NAME

slurm_create_partition, slurm_create_reservation, slurm_delete_partition, slurm_delete_reservation, slurm_init_part_desc_msg, slurm_init_resv_desc_msg, slurm_reconfigure, slurm_shutdown, slurm_takeover, ,slurm_init_update_node_msg slurm_update_node, slurm_update_partition, slurm_update_reservation − Slurm administrative functions

SYNTAX

#include <slurm/slurm.h>

int **slurm_create_partition** (
update_part_msg_t ***update_part_msg_ptr*
);

int **slurm_create_reservation** (
resv_desc_msg_t ***update_resv_msg_ptr*
);

int **slurm_delete_partition** (
delete_part_msg_t ***delete_part_msg_ptr*
);

int **slurm_delete_reservation** (
reservation_name_msg_t ***delete_resv_msg_ptr*
);

void **slurm_init_front_end_msg** (
update_front_end_msg_t ***update_front_end_msg_ptr*
);

void **slurm_init_part_desc_msg** (
update_part_msg_t ***update_part_msg_ptr*
);

void **slurm_init_resv_desc_msg** (
resv_desc_msg_t ***update_resv_msg_ptr*
);

void **slurm_init_update_node_msg**(
update_node_msg_t ***update_node_msg_ptr*
);

int **slurm_reconfigure** ( );

int **slurm_shutdown** (
uint16_t *shutdown_options*
);

int **slurm_takeover** ( );

int **slurm_update_front_end** (
update_front_end_msg_t ***update_front_end_msg_ptr*
);

int **slurm_update_node** (
update_node_msg_t ***update_node_msg_ptr*
);

int **slurm_update_partition** (
update_part_msg_t ***update_part_msg_ptr*
);

int **slurm_update_reservation** (
resv_desc_msg_t ***update_resv_msg_ptr*
);

ARGUMENTS

shutdown_options
0: all slurm daemons are shutdown
1: slurmctld generates a core file

2: only the slurmctld is shutdown (no core file)

delete_part_msg_ptr
Specifies the pointer to a partition delete request specification. See slurm.h for full details on the data structure's contents.

delete_resv_msg_ptr
Specifies the pointer to a reservation delete request specification. See slurm.h for full details on the data structure's contents.

update_front_end_msg_ptr
Specifies the pointer to a front end node update request specification. See slurm.h for full details on the data structure's contents.

update_node_msg_ptr
Specifies the pointer to a node update request specification. See slurm.h for full details on the data structure's contents.

update_part_msg_ptr
Specifies the pointer to a partition create or update request specification. See slurm.h for full details on the data structure's contents.

update_resv_msg_ptr
Specifies the pointer to a reservation create or update request specification. See slurm.h for full details on the data structure's contents.

DESCRIPTION

slurm_create_partition Request that a new partition be created. Initialize the data structure using the **slurm_init_part_desc_msg** function prior to setting values of the parameters to be changed. Note: **slurm_init_part_desc_msg** is not equivalent to setting the data structure values to zero. A partition name must be set for the call to succeed. This function may only be successfully executed by user root.

slurm_create_reservation Request that a new reservation be created. Initialize the data structure using the **slurm_init_resv_desc_msg** function prior to setting values of the parameters to be changed. Note: **slurm_init_resv_desc_msg** is not equivalent to setting the data structure values to zero. The reservation's time limits, user or account restrictions, and node names or a node count must be specified for the call to succeed. This function may only be successfully executed by user root.

slurm_delete_partition Request that the specified partition be deleted. All jobs associated with the identified partition will be terminated and purged. This function may only be successfully executed by user root.

slurm_delete_reservation Request that the specified reservation be deleted. This function may only be successfully executed by user root.

slurm_init_update_front_end_msg Initialize the contents of an update front end node descriptor with default values. Note: **slurm_init_update_front_end_msg** is not equivalent to setting the data structure values to zero. Execute this function before executing **slurm_update_front_end**.

slurm_init_part_desc_msg Initialize the contents of a partition descriptor with default values. Note: **slurm_init_part_desc_msg** is not equivalent to setting the data structure values to zero. Execute this function before executing **slurm_create_partition** or **slurm_update_partition**.

slurm_init_resv_desc_msg Initialize the contents of a reservation descriptor with default values. Note: **slurm_init_resv_desc_msg** is not equivalent to setting the data structure values to zero. Execute this function before executing **slurm_create_reservation** or **slurm_update_reservation**.

slurm_init_update_node_msg Initialize the contents of an update node descriptor with default values. Note: **slurm_init_update_node_msg** is not equivalent to setting the data structure values to zero. Execute this function before executing **slurm_update_node**.

slurm_reconfigure Request that the Slurm controller re−read its configuration file. The new configuration parameters take effect immediately. This function may only be successfully executed by user root.

slurm_shutdown Request that the Slurm controller terminate. This function may only be successfully executed by user root.

slurm_takeover Request that the Slurm primary controller shutdown immediately and the backup

controller take over. This function may only be successfully executed by user root.

slurm_update_front_end Request that the state of one or more front end nodes be updated. This function may only be successfully executed by user root. If used by some autonomous program, the state value most likely to be used is **NODE_STATE_DRAIN**.

slurm_update_node Request that the state of one or more nodes be updated. Note that the state of a node (e.g. DRAINING, IDLE, etc.) may be changed, but its hardware configuration may not be changed by this function. If the hardware configuration of a node changes, update the Slurm configuration file and execute the **slurm_reconfigure** function. This function may only be successfully executed by user root. If used by some autonomous program, the state value most likely to be used is **NODE_STATE_DRAIN** or **NODE_STATE_FAILING**. The node state flag **NODE_STATE_NO_RESPOND** may be specified without changing the underlying node state. Note that the node's **NODE_STATE_NO_RESPOND** flag will be cleared as soon as the slurmd daemon on that node communicates with the slurmctld daemon. Likewise the state **NODE_STATE_DOWN** indicates that the slurmd daemon is not responding (and has not responded for an interval at least as long as the **SlurmdTimeout** configuration parameter). The node will leave the **NODE_STATE_DOWN** state as soon as the slurmd daemon communicates.

slurm_update_partition Request that the configuration of a partition be updated. Note that most, but not all parameters of a partition may be changed by this function. Initialize the data structure using the **slurm_init_part_desc_msg** function prior to setting values of the parameters to be changed. Note: **slurm_init_part_desc_msg** is not equivalent to setting the data structure values to zero. This function may only be successfully executed by user root.

slurm_update_reservation Request that the configuration of a reservation be updated. Initialize the data structure using the **slurm_init_resv_desc_msg** function prior to setting values of the parameters to be changed. Note: **slurm_init_resv_desc_msg** is not equivalent to setting the data structure values to zero. This function may only be successfully executed by user root.

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and the Slurm error code is set appropriately.

Exception: A successful slurm_create_reservation call returns a string containing the name of the reservation, in memory to be freed by the caller. A failed call returns NULL and sets the Slurm error code.

ERRORS

SLURM_PROTOCOL_VERSION_ERROR Protocol version has changed, re−link your code.

ESLURM_INVALID_NODE_NAME The requested node name(s) is/are not valid.

ESLURM_INVALID_NODE_STATE The specified state node state or requested node state transition is not valid.

ESLURM_INVALID_PARTITION_NAME The requested partition name is not valid.

ESLURM_INVALID_AUTHTYPE_CHANGE The **AuthType** parameter can not be changed using the **slurm_reconfigure** function, but all Slurm daemons and commands must be restarted. See **slurm.conf**(5) for more information.

ESLURM_INVALID_SCHEDTYPE_CHANGE The **SchedulerType** parameter can not be changed using the **slurm_reconfigure** function, but the **slurmctld** daemon must be restarted. Manual changes to existing job parameters may also be required. See **slurm.conf**(5) for more information.

ESLURM_INVALID_SWITCHTYPE_CHANGE The **SwitchType** parameter can not be changed using the **slurm_reconfigure** function, but all Slurm daemons and commands must be restarted. All previously running jobs will be lost. See **slurm.conf**(5) for more information.

ESLURM_ACCESS_DENIED The requesting user lacks authorization for the requested action (e.g. trying to delete or modify another user's job).

SLURM_PROTOCOL_SOCKET_IMPL_TIMEOUT Timeout in communicating with Slurm controller.

ESLURM_RESERVATION_ACCESS Requestor is not authorized to access the reservation.

ESLURM_RESERVATION_INVALID Invalid reservation parameter given, e.g. wrong name given.

ESLURM_INVALID_TIME_VALUE Invalid time value.

ESLURM_RESERVATION_BUSY Reservation is busy, e.g. trying to delete a reservation while in use.

ESLURM_RESERVATION_NOT_USABLE Reservation not usable, e.g. trying to use an expired reservation.

EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <slurm/slurm.h>
#include <slurm/slurm_errno.h>

int main (int argc, char *argv[])
{
	update_node_msg_t	update_node_msg;
	update_part_msg_t	update_part_msg;
	delete_part_msg_t	delete_part_msg;
	resv_desc_msg_t		resv_msg;
	char			*resv_name = NULL;

	if (slurm_reconfigure ( )) {
		slurm_perror ("slurm_reconfigure error");
		exit (1);
	}

	slurm_init_part_desc_msg ( &update_part_msg );
	update_part_msg.name = "test.partition";
	update_part_msg.state_up = 0;  /* partition down */
	if (slurm_create_partition (&update_part_msg)) {
		slurm_perror ("slurm_create_partition error");
		exit (1);
	}

	update_part_msg.state_up = 1;  /* partition up */
	if (slurm_update_partition (&update_part_msg)) {
		slurm_perror ("slurm_update_partition error");
		exit (1);
	}

	delete_part_msg.name = "test.partition";
	if (slurm_delete_partition (&delete_part_msg)) {
		slurm_perror ("slurm_delete_partition error");
		exit (1);
	}

	slurm_init_update_node_msg (&update_node_msg);
	update_node_msg.node_names = "lx[10−12]";
	update_node_msg.node_state = NODE_STATE_DRAIN ;
	if (slurm_update_node (&update_node_msg)) {
		slurm_perror ("slurm_update_node error");
		exit (1);
	}

	slurm_init_resv_desc_msg ( &resv_msg );
	resv_msg.start_time = time(NULL) + 60*60;  /* One hour from now */
	resv_msg.duration = 720;  /* 12 hours/720 minutes */
	resv_msg.node_cnt = 10;
	resv_msg.accounts = "admin";
	resv_name = slurm_create_reservation (&resv_msg);
	if (!resv_name) {
		slurm_perror ("slurm_create_reservation error");
		exit (1);
	}
```

```
        free(resv_name);
        exit (0);
}
```

NOTE

These functions are included in the libslurm library, which must be linked to your process for use (e.g. "cc −lslurm myprog.c").

COPYING

SEE ALSO

scontrol(1), **slurm_get_errno**(3), **slurm_init_job_desc_msg**(3), **slurm_perror**(3), **slurm_strerror**(3), **slurm.conf**(5)