

# User-Level Compute Block Control

1. User-Level Compute Block Control
  1. Requesting a Non-Booted Block
  2. Discovering Allocated Bootable Blocks
  3. Booting a Block
  4. Running on a Block
  5. Freeing a Block
  6. Rebooting a Block
  7. Simple Multiboot Script

As of Cobalt 0.99.19, the Blue Gene/Q platform has been extended to provide users the ability to allocate and boot blocks within a Cobalt resource allocation within their script. This is a change from Blue Gene/P, and is now necessary as *runjob* only controls the execution of a job on a previously booting block, unlike *mpirun*, which encapsulated both the boot and execution steps. While this change makes certain parts of the system more stable, it has necessitated the creation of additional utilities within Cobalt to expose this control to the user in a safe and robust manner.

## Requesting a Non-Booted Block

When executing a script mode job, the user may request that Cobalt start the script without booting the block prior to execution. This is useful if one wishes to take the block allocated by Cobalt and then run on a number of blocks contained within the allocated block, for instance: a user that wishes to run a set of concurrent single-midplane jobs within a four midplane block. To do this submit the job with the `--disable_preboot` flag, for example:

```
qsub -n 2048 -t 6:00:00 -A Allocation --mode script --disable_preboot ./run_everything.sh
```

When the scheduler starts the job, the script will immediately run and the system will not try to boot the allocated block. `--disable_preboot` is ignored for non-script-mode jobs.

## Discovering Allocated Bootable Blocks

A list of blocks that are available to run within your job can be obtained through the `get-bootable-blocks` utility. This is included in the Cobalt distribution. This command takes a parent block as an argument, and also accepts `--size` and `--geometry` flags as constraints on the blocks returned. A block will only be listed if all of its resources are free to be booted, and that they are entirely contained within the specified block. Any block that is currently initialized or that is in a non-bootable state due to a software or a hardware failure will not be listed. If one wanted to list every 2048 node block within their allocated block, for instance, within your script you may specify:

```
get-bootable-blocks --size 2048 $COBALT_PARTNAME
```

Additionally the block must be defined in the control system, and at this time, it must be managed by Cobalt, and added to the list of blocks via `partadm -a`.

## Booting a Block

A block may be booted within a script via the `boot-block` command. If no options are specified then Cobalt will attempt to boot the block defined by the `$COBALT_PARTNAME` environment variable. A different block may be specified via the `--block` option. This block must be allocated to the user the script is running as within Cobalt, and must be a part of a currently running job owned by that user. Once initiated, the utility will complete with an exit status of 0 once the specified block is booted and ready to run jobs.

## Running on a Block

Once a block is booted, you may invoke `runjob` with the booted block as a target. For information on using IBM's `runjob` utility please consult the chapter 6 of the *IBM Blue Gene Solution Blue Gene/Q System Administration* Redbook that can be found at [here](#). Additionally you may wish to consult the `runjob` manpage installed on your Blue Gene/Q system.

## Freeing a Block

If you need to recover the resources of a booted block within your script, for instance to reboot a 2048 node part of your allocated block as a 1024 node block and a 512 block, you may free the currently booted resources of a block by using the `--free` flag. While not strictly necessary, it is advised that you free the blocks you have manually booted during your script as a part of any clean-up actions your script job may take.

## Rebooting a Block

In a script-mode job, you may run multiple jobs against a booted block with multiple invocations of `runjob`. The behavior of `runjob` is equivalent of Blue Gene/P `mpirun`'s `-nofree` flag. If your job does not exit cleanly, it may leave compute nodes in a "dirty" state, and the control system may report a "SoftwareFailure" state on the nodes of the aborted job, indicating that the block must be rebooted prior to another successful invocation of `runjob` on those resources. If this is the case the block may be cycled by running:

```
boot-block --block <name-of-failed-block> --reboot
```

This will free and reboot the block, returning the block in a state where you may run jobs on the block again. Note that this will not work, or exit successfully if the block's hardware has anything other than a "SoftwareFailure" error. Other error states indicate error states with the underlying hardware.

## Simple Multiboot Script

Here is an example of a script that may be used with the `--disable_preboot` flag to boot all of the blocks of a particular size within an allocated block. Do note that booting a block may render other blocks unbootable due to consuming wiring resources needed for other blocks of that size.

```
#!/bin/bash
BLOCKS=`get-bootable-blocks $COBALT_PARTNAME`
for BLOCK in $BLOCKS
```

```
do
    boot-block --block $BLOCK &
done
wait

for BLOCK in $BLOCKS
do
    runjob --block $BLOCK : `pwd`/my_binary &
done
wait

for BLOCK in $BLOCKS
do
    boot-block --block $BLOCK --free &
done
wait
```