULE** Fail to load the library.

**RDC\_ST\_INVALID\_HANDLER** Invalid handler.

**RDC\_ST\_BAD\_PARAMETER** A parameter is invalid.

**RDC\_ST\_NOT\_FOUND** Cannot find the value.

**RDC\_ST\_CONFLICT** Conflict with current state.

**RDC\_ST\_CLIENT\_ERROR** The RDC client error.

**RDC\_ST\_ALREADY\_EXIST** The item already exists.

**RDC\_ST\_MAX\_LIMIT** Max limit recording for the object.

**RDC\_ST\_INSUFF\_RESOURCES** Not enough resources to complete operation

**RDC\_ST\_FILE\_ERROR** Failed to access a file.

**RDC\_ST\_NO\_DATA** Data was requested, but none was found

**RDC\_ST\_PERM\_ERROR** Insufficient permission to complete operation

**RDC\_ST\_UNKNOWN\_ERROR** Unknown error.

#### 4.1.3.2 enum rdc\_group\_type\_t

type of GPU group

Enumerator

**RDC\_GROUP\_DEFAULT** All GPUs on the Node.

**RDC\_GROUP\_EMPTY** Empty group.

#### 4.1.3.3 enum rdc\_field\_t

These enums are used to specify a particular field to be retrieved.

Enumerator

**RDC\_FI\_INVALID** Identifier fields. Invalid field value

**RDC\_FI\_GPU\_COUNT** GPU count in the system.

**RDC\_FI\_DEV\_NAME** Name of the device.

**RDC\_FI\_GPU\_CLOCK** The current clock for the GPU.

**RDC\_FI\_MEM\_CLOCK** Clock for the memory.

**RDC\_FI\_MEMORY\_TEMP** Memory temperature for the device.

**RDC\_FI\_GPU\_TEMP** Current temperature for the device.

**RDC\_FI\_POWER\_USAGE** Power usage for the device.

**RDC\_FI\_PCIE\_TX** PCIe Tx utilization information.

**RDC\_FI\_PCIE\_RX** PCIe Rx utilization information.

**RDC\_FI\_GPU\_UTIL** GPU Utilization.

**RDC\_FI\_GPU\_MEMORY\_USAGE** Memory usage of the GPU instance.

**RDC\_FI\_GPU\_MEMORY\_TOTAL** Total memory of the GPU instance.

**RDC\_FI\_ECC\_CORRECT\_TOTAL** ECC related fields. Accumulated correctable ECC errors

**RDC\_FI\_ECC\_UNCORRECT\_TOTAL** Accumulated uncorrectable ECC errors.

**RDC\_EVT\_XGMI\_0\_NOP\_TX** NOPs sent to neighbor 0.

**RDC\_EVNT\_XGMI\_0\_REQ\_TX** Outgoing requests to neighbor 0

**RDC\_EVNT\_XGMI\_0\_RESP\_TX** Outgoing responses to neighbor 0

**RDC\_EVNT\_XGMI\_0\_BEATS\_TX** Data beats sent to neighbor 0; Each beat represents 32 bytes.

XGMI throughput can be calculated by multiplying a BEATS event such as `::RSMI_EVNT_XGMI_0_BEATS_TX` by 32 and dividing by the time for which event collection occurred, `::rsmi_counter_value_t.time_running` (which is in nanoseconds). To get bytes per second, multiply this value by  $10^9$ .

Throughput = BEATS/time\_running \*  $10^9$  (bytes/second)

**RDC\_EVNT\_XGMI\_1\_NOP\_TX** NOPs sent to neighbor 1.

**RDC\_EVNT\_XGMI\_1\_REQ\_TX** Outgoing requests to neighbor 1

**RDC\_EVNT\_XGMI\_1\_RESP\_TX** Outgoing responses to neighbor 1

**RDC\_EVNT\_XGMI\_1\_BEATS\_TX** Data beats sent to neighbor 1; Each beat represents 32 bytes

**RDC\_EVNT\_XGMI\_0\_THRPUT** Transmit throughput to XGMI neighbor 0 in bytes/sec

**RDC\_EVNT\_XGMI\_1\_THRPUT** Transmit throughput to XGMI neighbor 1 in bytes/sec

#### 4.1.4 Function Documentation

##### 4.1.4.1 `rdc_status_t rdc_init ( uint64_t init_flags )`

Initialize ROCm RDC.

When called, this initializes internal data structures, including those corresponding to sources of information that RDC provides. This must be called before [rdc\\_start\\_embedded\(\)](#) or [rdc\\_connect\(\)](#)

###### Parameters

<code>in</code>	<code>init_flags</code>	init_flags Bit flags that tell RDC how to initialize.
-----------------	-------------------------	---

###### Return values

<a href="#">RDC_ST_OK</a>	is returned upon successful call.
---------------------------	-----------------------------------

##### 4.1.4.2 `rdc_status_t rdc_shutdown ( )`

Shutdown ROCm RDC.

Do any necessary clean up.

##### 4.1.4.3 `rdc_status_t rdc_start_embedded ( rdc_operation_mode_t op_mode, rdc_handle_t * p_rdc_handle )`

Start embedded RDC agent within this process.

The RDC is loaded as library so that it does not require `rdcd` daemon. In this mode, the user has to periodically call [rdc\\_field\\_update\\_all\(\)](#) when `op_mode` is `RDC_OPERATION_MODE_MANUAL`, which tells RDC to collect the stats.

###### Parameters

<code>in</code>	<code>op_mode</code>	Operation modes. When <code>RDC_OPERATION_MODE_AUTO</code> , RDC schedules background task to collect the stats. When <code>RDC_OPERATION_MODE_MANUAL</code> , the user needs to call <a href="#">rdc_field_update_all()</a> periodically.
-----------------	----------------------	--

in, out	<i>p_rdc_handle</i>	Caller provided pointer to rdc_handle_t. Upon successful call, the value will contain the handler for following API calls.
---------	---------------------	--

## Return values

<a href="#">RDC_ST_OK</a>	is returned upon successful call.
---------------------------	-----------------------------------

## 4.1.4.4 rdc\_status\_t rdc\_stop\_embedded ( rdc\_handle\_t p\_rdc\_handle )

Stop embedded RDC agent.

Stop the embedded RDC agent, and p\_rdc\_handle becomes invalid after this call.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler that come from <a href="#">rdc_start_embedded()</a> .
----	---------------------	---

## Return values

<a href="#">RDC_ST_OK</a>	is returned upon successful call.
---------------------------	-----------------------------------

## 4.1.4.5 rdc\_status\_t rdc\_connect ( const char \* ipAndPort, rdc\_handle\_t \* p\_rdc\_handle, const char \* root\_ca, const char \* client\_cert, const char \* client\_key )

Connect to rdc daemon.

This method is used to connect to a remote stand-alone rdc daemon.

## Parameters

in	<i>ipAndPort</i>	The IP and port of the remote rdc. The ipAndPort can be specified in this x.x.x.x:yyyy format, where x.x.x.x is the IP address and yyyy is the port.
in, out	<i>p_rdc_handle</i>	Caller provided pointer to rdc_handle_t. Upon successful call, the value will contain the handler for following API calls.
in	<i>root_ca</i>	The root CA stored in the string in pem format. Set it as nullptr if the communication is not encrypted.
in	<i>client_cert</i>	The client certificate stored in the string in pem format. Set it as nullptr if the communication is not encrypted.
in	<i>client_key</i>	The client key stored in the string in pem format. Set it as nullptr if the communication is not encrypted.

## Return values

<a href="#">RDC_ST_OK</a>	is returned upon successful call.
---------------------------	-----------------------------------

## 4.1.4.6 rdc\_status\_t rdc\_disconnect ( rdc\_handle\_t p\_rdc\_handle )

Disconnect from rdc daemon.

Disconnect from rdc daemon, and p\_rdc\_handle becomes invalid after this call.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler that come from <a href="#">rdc_connect()</a> .
----	---------------------	--

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.7** `rdc_status_t rdc_job_start_stats ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t group_id, const char job_id[64], uint64_t update_freq )`

Request the RDC to watch the job stats.

This should be executed as part of job prologue. The summary job stats can be retrieved using [`rdc\_job\_get\_stats\(\)`](#). In RDC\_OPERATION\_MODE\_MANUAL, user must call `rdc_field_update_all(1)` at least once, before call [`rdc\_job\_get\_stats\(\)`](#)

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>group_id</i>	The group of GPUs to be watched.
in	<i>job_id</i>	The name of the job.
in	<i>update_freq</i>	How often to update this field in usec.

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.8** `rdc_status_t rdc_job_get_stats ( rdc_handle_t p_rdc_handle, const char job_id[64], rdc_job_info_t * p_job_info )`

Get the stats of the job using the job id.

The stats can be retrieved at any point when the job is in process.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>job_id</i>	The name of the job.
in,out	<i>p_job_info</i>	Caller provided pointer to <a href="#"><code>rdc_job_info_t</code></a> . Upon successful call, the value will contain the stats of the job.

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.9** `rdc_status_t rdc_job_stop_stats ( rdc_handle_t p_rdc_handle, const char job_id[64] )`

Request RDC to stop watching the stats of the job.

This should be execute as part of job epilogue. The job Id remains available to view the stats at any point. You must call `rdc_watch_job_fields()` before this call.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>job_id</i>	The name of the job.

## Return values



<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.10 `rdc_status_t rdc_job_remove ( rdc_handle_t p_rdc_handle, const char job_id[64] )`

Request RDC to stop tracking the job given by job\_id.

After this call, you will no longer be able to call [`rdc\_job\_get\_stats\(\)`](#) on this job\_id. But you will be able to reuse the job\_id after this call.

##### Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>job_id</i>	The name of the job.

##### Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.11 `rdc_status_t rdc_job_remove_all ( rdc_handle_t p_rdc_handle )`

Request RDC to stop tracking all the jobs.

After this call, you will no longer be able to call [`rdc\_job\_get\_stats\(\)`](#) on any job id. But you will be able to reuse the any previous used job id after this call.

##### Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
----	---------------------	------------------

##### Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.12 `rdc_status_t rdc_field_update_all ( rdc_handle_t p_rdc_handle, uint32_t wait_for_update )`

Request RDC to update all fields to be watched.

In RDC\_OPERATION\_MODE\_MANUAL, the user must call this method periodically.

##### Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>wait_for_update</i>	Whether or not to wait for the update loop to complete before returning to the caller 1=wait. 0=do not wait.

##### Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.13 `rdc_status_t rdc_device_get_all ( rdc_handle_t p_rdc_handle, uint32_t gpu_index_list[RDC_MAX_NUM_DEVICES], uint32_t* count )`

Get indexes corresponding to all the devices on the system.

Indexes represents RDC GPU Id corresponding to each GPU on the system and is immutable during the lifespan of the engine. The list should be queried again if the engine is restarted.

**Parameters**

in	<i>p_rdc_handle</i>	The RDC handler.
out	<i>gpu_index_list</i>	Array reference to fill GPU indexes present on the system.
out	<i>count</i>	Number of GPUs returned in <i>gpu_index_list</i> .

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.14 `rdc_status_t rdc_device_get_attributes ( rdc_handle_t p_rdc_handle, uint32_t gpu_index, rdc_device_attributes_t * p_rdc_attr )`

Gets device attributes corresponding to the *gpu\_index*.

Fetch the attributes, such as device name, of a GPU.

**Parameters**

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>gpu_index</i>	GPU index corresponding to which the attributes should be fetched
out	<i>p_rdc_attr</i>	GPU attribute corresponding to the <i>gpu_index</i> .

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.15 `rdc_status_t rdc_group_gpu_create ( rdc_handle_t p_rdc_handle, rdc_group_type_t type, const char * group_name, rdc_gpu_group_t * p_rdc_group_id )`

Create a group contains multiple GPUs.

This method can create a group contains multiple GPUs. Instead of executing an operation separately for each GPU, the RDC group enables the user to execute same operation on all the GPUs present in the group as a single API call.

**Parameters**

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>type</i>	The type of the group. <code>RDC_GROUP_DEFAULT</code> includes all the GPUs on the node, and <code>RDC_GROUP_EMPTY</code> creates an empty group.
in	<i>group_name</i>	The group name specified as NULL terminated C String
in, out	<i>p_rdc_group_id</i>	Caller provided pointer to <code>rdc_gpu_group_t</code> . Upon successful call, the value will contain the group id for following group API calls.

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

#### 4.1.4.16 `rdc_status_t rdc_group_gpu_add ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t group_id, uint32_t gpu_index )`

Add a GPU to the group.

This method can add a GPU to the group

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>group_id</i>	The group id to which the GPU will be added.
in	<i>gpu_index</i>	The GPU index to be added to the group.

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.17** `rdc_status_t rdc_group_gpu_get_info ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t p_rdc_group_id, rdc_group_info_t * p_rdc_group_info )`

Get information about a GPU group.

Get detail information about a GPU group created by `rdc_group_gpu_create`

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>p_rdc_group_id</i>	The GPU group handler created by <code>rdc_group_gpu_create</code>
out	<i>p_rdc_group_info</i>	The information of the GPU group <code>p_rdc_group_id</code> .

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.18** `rdc_status_t rdc_group_get_all_ids ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t group_id_list[], uint32_t * count )`

Used to get information about all GPU groups in the system.

Get the list of GPU group ids in the system.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
out	<i>group_id_list</i>	Array reference to fill GPU group ids in the system.
out	<i>count</i>	Number of GPU group returned in <code>group_id_list</code> .

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

**4.1.4.19** `rdc_status_t rdc_group_gpu_destroy ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t p_rdc_group_id )`

Destroy GPU group represented by `p_rdc_group_id`.

Delete the logic group represented by `p_rdc_group_id`

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>p_rdc_group_id</i>	The group id

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.20 **rdc\_status\_t** rdc\_group\_field\_create ( **rdc\_handle\_t** *p\_rdc\_handle*, **uint32\_t** *num\_field\_ids*, **rdc\_field\_t** \* *field\_ids*, **const char** \* *field\_group\_name*, **rdc\_field\_grp\_t** \* *rdc\_field\_group\_id* )

create a group of fields

The user can create a group of fields and perform an operation on a group of fields at once.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>num_field_ids</i>	Number of field IDs that are being provided in <i>field_ids</i> .
in	<i>field_ids</i>	Field IDs to be added to the newly-created field group.
in	<i>field_group_name</i>	Unique name for this group of fields.
out	<i>rdc_field_group_id</i>	Handle to the newly-created field group

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.21 **rdc\_status\_t** rdc\_group\_field\_get\_info ( **rdc\_handle\_t** *p\_rdc\_handle*, **rdc\_field\_grp\_t** *rdc\_field\_group\_id*, **rdc\_field\_group\_info\_t** \* *field\_group\_info* )

Get information about a field group.

Get detail information about a field group created by `rdc_group_field_create`

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>rdc_field_group_id</i>	The field group handler created by <code>rdc_group_field_create</code>
out	<i>field_group_info</i>	The information of the field group <code>rdc_field_group_id</code> .

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.22 **rdc\_status\_t** rdc\_group\_field\_get\_all\_ids ( **rdc\_handle\_t** *p\_rdc\_handle*, **rdc\_field\_grp\_t** *field\_group\_id\_list*[], **uint32\_t** \* *count* )

Used to get information about all field groups in the system.

Get the list of field group ids in the system.

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
out	<i>field_group_id_list</i>	Array reference to fill field group ids in the system.

out	count	Number of field group returned in field_group_id_list.
-----	-------	--

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.23 `rdc_status_t rdc_group_field_destroy ( rdc_handle_t p_rdc_handle, rdc_field_grp_t rdc_field_group_id )`

Destroy field group represented by rdc\_field\_group\_id.

Delete the logic group represented by rdc\_field\_group\_id

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>rdc_field_group_id</i>	The field group id

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.24 `rdc_status_t rdc_field_watch ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t group_id, rdc_field_grp_t field_group_id, uint64_t update_freq, double max_keep_age, uint32_t max_keep_samples )`

Request the RDC start recording updates for a given field collection.

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, user must call rdc\_field\_update\_all(1)

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>group_id</i>	The group of GPUs to be watched.
in	<i>field_group_id</i>	The collection of fields to record
in	<i>update_freq</i>	How often to update fields in usec.
in	<i>max_keep_age</i>	How long to keep data for fields in seconds.
in	<i>max_keep_samples</i>	Maximum number of samples to keep. 0=no limit.

## Return values

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.25 `rdc_status_t rdc_field_get_latest_value ( rdc_handle_t p_rdc_handle, uint32_t gpu_index, rdc_field_t field, rdc_field_value * value )`

Request a latest cached field of a GPU.

Note that the field can be cached after called rdc\_field\_watch

## Parameters

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>gpu_index</i>	The GPU index.

in	<i>field</i>	The field id
out	<i>value</i>	The field value got from cache.

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.26 `rdc_status_t rdc_field_get_value_since ( rdc_handle_t p_rdc_handle, uint32_t gpu_index, rdc_field_t field, uint64_t since_time_stamp, uint64_t * next_since_time_stamp, rdc_field_value * value )`

Request a history cached field of a GPU.

Note that the field can be cached after called `rdc_field_watch`

**Parameters**

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>gpu_index</i>	The GPU index.
in	<i>field</i>	The field id
in	<i>since_time_ - stamp</i>	Timestamp to request values since in usec since 1970.
out	<i>next_since_time - stamp</i>	Timestamp to use for sinceTimestamp on next call to this function
out	<i>value</i>	The field value got from cache.

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.27 `rdc_status_t rdc_field_unwatch ( rdc_handle_t p_rdc_handle, rdc_gpu_group_t group_id, rdc_field_grp_t field_group_id )`

Stop record updates for a given field collection.

The cache of those fields will not be updated after this call

**Parameters**

in	<i>p_rdc_handle</i>	The RDC handler.
in	<i>group_id</i>	The GPU group id.
in	<i>field_group_id</i>	The field group id.

**Return values**

<a href="#"><i>RDC_ST_OK</i></a>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.4.28 `const char* rdc_status_string ( rdc_status_t status )`

Get a description of a provided RDC error status.

return the string in human readable format.

**Parameters**

in	<i>status</i>	The RDC status.
----	---------------	-----------------

## Return values

	<i>The</i>	string to describe the RDC status.
--	------------	------------------------------------

4.1.4.29 `const char* field_id_string ( rdc_field_t field_id )`

Get the name of a field.

return the string in human readable format.

## Parameters

in	<i>field_id</i>	The field id.
----	-----------------	---------------

## Return values

	<i>The</i>	string to describe the field.
--	------------	-------------------------------

4.1.4.30 `rdc_field_t get_field_id_from_name ( const char * name )`

Get the field id from name.

return the field id from field name.

## Parameters

in	<i>name</i>	The field name.
----	-------------	-----------------

## Return values

	<i>return</i>	RDC_FI_INVALID if the field name is invalid.
--	---------------	--

# Index

entity\_ids  
    rdc\_group\_info\_t, 8

field\_id\_string  
    rdc.h, 25

field\_ids  
    rdc\_field\_group\_info\_t, 6

get\_field\_id\_from\_name  
    rdc.h, 25

RDC\_EVNT\_XGMI\_0\_BEATS\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_0\_NOP\_TX  
    rdc.h, 15

RDC\_EVNT\_XGMI\_0\_REQ\_TX  
    rdc.h, 15

RDC\_EVNT\_XGMI\_0\_RESP\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_0\_THRPUT  
    rdc.h, 16

RDC\_EVNT\_XGMI\_1\_BEATS\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_1\_NOP\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_1\_REQ\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_1\_RESP\_TX  
    rdc.h, 16

RDC\_EVNT\_XGMI\_1\_THRPUT  
    rdc.h, 16

RDC\_FI\_DEV\_NAME  
    rdc.h, 15

RDC\_FI\_ECC\_CORRECT\_TOTAL  
    rdc.h, 15

RDC\_FI\_ECC\_UNCORRECT\_TOTAL  
    rdc.h, 15

RDC\_FI\_GPU\_CLOCK  
    rdc.h, 15

RDC\_FI\_GPU\_COUNT  
    rdc.h, 15

RDC\_FI\_GPU\_MEMORY\_TOTAL  
    rdc.h, 15

RDC\_FI\_GPU\_MEMORY\_USAGE  
    rdc.h, 15

RDC\_FI\_GPU\_TEMP  
    rdc.h, 15

RDC\_FI\_GPU\_UTIL  
    rdc.h, 15

RDC\_FI\_INVALID

    rdc.h, 15

RDC\_FI\_MEM\_CLOCK  
    rdc.h, 15

RDC\_FI\_MEMORY\_TEMP  
    rdc.h, 15

RDC\_FI\_PCIE\_RX  
    rdc.h, 15

RDC\_FI\_PCIE\_TX  
    rdc.h, 15

RDC\_FI\_POWER\_USAGE  
    rdc.h, 15

RDC\_GROUP\_DEFAULT  
    rdc.h, 15

RDC\_GROUP\_EMPTY  
    rdc.h, 15

RDC\_ST\_ALREADY\_EXIST  
    rdc.h, 15

RDC\_ST\_BAD\_PARAMETER  
    rdc.h, 15

RDC\_ST\_CLIENT\_ERROR  
    rdc.h, 15

RDC\_ST\_CONFLICT  
    rdc.h, 15

RDC\_ST\_FAIL\_LOAD\_MODULE  
    rdc.h, 15

RDC\_ST\_FILE\_ERROR  
    rdc.h, 15

RDC\_ST\_INSUFF\_RESOURCES  
    rdc.h, 15

RDC\_ST\_INVALID\_HANDLER  
    rdc.h, 15

RDC\_ST\_MAX\_LIMIT  
    rdc.h, 15

RDC\_ST\_MSI\_ERROR  
    rdc.h, 15

RDC\_ST\_NO\_DATA  
    rdc.h, 15

RDC\_ST\_NOT\_FOUND  
    rdc.h, 15

RDC\_ST\_NOT\_SUPPORTED  
    rdc.h, 14

RDC\_ST\_OK  
    rdc.h, 14

RDC\_ST\_PERM\_ERROR  
    rdc.h, 15

RDC\_ST\_UNKNOWN\_ERROR  
    rdc.h, 15

rdc.h

    RDC\_EVNT\_XGMI\_0\_BEATS\_TX, 16



[RDC\\_EVT\\_XGMI\\_0\\_NOP\\_TX](#), [15](#)  
[RDC\\_EVT\\_XGMI\\_0\\_REQ\\_TX](#), [15](#)  
[RDC\\_EVT\\_XGMI\\_0\\_RESP\\_TX](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_0\\_THRPUT](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_1\\_BEATS\\_TX](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_1\\_NOP\\_TX](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_1\\_REQ\\_TX](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_1\\_RESP\\_TX](#), [16](#)  
[RDC\\_EVT\\_XGMI\\_1\\_THRPUT](#), [16](#)  
[RDC\\_FI\\_DEV\\_NAME](#), [15](#)  
[RDC\\_FI\\_ECC\\_CORRECT\\_TOTAL](#), [15](#)  
[RDC\\_FI\\_ECC\\_UNCORRECT\\_TOTAL](#), [15](#)  
[RDC\\_FI\\_GPU\\_CLOCK](#), [15](#)  
[RDC\\_FI\\_GPU\\_COUNT](#), [15](#)  
[RDC\\_FI\\_GPU\\_MEMORY\\_TOTAL](#), [15](#)  
[RDC\\_FI\\_GPU\\_MEMORY\\_USAGE](#), [15](#)  
[RDC\\_FI\\_GPU\\_TEMP](#), [15](#)  
[RDC\\_FI\\_GPU\\_UTIL](#), [15](#)  
[RDC\\_FI\\_INVALID](#), [15](#)  
[RDC\\_FI\\_MEM\\_CLOCK](#), [15](#)  
[RDC\\_FI\\_MEMORY\\_TEMP](#), [15](#)  
[RDC\\_FI\\_PCIE\\_RX](#), [15](#)  
[RDC\\_FI\\_PCIE\\_TX](#), [15](#)  
[RDC\\_FI\\_POWER\\_USAGE](#), [15](#)  
[RDC\\_GROUP\\_DEFAULT](#), [15](#)  
[RDC\\_GROUP\\_EMPTY](#), [15](#)  
[RDC\\_ST\\_ALREADY\\_EXIST](#), [15](#)  
[RDC\\_ST\\_BAD\\_PARAMETER](#), [15](#)  
[RDC\\_ST\\_CLIENT\\_ERROR](#), [15](#)  
[RDC\\_ST\\_CONFLICT](#), [15](#)  
[RDC\\_ST\\_FAIL\\_LOAD\\_MODULE](#), [15](#)  
[RDC\\_ST\\_FILE\\_ERROR](#), [15](#)  
[RDC\\_ST\\_INSUFF\\_RESOURCES](#), [15](#)  
[RDC\\_ST\\_INVALID\\_HANDLER](#), [15](#)  
[RDC\\_ST\\_MAX\\_LIMIT](#), [15](#)  
[RDC\\_ST\\_MSI\\_ERROR](#), [15](#)  
[RDC\\_ST\\_NO\\_DATA](#), [15](#)  
[RDC\\_ST\\_NOT\\_FOUND](#), [15](#)  
[RDC\\_ST\\_NOT\\_SUPPORTED](#), [14](#)  
[RDC\\_ST\\_OK](#), [14](#)  
[RDC\\_ST\\_PERM\\_ERROR](#), [15](#)  
[RDC\\_ST\\_UNKNOWN\\_ERROR](#), [15](#)  
[rdc.h](#), [11](#)  
[field\\_id\\_string](#), [25](#)  
[get\\_field\\_id\\_from\\_name](#), [25](#)  
[rdc\\_connect](#), [17](#)  
[rdc\\_device\\_get\\_all](#), [19](#)  
[rdc\\_device\\_get\\_attributes](#), [20](#)  
[rdc\\_disconnect](#), [17](#)  
[rdc\\_field\\_get\\_latest\\_value](#), [23](#)  
[rdc\\_field\\_get\\_value\\_since](#), [24](#)  
[rdc\\_field\\_t](#), [15](#)  
[rdc\\_field\\_unwatch](#), [24](#)  
[rdc\\_field\\_update\\_all](#), [19](#)  
[rdc\\_field\\_watch](#), [23](#)  
[rdc\\_group\\_field\\_create](#), [22](#)  
[rdc\\_group\\_field\\_destroy](#), [23](#)  
[rdc\\_group\\_field\\_get\\_all\\_ids](#), [22](#)  
[rdc\\_group\\_field\\_get\\_info](#), [22](#)  
[rdc\\_group\\_get\\_all\\_ids](#), [21](#)  
[rdc\\_group\\_gpu\\_add](#), [20](#)  
[rdc\\_group\\_gpu\\_create](#), [20](#)  
[rdc\\_group\\_field\\_get\\_info](#), [22](#)  
[rdc\\_group\\_get\\_all\\_ids](#), [21](#)  
[rdc\\_group\\_gpu\\_add](#), [20](#)  
[rdc\\_group\\_gpu\\_create](#), [20](#)  
[rdc\\_group\\_gpu\\_destroy](#), [21](#)  
[rdc\\_group\\_gpu\\_get\\_info](#), [21](#)  
[rdc\\_group\\_type\\_t](#), [15](#)  
[rdc\\_handle\\_t](#), [14](#)  
[rdc\\_init](#), [16](#)  
[rdc\\_job\\_get\\_stats](#), [18](#)  
[rdc\\_job\\_remove](#), [19](#)  
[rdc\\_job\\_remove\\_all](#), [19](#)  
[rdc\\_job\\_start\\_stats](#), [18](#)  
[rdc\\_job\\_stop\\_stats](#), [18](#)  
[rdc\\_shutdown](#), [16](#)  
[rdc\\_start\\_embedded](#), [16](#)  
[rdc\\_status\\_string](#), [24](#)  
[rdc\\_status\\_t](#), [14](#)  
[rdc\\_stop\\_embedded](#), [17](#)  
[rdc\\_connect](#)  
[rdc.h](#), [17](#)  
[rdc\\_device\\_attributes\\_t](#), [5](#)  
[rdc\\_device\\_get\\_all](#)  
[rdc.h](#), [19](#)  
[rdc\\_device\\_get\\_attributes](#)  
[rdc.h](#), [20](#)  
[rdc\\_disconnect](#)  
[rdc.h](#), [17](#)  
[rdc\\_field\\_get\\_latest\\_value](#)  
[rdc.h](#), [23](#)  
[rdc\\_field\\_get\\_value\\_since](#)  
[rdc.h](#), [24](#)  
[rdc\\_field\\_group\\_info\\_t](#), [5](#)  
[field\\_ids](#), [6](#)  
[rdc\\_field\\_t](#)  
[rdc.h](#), [15](#)  
[rdc\\_field\\_unwatch](#)  
[rdc.h](#), [24](#)  
[rdc\\_field\\_update\\_all](#)  
[rdc.h](#), [19](#)  
[rdc\\_field\\_value](#), [6](#)  
[value](#), [6](#)  
[rdc\\_field\\_watch](#)  
[rdc.h](#), [23](#)  
[rdc\\_gpu\\_usage\\_info\\_t](#), [6](#)  
[rdc\\_group\\_field\\_create](#)  
[rdc.h](#), [22](#)  
[rdc\\_group\\_field\\_destroy](#)  
[rdc.h](#), [23](#)  
[rdc\\_group\\_field\\_get\\_all\\_ids](#)  
[rdc.h](#), [22](#)  
[rdc\\_group\\_field\\_get\\_info](#)  
[rdc.h](#), [22](#)  
[rdc\\_group\\_get\\_all\\_ids](#)  
[rdc.h](#), [21](#)  
[rdc\\_group\\_gpu\\_add](#)  
[rdc.h](#), [20](#)  
[rdc\\_group\\_gpu\\_create](#)

- rdc.h, [20](#)
- rdc\_group\_gpu\_destroy
  - rdc.h, [21](#)
- rdc\_group\_gpu\_get\_info
  - rdc.h, [21](#)
- rdc\_group\_info\_t, [7](#)
  - entity\_ids, [8](#)
- rdc\_group\_type\_t
  - rdc.h, [15](#)
- rdc\_handle\_t
  - rdc.h, [14](#)
- rdc\_init
  - rdc.h, [16](#)
- rdc\_job\_get\_stats
  - rdc.h, [18](#)
- rdc\_job\_group\_info\_t, [8](#)
- rdc\_job\_info\_t, [8](#)
  - summary, [9](#)
- rdc\_job\_remove
  - rdc.h, [19](#)
- rdc\_job\_remove\_all
  - rdc.h, [19](#)
- rdc\_job\_start\_stats
  - rdc.h, [18](#)
- rdc\_job\_stop\_stats
  - rdc.h, [18](#)
- rdc\_shutdown
  - rdc.h, [16](#)
- rdc\_start\_embedded
  - rdc.h, [16](#)
- rdc\_stats\_summary\_t, [9](#)
- rdc\_status\_string
  - rdc.h, [24](#)
- rdc\_status\_t
  - rdc.h, [14](#)
- rdc\_stop\_embedded
  - rdc.h, [17](#)
- summary
  - rdc\_job\_info\_t, [9](#)
- value
  - rdc\_field\_value, [6](#)