DEOS-FY2013-EA-01E

D-RE APIs Specification

Version E1.02

2013/09/01

Edited by DEOS R&D Center

DEOS Project

JST-CREST

Research Area

"Dependable Operating Systems for Embedded Systems Aiming at Practical Applications"

Japan Science and Technology Agency

Contents

1 APIs for the Containers 5
1.1 Common 5
1.1.1 Lifecycle 5
1.1.2 Structures 5
1.1.2.1 dre_err_t 5
1.1.3 Type declarations of the other 6
1.1.3.1 dre_list_t 6
1.1.3.2 dre_name_t 6
1.1.3.3 dre_return_t 6
1.2 APIs for the Application Containers 7
1.2.1 How to use 7
1.2.2 Constants and Macros 7
1.2.2.1 dre_app_resource_t 7
1.2.2.2 dre_app_status_t 7
1.2.2.3 dre_app_type_t 7
1.2.3 Structures 8
1.2.3.1 dre_app_data 8
1.2.3.2 dre_app_handle 8
1.2.3.3 dre_app_pid_t 8
1.2.3.4 dre_app_resource_value_t 9
1.2.4 Type declarations 9
1.2.4.1 dre_app_pid_list_t 9
1.2.5 Functions 9
1.2.5.1 dre_app_bind_pid 9
1.2.5.2 dre_app_bind_vpid 10
1.2.5.3 dre_app_checkpoint 10
1.2.5.4 dre_app_close 11
1.2.5.5 dre_app_commit_snapshot 11
1.2.5.6 dre_app_create_cgroup 12
1.2.5.7 dre_app_create_container 12
1.2.5.8 dre_app_destroy_cgroup 12
1.2.5.9 dre_app_destroy_container 13
1.2.5.10 dre_app_get_base_list 13
1.2.5.11 dre_app_get_checkpoint_list 14
1.2.5.12 dre_app_get_container_list 14
1.2.5.13 dre_app_get_pid 15
1.2.5.14 dre_app_get_registered_pid_list 15
1.2.5.15 dre_app_get_resource 15
1.2.5.16 dre_app_get_snapshot_list 16
1.2.5.17 dre_app_load_snapshot 16
1.2.5.18 dre_app_open 17
1.2.5.19 dre_app_register_pid 17
1.2.5.20 dre_app_register_vpid 18
1.2.5.21 dre_app_release_base_list 18
1.2.5.22 dre_app_release_checkpoint_list 19
1.2.5.23 dre_app_release_container_list 19
1.2.5.24 dre_app_release_registered_pid_list 19
1.2.5.25 dre_app_release_snapshot_list 20

1.2.5.26 dre_app_remove_base....20
1.2.5.27 dre_app_remove_checkpoint....20
1.2.5.28 dre_app_remove_snapshot....21
1.2.5.29 dre_app_restart....21
1.2.5.30 dre_app_resume....22
1.2.5.31 dre_app_save_snapshot....22
1.2.5.32 dre_app_set_resource....23
1.2.5.33 dre_app_start....23
1.2.5.34 dre_app_status....24
1.2.5.35 dre_app_stop....24
1.2.5.36 dre_app_suspend....24
1.2.5.37 dre_app_unbind_pid....25
1.2.5.38 dre_app_unbind_vpid....25
1.2.5.39 dre_app_unregister_pid....26
1.2.5.40 dre_app_unregister_vpid....26
1.2.5.41 dre_app_unset_resource....27
1.3 APIs for the System Containers....28
1.3.1 How to use....28
1.3.2 Constants and Macros....28
1.3.2.1 dre_sys_resource_t....28
1.3.2.2 dre_sys_status_t....28
1.3.2.3 dre_sys_type_t....28
1.3.3 Structures....29
1.3.3.1 dre_sys_data....29
1.3.3.2 dre_sys_handle....29
1.3.3.3 dre_sys_resource_value_t....29
1.3.4 Type declarations....30
1.3.5 Functions....30
1.3.5.1 dre_sys_bind_pid....30
1.3.5.2 dre_sys_checkpoint....30
1.3.5.3 dre_sys_close....31
1.3.5.4 dre_sys_commit_snapshot....31
1.3.5.5 dre_sys_create_cgroup....32
1.3.5.6 dre_sys_create_container....32
1.3.5.7 dre_sys_destroy_cgroup....33
1.3.5.8 dre_sys_destroy_container....33
1.3.5.9 dre_sys_get_base_list....33
1.3.5.10 dre_sys_get_checkpoint_list....34
1.3.5.11 dre_sys_get_container_list....34
1.3.5.12 dre_sys_get_pid....35
1.3.5.13 dre_sys_get_resource....35
1.3.5.14 dre_sys_get_snapshot_list....36
1.3.5.15 dre_sys_immigration_start....36
1.3.5.16 dre_sys_load_snapshot....36
1.3.5.17 dre_sys_migration_start....37
1.3.5.18 dre_sys_open....37
1.3.5.19 dre_sys_release_base_list....38
1.3.5.20 dre_sys_release_checkpoint_list....38
1.3.5.21 dre_sys_release_container_list....39

1.3.5.22 dre_sys_release_snapshot_list 39
1.3.5.23 dre_sys_remove_base 39
1.3.5.24 dre_sys_remove_checkpoint 40
1.3.5.25 dre_sys_remove_snapshot 40
1.3.5.26 dre_sys_restart 41
1.3.5.27 dre_sys_resume 41
1.3.5.28 dre_sys_save_snapshot 41
1.3.5.29 dre_sys_set_resource 42
1.3.5.30 dre_sys_share 42
1.3.5.31 dre_sys_start 43
1.3.5.32 dre_sys_status 43
1.3.5.33 dre_sys_stop 44
1.3.5.34 dre_sys_suspend 44
1.3.5.35 dre_sys_unbind_pid 44
1.3.5.36 dre_sys_unset_resource 45
1.3.5.37 dre_sys_unshare 45
2 APIs for the D-Aware Applications 47
2.1 Lifecycle 47
2.2 How to use 47
2.3 Constants and Macros 47
2.3.1 DLOG 47
2.3.2 DLOG_EMERG/DLOG_ALERT/DLOG_CRIT/DLOG_ERR/DLOG_WARNING/DLOG_NOTICE/DLOG_INFO/DLOG_DEBUG 47
2.4 Structures 48
2.5 Type declarations 48
2.5.1 DTermination 48
2.6 Functions 48
2.6.1 daware_log 48
2.6.2 daware_log_close 49
2.6.3 daware_log_open 49
2.6.4 daware_log_set_format 50
2.6.5 daware_termination_register_callback 50
2.6.6 daware_termination_unregister_callback 51
2.7 Others 51
2.7.1 facility 51
2.7.2 level 52
2.7.3 option 52

History

Version	変更内容	変更者	日付
1.02	Corrected the omittions and typo	Dependable Embedded OS R&D Center	2013.Sep.1

1 APIs for the Containers

1.1 Common

1.1.1 Lifecycle

The lifecycle of the container is "create" -> "start" -> "stop" -> "destroy". Once a container is created, you can reuse it any number of times until it is destroyed.

In addition, the lifecycle of the container with the programs that use these APIs is getting the handle ("open") -> the operation of the container (such as "start" or "stop") -> releasing the handle of the container ("close"). Once a handle is opened, you can reuse it any number of times until it is closed.

1.1.2 Structures

1.1.2.1 dre_err_t

Summary

Error code

Implementation

```
typedef enum {
    DRE_ERR_INVALID_ARGUMENT          = -101,   // Invalid argument
    DRE_ERR_PERMISSION_DENIED         = -102,   // No privilege to run
    DRE_ERR_DAWARE_MOD_NONEXISTENCE   = -103,   // Daware.mod is not available
    DRE_ERR_NO_VT                     = -104,   // Intel-VT and AMD-V are not available
    DRE_ERR_CONTAINER_NONEXISTENCE    = -111,   // The container does not exist
    DRE_ERR_CONTAINER_EXISTENCE       = -112,   // The container already exists
    DRE_ERR_CONTAINER_RUNNING         = -113,   // The container is running contrary to expectations
    DRE_ERR_CONTAINER_STOPPED         = -114,   // The container is stopped contrary to expectations
    DRE_ERR_CONTAINER_SUSPENDED       = -115,   // The container is suspended contrary to expectations
    DRE_ERR_CONTAINER_BUSY            = -116,   // The container is busy
    DRE_ERR_CONTAINER_UNKNOWN_STATUS  = -117,   // The status of the container is unknown
    DRE_ERR_BASE_NONEXISTENCE         = -121,   // The Base Image does not exist
    DRE_ERR_BASE_EXISTENCE            = -122,   // The Base Image already exists
    DRE_ERR_REMOVE_BASE_FAILED        = -123,   // Failed to remove the Base Image
    DRE_ERR_SNAPSHOT_NONEXISTENCE     = -131,   // The snapshot does not exist
    DRE_ERR_SNAPSHOT_EXISTENCE        = -132,   // The snapshot already exists
    DRE_ERR_SAVE_SNAPSHOT_FAILED      = -133,   // Failed to save the snapshot
    DRE_ERR_LOAD_SNAPSHOT_FAILED      = -134,   // Failed to load the snapshot
    DRE_ERR_REMOVE_SNAPSHOT_FAILED    = -135,   // Failed to remove the snapshot/checkpoint
    DRE_ERR_COMMIT_SNAPSHOT_FAILED    = -136,   // Failed to commit the snapshot
    DRE_ERR_CGROUP_NONEXISTENCE       = -141,   // The Control Group does not exist
    DRE_ERR_CGROUP_EXISTENCE          = -142,   // The Control Group already exists
    DRE_ERR_PID_NONEXISTENCE          = -143,   // The process with the specified PID does not exist
    DRE_ERR_SET_LIMIT_FAILED          = -144,   // Failed to set the limitation of resource usage
    DRE_ERR_UNSET_LIMIT_FAILED        = -145,   // Failed to unset the limitation of resource usage
    DRE_ERR_DESTROY_CGROUP_FAILED     = -146,   // Failed to remove the Control Group
```

```
        DRE_ERR_DEVICE_BUSY             = -151,    // No response from the device
        DRE_ERR_NO_TAP_AVAILABLE        = -152,    // No TAP device is available
        DRE_ERR_NO_SOCKET_AVAILABLE     = -153,    // No socket is available
        DRE_ERR_NFS_MOUNT_FAILED        = -154,    // Failed to mount NFS server
        DRE_ERR_BIND_FAILED             = -155,    // Failed to bind PID
        DRE_ERR_INVALID_HOSTNAME        = -156,    // Invalid host name
        DRE_ERR_NULL_POINTER            = -201,    // Null pointer
        DRE_ERR_REQUEST_TOO_LONG        = -202,    // Request is too long
    } dre_err_t;
```

1.1.3 Type declarations of the other

1.1.3.1 dre_list_t

Summary

The list of the containers

Implementation

```
typedef dre_name_t *        dre_list_t;
```

1.1.3.2 dre_name_t

Summary

The name of the container

Implementation

```
typedef char *              dre_name_t;
```

1.1.3.3 dre_return_t

Summary

Return values

Implementation

```
typedef int                 dre_return_t;
```

1.2 APIs for the Application Containers

1.2.1How to use

To use these APIs in your program, include "dre_app.h" and link the libraries of D-RE (libdreapp.so or libdreapp.a) and D-Aware (libdaware.so or libdaware.a).

1.2.2Constants and Macros

1.2.2.1 dre_app_resource_t

Summary

The type of the resource

Implementation

```
typedef enum {
        DRE_APP_CONTAINER_RESOURCE_CPU = 1,    // CPU
        DRE_APP_CONTAINER_RESOURCE_CPU_CORE,  // CPU core(s)
        DRE_APP_CONTAINER_RESOURCE_MEMORY,    // Memory
        DRE_APP_CONTAINER_RESOURCE_MEMSW      // Memory and swap
} dre_app_resource_t;
```

1.2.2.2 dre_app_status_t

Summary

The status of the container

Implementation

```
typedef enum {
        DRE_APP_CONTAINER_STATUS_STOP = 11,   // The container is stopped.
        DRE_APP_CONTAINER_STATUS_RUNNING,     // The container is running.
        DRE_APP_CONTAINER_STATUS_SUSPENDED,   // The container is suspended.
        DRE_APP_CONTAINER_STATUS_BUSY         // The container is busy.
} dre_app_status_t;
```

1.2.2.3 dre_app_type_t

Summary

The type of the Application Container

Implementation

```
typedef enum {
        DRE_APP_CONTAINER_TYPE_LXC = 1        // LXC
} dre_app_type_t;
```

1.2.3Structures

1.2.3.1 dre_app_data

Summary

The configuration of the container

Implementation

```
typedef struct {
        dre_app_type_t type;     // The type of the container
        union {
                union {
                        int port;       // The port number for SSH on the container
                        char *base;     // The name of the Base Image (NULL for default)
                } lxc;
        } udata;
} dre_app_handle;
```

The name of the Base Image created by "dre-app commit" can be specified as "base", which is a member of the above structure. If the Base Image is specified, it can be used as the file system of the application container.

1.2.3.2 dre_app_handle

Summary

The handle of the Application Container

Implementation

```
typedef struct {
        dre_app_type_t type;     // The type of the container
        char *name;              // The name of the container
} dre_app_handle;
```

1.2.3.3 dre_app_pid_t

Summary

The Process ID of a process in the Application Container

Implementation

```
typedef struct {
        int pid;                 // The Process ID assigned at the Host OS
        int vpid;                // The Process ID assigned at the Application Container
} dre_app_pid_t;
```

1.2.3.4 dre_app_resource_value_t

Summary

Resource usage

Implementation

```
typedef struct {
        dre_app_resource_t type;                // The type of the resource
                union {
                        union {
                                int parcent;            // CPU usage (%)
                                char *cores;            // CPU core(s)
                        } cpu;
                union {
                        int usage_in_bytes;             // Memory usage (Bytes)
                        int memsw_usage_in_bytes;       // Memory and swap usage (Bytes)
                } mem;
        } udata;
} dre_app_resource_value_t;
```

1.2.4Type declarations

1.2.4.1 dre_app_pid_list_t

Summary

The list of the PIDs of the processes in the Application Container

Implementation

```
typedef dre_app_pid_t **        dre_app_pid_list_t;
```

1.2.5Functions

1.2.5.1 dre_app_bind_pid

Summary

Bind a process in the Host OS into a Control Group
(This function is available only while the container is running.)

Function interface

```
dre_return_t  dre_app_bind_pid  (
        const dre_app_handle *          h,
        const char *                    cgroup,
        const int                       pid
);
```

Arguments

```
const dre_app_handle *          h       // Handle of the container
const char *                    cgroup  // The name of the Control Group
const int                       pid     // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.2 dre_app_bind_vpid

Summary

Bind a process in the Application Container into a Control Group
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_bind_vpid (
        const dre_app_handle *      h,
        const char *                cgroup,
        const int                   vpid
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
const int                       vpid    // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.3 dre_app_checkpoint

Summary

Generate a checkpoint of an Application Container (Not implemented yet)

Function interface

```
dre_return_t dre_app_checkpoint (
        const dre_app_handle *      h,
        const char *                tag
);
```

Arguments

```
const dre_app_handle *          h   // The handle of the container
```

```
const char *                    tag  // The name of the checkpoint
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.4 dre_app_close

Summary

Release a handle of an Application Container

Function interface

```
dre_return_t dre_app_close (
        const dre_app_handle *          h
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.5 dre_app_commit_snapshot

Summary

Create a new Base Image of the Application Container from a snapshot
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_app_commit_snapshot (
        const dre_app_handle *          h,
        const char *                    tag,
        const char *                    base
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    tag     // The name of the snapshot
const char *                    base    // The name of the Base Image
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.6 dre_app_create_cgroup

Summary

Create a Control Group in the Application Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_create_cgroup (
        const dre_app_handle *          h,
        const char *                    cgroup
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.2.5.7 dre_app_create_container

Summary

Create an Application Container

Function interface

```
dre_return_t dre_app_create_container (
        const char *                    name,
        const dre_app_data *            data
);
```

Arguments

```
const char *                    name    // The name of the container
const dre_app_data *            data    // The configuration of the container
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.2.5.8 dre_app_destroy_cgroup

Summary

Remove a Control Group on the container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_destroy_cgroup (
        const dre_app_handle *          h,
        const char *                    cgroup
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.9 dre_app_destroy_container

Summary

Remove a container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_app_destroy_container (
        const char *                    name
);
```

Arguments

```
const char *                    name    // The name of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.10 dre_app_get_base_list

Summary

List the Base Images of the Application Containers
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_app_release_base_list()" after using this function.)

Function interface

```
dre_return_t dre_app_get_base_list (
        dre_list_t *                    list
);
```

Arguments

dre_list_t *	list // The pointer to the list of the containers

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.11 dre_app_get_checkpoint_list

Summary

List the checkpoints of an Application Container (Not implemented yet)

Function interface

```
dre_return_t dre_app_get_checkpoint_list (
        const dre_app_handle *          h,
        dre_list_t *                    list
);
```

Arguments

const dre_app_handle *	h // The handle of the container
dre_list_t *	list // The pointer to the list of the checkpoints

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.12 dre_app_get_container_list

Summary

List the Application Containers on the system
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_app_release_container_list()" after using this function.)

Function interface

```
dre_return_t dre_app_get_container_list (
        dre_list_t *                    list
);
```

Arguments

dre_list_t *	list // The pointer to the list of the containers

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.13 dre_app_get_pid

Summary

Get the Process ID of an Application Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_get_pid (
        const dre_app_handle *          h,
        int *                           pid
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
int *                           pid   // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.14 dre_app_get_registered_pid_list

Summary

Get a list of the PIDs under the control of the D-Aware module
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_app_release_registered_pid_list()" after using this function.)

Function interface

```
dre_return_t dre_app_get_registered_pid_list (
        const dre_app_handle *          h,
        dre_pid_list_t *                list
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
dre_app_pid_list_t *            list  // The pointer to the list of PIDs
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.15 dre_app_get_resource

Summary

Obtain the information of the resource usage
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_get_resource (
        const dre_app_handle *          h,
        const int                       pid,
        dre_app_resource_value_t *      resource
);
```

Arguments

const dre_app_handle *	h	// The handle of the container
const int	pid	// Process ID
dre_app_resource_value_t *	resource	// Resource usage

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.16 dre_app_get_snapshot_list

Summary

List the snapshots of an Application Container
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_app_release_snapshot_list()" after using this function.)

Function interface

```
dre_return_t dre_app_get_snapshot_list (
        const dre_app_handle *          h,
        dre_list_t *                    list
);
```

Arguments

const dre_app_handle *	h	// The handle of the container
dre_list_t *	list	// The pointer to the list of PID

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.17 dre_app_load_snapshot

Summary

Restore the file system of an Application Container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_app_load_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
const char *                    tag   // The name of the snapshot
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.2.5.18 dre_app_open

Summary

Get a handle of an Application Container

Function interface

```
dre_app_handle * dre_app_open (
        const char *                    name
);
```

Arguments

```
const char *                    name     // The name of the container
```

Return value

```
NULL                            Failed to get a handle
Not NULL                        Pointer to the handle of the container
```

1.2.5.19 dre_app_register_pid

Summary

Put a process in the Host OS under the control of the D-Aware module

Function interface

```
dre_return_t dre_app_register_pid (
        const dre_app_handle *          h,
        const int                       pid
);
```

Arguments

const dre_app_handle *	h	// The handle of the container
const int	pid	// Process ID

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.20 dre_app_register_vpid

Summary

Put a process in the Application Container under the control of the D-Aware module

Function interface

```
dre_return_t dre_app_register_vpid (
        const dre_app_handle *          h,
        const int                       vpid
);
```

Arguments

const dre_app_handle *	h	// The handle of the container
const int	vpid	// Process ID

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.21 dre_app_release_base_list

Summary

Release a list of the Base Images of the Application Containers

Function interface

```
dre_return_t dre_app_release_base_list (
        dre_list_t                      list
);
```

Arguments

dre_list_t	list	// The list of the Application Containers

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.22 dre_app_release_checkpoint_list

Summary

Release a list of the checkpoints of an Application Container (Not implemented yet)

Function interface

```
dre_return_t dre_app_release_checkpoint_list (
        dre_list_t                      list
);
```

Arguments

dre_list_t	list // The pointer to the list of the checkpoints

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.23 dre_app_release_container_list

Summary

Release a list of the Application Containers

Function interface

```
dre_return_t dre_app_release_container_list (
        dre_list_t                      list
);
```

Arguments

dre_list_t	list // The list of the Application Containers

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.24 dre_app_release_registered_pid_list

Summary

Release a list of the PIDs under the control of the D-Aware module

Function interface

```
dre_return_t dre_app_release_registered_pid_list (
        dre_app_pid_list_t              list
);
```

Arguments

dre_app_pid_list_t	list // The list of the PIDs

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.25 dre_app_release_snapshot_list

Summary

Release a list of the snapshots of an Application Container

Function interface

```
dre_return_t dre_app_release_snapshot_list (
        dre_list_t                  list
);
```

Arguments

dre_list	list // The list of the snapshots

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.26 dre_app_remove_base

Summary

Remove a Base Image of the Application Containers

Function interface

```
dre_return_t dre_app_remove_base (
        const char *                base
);
```

Arguments

const char *	base // The name of the Base Image

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.27 dre_app_remove_checkpoint

Summary

Remove a checkpoint of an Application Container (Not implemented yet)

Function interface

```
dre_return_t dre_app_remove_checkpoint (
        const dre_app_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
const char *                    tag  // The name of the checkpoint
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.28 dre_app_remove_snapshot

Summary

Remove a snapshot of an Application Container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_app_remove_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
const char *                    tag  // The name of the snapshot
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.29 dre_app_restart

Summary

Restart an Application Container from a checkpoint (Not implemented yet)

Function interface

```
dre_return_t dre_app_restart (
```

```
        const dre_app_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
const char *                    tag  // The name of the checkpoint
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.30 dre_app_resume

Summary

Resume an Application Container which is suspended

Function interface

```
dre_return_t dre_app_resume (
        const dre_app_handle *          h
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.31 dre_app_save_snapshot

Summary

Generate a snapshot of an Application Container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_app_save_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
const char *                    tag  // The name of the snapshot
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.32 dre_app_set_resource

Summary

Set a limitation on resource usage of an Application Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_set_resource (
        const dre_app_handle *          h,
        const char *                    cgroup,
        dre_app_resource_value_t *      resource
);
```

Arguments

```
const dre_app_handle *          h         // The handle of the container
const char *                    cgroup    // The name of the Control Group
dre_app_resource_value_t *      resource  // Resource usage
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.33 dre_app_start

Summary

Start an Application Container

Function interface

```
dre_return_t dre_app_start (
        const dre_app_handle *          h
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.34 dre_app_status

Summary

Get the status of an Application Container

Function interface

```
dre_return_t dre_app_status (
        const dre_app_handle *          h
);
```

Arguments

```
const dre_app_handle *          h    // The handle of the container
```

Return value

DRE_APP_CONTAINER_STATUS_STOP	The container is stopped
DRE_APP_CONTAINER_STATUS_RUNNING	The container is running
DRE_APP_CONTAINER_STATUS_SUSPENDED	The container is suspended
DRE_APP_CONTAINER_STATUS_BUSY	The container is busy
Others	Failed (See 1.1.2.1 for detail.)

1.2.5.35 dre_app_stop

Summary

Stop an Application Container

Function interface

```
dre_return_t dre_app_stop (
        const dre_app_handle *          h
);
```

Arguments

```
const dre_app_handle *          h    // the handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.36 dre_app_suspend

Summary

Suspend an Application Container which is running

Function interface

```
dre_return_t dre_app_suspend (
```

```
    const dre_app_handle *      h
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.37 dre_app_unbind_pid

Summary

Unbind a process which is bound to the Control Group in the Host OS
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_unbind_pid (
    const dre_app_handle *      h,
    const char *                cgroup,
    const int                   pid
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
const int                       pid     // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.38 dre_app_unbind_vpid

Summary

Unbind a process which is bound to the Control Group in an Application Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_unbind_vpid (
    const dre_app_handle *      h,
    const char *                cgroup,
    const int                   vpid
);
```

Arguments

```
const dre_app_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
const int                       vpid    // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.39 dre_app_unregister_pid

Summary

Exclude a process in the Host OS from the control of the D-Aware module

Function interface

```
dre_return_t dre_app_unregister_pid (
    const dre_app_handle *      h,
    const int                   pid
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
const int                       pid   // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.40 dre_app_unregister_vpid

Summary

Exclude a process in the Application Container from the control of the D-Aware module

Function interface

```
dre_return_t dre_app_unregister_vpid (
    const dre_app_handle *      h,
    const int                   vpid
);
```

Arguments

```
const dre_app_handle *          h     // The handle of the container
const int                       vpid  // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.2.5.41 dre_app_unset_resource

Summary

Release the limitation on resource usage of an Application Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_app_unset_resource (
        const dre_app_handle *      h,
        const char *                cgroup,
        dre_app_resource_value_t *  resource
);
```

Arguments

```
const dre_app_handle *          h         // the handle of the container
const char *                    cgroup    // the name of the Control Group
dre_app_resource_value_t *      resource  // resource usage
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3 APIs for the System Containers

1.3.1How to use

To use these APIs in your program, include "dre_sys.h" and link the libraries of D-RE (libdresys.so or libdresys.a) and D-Aware (libdaware.so or libdaware.a).

1.3.2Constants and Macros

1.3.2.1 dre_sys_resource_t

Summary

The type of the resource

Implementation

```
typedef enum {
        DRE_SYS_CONTAINER_RESOURCE_CPU = 1,      // CPU
        DRE_SYS_CONTAINER_RESOURCE_CPU_CORE,     // CPU core(s)
        DRE_SYS_CONTAINER_RESOURCE_MEMORY,       // Memory
        DRE_SYS_CONTAINER_RESOURCE_MEMSW         // Memory and swap
} dre_sys_resource_t;
```

1.3.2.2 dre_sys_status_t

Summary

The status of the container

Implementation

```
typedef enum {
        DRE_SYS_CONTAINER_STATUS_STOP = 11,      // The container is stopped.
        DRE_SYS_CONTAINER_STATUS_RUNNING,        // The container is running.
        DRE_SYS_CONTAINER_STATUS_SUSPENDED,      // The container is suspended.
        DRE_SYS_CONTAINER_STATUS_BUSY            // The container is busy.
} dre_sys_status_t;
```

1.3.2.3 dre_sys_type_t

Summary

The type of the container

Implementation

```
typedef enum {
        DRE_SYS_CONTAINER_TYPE_KVM = 100   // KVM
} dre_sys_type_t;
```

1.3.3Structures

1.3.3.1 dre_sys_data

Summary

The configuration of the container

Implementation

```
typedef struct {
        dre_sys_type_t type;            // The type of the container
        union {
                union {
                        int memory_size;        // The amount of the memory allocated by the container
                        char *base;             // The name of the Base Image (NULL for default)
                        char *rootfs;           // The name of the partition to be mounted instead of the base image
                } kvm;
        } udata;
} dre_sys_data;
```

The name of the Base Image created by "dre-sys commit" is available as the "base", which is the member of the above structure. If the Base Image is specified, the diff image based on it is created as the KVM Image of the System Container. If "rootfs" is specified, the OS installed on the specified partition can be used as the Guest OS.

1.3.3.2 dre_sys_handle

Summary

Handle of the container

Implementation

```
typedef struct {
        dre_sys_type_t type;        // The type of the container
        char *name;                 // The name of the container
} dre_sys_handle;
```

1.3.3.3 dre_sys_resource_value_t

Summary

Resource usage

Implementation

```
typedef struct {
        dre_sys_resource_t type;                // The type of the resource
        union {
```

```
        union {
            int parcent;                // CPU usage (%)
            char *cores;                // CPU core(s)
        } cpu;
        union {
            int usage_in_bytes;         // Memory usage (Bytes)
            int memsw_usage_in_bytes;   // Memory and swap usage (Bytes)
        } mem;
    } udata;
} dre_sys_resource_value_t;
```

1.3.4Type declarations

None

1.3.5Functions

1.3.5.1 dre_sys_bind_pid

Summary

Bind a process in the Host OS into a Control Group

Function interface

```
dre_return_t  dre_sys_bind_pid  (
    const dre_sys_handle *      h,
    const char *                cgroup,
    const int                   pid
);
```

Arguments

```
const dre_sys_handle *      h       // The handle of the container
const char *                cgroup  // The name of the Control Group
const int                   pid     // Process ID
```

Return value

```
0                           Succeeded
Not 0                       Failed (See 1.1.2.1 for detail.)
```

1.3.5.2 dre_sys_checkpoint

Summary

Generate a checkpoint of a System Container
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_sys_checkpoint (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
const char *                    tag  // The name of the checkpoint
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.3 dre_sys_close

Summary

Release a handle of the System Container

Function interface

```
dre_return_t dre_sys_close (
        const dre_sys_handle *          h
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.4 dre_sys_commit_snapshot

Summary

Create a new Base Image of the System Container from a snapshot
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_sys_commit_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag,
        const char *                    base
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
```

```
const char *                    tag  // The name of the snapshot
const char *                    base// The name of the Base Image
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.5 dre_sys_create_cgroup

Summary

Create a Control Group in the System Container

Function interface

```
dre_return_t dre_sys_create_cgroup (
        const dre_sys_handle *          h,
        const char *                    cgroup
);
```

Arguments

```
const dre_sys_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.6 dre_sys_create_container

Summary

Create a System Container

Function interface

```
dre_return_t dre_sys_create_container (
        const char *                    name,
        const dre_sys_data *            data
);
```

Arguments

```
const char *                    name  // The name of the container
const dre_sys_data *            data  // The configuration of the container
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.7 dre_sys_destroy_cgroup

Summary

Remove a Control Group in a container

Function interface

```
dre_return_t dre_sys_destroy_cgroup (
        const dre_sys_handle *          h,
        const char *                    cgroup
);
```

Arguments

```
const dre_sys_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.8 dre_sys_destroy_container

Summary

Remove a container

Function interface

```
dre_return_t dre_sys_destroy_container (
        const char *                    name
);
```

Arguments

```
const char *                    name  // The name of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.9 dre_sys_get_base_list

Summary

List the Base Images of the System Containers
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_sys_release_base_list()" after using this function.)

Function interface

```
dre_return_t dre_sys_get_base_list (
        dre_list_t *                    list
);
```

Arguments

```
dre_list_t *                    list   // The pointer to the list of the containers
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.10 dre_sys_get_checkpoint_list

Summary

List the checkpoints of a System Container
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_sys_release_checkpoint_list()" after using this function.)

Function interface

```
dre_return_t dre_sys_get_checkpoint_list (
        const dre_sys_handle *          h,
        dre_list_t *                    list
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
dre_list_t *                    list  // The pointer to the list of the checkpoints
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.11 dre_sys_get_container_list

Summary

List the System Containers on the system
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_sys_release_container_list()" after using this function.)

Function interface

```
dre_return_t dre_sys_get_container_list (
        dre_list_t *                    list
);
```

Arguments

```
dre_list_t *                    list   // The pointer to the list of the containers
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.12 dre_sys_get_pid

Summary

Get the Process ID of a System Container

Function interface

```
dre_return_t dre_sys_get_pid (
        const dre_sys_handle *      h,
        int *                       pid
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
int *                           pid   // Process ID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.13 dre_sys_get_resource

Summary

Obtain the information of the resource usage

Function interface

```
dre_return_t dre_sys_get_resource (
        const dre_sys_handle *      h,
        const int                   pid,
        dre_sys_resource_value_t *  resource
);
```

Arguments

```
const dre_sys_handle *          h         // The handle of the container
const int                       pid       // Process ID
dre_sys_resource_value_t *      resource  // Resource usage
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.14 dre_sys_get_snapshot_list

Summary

List the snapshots of a System Container
(Note: Since this function allocates the memory internally, you need to release the memory by "dre_sys_release_snapshot_list()" after using this function.)

Function interface

```
dre_return_t dre_sys_get_snapshot_list (
        const dre_sys_handle *          h,
        dre_list_t *                    list
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
dre_list_t *                    list  // The pointer to the list of PID
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.15 dre_sys_immigration_start

Summary

Start a container in the target host of immigration in order to perform immigration of the System Container

Function interface

```
dre_return_t dre_sys_immigration_start (
        const char *                    name,
        const int                       port
);
```

Arguments

```
const char *                    name  // The name of the container (the same as the migration source)
const int                       port  // The port number for the migration
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.16 dre_sys_load_snapshot

Summary

Restore the file system of a System Container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_sys_load_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
const char *                    tag   // The name of the snapshot
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.17 dre_sys_migration_start

Summary

Perform the migration of a System Container

Function interface

```
dre_return_t dre_sys_migration_start (
        const dre_sys_handle *          h,
        const char *                    target,
        const int                       port
);
```

Arguments

```
const dre_sys_handle *          h       // The handle of the container
const char *                    target  // The IP address of the migration target
const int                       port    // The port number for the migration
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.18 dre_sys_open

Summary

Get a handle of a System Container

Function interface

```
dre_sys_handle * dre_sys_open (
```

```
        const char *                    name
);
```

Arguments

```
const char *                    name   // The name of the container
```

Return value

NULL	Failed to get a handle
Not NULL	Pointer to the handle of the container

1.3.5.19 dre_sys_release_base_list

Summary

Release a list of the Base Images of the System Containers

Function interface

```
dre_return_t dre_sys_release_base_list (
        dre_list_t                  list
);
```

Arguments

```
dre_list_t                      list   // The list of the System Containers
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.20 dre_sys_release_checkpoint_list

Summary

Release a list of the checkpoints of a System Container

Function interface

```
dre_return_t dre_sys_release_checkpoint_list (
        dre_list_t                  list
);
```

Arguments

```
dre_list                        list   // The pointer to the list of the checkpoints
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.21 dre_sys_release_container_list

Summary

Release a list of the System Containers

Function interface

```
dre_return_t dre_sys_release_container_list (
        dre_list_t                      list
);
```

Arguments

dre_list_t	list	// The list of the System Containers

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.22 dre_sys_release_snapshot_list

Summary

Release a list of snapshots of a System Container

Function interface

```
dre_return_t dre_sys_release_snapshot_list (
        dre_list_t                      list
);
```

Arguments

dre_list	list	// The list of the snapshots

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.23 dre_sys_remove_base

Summary

Remove a Base Image of the System Containers

Function interface

```
dre_return_t dre_sys_remove_base (
        const char *                    base
);
```

Arguments

const char *	base // The name of the Base Image

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.24 dre_sys_remove_checkpoint

Summary

Remove a checkpoint of a System Container

Function interface

```
dre_return_t dre_sys_remove_checkpoint (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

const dre_sys_handle *	h // The handle of the container
const char *	tag // The name of the checkpoint

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.25 dre_sys_remove_snapshot

Summary

Remove a snapshot of a System Container
(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_sys_remove_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

const dre_sys_handle *	h // The handle of the container
const char *	tag // The name of the snapshot

Return value

0	Succeeded

Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.26 dre_sys_restart

Summary

Resume the running of a System Container from a checkpoint
(This function is available only while the container is running.)

Function interface

```
dre_return_t dre_sys_restart (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
const char *                    tag  // The name of the checkpoint
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.27 dre_sys_resume

Summary

Resume the running of a System Container which is suspended

Function interface

```
dre_return_t dre_sys_resume (
        const dre_sys_handle *          h
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.28 dre_sys_save_snapshot

Summary

Generate a snapshot of a System Container

(This function is available only while the container is stopped.)

Function interface

```
dre_return_t dre_sys_save_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

Arguments

```
const dre_sys_handle *          h    // The handle of the container
const char *                    tag  // The name of the snapshot
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.29 dre_sys_set_resource

Summary

Set a limitation on resource usage of a System Container

Function interface

```
dre_return_t dre_sys_set_resource (
        const dre_sys_handle *          h,
        const char *                    cgroup,
        dre_sys_resource_value_t *      resource
);
```

Arguments

```
const dre_sys_handle *          h         // The handle of the container
const char *                    cgroup    // The name of the Control Group
dre_sys_resource_value_t *      resource  // Resource usage
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.30 dre_sys_share

Summary

Share a System Container between the Host OS and the NFS server

Function interface

```
dre_return_t dre_sys_share (
        const dre_sys_handle *          h
```

```
);
```

Arguments

const dre_sys_handle *	h	// The handle of the container

Return value

0	Succeeded
0 以外	Failed (See 1.1.2.1 for detail.)

1.3.5.31 dre_sys_start

Summary

Start a System Container

Function interface

```
dre_return_t dre_sys_start (
    const dre_sys_handle *      h
);
```

Arguments

const dre_sys_handle *	h	// The handle of the container

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.32 dre_sys_status

Summary

Get the status of a System Container

Function interface

```
dre_return_t dre_sys_status (
    const dre_sys_handle *      h
);
```

Arguments

const dre_sys_handle *	h	// the handle of the container

Return value

DRE_SYS_CONTAINER_STATUS_STOP	The container is stopped
DRE_SYS_CONTAINER_STATUS_RUNNING	The container is running
DRE_SYS_CONTAINER_STATUS_SUSPENDED	The container is suspended
DRE_SYS_CONTAINER_STATUS_BUSY	The container is busy

Others	Failed (See 1.1.2.1 for detail.)

1.3.5.33 dre_sys_stop

Summary

Stop a System Container

Function interface

```
dre_return_t dre_sys_stop (
        const dre_sys_handle *          h
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.34 dre_sys_suspend

Summary

Suspend a System Container which is running

Function interface

```
dre_return_t dre_sys_suspend (
        const dre_sys_handle *          h
);
```

Arguments

```
const dre_sys_handle *          h     // The handle of the container
```

Return value

0	Succeeded
Not 0	Failed (See 1.1.2.1 for detail.)

1.3.5.35 dre_sys_unbind_pid

Summary

Unbind a process which is bound to the Control Group in a System Container

Function interface

```
dre_return_t dre_sys_unbind_pid (
```

```
    const dre_sys_handle *          h,
    const char *                    cgroup,
    const int                       pid
);
```

Arguments

```
const dre_sys_handle *          h       // The handle of the container
const char *                    cgroup  // The name of the Control Group
const int                       pid     // Process ID
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.36 dre_sys_unset_resource

Summary

Release the limitation of resource usage of a System Container

Function interface

```
dre_return_t dre_sys_unset_resource (
    const dre_sys_handle *          h,
    const char *                    cgroup,
    dre_sys_resource_value_t *      resource
);
```

Arguments

```
const dre_sys_handle *          h         // The handle of the container
const char *                    cgroup    // The name of the Control Group
dre_sys_resource_value_t *      resource  // Resource usage
```

Return value

```
0                               Succeeded
Not 0                           Failed (See 1.1.2.1 for detail.)
```

1.3.5.37 dre_sys_unshare

Summary

Stop the sharing of the System Container between the Host OS and the NFS server

Function interface

```
dre_return_t dre_sys_unshare (
    const dre_sys_handle *          h
);
```

Arguments

const dre_sys_handle *	h	// The handle of the container

Return value

0	Succeeded
0 以外	Failed (See 1.1.2.1 for detail.)

2 APIs for the D-Aware Applications

2.1 Lifecycle

We call the programs that use these APIs "D-Aware Applications".

D-Aware Applications need to call " daware_termination_register_callback()" when they start. "daware_termination_register_callback()" specifies the signal that the program can receive and the function which is called back when the program receives the signal.

D-Aware Applications are designed to work on the Application Containers, and they have two Process IDs called PID and VPID. PID means the Process ID assigned in the Host OS, and it can be referred to only from the Host OS. VPID means the Process ID assigned in the Application Container and it can be referred to only from the Application Container.

When "daware_termination_register_callback()" is called, the PID is correlated to the VPID, and they are registered to "/proc/daware".

D-Aware Applications can output the log in any timing by "daware_log()". The format of the log can be specified by " daware_log_set_format()".

D-Aware Applications need to call "daware_termination_unregister_callback()" when they terminate. This function unregisters the PID/VPID of the program from "/proc/daware".

2.2 How to use

To use these APIs in your program, include "daware.h" and link the library of D-Aware (libdaware.so or libdaware.a).

2.3 Constants and Macros

2.3.1 DLOG

Summary

output the log for the specified priority

Implementation

DLOG (priority, fmt, ...)

Arguments

priority	// The logical sum of the facility and the level
fmt, ...	// The same as "printf(fmt, ...)"

2.3.2 DLOG_EMERG/DLOG_ALERT/DLOG_CRIT/DLOG_ERR/DLOG_WARNING/DLOG_NOTICE/DLOG_INFO/DLOG_DEBUG

Summary

output the log in each priority

Implementation

DLOG_EMERG(fmt, ...)

```
DLOG_ALERT(fmt, ...)
DLOG_CRIT(fmt, ...)
DLOG_ERR(fmt, ...)
DLOG_WARNING(fmt, ...)
DLOG_NOTICE(fmt, ...)
DLOG_INFO(fmt, ...)
DLOG_DEBUG(fmt, ...)
```

Arguments

```
fmt, ...                // The same as "printf(fmt, ...)"
```

2.4 Structures

None

2.5 Type declarations

2.5.1 DTermination

Summary

The handle of the termination callback

Implementation

```
typedef long            DTermination;
```

2.6 Functions

2.6.1 daware_log

Summary

Output the log

Function interface

```
daware_log (
        int             priority,
        const char *    file,
        int             line,
        const char *    func,
        const char *    fmt, ...
);
```

Arguments

```
int                     priority        // The logical sum of the facility and the level
const char *            file            // File name (__FILE__)
```

```
int             line        // Line number (__LINE__)
const char *    func        // Function name (__func__)
const char *    fmt, ...    // The same as "printf(fmt, ...)"
```

Return value

None

2.6.2 daware_log_close

Summary

Close the log (Not required)

Function interface

```
void daware_log_close ();
```

Arguments

None

Return value

None

2.6.3 daware_log_open

Summary

Open the log (Not required)

Function interface

```
void daware_log_open (
    const char *    ident,
    int             option,
    int             facility
);
```

Arguments

```
const char *    ident       // The string to identify the program which outputted the log
                            // (This string is logged.)
int             option      // The same as "openlog()"
                            // (The 2nd argument of "openlog()" is the logical sum of the values
                            // written in 2.7.3.)
                            // Run "man syslog" for "openlog()".)
int             facility    // Specify the default value of the facility (See 2.7.1 for detail.)
```

Return value

None

2.6.4 daware_log_set_format

Summary

Specify the format of the log

Function interface

```
daware_log_set_format (
        const char *        logfmt
);
```

Arguments

```
const char *            logfmt              // The format of the log
                                            // The following tags are available

                        %faci%   :   Syslog facility{cron|daemon|main|user|...}
                        %lvl%    :   Syslog level{err|warning|notice|debug|...}
                        %time%   :   Date and time
                        %host%   :   Host name
                        %prog%   :   Program name
                        %pid%    :   PID
                        %msg%    :   Message
                        %file%   :   Filename
                        %line%   :   Line number
                        %func%   :   Function name

                    Example
                        daware_log_set_format("%time% %prog%[%pid%] %lvl%: %msg%");
```

Return value

None

2.6.5 daware_termination_register_callback

Summary

Register the termination callback

Function interface

```
DTermination * daware_termination_register_callback (
        int             signo,
        int             (*func)(int, void *)
        void *          arg
);
```

Arguments

```
int                     signo               // The signal
                                            // (This function calls the callback function
```

```
                                        // when the program receives the specified signal.)
int             (*func)(int, void *)    // The termination callback function
void *          arg                     // The arguments of the termination callback function
```

Return value

the handle of the termination callback	Succeeded
NULL	Failed

2.6.6 daware_termination_unregister_callback

Summary

Unregister the termination callback

Function interface

```
void daware_termination_unregister_callback (
    DTermination *      dterm
);
```

Arguments

```
DTermination *          dterm               // The handle of the termination callback
```

Return value

None

2.7 Others

2.7.1 facility

Summary

The type of the program which records the messages

Available values

LOG_AUTH
: The message of the security authentication (Not recommended. Use "LOG_AUTHPRIV" instead.)

LOG_AUTHPRIV
: The message of the security authentication (Private)

LOG_CRON
: The crock daemon ("cron" and "at")

LOG_DAEMON
: The system daemon with no facility

LOG_FTP
: The daemon of the FTP

LOG_KERN
: The message from the kernel (Cannot be created by the user processes.)

LOG_LOCAL0, LOG_LOCAL1… LOG_LOCAL7

Reserved for the local usage

LOG_LPR
: The sub system of the line printers

LOG_MAIL
: The sub system of the mails

LOG_NEWS
: The sub system of the USENET news

LOG_SYSLOG
: The internal messages from "syslogd(8)"

LOG_USER (default)
: The general messages of the user level

LOG_UUCP
: The sub system of the UUCP

2.7.2 level

Summary

the priority of the message

Available values

LOG_EMERG
: A panic condition. This is normally broadcast to all users.

LOG_ALERT
: A condition that should be corrected mmediately, such as a corrupted system database.

LOG_CRIT
: Critical conditions, such as hard device errors.

LOG_ERR
: Errors.

LOG_WARNING
: Warning messages.

LOG_NOTICE
: Conditions that are not error conditions, but that may require special handling.

LOG_INFO
: Informational messages.

LOG_DEBUG
: Messages that contain information normally of use only when debugging a program.

2.7.3 option

Summary

The values for "option" which is the 2nd argument of "openlog()"

Available values

LOG_CONS
: Write messages to the system console if they cannot be sent to syslogd(1M). This option is safe to use in daemon processes that have no controlling terminal, since syslog() forks before opening the console.

LOG_NDELAY

Open the connection to syslogd(1M) immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.

(Usually, the connection is established when the first message is recorded.)

LOG_NOWAIT

Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using SIGCHLD, since syslog() may otherwise block waiting for a child whose exit status has already been collected.

(GNU C libraries do not fork a child process. Therefore, this option is not valid for Linux.)

LOG_ODELAY (default)

Delay open until syslog() is called.

LOG_PERROR

Print to stderr as well. (This is not defined in "POSIX.1-2001".)

LOG_PID

Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork).