

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	7
3.3.2.1 value	7
3.4 rdc_gpu_usage_info_t Struct Reference	7
3.4.1 Detailed Description	8
3.5 rdc_group_info_t Struct Reference	8
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	9
3.7 rdc_job_info_t Struct Reference	9
3.7.1 Detailed Description	9
3.7.2 Field Documentation	9
3.7.2.1 summary	10
3.8 rdc_stats_summary_t Struct Reference	10
3.8.1 Detailed Description	10
4 File Documentation	11
4.1 rdc.h File Reference	11
4.1.1 Detailed Description	14
4.1.2 Macro Definition Documentation	15
4.1.2.1 RDC_FI_GPU_MEMORY_USAGE	15
4.1.2.2 RDC_FI_GPU_MEMORY_TOTAL	15
4.1.2.3 RDC_FI_POWER_USAGE	15
4.1.2.4 RDC_FI_GPU_CLOCK	15
4.1.2.5 RDC_FI_MEM_CLOCK	15
4.1.2.6 RDC_FI_PCIE_TX	15
4.1.2.7 RDC_FI_PCIE_RX	15

4.1.2.8 RDC_FI_GPU_UTIL	16
4.1.2.9 RDC_FI_ECC_CORRECT_TOTAL	16
4.1.2.10 RDC_FI_ECC_UNCORRECT_TOTAL	16
4.1.2.11 RDC_FI_MEMORY_TEMP	16
4.1.2.12 RDC_FI_GPU_TEMP	16
4.1.2.13 RDC_FI_GPU_COUNT	16
4.1.2.14 RDC_FI_DEV_NAME	16
4.1.3 Typedef Documentation	16
4.1.3.1 rdc_handle_t	17
4.1.4 Enumeration Type Documentation	17
4.1.4.1 rdc_status_t	17
4.1.4.2 rdc_group_type_t	17
4.1.5 Function Documentation	18
4.1.5.1 rdc_init()	18
4.1.5.2 rdc_shutdown()	18
4.1.5.3 rdc_start_embedded()	18
4.1.5.4 rdc_stop_embedded()	19
4.1.5.5 rdc_connect()	19
4.1.5.6 rdc_disconnect()	20
4.1.5.7 rdc_job_start_stats()	20
4.1.5.8 rdc_job_get_stats()	21
4.1.5.9 rdc_job_stop_stats()	21
4.1.5.10 rdc_job_remove()	22
4.1.5.11 rdc_job_remove_all()	22
4.1.5.12 rdc_field_update_all()	23
4.1.5.13 rdc_device_get_all()	23
4.1.5.14 rdc_device_get_attributes()	24
4.1.5.15 rdc_group_gpu_create()	24
4.1.5.16 rdc_group_gpu_add()	25
4.1.5.17 rdc_group_gpu_get_info()	25
4.1.5.18 rdc_group_get_all_ids()	26
4.1.5.19 rdc_group_gpu_destroy()	26
4.1.5.20 rdc_group_field_create()	27
4.1.5.21 rdc_group_field_get_info()	27
4.1.5.22 rdc_group_field_get_all_ids()	28
4.1.5.23 rdc_group_field_destroy()	28
4.1.5.24 rdc_field_watch()	29
4.1.5.25 rdc_field_get_latest_value()	29
4.1.5.26 rdc_field_get_value_since()	30
4.1.5.27 rdc_field_unwatch()	30
4.1.5.28 rdc_status_string()	31
4.1.5.29 field_id_string()	31

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_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	8
rdc_job_info_t	The structure to hold the job stats	9
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

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
- uint32_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

```
uint32_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

- uint32_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 [l_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 value

```
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](#)
Uncorrrtable 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 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](#)

3.6 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 statictics by GPU.

3.7.1 Detailed Description

The structure to hold the job stats.

3.7.2 Field Documentation

3.7.2.1 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](#)

3.8 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.

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>
```

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.
- #define `RDC_FI_GPU_MEMORY_USAGE` 525
- #define `RDC_FI_GPU_MEMORY_TOTAL` 580
- #define `RDC_FI_POWER_USAGE` 155
- #define `RDC_FI_GPU_CLOCK` 100
- #define `RDC_FI_MEM_CLOCK` 101
- #define `RDC_FI_PCIE_TX` 200
- #define `RDC_FI_PCIE_RX` 201
- #define `RDC_FI_GPU_UTIL` 203
- #define `RDC_FI_ECC_CORRECT_TOTAL` 312
- #define `RDC_FI_ECC_UNCORRECT_TOTAL` 313
- #define `RDC_FI_MEMORY_TEMP` 140
- #define `RDC_FI_GPU_TEMP` 150
- #define `RDC_FI_GPU_COUNT` 4
- #define `RDC_FI_DEV_NAME` 50

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` }
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

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, uint32_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, uint32_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, uint32_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 (uint32_t field_id)`
Get the name of a field.

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 Macro Definition Documentation

4.1.2.1 RDC_FI_GPU_MEMORY_USAGE

```
#define RDC_FI_GPU_MEMORY_USAGE 525
```

Memory usage of the GPU instance

4.1.2.2 RDC_FI_GPU_MEMORY_TOTAL

```
#define RDC_FI_GPU_MEMORY_TOTAL 580
```

Total memory of the GPU instance

4.1.2.3 RDC_FI_POWER_USAGE

```
#define RDC_FI_POWER_USAGE 155
```

Power usage for the device

4.1.2.4 RDC_FI_GPU_CLOCK

```
#define RDC_FI_GPU_CLOCK 100
```

The current clock for the GPU

4.1.2.5 RDC_FI_MEM_CLOCK

```
#define RDC_FI_MEM_CLOCK 101
```

Clock for the memory

4.1.2.6 RDC_FI_PCIE_TX

```
#define RDC_FI_PCIE_TX 200
```

PCIe Tx utilization information

4.1.2.7 RDC_FI_PCIE_RX

```
#define RDC_FI_PCIE_RX 201
```

PCIe Rx utilization information

4.1.2.8 RDC_FI_GPU_UTIL

```
#define RDC_FI_GPU_UTIL 203
```

GPU Utilization

4.1.2.9 RDC_FI_ECC_CORRECT_TOTAL

```
#define RDC_FI_ECC_CORRECT_TOTAL 312
```

Accumulated correctable ECC errors

4.1.2.10 RDC_FI_ECC_UNCORRECT_TOTAL

```
#define RDC_FI_ECC_UNCORRECT_TOTAL 313
```

Accumulated uncorrectable ECC errors

4.1.2.11 RDC_FI_MEMORY_TEMP

```
#define RDC_FI_MEMORY_TEMP 140
```

Memory temperature for the device

4.1.2.12 RDC_FI_GPU_TEMP

```
#define RDC_FI_GPU_TEMP 150
```

Current temperature for the device

4.1.2.13 RDC_FI_GPU_COUNT

```
#define RDC_FI_GPU_COUNT 4
```

GPU count in the system

4.1.2.14 RDC_FI_DEV_NAME

```
#define RDC_FI_DEV_NAME 50
```

Name of the device

4.1.3 Typedef Documentation

4.1.3.1 rdc_handle_t

```
typedef void* rdc_handle_t
```

handlers used in various rdc calls

Handle used for an RDC session

4.1.4 Enumeration Type Documentation

4.1.4.1 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.

4.1.4.2 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.

4.1.5 Function Documentation

4.1.5.1 `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

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.2 `rdc_shutdown()`

```
rdc_status_t rdc_shutdown ( )
```

Shutdown ROCm RDC.

Do any necessary clean up.

4.1.5.3 `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	<i>op_mode</i>	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	<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

RDC_ST_OK	is returned upon successful call.
---------------------------	-----------------------------------

4.1.5.4 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	<i>p_rdc_handle</i>	The RDC handler that come from rdc_start_embedded() .
----	---------------------	---

Return values

RDC_ST_OK	is returned upon successful call.
---------------------------	-----------------------------------

4.1.5.5 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	<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

RDC_ST_OK	is returned upon successful call.
---------------------------	-----------------------------------

4.1.5.6 rdc_disconnect()

```
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 rdc_connect() .
----	---------------------	--

Return values

RDC_ST_OK	is returned upon successful call.
---------------------------	-----------------------------------

4.1.5.7 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	<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

<i>RDC_ST_OK</i>	is returned upon successful call.
----------------------------------	-----------------------------------

4.1.5.8 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	<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 <i>rdc_job_info_t</i> . Upon successful call, the value will contain the stats of the job.

Return values

<i>RDC_ST_OK</i>	is returned upon successful call.
----------------------------------	-----------------------------------

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

Return values

<i>RDC_ST_OK</i>	is returned upon successful call.
----------------------------------	-----------------------------------

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

Return values

<i>RDC_ST_OK</i>	is returned upon successful call.
----------------------------------	-----------------------------------

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.12 `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	<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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.13 `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	<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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.14 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	<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 <code>gpu_index</code> .

Return values

<i>RDC_ST_OK</i>	is returned upon successful call.
------------------	-----------------------------------

4.1.5.15 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	<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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.18 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	<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 group_id_list.

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.20 rdc_group_field_create()

```
rdc_status_t rdc_group_field_create (
    rdc_handle_t p_rdc_handle,
    uint32_t num_field_ids,
    uint32_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

<i>RDC_ST_OK</i>	is returned upon successful call.
------------------	-----------------------------------

4.1.5.21 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	<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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.22 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	<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	<i>count</i>	Number of field group returned in field_group_id_list.

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

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

Return values

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.24 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	<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

<i>RDC_ST_OK</i>	is returned upon successful call.
------------------	-----------------------------------

4.1.5.25 rdc_field_get_latest_value()

```
rdc_status_t rdc_field_get_latest_value (
    rdc_handle_t p_rdc_handle,
    uint32_t gpu_index,
    uint32_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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.26 `rdc_field_get_value_since()`

```
rdc_status_t rdc_field_get_value_since (
    rdc_handle_t p_rdc_handle,
    uint32_t gpu_index,
    uint32_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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.27 `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	<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

<code>RDC_ST_OK</code>	is returned upon successful call.
------------------------	-----------------------------------

4.1.5.28 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	<code>status</code>	The RDC status.
----	---------------------	-----------------

Return values

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

4.1.5.29 field_id_string()

```
const char* field_id_string (
    uint32_t field_id )
```

Get the name of a field.

return the string in human readable format.

Parameters

in	<code>field_id</code> <code>_id</code>	The field id.
----	---	---------------

Return values

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

Index

- entity_ids
 - rdc_group_info_t, 8
- field_id_string
 - rdc.h, 31
- field_ids
 - rdc_field_group_info_t, 6
- rdc.h, 11
 - field_id_string, 31
 - rdc_connect, 19
 - rdc_device_get_all, 23
 - rdc_device_get_attributes, 24
 - rdc_disconnect, 20
 - RDC_FI_DEV_NAME, 16
 - RDC_FI_ECC_CORRECT_TOTAL, 16
 - RDC_FI_ECC_UNCORRECT_TOTAL, 16
 - RDC_FI_GPU_CLOCK, 15
 - RDC_FI_GPU_COUNT, 16
 - RDC_FI_GPU_MEMORY_TOTAL, 15
 - RDC_FI_GPU_MEMORY_USAGE, 15
 - RDC_FI_GPU_TEMP, 16
 - RDC_FI_GPU_UTIL, 15
 - RDC_FI_MEM_CLOCK, 15
 - RDC_FI_MEMORY_TEMP, 16
 - RDC_FI_PCIE_RX, 15
 - RDC_FI_PCIE_TX, 15
 - RDC_FI_POWER_USAGE, 15
 - rdc_field_get_latest_value, 29
 - rdc_field_get_value_since, 30
 - rdc_field_unwatch, 30
 - rdc_field_update_all, 23
 - rdc_field_watch, 29
 - RDC_GROUP_DEFAULT, 17
 - RDC_GROUP_EMPTY, 17
 - rdc_group_field_create, 27
 - rdc_group_field_destroy, 28
 - rdc_group_field_get_all_ids, 28
 - rdc_group_field_get_info, 27
 - rdc_group_get_all_ids, 26
 - rdc_group_gpu_add, 25
 - rdc_group_gpu_create, 24
 - rdc_group_gpu_destroy, 26
 - rdc_group_gpu_get_info, 25
 - rdc_group_type_t, 17
 - rdc_handle_t, 16
 - rdc_init, 18
 - rdc_job_get_stats, 21
 - rdc_job_remove, 22
 - rdc_job_remove_all, 22
 - rdc_job_start_stats, 20
 - rdc_job_stop_stats, 21
 - rdc_shutdown, 18
 - 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_INVALID_HANDLER, 17
 - RDC_ST_MAX_LIMIT, 17
 - RDC_ST_MSI_ERROR, 17
 - RDC_ST_NOT_FOUND, 17
 - RDC_ST_NOT_SUPPORTED, 17
 - RDC_ST_OK, 17
 - rdc_start_embedded, 18
 - rdc_status_string, 31
 - rdc_status_t, 17
 - rdc_stop_embedded, 19
- rdc_connect
 - rdc.h, 19
- rdc_device_attributes_t, 5
- rdc_device_get_all
 - rdc.h, 23
- rdc_device_get_attributes
 - rdc.h, 24
- rdc_disconnect
 - rdc.h, 20
- RDC_FI_DEV_NAME
 - rdc.h, 16
- RDC_FI_ECC_CORRECT_TOTAL
 - rdc.h, 16
- RDC_FI_ECC_UNCORRECT_TOTAL
 - rdc.h, 16
- RDC_FI_GPU_CLOCK
 - rdc.h, 15
- RDC_FI_GPU_COUNT
 - rdc.h, 16
- RDC_FI_GPU_MEMORY_TOTAL
 - rdc.h, 15
- RDC_FI_GPU_MEMORY_USAGE
 - rdc.h, 15
- RDC_FI_GPU_TEMP
 - rdc.h, 16
- RDC_FI_GPU_UTIL
 - rdc.h, 15
- RDC_FI_MEM_CLOCK
 - rdc.h, 15
- RDC_FI_MEMORY_TEMP
 - rdc.h, 16

RDC_FI_PCIE_RX
 rdc.h, 15
 RDC_FI_PCIE_TX
 rdc.h, 15
 RDC_FI_POWER_USAGE
 rdc.h, 15
 rdc_field_get_latest_value
 rdc.h, 29
 rdc_field_get_value_since
 rdc.h, 30
 rdc_field_group_info_t, 5
 field_ids, 6
 rdc_field_unwatch
 rdc.h, 30
 rdc_field_update_all
 rdc.h, 23
 rdc_field_value, 6
 value, 7
 rdc_field_watch
 rdc.h, 29
 rdc_gpu_usage_info_t, 7
 RDC_GROUP_DEFAULT
 rdc.h, 17
 RDC_GROUP_EMPTY
 rdc.h, 17
 rdc_group_field_create
 rdc.h, 27
 rdc_group_field_destroy
 rdc.h, 28
 rdc_group_field_get_all_ids
 rdc.h, 28
 rdc_group_field_get_info
 rdc.h, 27
 rdc_group_get_all_ids
 rdc.h, 26
 rdc_group_gpu_add
 rdc.h, 25
 rdc_group_gpu_create
 rdc.h, 24
 rdc_group_gpu_destroy
 rdc.h, 26
 rdc_group_gpu_get_info
 rdc.h, 25
 rdc_group_info_t, 8
 entity_ids, 8
 rdc_group_type_t
 rdc.h, 17
 rdc_handle_t
 rdc.h, 16
 rdc_init
 rdc.h, 18
 rdc_job_get_stats
 rdc.h, 21
 rdc_job_group_info_t, 8
 rdc_job_info_t, 9
 summary, 9
 rdc_job_remove
 rdc.h, 22
 rdc_job_remove_all
 rdc.h, 22
 rdc_job_start_stats
 rdc.h, 20
 rdc_job_stop_stats
 rdc.h, 21
 rdc_shutdown
 rdc.h, 18
 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_INVALID_HANDLER
 rdc.h, 17
 RDC_ST_MAX_LIMIT
 rdc.h, 17
 RDC_ST_MSI_ERROR
 rdc.h, 17
 RDC_ST_NOT_FOUND
 rdc.h, 17
 RDC_ST_NOT_SUPPORTED
 rdc.h, 17
 RDC_ST_OK
 rdc.h, 17
 rdc_start_embedded
 rdc.h, 18
 rdc_stats_summary_t, 10
 rdc_status_string
 rdc.h, 31
 rdc_status_t
 rdc.h, 17
 rdc_stop_embedded
 rdc.h, 19
 summary
 rdc_job_info_t, 9
 value
 rdc_field_value, 7