RDC

Generated by Doxygen 1.8.18

1 Data Structure Index 1

1.1 Data Structures . . . 1

2 File Index 3

2.1 File List . . . 3

3 Data Structure Documentation 5

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 . . . 6

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 . . . 7

3.4 rdc_field_value_data Union Reference . . . 7

3.4.1 Detailed Description . . . 7

3.5 rdc_gpu_usage_info_t Struct Reference . . . 7

3.5.1 Detailed Description . . . 8

3.6 rdc_group_info_t Struct Reference . . . 8

3.6.1 Detailed Description . . . 9

3.6.2 Field Documentation . . . 9

3.6.2.1 entity_ids . . . 9

3.7 rdc_job_group_info_t Struct Reference . . . 9

3.7.1 Detailed Description . . . 10

3.8 rdc_job_info_t Struct Reference . . . 10

3.8.1 Detailed Description . . . 10

3.8.2 Field Documentation . . . 10

3.8.2.1 summary . . . 10

3.9 rdc_stats_summary_t Struct Reference . . . 10

3.9.1 Detailed Description . . . 11

4 File Documentation 13

4.1 rdc.h File Reference . . . 13

4.1.1 Detailed Description . . . 17

4.1.2 Typedef Documentation . . . 17

4.1.2.1 rdc_handle_t . . . 17

4.1.3 Enumeration Type Documentation . . . 17

4.1.3.1 rdc_status_t . . . 17

4.1.3.2 rdc_group_type_t . . . 18

4.1.3.3 rdc_field_t . . . 18

4.1.4 Function Documentation 20
4.1.4.1 rdc_init() 20
4.1.4.2 rdc_shutdown() 20
4.1.4.3 rdc_start_embedded() 20
4.1.4.4 rdc_stop_embedded() 21
4.1.4.5 rdc_connect() 21
4.1.4.6 rdc_disconnect() 22
4.1.4.7 rdc_job_start_stats() 22
4.1.4.8 rdc_job_get_stats() 23
4.1.4.9 rdc_job_stop_stats() 23
4.1.4.10 rdc_job_remove() 24
4.1.4.11 rdc_job_remove_all() 24
4.1.4.12 rdc_field_update_all() 25
4.1.4.13 rdc_device_get_all() 25
4.1.4.14 rdc_device_get_attributes() 26
4.1.4.15 rdc_group_gpu_create() 26
4.1.4.16 rdc_group_gpu_add() 27
4.1.4.17 rdc_group_gpu_get_info() 27
4.1.4.18 rdc_group_get_all_ids() 28
4.1.4.19 rdc_group_gpu_destroy() 28
4.1.4.20 rdc_group_field_create() 29
4.1.4.21 rdc_group_field_get_info() 29
4.1.4.22 rdc_group_field_get_all_ids() 30
4.1.4.23 rdc_group_field_destroy() 30
4.1.4.24 rdc_field_watch() 31
4.1.4.25 rdc_field_get_latest_value() 31
4.1.4.26 rdc_field_get_value_since() 32
4.1.4.27 rdc_field_unwatch() 32
4.1.4.28 rdc_status_string() 33
4.1.4.29 field_id_string() 33
4.1.4.30 get_field_id_from_name() 33

Index **35**

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

rdc_device_attributes_t
: Represents attributes corresponding to a device . . . 5

rdc_field_group_info_t
: The structure to store the field group info . . . 5

rdc_field_value
: The structure to store the field value . . . 6

rdc_field_value_data
: Field value data . . . 7

rdc_gpu_usage_info_t
: The structure to hold the GPU usage information . . . 7

rdc_group_info_t
: The structure to store the group info . . . 8

rdc_job_group_info_t
: The structure to store the job info . . . 9

rdc_job_info_t
: The structure to hold the job stats . . . 10

rdc_stats_summary_t
: The structure to store summary of data . . . 10

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 13

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 field_ids

```
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.
- rdc_field_value_data value

3.3.1 Detailed Description

The structure to store the field value.

3.3.2 Field Documentation

3.3.2.1 value

`rdc_field_value_data 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_field_value_data Union Reference

Field value data.

`#include <rdc.h>`

Data Fields

- int64_t **l_int**
- double **dbl**
- char **str** [RDC_MAX_STR_LENGTH]

3.4.1 Detailed Description

Field value data.

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

- rdc.h

3.5 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

  Uncorrtable 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.

3.5.1 Detailed Description

The structure to hold the GPU usage information.

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

- rdc.h

3.6 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]

3.6.1 Detailed Description

The structure to store the group info.

3.6.2 Field Documentation

3.6.2.1 entity_ids

```
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

3.7 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

3.7.1 Detailed Description

The structure to store the job info.

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

- rdc.h

3.8 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 staticstics by GPU.

3.8.1 Detailed Description

The structure to hold the job stats.

3.8.2 Field Documentation

3.8.2.1 summary

`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

3.9 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.

3.9.1 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.
- union rdc_field_value_data

  Field value data.
- 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.
- #define **RDC_EVNT_IS_NOTIF_FIELD**(FIELD) ((FIELD) >= RDC_EVNT_NOTIF_FIRST && (FIELD) <= RDC_EVNT_NOTIF_LAST)

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_FI_ECC_SDMA_SEC,
  RDC_FI_ECC_SDMA_DED, RDC_FI_ECC_GFX_SEC, RDC_FI_ECC_GFX_DED, RDC_FI_ECC_MMHUB_SEC,
  RDC_FI_ECC_MMHUB_DED, RDC_FI_ECC_ATHUB_SEC, RDC_FI_ECC_ATHUB_DED, RDC_FI_ECC_BIF_SEC,
  RDC_FI_ECC_BIF_DED, RDC_FI_ECC_HDP_SEC, RDC_FI_ECC_HDP_DED, RDC_FI_ECC_XGMI_WAFL_SEC,
  RDC_FI_ECC_XGMI_WAFL_DED, RDC_FI_ECC_DF_SEC, RDC_FI_ECC_DF_DED, RDC_FI_ECC_SMN_SEC,
  RDC_FI_ECC_SMN_DED, RDC_FI_ECC_SEM_SEC, RDC_FI_ECC_SEM_DED, RDC_FI_ECC_MP0_SEC,
  RDC_FI_ECC_MP0_DED, RDC_FI_ECC_MP1_SEC, RDC_FI_ECC_MP1_DED, RDC_FI_ECC_FUSE_SEC,
  RDC_FI_ECC_FUSE_DED, RDC_FI_ECC_UMC_SEC, RDC_FI_ECC_UMC_DED, 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, RDC_EVNT_XGMI_2_THRPUT, RDC_EVNT_XGMI_3_THRPUT,
  RDC_EVNT_XGMI_4_THRPUT,
  RDC_EVNT_XGMI_5_THRPUT, RDC_EVNT_NOTIF_VMFAULT = 2000, **RDC_EVNT_NOTIF_FIRST** =
  RDC_EVNT_NOTIF_VMFAULT, RDC_EVNT_NOTIF_THERMAL_THROTTLE,
  RDC_EVNT_NOTIF_PRE_RESET, RDC_EVNT_NOTIF_POST_RESET, **RDC_EVNT_NOTIF_LAST** = R↩
  DC_EVNT_NOTIF_POST_RESET }

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 rdc_handle_t

```
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 rdc_status_t

```
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 rdc_group_type_t

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 rdc_field_t

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_FI_ECC_SDMA_SEC	SDMA Single Error Correction.
RDC_FI_ECC_SDMA_DED	SDMA Double Error Detection.
RDC_FI_ECC_GFX_SEC	GFX Single Error Correction.
RDC_FI_ECC_GFX_DED	GFX Double Error Detection.
RDC_FI_ECC_MMHUB_SEC	MMHUB Single Error Correction.
RDC_FI_ECC_MMHUB_DED	MMHUB Double Error Detection.
RDC_FI_ECC_ATHUB_SEC	ATHUB Single Error Correction.
RDC_FI_ECC_ATHUB_DED	ATHUB Double Error Detection.
RDC_FI_ECC_BIF_SEC	BIF Single Error Correction.
RDC_FI_ECC_BIF_DED	BIF Double Error Detection.
RDC_FI_ECC_HDP_SEC	HDP Single Error Correction.

Enumerator

RDC_FI_ECC_HDP_DED	HDP Double Error Detection.
RDC_FI_ECC_XGMI_WAFL_SEC	XGMI WAFL Single Error Correction.
RDC_FI_ECC_XGMI_WAFL_DED	XGMI WAFL Double Error Detection.
RDC_FI_ECC_DF_SEC	DF Single Error Correction.
RDC_FI_ECC_DF_DED	DF Double Error Detection.
RDC_FI_ECC_SMN_SEC	SMN Single Error Correction.
RDC_FI_ECC_SMN_DED	SMN Double Error Detection.
RDC_FI_ECC_SEM_SEC	SEM Single Error Correction.
RDC_FI_ECC_SEM_DED	SEM Double Error Detection.
RDC_FI_ECC_MP0_SEC	MP0 Single Error Correction.
RDC_FI_ECC_MP0_DED	MP0 Double Error Detection.
RDC_FI_ECC_MP1_SEC	MP1 Single Error Correction.
RDC_FI_ECC_MP1_DED	MP1 Double Error Detection.
RDC_FI_ECC_FUSE_SEC	FUSE Single Error Correction.
RDC_FI_ECC_FUSE_DED	FUSE Double Error Detection.
RDC_FI_ECC_UMC_SEC	UMC Single Error Correction.
RDC_FI_ECC_UMC_DED	UMC Double Error Detection.
RDC_EVNT_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 represnts 32 bytes
RDC_EVNT_XGMI_0_THRPUT	Transmit throughput to XGMI neighbor 0 in byes/sec
RDC_EVNT_XGMI_1_THRPUT	Transmit throughput to XGMI neighbor 1 in byes/sec
RDC_EVNT_XGMI_2_THRPUT	Transmit throughput to XGMI neighbor 2 in byes/sec
RDC_EVNT_XGMI_3_THRPUT	Transmit throughput to XGMI neighbor 3 in byes/sec
RDC_EVNT_XGMI_4_THRPUT	Transmit throughput to XGMI neighbor 4 in byes/sec
RDC_EVNT_XGMI_5_THRPUT	Transmit throughput to XGMI neighbor 5 in byes/sec
RDC_EVNT_NOTIF_VMFAULT	VM page fault.
RDC_EVNT_NOTIF_THERMAL_THROTTLE	Clock frequency has decreased due to temperature rise
RDC_EVNT_NOTIF_PRE_RESET	GPU reset is about to occur.
RDC_EVNT_NOTIF_POST_RESET	GPU reset just occurred.

4.1.4 Function Documentation

4.1.4.1 rdc_init()

```
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

in	*init_flags*	init_flags Bit flags that tell RDC how to initialize.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.2 rdc_shutdown()

```
rdc_status_t rdc_shutdown ( )
```

Shutdown ROCm RDC.

Do any necessary clean up.

4.1.4.3 rdc_start_embedded()

```
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

in	*op_mode*	Operation modes. When RDC_OPERATION_MODE_AUTO, RDC schedules background task to collect the stats. When RDC_OPERATION_MODE_MANUAL, the user needs to call rdc_field_update_all() periodically.
in,out	*p_rdc_handle*	Caller provided pointer to rdc_handle_t. Upon successful call, the value will contain the handler for following API calls.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.4 rdc_stop_embedded()

```
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	*p_rdc_handle*	The RDC handler that come from rdc_start_embedded().

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.5 rdc_connect()

```
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.

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

Parameters

in	*ipAndPort*	The IP and port of the remote rdcd. 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	*p_rdc_handle*	Caller provided pointer to rdc_handle_t. Upon successful call, the value will contain the handler for following API calls.
in	*root_ca*	The root CA stored in the string in pem format. Set it as nullptr if the communication is not encrypted.
in	*client_cert*	The client certificate stored in the string in pem format. Set it as nullptr if the communication is not encrypted.
in	*client_key*	The client key stored in the string in pem format. Set it as nullptr if the communication is not encrypted.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.6 rdc_disconnect()

```
rdc_status_t rdc_disconnect (
             rdc_handle_t p_rdc_handle )
```

Disconnect from rdcd daemon.

Disconnect from rdcd daemon, and p_rdc_handle becomes invalid after this call.

Parameters

in	*p_rdc_handle*	The RDC handler that come from rdc_connect().

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.7 rdc_job_start_stats()

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

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.8 rdc_job_get_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.

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

Parameters

in	*p_rdc_handle*	The RDC handler.
in	*job_id*	The name of the job.
in,out	*p_job_info*	Caller provided pointer to rdc_job_info_t. Upon successful call, the value will contain the stats of the job.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.9 rdc_job_stop_stats()

```
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	*p_rdc_handle*	The RDC handler.
in	*job_id*	The name of the job.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.10 rdc_job_remove()

```
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	*p_rdc_handle*	The RDC handler.
in	*job_id*	The name of the job.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.11 rdc_job_remove_all()

```
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	*p_rdc_handle*	The RDC handler.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.12 rdc_field_update_all()

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

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.13 rdc_device_get_all()

```
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	*p_rdc_handle*	The RDC handler.
out	*gpu_index_list*	Array reference to fill GPU indexes present on the system.
out	*count*	Number of GPUs returned in gpu_index_list.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.14 rdc_device_get_attributes()

```
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	*p_rdc_handle*	The RDC handler.
in	*gpu_index*	GPU index corresponding to which the attributes should be fetched
out	*p_rdc_attr*	GPU attribute corresponding to the gpu_index.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.15 rdc_group_gpu_create()

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

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.16 rdc_group_gpu_add()

```
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	*p_rdc_handle*	The RDC handler.
in	*group_id*	The group id to which the GPU will be added.
in	*gpu_index*	The GPU index to be added to the group.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.17 rdc_group_gpu_get_info()

```
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	*p_rdc_handle*	The RDC handler.
in	*p_rdc_group_id*	The GPU group handler created by rdc_group_gpu_create
out	*p_rdc_group_info*	The information of the GPU group p_rdc_group_id.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.18 rdc_group_get_all_ids()

```
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	*p_rdc_handle*	The RDC handler.
out	*group_id_list*	Array reference to fill GPU group ids in the system.
out	*count*	Number of GPU group returned in group_id_list.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.19 rdc_group_gpu_destroy()

```
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	*p_rdc_handle*	The RDC handler.
in	*p_rdc_group↩ _id*	The group id

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.20 rdc_group_field_create()

```
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	*p_rdc_handle*	The RDC handler.
in	*num_field_ids*	Number of field IDs that are being provided in field_ids.
in	*field_ids*	Field IDs to be added to the newly-created field group.
in	*field_group_name*	Unique name for this group of fields.
out	*rdc_field_group↩ _id*	Handle to the newly-created field group

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.21 rdc_group_field_get_info()

```
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	*p_rdc_handle*	The RDC handler.
in	*rdc_field_group↩ _id*	The field group handler created by rdc_group_field_create
out	*field_group_info*	The information of the field group rdc_field_group_id.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.22 rdc_group_field_get_all_ids()

```
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	*p_rdc_handle*	The RDC handler.
out	*field_group_id_list*	Array reference to fill field group ids in the system.
out	*count*	Number of field group returned in field_group_id_list.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.23 rdc_group_field_destroy()

```
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	*p_rdc_handle*	The RDC handler.
in	*rdc_field_group↩ _id*	The field group id

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.24 rdc_field_watch()

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

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.25 rdc_field_get_latest_value()

```
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	*p_rdc_handle*	The RDC handler.
in	*gpu_index*	The GPU index.
in	*field*	The field id
out	*value*	The field value got from cache.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.26 rdc_field_get_value_since()

```
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	*p_rdc_handle*	The RDC handler.
in	*gpu_index*	The GPU index.
in	*field*	The field id
in	*since_time_stamp*	Timestamp to request values since in usec since 1970.
out	*next_since_time_stamp*	Timestamp to use for sinceTimestamp on next call to this function
out	*value*	The field value got from cache.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.27 rdc_field_unwatch()

```
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	*p_rdc_handle*	The RDC handler.
in	*group_id*	The GPU group id.
in	*field_group_id*	The field group id.

Return values

RDC_ST_OK	is returned upon successful call.

4.1.4.28 rdc_status_string()

```
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	*status*	The RDC status.

Return values

The	string to describe the RDC status.

4.1.4.29 field_id_string()

```
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	*field_id*	The field id.

Return values

The	string to describe the field.

4.1.4.30 get_field_id_from_name()

```
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	*name*	The field name.

Return values

return	RDC_FI_INVALID if the field name is invalid.

Index

entity_ids
rdc_group_info_t, 9

field_id_string
rdc.h, 33
field_ids
rdc_field_group_info_t, 6

get_field_id_from_name
rdc.h, 33

rdc.h, 13
field_id_string, 33
get_field_id_from_name, 33
rdc_connect, 21
rdc_device_get_all, 25
rdc_device_get_attributes, 26
rdc_disconnect, 22
RDC_EVNT_NOTIF_POST_RESET, 19
RDC_EVNT_NOTIF_PRE_RESET, 19
RDC_EVNT_NOTIF_THERMAL_THROTTLE, 19
RDC_EVNT_NOTIF_VMFAULT, 19
RDC_EVNT_XGMI_0_BEATS_TX, 19
RDC_EVNT_XGMI_0_NOP_TX, 19
RDC_EVNT_XGMI_0_REQ_TX, 19
RDC_EVNT_XGMI_0_RESP_TX, 19
RDC_EVNT_XGMI_0_THRPUT, 19
RDC_EVNT_XGMI_1_BEATS_TX, 19
RDC_EVNT_XGMI_1_NOP_TX, 19
RDC_EVNT_XGMI_1_REQ_TX, 19
RDC_EVNT_XGMI_1_RESP_TX, 19
RDC_EVNT_XGMI_1_THRPUT, 19
RDC_EVNT_XGMI_2_THRPUT, 19
RDC_EVNT_XGMI_3_THRPUT, 19
RDC_EVNT_XGMI_4_THRPUT, 19
RDC_EVNT_XGMI_5_THRPUT, 19
RDC_FI_DEV_NAME, 18
RDC_FI_ECC_ATHUB_DED, 18
RDC_FI_ECC_ATHUB_SEC, 18
RDC_FI_ECC_BIF_DED, 18
RDC_FI_ECC_BIF_SEC, 18
RDC_FI_ECC_CORRECT_TOTAL, 18
RDC_FI_ECC_DF_DED, 19
RDC_FI_ECC_DF_SEC, 19
RDC_FI_ECC_FUSE_DED, 19
RDC_FI_ECC_FUSE_SEC, 19
RDC_FI_ECC_GFX_DED, 18
RDC_FI_ECC_GFX_SEC, 18
RDC_FI_ECC_HDP_DED, 19
RDC_FI_ECC_HDP_SEC, 18
RDC_FI_ECC_MMHUB_DED, 18
RDC_FI_ECC_MMHUB_SEC, 18
RDC_FI_ECC_MP0_DED, 19
RDC_FI_ECC_MP0_SEC, 19
RDC_FI_ECC_MP1_DED, 19
RDC_FI_ECC_MP1_SEC, 19
RDC_FI_ECC_SDMA_DED, 18
RDC_FI_ECC_SDMA_SEC, 18
RDC_FI_ECC_SEM_DED, 19
RDC_FI_ECC_SEM_SEC, 19
RDC_FI_ECC_SMN_DED, 19
RDC_FI_ECC_SMN_SEC, 19
RDC_FI_ECC_UMC_DED, 19
RDC_FI_ECC_UMC_SEC, 19
RDC_FI_ECC_UNCORRECT_TOTAL, 18
RDC_FI_ECC_XGMI_WAFL_DED, 19
RDC_FI_ECC_XGMI_WAFL_SEC, 19
RDC_FI_GPU_CLOCK, 18
RDC_FI_GPU_COUNT, 18
RDC_FI_GPU_MEMORY_TOTAL, 18
RDC_FI_GPU_MEMORY_USAGE, 18
RDC_FI_GPU_TEMP, 18
RDC_FI_GPU_UTIL, 18
RDC_FI_INVALID, 18
RDC_FI_MEM_CLOCK, 18
RDC_FI_MEMORY_TEMP, 18
RDC_FI_PCIE_RX, 18
RDC_FI_PCIE_TX, 18
RDC_FI_POWER_USAGE, 18
rdc_field_get_latest_value, 31
rdc_field_get_value_since, 32
rdc_field_t, 18
rdc_field_unwatch, 32
rdc_field_update_all, 25
rdc_field_watch, 31
RDC_GROUP_DEFAULT, 18
RDC_GROUP_EMPTY, 18
rdc_group_field_create, 29
rdc_group_field_destroy, 30
rdc_group_field_get_all_ids, 30
rdc_group_field_get_info, 29
rdc_group_get_all_ids, 28
rdc_group_gpu_add, 27
rdc_group_gpu_create, 26
rdc_group_gpu_destroy, 28
rdc_group_gpu_get_info, 27
rdc_group_type_t, 18
rdc_handle_t, 17
rdc_init, 20

rdc_job_get_stats, 23
rdc_job_remove, 24
rdc_job_remove_all, 24
rdc_job_start_stats, 22
rdc_job_stop_stats, 23
rdc_shutdown, 20
RDC_ST_ALREADY_EXIST, 17
RDC_ST_BAD_PARAMETER, 17
RDC_ST_CLIENT_ERROR, 17
RDC_ST_CONFLICT, 17
RDC_ST_FAIL_LOAD_MODULE, 17
RDC_ST_FILE_ERROR, 17
RDC_ST_INSUFF_RESOURCES, 17
RDC_ST_INVALID_HANDLER, 17
RDC_ST_MAX_LIMIT, 17
RDC_ST_MSI_ERROR, 17
RDC_ST_NO_DATA, 17
RDC_ST_NOT_FOUND, 17
RDC_ST_NOT_SUPPORTED, 17
RDC_ST_OK, 17
RDC_ST_PERM_ERROR, 17
RDC_ST_UNKNOWN_ERROR, 17
rdc_start_embedded, 20
rdc_status_string, 33
rdc_status_t, 17
rdc_stop_embedded, 21
rdc_connect
rdc.h, 21
rdc_device_attributes_t, 5
rdc_device_get_all
rdc.h, 25
rdc_device_get_attributes
rdc.h, 26
rdc_disconnect
rdc.h, 22
RDC_EVNT_NOTIF_POST_RESET
rdc.h, 19
RDC_EVNT_NOTIF_PRE_RESET
rdc.h, 19
RDC_EVNT_NOTIF_THERMAL_THROTTLE
rdc.h, 19
RDC_EVNT_NOTIF_VMFAULT
rdc.h, 19
RDC_EVNT_XGMI_0_BEATS_TX
rdc.h, 19
RDC_EVNT_XGMI_0_NOP_TX
rdc.h, 19
RDC_EVNT_XGMI_0_REQ_TX
rdc.h, 19
RDC_EVNT_XGMI_0_RESP_TX
rdc.h, 19
RDC_EVNT_XGMI_0_THRPUT
rdc.h, 19
RDC_EVNT_XGMI_1_BEATS_TX
rdc.h, 19
RDC_EVNT_XGMI_1_NOP_TX
rdc.h, 19
RDC_EVNT_XGMI_1_REQ_TX
rdc.h, 19
RDC_EVNT_XGMI_1_RESP_TX
rdc.h, 19
RDC_EVNT_XGMI_1_THRPUT
rdc.h, 19
RDC_EVNT_XGMI_2_THRPUT
rdc.h, 19
RDC_EVNT_XGMI_3_THRPUT
rdc.h, 19
RDC_EVNT_XGMI_4_THRPUT
rdc.h, 19
RDC_EVNT_XGMI_5_THRPUT
rdc.h, 19
RDC_FI_DEV_NAME
rdc.h, 18
RDC_FI_ECC_ATHUB_DED
rdc.h, 18
RDC_FI_ECC_ATHUB_SEC
rdc.h, 18
RDC_FI_ECC_BIF_DED
rdc.h, 18
RDC_FI_ECC_BIF_SEC
rdc.h, 18
RDC_FI_ECC_CORRECT_TOTAL
rdc.h, 18
RDC_FI_ECC_DF_DED
rdc.h, 19
RDC_FI_ECC_DF_SEC
rdc.h, 19
RDC_FI_ECC_FUSE_DED
rdc.h, 19
RDC_FI_ECC_FUSE_SEC
rdc.h, 19
RDC_FI_ECC_GFX_DED
rdc.h, 18
RDC_FI_ECC_GFX_SEC
rdc.h, 18
RDC_FI_ECC_HDP_DED
rdc.h, 19
RDC_FI_ECC_HDP_SEC
rdc.h, 18
RDC_FI_ECC_MMHUB_DED
rdc.h, 18
RDC_FI_ECC_MMHUB_SEC
rdc.h, 18
RDC_FI_ECC_MP0_DED
rdc.h, 19
RDC_FI_ECC_MP0_SEC
rdc.h, 19
RDC_FI_ECC_MP1_DED
rdc.h, 19
RDC_FI_ECC_MP1_SEC
rdc.h, 19
RDC_FI_ECC_SDMA_DED
rdc.h, 18
RDC_FI_ECC_SDMA_SEC
rdc.h, 18
RDC_FI_ECC_SEM_DED

rdc.h, 19
RDC_FI_ECC_SEM_SEC
rdc.h, 19
RDC_FI_ECC_SMN_DED
rdc.h, 19
RDC_FI_ECC_SMN_SEC
rdc.h, 19
RDC_FI_ECC_UMC_DED
rdc.h, 19
RDC_FI_ECC_UMC_SEC
rdc.h, 19
RDC_FI_ECC_UNCORRECT_TOTAL
rdc.h, 18
RDC_FI_ECC_XGMI_WAFL_DED
rdc.h, 19
RDC_FI_ECC_XGMI_WAFL_SEC
rdc.h, 19
RDC_FI_GPU_CLOCK
rdc.h, 18
RDC_FI_GPU_COUNT
rdc.h, 18
RDC_FI_GPU_MEMORY_TOTAL
rdc.h, 18
RDC_FI_GPU_MEMORY_USAGE
rdc.h, 18
RDC_FI_GPU_TEMP
rdc.h, 18
RDC_FI_GPU_UTIL
rdc.h, 18
RDC_FI_INVALID
rdc.h, 18
RDC_FI_MEM_CLOCK
rdc.h, 18
RDC_FI_MEMORY_TEMP
rdc.h, 18
RDC_FI_PCIE_RX
rdc.h, 18
RDC_FI_PCIE_TX
rdc.h, 18
RDC_FI_POWER_USAGE
rdc.h, 18
rdc_field_get_latest_value
rdc.h, 31
rdc_field_get_value_since
rdc.h, 32
rdc_field_group_info_t, 5
field_ids, 6
rdc_field_t
rdc.h, 18
rdc_field_unwatch
rdc.h, 32
rdc_field_update_all
rdc.h, 25
rdc_field_value, 6
value, 6
rdc_field_value_data, 7
rdc_field_watch
rdc.h, 31
rdc_gpu_usage_info_t, 7
RDC_GROUP_DEFAULT
rdc.h, 18
RDC_GROUP_EMPTY
rdc.h, 18
rdc_group_field_create
rdc.h, 29
rdc_group_field_destroy
rdc.h, 30
rdc_group_field_get_all_ids
rdc.h, 30
rdc_group_field_get_info
rdc.h, 29
rdc_group_get_all_ids
rdc.h, 28
rdc_group_gpu_add
rdc.h, 27
rdc_group_gpu_create
rdc.h, 26
rdc_group_gpu_destroy
rdc.h, 28
rdc_group_gpu_get_info
rdc.h, 27
rdc_group_info_t, 8
entity_ids, 9
rdc_group_type_t
rdc.h, 18
rdc_handle_t
rdc.h, 17
rdc_init
rdc.h, 20
rdc_job_get_stats
rdc.h, 23
rdc_job_group_info_t, 9
rdc_job_info_t, 10
summary, 10
rdc_job_remove
rdc.h, 24
rdc_job_remove_all
rdc.h, 24
rdc_job_start_stats
rdc.h, 22
rdc_job_stop_stats
rdc.h, 23
rdc_shutdown
rdc.h, 20
RDC_ST_ALREADY_EXIST
rdc.h, 17
RDC_ST_BAD_PARAMETER
rdc.h, 17
RDC_ST_CLIENT_ERROR
rdc.h, 17
RDC_ST_CONFLICT
rdc.h, 17
RDC_ST_FAIL_LOAD_MODULE
rdc.h, 17
RDC_ST_FILE_ERROR
rdc.h, 17

RDC_ST_INSUFF_RESOURCES
rdc.h, 17
RDC_ST_INVALID_HANDLER
rdc.h, 17
RDC_ST_MAX_LIMIT
rdc.h, 17
RDC_ST_MSI_ERROR
rdc.h, 17
RDC_ST_NO_DATA
rdc.h, 17
RDC_ST_NOT_FOUND
rdc.h, 17
RDC_ST_NOT_SUPPORTED
rdc.h, 17
RDC_ST_OK
rdc.h, 17
RDC_ST_PERM_ERROR
rdc.h, 17
RDC_ST_UNKNOWN_ERROR
rdc.h, 17
rdc_start_embedded
rdc.h, 20
rdc_stats_summary_t, 10
rdc_status_string
rdc.h, 33
rdc_status_t
rdc.h, 17
rdc_stop_embedded
rdc.h, 21

summary
rdc_job_info_t, 10

value
rdc_field_value, 6