Contents

1 About this Document 11
  1.1 State and Completeness . . . . . . . . . . . . . . . . 11
  1.2 Audience and Purpose . . . . . . . . . . . . . . . . . . . . 11
  1.3 Document Organization . . . . . . . . . . . . . . . . . . . . 11
  1.4 Related Documents . . . . . . . . . . . . . . . . . . . . . . 12
  1.5 Typographic Conventions . . . . . . . . . . . . . . . . . . . 13

2 What is OpenCCS 15
  2.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.2 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    2.2.1 Command Naming Schema . . . . . . . . . . . . . . . 17
    2.2.2 Request States . . . . . . . . . . . . . . . . . . . . . 17
  2.3 Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    2.3.1 Planning Based . . . . . . . . . . . . . . . . . . . . . 18
    2.3.2 Authentication . . . . . . . . . . . . . . . . . . . . . 19
    2.3.3 Authorization . . . . . . . . . . . . . . . . . . . . . . 19
    2.3.4 User Roles . . . . . . . . . . . . . . . . . . . . . . . 20
    2.3.5 Job Trace . . . . . . . . . . . . . . . . . . . . . . . . 20
    2.3.6 Reliability and Fault Tolerance . . . . . . . . . . . . 20
    2.3.7 Customizing . . . . . . . . . . . . . . . . . . . . . . . 22
  2.4 Differences . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
    2.4.1 Validity . . . . . . . . . . . . . . . . . . . . . . . . . 23
    2.4.2 Limits . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
    2.4.3 FreePools . . . . . . . . . . . . . . . . . . . . . . . . 24

3 New Features 25
  3.1 New Features in OpenCCS 0.9.8-5 . . . . . . . . . . . . . 25
  3.2 New Features in OpenCCS 0.9.8-4 . . . . . . . . . . . . . 25
  3.3 New Features in OpenCCS 0.9.8-3 . . . . . . . . . . . . . 26
  3.4 New Features in OpenCCS 0.9.8-2 . . . . . . . . . . . . . 26
  3.5 New Features in OpenCCS 0.9.8-1 . . . . . . . . . . . . . 26
  3.6 New Features in OpenCCS 0.9.8 . . . . . . . . . . . . . . . 26
  3.7 New Features in OpenCCS 0.9.7-5 . . . . . . . . . . . . . 26
CONTENTS

3.8 New Features in OpenCCS 0.9.7-4 . . . . . . . . . . . . . . . 27
3.9 New Features in OpenCCS 0.9.7 . . . . . . . . . . . . . . . . 28
3.10 New Features in OpenCCS 0.9.6 . . . . . . . . . . . . . . . . 28

4 Getting Started with OpenCCS 29
4.1 User Roles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
  4.1.1 User-Role . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
  4.1.2 Manager-Role . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
  4.1.3 Administrator-Role . . . . . . . . . . . . . . . . . . . . . . . . 30
4.2 Principle Usage . . . . . . . . . . . . . . . . . . . . . . . . . . 30
4.3 Request Types and Identifier . . . . . . . . . . . . . . . . . . . 31
4.4 Request States . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
4.5 OpenCCS Command Line Interfaces . . . . . . . . . . . . . . . 32
  4.5.1 CLI Argument Parsing . . . . . . . . . . . . . . . . . . . . . . . 32
  4.5.2 CLI Default Values . . . . . . . . . . . . . . . . . . . . . . . . 33
4.6 Setup the OpenCCS Environment . . . . . . . . . . . . . . . . . 35
4.7 Message of the Day . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
4.8 Job Priorities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.9 Job Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
  4.9.1 Boot Node Environment Variables . . . . . . . . . . . . . . . 37
  4.9.2 The OpenCCS Node File . . . . . . . . . . . . . . . . . . . . . 39

5 Resources 41
5.1 Resource Categories . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
5.2 Resource Formats . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
5.3 Built-In Resources . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.4 Specifying Resources . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
  5.4.1 Syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
  5.4.2 Resource Set / Chunk Specification . . . . . . . . . . . . . . . . 45
  5.4.3 Placement Specification . . . . . . . . . . . . . . . . . . . . . . . 46
  5.4.4 Job-Wide Resources . . . . . . . . . . . . . . . . . . . . . . . . . 47
  5.4.5 Specifying Resource Values . . . . . . . . . . . . . . . . . . . . . 47
  5.4.6 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
5.5 Resource Assignment to Jobs . . . . . . . . . . . . . . . . . . . . . . 49
  5.5.1 Default and Force Resources . . . . . . . . . . . . . . . . . . . . . 49
  5.5.2 Matching Unset Resources . . . . . . . . . . . . . . . . . . . . . 49
5.6 Resources and Limits . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
  5.6.1 Limit Enforcement . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
  5.6.2 Limits on Exclusively Used Nodes . . . . . . . . . . . . . . . . . 51
  5.6.3 Resource Credits . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
## 6 Submitting Jobs

6.1 Introduction ........................................... 53
6.2 Script Jobs ............................................ 53
   6.2.1 Submitting a Job Script ......................... 53
   6.2.2 Changing the Job’s CCS Directive ............... 54
   6.2.3 Passing Arguments to Job Scripts ................ 55
   6.2.4 Jobs Without a Job Script ....................... 55
6.3 Interactive Jobs ....................................... 55
   6.3.1 The Virtual Terminal .............................. 55
6.4 Background Jobs ....................................... 56
6.5 Job Submission Options ............................... 56
   6.5.1 Time Related Attributes ......................... 57
   6.5.2 Request Name .................................... 58
   6.5.3 Email Notification ............................... 58
   6.5.4 Email Recipients ................................. 59
   6.5.5 Job Notification ................................ 59
   6.5.6 Input, Output and Error Files ................... 60
   6.5.7 Job Trace File .................................. 61

## 7 Predicting Job Start Times

7.0.1 Resource Syntax ................................. 63
    7.0.2 Iterator Syntax ................................ 63
    7.0.3 Examples ...................................... 65

## 8 Checking Job and System Status

8.1 Schedule Status ...................................... 67
   8.1.1 Summary ....................................... 67
   8.1.2 Job Distribution ............................... 68
   8.1.3 Filtering the Data ............................. 68
   8.1.4 Formatting the Output ......................... 69
   8.1.5 Examples ...................................... 71
8.2 System Status ....................................... 71
   8.2.1 Node Status .................................... 71
   8.2.2 Available Workers ............................. 74
   8.2.3 Allocatable Resources ......................... 75
   8.2.4 FreePools ..................................... 76
8.3 Group / User Related Infos ......................... 77
   8.3.1 Group Membership ............................. 77
   8.3.2 Limits and Privileges ......................... 77
   8.3.3 Default and Force Values ..................... 78
8.4 Request Status ..................................... 79
Chapter 1

About this Document

1.1 State and Completeness
This document is a general user manual. Therefore, it does not always reflect the state of the installed release.

1.2 Audience and Purpose
This manual is intended for people using OpenCCS.

1.3 Document Organization

- Chapter 2 gives an overview about the OpenCCS architecture, its features, and the differences to queueing based systems. If you are impatient, skip it.
- Chapter 3 describes the new features of the current OpenCCS release.
- Chapter 4 describes how to get started with OpenCCS.
- Chapter 5 describes the resources OpenCCS manages and how to specify them.
- Chapter 6 describes how to submit jobs to OpenCCS.
- Chapter 8 describes how to get information about the status of nodes, systems, and jobs.
- Chapter 9 describes how to manage OpenCCS jobs after submission.
- Chapter 10 describes how to reserve resources and how to use them.
- Chapter 11 describes how to use job arrays.
• A description of OpenCCS specific items can be found in the glossary (Appendix M).

1.4 Related Documents

• http://openccs.eu: online documentation.

• The OpenCCS man pages
1.5 Typographic Conventions

This document uses the following typographic conventions:

- Important items are marked by a small box at the margin (e.g. FOO). They also appear in the index.

- Function and variable names, examples of screen output, names of directories, files, and file contents appear in monospace type.

In addition the following symbols appear in command syntax definitions, both in the documentation and in OpenCCS online help statements. When issuing a command, do not type these symbols.

Square brackets [ ]
Surround optional parameters.

Angle brackets < >
Surround user-supplied values in OpenCCS commands.

Pipe symbol |
In a command syntax statement separates mutually exclusive values for an argument.

Percent sign %
Represents the regular command shell prompt. Some operating systems possibly use a different character for this prompt.

Number sign #
Represents the command shell prompt for the local superuser root. Some operating systems possibly use a different character for this prompt.
Chapter 2

What is OpenCCS

2.1 Overview

The Computing Center Software is a completely planning based Workload Management system (WLM). It has been initially designed to serve two purposes: For HPC users it should provide a homogeneous interface to a pool of different HPC systems. For system administrators it should provide a means for describing, organizing, and managing HPC systems that are operated in a computing center. Hence the name “Computing Center Software”, CCS for short. CCS is released under the terms of the GNU General Public License. Therefore, it is also called OpenCCS.

2.2 Architecture

OpenCCS consists of several modules, which may run on multiple hosts to improve the response time. OpenCCS is based on events (e.g., timers, messages, signals), and the communication is stateless and asynchronous. The modules are multi-threaded but single-tasked. The submission syntax and resource description is strongly PBSPro compatible to ease the integration of commercial applications.

Figure 2.1 depicts the OpenCCS modules (described below) and the event handling.

User Interface(UI)

Provides a single access point to one or more systems via a Command Line interface.

Access Manager(AM)

Manages the user interfaces and is responsible for authentication, authorization, and accounting.

Planning Manager(PM)

Schedules and maps the user requests onto the machine.
CHAPTER 2. WHAT IS OPENCCS

Machine Manager (MM)
Provides machine specific features like node management or job controlling.

Island Manager (IM)
Provides OpenCCS internal name services and watchdog facilities to keep the island in a stable condition.

Operator Shell (OS)
Is the main interface for system administrators to control OpenCCS, e.g. by connecting to the system modules.

Node Session Manager (NSM)
Runs with root privileges on each node managed by OpenCCS. The NSM is responsible for node access and job controlling. At allocation time, the NSM starts an EM for each job.

Execution Manager (EM)
Establishes the user environment (UID, shell settings, environment variables, etc.) and starts the application.
2.2. Command Naming Schema

- All user commands start with **ccs**.
- All graphical user commands start with **ccsx**.
- All administrator commands start with **ccs-**.
- All graphical administrator user commands start with **ccs_x**.

2.2.2 Request States

A request (i.e., a reservation or a job) in OpenCCS may enter these states.

**PLANNING**  The scheduler is assigning a start time to the request.

**PLANNED**  The scheduler has successfully assigned a start time to the request.

**ALLOCATING**  The resources assigned to the request are currently initialized.

**ALLOCATED**  The resources assigned to the request are ready for work. The following substates are possible for jobs:

- **PREPROC**  Preprocessing
- **EXEC**  Job is running
- **ACCOUNTED**  Job has accounted
- **POSTPROC**  Postprocessing
- **KILL**  Job is killed

**STOPPING**  The resources assigned to the request are being released.

**STOPPED**  The resources are released. Job related data will be available for users for an administrator defined interval (often 30m) before OpenCCS removes the job completely. The command **ccstracejob** may be used to print job data if the job has been removed in the CCS runtime database.

**WAITING**  The request is in a “waiting room” because OpenCCS is not able to assign a start time. This state is reached for two reasons:

1. If resources become unavailable while a request is in state **PLANNED**, In such a case the request keeps its place in the schedule because the resources may become available again before switching to state **ALLOCATING**. If not
enough resources available, OpenCCS tries to replan the request. If this is not possible, the request state switches to WAITING. If the resources become available again, OpenCCS automatically tries to replan waiting requests.

2. The job has been held via ccsalter. Refer to 9.2. The following substates are possible for held jobs:

- **HOLD_BY_USER**
  The job has been held by the job owner.

- **HOLD_BY_GROUP_MANAGER**
  The job has been held by a group manager.

- **HOLD_BY_GROUP_ADMIN**
  The job has been held by the OpenCCS administration.

2.3 Features

2.3.1 Planning Based

*Queuing based WLMs* mainly try to utilize currently free resources with waiting resource requests. Resource planning for waiting requests is often not done/possible.

*Planning based systems* in contrast plan for the present and future. Planned start times are assigned to requests and a schedule about the future resource usage is computed and made available to the users. This approach has some important implications:

- There are no explicit queues in OpenCCS.

- Users are supposed to specify the expected runtime of their requests. If no duration is specified, OpenCCS assigns a site specific one.

- Authorization and limitation is attached to groups and / or users.

- OpenCCS requires the users to specify the expected runtime of their requests.

- Entities (e.g., user, group, resource, limit, or FreePool) may have a validity. If the validity is exceeded, the entity is disabled.

**Advance Reservations** All users can reserve resources for a given period of time in advance. After planning the request, the user is guaranteed access to the reserved resources. Jobs can then be submitted to the reserved resources. The administrator may deny the reservation privilege. Refer to chapter 10 for more detailed information.
2.3. FEATURES

Specifying Start Times  One can specify when exactly a job should start or the time after which the job is eligible for execution. Refer to section 6.5.1 for more detailed information.

Deadlines  Batch jobs can be submitted with a deadline. Once a job has been accepted, OpenCCS guarantees the job to be completed at (or before) the specified time. Refer to section 6.5.1 for more detailed information.

Showing Planned Start Times  The OpenCCS command line interface shows the estimated start time of interactive requests directly after the submitted request has been planned. In case of an interactive job, this output is updated whenever the schedule changes significantly. Refer to chapter 8 for more detailed information.

Limits  Limits are related to resources and a consumer and have a validity. Resource is the name of a resource class (e.g. ncpus). Consumer is either a user or a group. User limits may overwrite group limits.

Job Notification  X minutes before the maximum runtime ends, OpenCCS may send a signal to the job or may start an executable. Refer to 6.5.5 for more detailed information.

2.3.2 Authentication

When requesting resources, users must specify a group name. A group may be a UNIX/LDAP group or an inner OpenCCS group. Users may be member of several groups at the same time. OpenCCS checks if the user is member of the group and if the validity of group and user is OK.

2.3.3 Authorization

Authorization in OpenCCS is group and user based. One has to specify a group at submit time. Limitations and privileges can be granted to either a whole group or to group members. Examples are:

- Privileges (e.g., allocate-/reserve/change resource requests)
- Maximum number of concurrently used resources
- Maximum allowed time of resource usage
2.3.4 User Roles

OpenCCS allows certain privileges based on what role a person has. OpenCCS recognizes only three roles, and all those using OpenCCS must be assigned at least one of these roles. The roles are User, Manager, and Administrator. Roles are assigned by OpenCCS-Owner only (via \%ISLAND\_AAL\_FILE). No roles can be added, and roles cannot be modified; the function of roles is hardcoded in the servers.

Resource Credit Management  The operator may assign resource "credits" for any consumable resource to a consumer. (e.g. 200 cpu hours or 100 GPU hours). After consuming the credit, OpenCCS denies resource reservation or job allocation.

2.3.5 Job Trace

A user may specify a path to a trace file at submission time. OpenCCS then writes all state changes into this file. Example 2.3.1 depicts such a file. Additionally, using ccstracejob users may print log and accounting data of completed jobs (H).

2.3.6 Reliability and Fault Tolerance

Before allocating a node, OpenCCS may check its integrity (e.g. disk space, network, memory, processes, ...). If the check fails OpenCCS tries to fix the problem. If this is not possible, the allocation fails, the user is notified, and the operator gets an e-mail describing what went wrong. Additionally, OpenCCS may perform a post-processing after each job.

Automatic Disabling of Defect Nodes  On workstation clusters, all nodes are frequently checked. If OpenCCS detects that a node of the managed machine does not answer in time, it tries to reconnect. If this is not possible, this node is automatically marked offline. Concerned jobs are stopped and OpenCCS sends an email to the user and the operator describing the problem.

Alive Checks  All OpenCCS modules are sending "heartbeat" messages to their communication partners. When a OpenCCS module detects a breakdown while communicating with another module, it closes the connection to this module and requests the IM to re-establish the link.

For recovery, each OpenCCS module periodically saves its state. At boot time the modules read their information and synchronize with their communication partners.
Example 2.3.1: A job trace file

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event Description</th>
<th>Result Code</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-04-11</td>
<td>14:37:14</td>
<td>REQUEST_PLANNED</td>
<td></td>
<td>Planned start is: 22:00:00</td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:38:44</td>
<td>REQUEST_ALTER</td>
<td>+0200</td>
<td>Altered: start,,1,nrset,ncpus=1:vmem=1m,2</td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:38:45</td>
<td>RESOURCE_AVAILABLE</td>
<td></td>
<td>Nodes: kel-ubuntu910</td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:38:45</td>
<td>JOB-START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:41:47</td>
<td>JOB-OVER-LIMIT</td>
<td>+0200</td>
<td>Killing the job: vmem 3236k exceeded limit 1024k on host kel-ubuntu910</td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:41:49</td>
<td>JOB-KILL</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2012-04-11</td>
<td>14:41:49</td>
<td>RESOURCE_RELEASED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Exec-host : kel-ubuntu910
- Exit Status : terminated due to signal 1(SIGHUP)
- Used CPU-time : 1896k
- Used mem : 7696k
- Used walltime : 7s

job terminated
**Virtual Terminal** Interactive applications are not stopped when the connection between the user interface and the application has been broken (e.g. due to a frontend failure). STDIN, STDOUT and STDERR will be redirected to specified files. It is possible to re-bind to a running interactive session.

### 2.3.7 Customizing

**Worker Concept** Cluster systems comprise nodes with full operating system capabilities and software packages like debuggers, performance analyzers, numerical libraries, and runtime environments. Often these software packages require specific pre and post-processing. OpenCCS supports this with the so-called *worker concept*. s are tools to start jobs under specific runtime environments. They hide specific procedures (e.g. starting of a daemon or setting of environment variables) and provide a convenient way to start and control programs. A worker’s behavior is specified in a configuration file.

The *gaussian worker* may serve as an example to illustrate what can be done with a worker: It checks whether Gaussian should run in parallel. If so the worker copies the gaussian worker file and inserts the necessary LINDA directives to start the job in parallel. Before starting the job, a host file is generated and the user’s environment is extended by all environment variables required by gaussian. After the job has terminated, the worker cleans up all.

**Adapting to the Local Environment** Since OpenCCS is able to manage heterogeneous systems it is possible that the process environment may differ on the UI host and on the compute node. OpenCCS copes with problems like this by modifying the process environment of an application before starting it. Environment variables like `PATH` will be explored and modified with respect to the host name. This is done in the UI and in the EM.

### 2.4 Differences

OpenCCS needs a shared file system. It does not stage-in / stage-out executables or in-/out-put data to or from nodes. All nodes, frontends, and management nodes have to be connected to a shared file system.

Since OpenCCS has no explicit queues things like default values, resource limits, or authentication are handled differently. In the following the terms validity, limit, and FreePool are explained in more detail.
2.4. DIFFERENCES

2.4.1 Validity

Specifies the time period in which an entity is valid. It can be given as:

**Date**

Specifies an absolute end-date. Given as Datetime.
E.g. 15:00:31.12.2015.
If time is not given, it is assumed as 23:59:59.

**Date1 - Date2**

Specifies an absolute start- and an absolute end-date, both given as Datetime.
E.g. 15:00:01.01.2012 - 31.12.2015.
If time1 is not given, it is assumed as 00:00:00 If time2 is not given, it is assumed as 23:59:59

**Cron**

Specifies repeated intervals. Given as Cron. If Cron is given, Date1 or Date1 - Date2 may specify the interval in which the cron is valid.

2.4.2 Limits

If a consumer has no limit assigned this means all resources are available forever. A consumer must not have more than one limit per resource. Exception of the rule: A '*' limit can be overwritten by a specific resource limit. A limit consists of the following items:

**validity**

The validity period of a limit (section 2.4.1).
Defaults to no validity which means limit is always valid.

**items**

The maximum number of allocatable items.
Syntax: `<min[/max]>`
min is of type size and max specifies the percent of currently available items. If both given, OpenCCS takes the maximum of min,max. Example: 30/45% denotes a limit 30 items or 45% of the available items. The default-value is no limit.

**duration**

The maximum timespan the resource may be used.
Default value is no limit.

**area**

The maximum area in the complete schedule.
Given as `duration*count`. The area may be given as an absolute value or as float value. In the latter case the limit is computed at runtime:

`limit ::= <value> * duration-limit * item-limit`

If the limit is exceeded, the job will be accepted but placed in the waiting room (i.e., not planned). Such jobs will be re-planned as soon as the limit allows. Default value is no area limit.
credit Defines a resource credit given as area. Default value is infinite credit.

If a time dependent limit is exceeded, the affected request will be scheduled to a later or earlier slot (depending on the request type).

### 2.4.3 FreePools

FreePools are like limits, but describe the conditions for resources to be kept free (i.e., they constrain the access to resources). A FreePool consists of the following items:

- **NAME** The name of the FreePool.
- **WHAT** The name of the resource to be kept free. ‘*’ means all resources.
- **QUANTITY** How many of WHAT should be kept free. The maximum number of allocatable items.
  Syntax: \(<\text{min}[/\text{max}]>\)
  \(\text{min}\) is of type size and \(\text{max}\) specifies the percent of currently available items. If both given, OpenCCS takes the maximum of \(\text{min}, \text{max}\). Example: 30/45\% denotes 30 items or 45\% of the available items.
- **ALLOWED** Specifies the conditions to get access to WHAT.
  4 conditions which may be logically connected
  1. GROUP: A comma separated lists of groups.
  2. USER: A comma seperated lists of accounts.
  3. COUNT: The maximal number of requested items.
  4. DURATION: The maximal allowed duration.
- **VALIDITY** The validity period of the FreePool (section 2.4.1).
  Defaults to no validity which means the FreePool is always valid.

Two examples may show what can be done with a FreePool:

- Keep free 20 percent of the available CPUs but at minimum 10 CPUs for jobs which request less than 4 CPUs for less than 1h.
- Keep all nodes owned by X free for the groups \(P1\) and \(P2\) and user \(alice\). All others may use the nodes only for a maximum of 2 hours.
Chapter 3

New Features

3.1 New Features in OpenCCS 0.9.8-5

- ccstracejob. Allows users to trace log- and accounting-data of completed jobs. Refer to H.
- Manager Role. Managers have more privileges than users, and less privileges than Administrators. Managers are assigned to groups within CCS. A group can have more than one manager. Refer to 4.1 for more details.
- Hold / resume. A user may put a request in state hold, which means it is ignored in planning. Altering to state resume, will enable planning again. Refer to 9.2. for more details.
- Exclude hosts from mapping (place=ignore=h1) Refer to 5.4.3.
- Background Jobs. These are jobs, which have assigned the lowest priority. Refer to 6.4 for more details.

3.2 New Features in OpenCCS 0.9.8-4

- The OpenCCS administration may assign credits to a consumer (group or user) for any consumable resource. If a credit is consumed, OpenCCS will (depending on the specified policy):
  - Deny resource reservation or job allocation for all jobs requesting such resources.
  - Set the job priority to the “background job” priority.
  - Do nothing.
Users can inspect their current credits and the active policy. Refer to 5.6.3 for more details.
CHAPTER 3. NEW FEATURES

3.3 New Features in OpenCCS 0.9.8-3

- The OpenCCS administration may enable a policy for dynamic partitioning of nodes to enforce the mapping of "small jobs" to specific nodes to increase the likelihood for large jobs to start.

- New `ccsinfo` format specifier `%a`. It shows job attributes. Each printed letter stands for a attribute. Refer to section 8.1.4 for more details.

- Predicting which resources are when available. Refer to chapter 7.

3.4 New Features in OpenCCS 0.9.8-2

- `place=group=<resource>`. Refer to 5.4.3.

- Node specific access control lists (ACLs). Refer to Appendix:L.

3.5 New Features in OpenCCS 0.9.8-1

- Support for multiple guest devices on a host (GPU, PHI, FPGA, ...). `NSM` maps the jobs to the guest devices and `EM` and `ccsattach` set the appropriate environment variables (e.g., `CUDA_VISIBLE_DEVICES`).

- Island specific uirc files. Refer to 4.5.2.

- Admin may change the enforcement of job limits to none, all, or hybrid.

3.6 New Features in OpenCCS 0.9.8

- Job-Arrays to run almost identical jobs. Refer to section 11 for more details.

- `ccsinfo -s --fmt=%c` shows core usage and it's efficiency. Refer to section 8.1.4 for more details.

- `ccsinfo <reqID>` shows resource usage for each used node.

3.7 New Features in OpenCCS 0.9.7-5

- Automatic setting of job-priorities related to resource requirements. Priorities are divided in 4 classes: best-effort, deadline, fix start time, and reservations. In each class the priority may be increased if
  - the job requests "expensive resources" (i.e. chunks with many cores),
– the job is parallel (i.e., requests a lot of cores),
– the job requests a high priority resource specified by the administration (i.e., a Tesla card).

At backfill the jobs are processed by their priority in descending order. At replan (e.g., due to a resource outage or an admin command) jobs with lower priority are displaced. The priority of displaced jobs is incremented to avoid starving.

- Node specific minimum resources.
The administrator may specify minimum resource amounts and/or a maximum duration which have to be requested at least by a job to be mapped on that node.

- ccsinfo -s --fmt=%q shows priority. Refer to section 8.1.4 for more details.
- ccsinfo -n --fmt=%p shows node properties. Refer to section 8.2.1 and appendix L for more details.
- ccsinfo -n --fmt=%m shows the node specific minimum resource limits. Refer to section 8.2.1 and appendix L for more details.

- Faster scheduler.

3.8 New Features in OpenCCS 0.9.7-4

- Matching Unset Resources.
  Refer to section 5.5.2 for more details.

- Increasing the walltime of running jobs is possible via ccsalter.

- Administrator may specify a limit with respect to alter walltime. This limit is checked for running jobs.

- Administrator may specify a job count limit to a group or user.

- Administrator may specify attributes which overwrite the user given ones. Refer to section 5.5.1 for more details.

- ccsinfo -l --user=ALL shows limits of all specified users in the group. Refer to section 8.3.2 for more details.

- ccsinfo -s --dist=FILTER shows job distribution. Refer to section 8.1.2 for more details.

- ccsinfo -s --fmt=%v shows elapsed time.

- ccsinfo -s --fmt=%d shows elapsed time in percent of max. duration.
3.9 New Features in OpenCCS 0.9.7

- Default memory is now per core instead of chunk. Refer to section 5.5.1 for more details.

- Support for Intel-Xeon-Phi cards
  Users may request Intel-Xeon-Phi cards in native mode. For example in MPI applications.

- Increasing duration of running jobs may displace planned best effort jobs.

- `ccsinfo -n --fmt=%O`
  Added nodes uptime specifier %O. Refer to section 8.2.1 for more details.

- `ccsinfo -s --dist`
  Shows job distribution for users / groups, i.e., how many jobs are in which state. Refer to section 8.1.2 for more details.

3.10 New Features in OpenCCS 0.9.6

- `ncpus` limit enforcement
  The NSM now observes that a job will not use more ncpus than requested. Refer to section 5.6.1 for more details.

- Option `--fmt` to `ccsinfo -s` and `ccsinfo -n`
  Using the option `--fmt` one can specify which information should be shown, the field format, and their order. Refer to section 8.1.4 and section 8.2.1 for more details.

- Option `--raw` to `ccsinfo -s` and `ccsinfo -n`
  Using the option `--raw` prints the fields in a raw format: No headline, no field formatting. Fields are separated by blanks. Refer to section 8.1.4 and section 8.2.1 for more details.

- Option `--reqid` to `ccsinfo -n`
  Prints information about nodes assigned to request reqID. Refer to section 8.2.1 for more details.

- `ccsinfo -s` accepts reqIDs as a filter.
  E.g., `ccsinfo -s 123 456` prints info only for the requests with reqID 123 or 456.
Chapter 4

Getting Started with OpenCCS

This chapter introduces the different OpenCCS user interfaces and some basic concepts like request types. It also explains how the OpenCCS user interface can be configured and describes the environment variables set for jobs.

4.1 User Roles

OpenCCS allows certain privileges based on what role a person has. OpenCCS recognizes only three roles, and all those using OpenCCS must be assigned at least one of these roles. The roles are User, Manager, and Administrator.

4.1.1 User-Role

Users are those who submit jobs to CCS. Users have the lowest level of privilege. Users are referred to in the CCS documentation as users. Users may operate only on their own jobs. They can do the following:

- Submit jobs,
- alter their jobs,
- send messages or signals to their jobs,
- hold, resume, or kill their jobs,
- get all information about their jobs.
4.1.2 Manager-Role

Managers have more privileges than users, and less privileges than Administrators. Managers are assigned to groups. A group can have more than one manager. ccsinfo -l prints the group managers. Managers can do the following:

- Run any command on jobs owned by their group (ccsalter, ccsbind, ccsinfo, ccskill, ccsmmsg, ccssignal, ccstracejob).
  
  Note: ccskill --all will only kill jobs owned by the caller. The Manager privilege is ignored if using “--all”.

4.1.3 Administrator-Role

Administrators can do all operations that Managers can perform for all groups. The OpenCCS CLI knows the option --admin. If this option is given the CLI checks if the calling user is a known OpenCCS administrator. The AM logs all connections of an Administrator user interface.

Note: ccskill --all will only kill jobs owned by the caller. The Administrator privilege is ignored if using “--all”.

4.2 Principle Usage

You specify the tasks you want to execute and OpenCCS takes care of running these tasks. You may create a batch job file (i.e. a shell script file) and submit it to OpenCCS. This file includes the set of commands you want to execute and directives specifying the characteristics of the job and its resource requirements. Here is a small example:

```bash
#!/bin/sh
#CCS -s 17:00:23.04.2015
#CCS -t 4h
#CCS --res=rset=4:mem=400M:ncpus=4
./my_application
ccsworker ompi my_MPI
```

Example 4.2.1: A simple job script

You may also submit executables directly or by using a worker. By default, submitted jobs run in batch mode. However, you may run them in interactive mode, as well. You may also request an interactive shell on a node. Resources can be reserved in advance and jobs might be submitted to these resources. Access to reserved resources may be granted to other users or groups. All details are explained in chapter 6.
4.3 Request Types and Identifier

OpenCCS knows three types of “requests”:

1. Jobs: Are commands running on requested resources.
2. Job Arrays: Are used to group closely related work into a set which can be processed as a unit.
3. Reservations: Users may reserve resources in advance and then submit jobs or job arrays to them.

Users may assign a name to a request in the submission. Please, note that request names are not unique.

After accepting a submitted request OpenCCS assigns a unique numerical identifier to the request. The so called request-ID or reqID.

All commands related to an accepted request use a request identifier to identify a request. For this purpose one may use the reqID, the request name, or, in case of a job array subjob, a subjob identifier (SJID). All CLI commands know the option --all. It means: all owned requests. For example ccskill --all kills all my requests.

4.4 Request States

A request (i.e., a reservation or a job) in OpenCCS may enter these states.

PLANNING The scheduler is assigning a start time to the request.

PLANNED The scheduler has successfully assigned a start time to the request.

ALLOCATING The resources assigned to the request are currently initialized.

ALLOCATED The resources assigned to the request are ready for work. The following substates are possible for jobs:

PREPROC Preprocessing
EXEC Job is running
ACCOUNTED Job has accounted
POSTPROC Postprocessing
KILL Job is killed

STOPPING The resources assigned to the request are being released.
CHAPTER 4. GETTING STARTED WITH OPENCCS

STOPPED The resources are released. Job related data will be available for users for an administrator defined interval (often 30m) before OpenCCS removes the job completely. The command ccstracejob may be used to print job data if the job has been removed in the CCS runtime database.

WAITING The request is in a “waiting room” because OpenCCS is not able to assign a start time. This state is reached for two reasons:

1. If resources become unavailable while a request is in state PLANNED. In such a case the request keeps its place in the schedule because the resources may become available again before switching to state ALLOCATING. If not enough resources available, OpenCCS tries to replan the request. If this is not possible, the request state switches to WAITING. If the resources become available again, OpenCCS automatically tries to replan waiting requests.

2. The job has been held via ccsalter. Refer to 9.2. The following substates are possible for held jobs:
   - HOLD_BY_USER
     The job has been held by the job owner.
   - HOLD_BY_GROUP_MANAGER
     The job has been held by a group manager.
   - HOLD_BY_GROUP_ADMIN
     The job has been held by the OpenCCS administration.

4.5 OpenCCS Command Line Interfaces

OpenCCS provides a command line interface (CLI). Table 4.1 gives an overview.

All common UNIX mechanisms for I/O redirection and shell scripts can be used. All job control signals (ctl-s, ctl-q, ctl-z, ctl-c, ...) are supported and forwarded to the application if running an interactive job.

4.5.1 CLI Argument Parsing

The OpenCCS CLIs support GNU style command line arguments. The parsing follows these rules:

- The CLI arguments are the whitespace-separated tokens given in the shell command used to invoke the program.

- A "short option" is a single character argument beginning with a hyphen delimiter (‘-’).
4.5. OPENCCS COMAND LINE INTERFACES

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccsalloc</td>
<td>Submit a job or a reservation</td>
</tr>
<tr>
<td>ccsalter</td>
<td>Alter job(s) / reservations(s)</td>
</tr>
<tr>
<td>ccsbind</td>
<td>Re-bind an interactive job</td>
</tr>
<tr>
<td>ccsinfo</td>
<td>Get status information about system, nodes, and jobs</td>
</tr>
<tr>
<td>ccskill</td>
<td>Delete job(s) / reservation(s)</td>
</tr>
<tr>
<td>ccsmsg</td>
<td>Send a message to job(s)</td>
</tr>
<tr>
<td>ccssignal</td>
<td>Send a signal to job(s)</td>
</tr>
<tr>
<td>ccstracejob</td>
<td>print log / accounting data of completed job(s)</td>
</tr>
<tr>
<td>ccsworker</td>
<td>Call a OpenCCS worker</td>
</tr>
</tbody>
</table>

Table 4.1: OpenCCS CLI User Commands

- Multiple short options may follow a hyphen delimiter in a single token if they do not take arguments. Thus, ’-abc’ is equivalent to ’-a -b -c’.

- An option and its argument may or may not appear as separate tokens. (In other words, the whitespace separating them is optional.) Thus, ’-o foo’ and ’-ofoo’ are equivalent.

- A ”long option” consists of ‘--’ followed by a name made of alphanumeric characters and dashes. Users can abbreviate the option names as long as the abbreviations are unique.

- To specify an argument for a long option, write ’--NAME=VALUE’.

- Options may appear only once.

4.5.2 CLI Default Values

For some of the CLI options default values can be specified in the process environment or in a file. All OpenCCS command line interface commands may scan for default values.

The File uirc

OpenCCS rc (resource) files are normally located in the director $HOME/.ccsrc. The file uirc contains default values for the OpenCCS CLI commands.

The file syntax is: ”<name> <value>”. The name prefix is ”CCS_UI_”. A word beginning with ‘#’ and all the following characters up to a NEWLINE are ignored. The character tilde (‘~’) will be replaced with the caller’s home-directory. One can have an uirc file for each known island. The CLI commands first try to read $HOME/.ccsrc/uirc.ISLAND_NAME where
ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, they search for $HOME/.ccsrc/uirc.

**Search Order**

If a mandatory value is not specified via a CLI option, the CLI first looks for a corresponding environment variable. If it does not exist, the CLI tries to read the corresponding uirc file as described above. If no value is found, a compile time defined default value will be taken.

**Available Default Values**

- **CCS_UI_ADMIN** < ON|OFF >
  - Activate admin mode.
  - Defaults to: OFF.

- **CCS_UI_BG_OUTPUT** < path >
  - Related CLI switch -o.
  - Defaults to: /dev/null.

- **CCS_UI_DEBUG** < debug − level >
  - Related CLI switch --debug.
  - Defaults to: no debug mode.

- **CCS_UI_DEF_DURATION** < timespan >
  - Related CLI switch -d.
  - Defaults to: '10m'.

- **CCS_UI_DEF_EMAIL_RECIPIENTS** < emailaddress,... >
  - Related CLI switch --mail.
  - Defaults to: not specified.

- **CCS_UI_DEF_GROUP** < name >
  - Related CLI switch --group.
  - Defaults to: not specified.

- **CCS_UI_DEF_ISLAND** < name >
  - Related CLI switch -i.
  - Defaults to: not specified.

- **CCS_UI_DEF_NOTIFY_JOB** < signal|command leadtime >
  - Related CLI switch --notifyjob.
  - Defaults to: no notification.

- **CCS_UI_DEF_NOTIFY_USER** < level >
  - Related CLI switch --notifyuser.
  - Defaults to: no notification.

- **CCS_UI_DEF_NODE_FMT** < fmt − string >
  - Related CLI switch ccsinfo -n --fmt.
  - Defaults to: not specified.
4.6. SETUP THE OPENCCS ENVIRONMENT

CCS_UI_DEF_SCHED_FMT < fmt – string >
   Related CLI switch ccsinfo -s --fmt.
   Defaults to: not specified.

CCS_UI_NOHUP < ON|OFF >
   If set to ON it prevents the user-interface to break the con-
   nection to a running, interactive job if catching the SIGHUP
   signal. The caught signal will be sent to the job instead.
   Defaults to: OFF.

CCS_UI_REQ_NAME < name >
   Related CLI switch --name.
   Defaults to the base name of the job script file or the exe-
   cutable specified on the command line. If an interactive shell is
   requested, the request name will be set to INT. If a reservation
   is requested, the request name will be set to RSV.

CCS_UI_RC_FILE < path >
   Specifies an alternative CLI rc file.
   Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be speci-
   fied in the environment.

CCS_UI_WORKER_FILE < path >
   Specifies an alternative worker configuration file.
   Defaults to: $CCS/etc/<island>/worker.conf. NOTE: Needs
   admin privileges and can only be specified in the environment.

4.6 Setup the OpenCCS Environment

To use OpenCCS, a user has to modify its environment. Normally, this done
by the site administration which will establish the necessary environment
settings in system files (e.g., /etc/profile.d). Hence, everything should
work. If not, the following settings have to be made.

- Setup the environment variable CCS to the installation dependent Open-
  CCS directory.

- Expand the search path by two directories:
  1. $CCS/bin
  2. $CCS/bin/$CCS_ARCH

- Expand the manual search path by the directory: $CCS/man.

ccsgenrcfiles

To ease the installation of a new user, OpenCCS comes with the script
ccsgenrcfiles which may be called by the user and makes the following:
CHAPTER 4. GETTING STARTED WITH OPENCCS

- It creates the directory `.ccsrc` in the user’s home directory.
- It creates the default UI configuration file `.ccsrc/uirc` (refer to section 4.5.2).

```bash
%ccsgenrcfiles SNF

Your environment has been prepared for using CCS.
This script has:
- created the directory /home/kel/.ccsrc
- created the file /home/kel/.ccsrc/uirc
  which sets default values for the CCS
  command line user interface

For further information see ccsalloc(1) or http://openccs.eu
```

Example 4.6.1: Calling ccsgenrcfiles

4.7 Message of the Day

The administrator can establish a "message of the day" which is printed by ccsalloc, ccsalter, ccsbind, ccskill, and ccsinfo like this:

```bash
ccskill 23

********** CCS-motd from: Tue Jun,25 2018 18:30 **********
Do not meddle in the affairs of sysadmins, for they are subtle and quick to anger.
************************************************************
Request 23 will be killed
Request (23 / simul): is stopped
```

Example 4.7.1: Message of the day

If using the option `-q` printing of the motd is suppressed. The user can explicitely print the message of the day by calling ccsinfo --motd.
4.8 Job Priorities

OpenCCS jobs have priorities which are assigned automatically. There are 5 priority classes:

- **0-99**: Background jobs (ccsalloc -g background),
- **100-199**: Best-effort and earliest start time (ccsalloc -a),
- **200-299**: deadline jobs (ccsalloc -e),
- **300-399**: fix start time (ccsalloc -s),
- **400-499**: reservations.

In each class the priority is increased if:

- the job requests “expensive” resources (e.g., chunks with a lot of cores or memory,
- the job is running on more than one node,
- the job requests “high priority” resources.

A job’s priority can be printed using ccsinfo. The priority is used at backfilling and while re-planning a job. At backfill the list of backfilling jobs is sorted by the priority in descending order. At re-plan jobs with lower priority are displaced. This happens if:

- a job could not be allocated in time due to a system error,
- a use increased the maximum duration of a running job,
- the admin initiates a re-plan for a job using the operator interface.

The priority of a displaced job is automatically increased (if not a background job) to avoid starving.

4.9 Job Environment

When submitting a job, OpenCCS copies the process environment and rebuilds it on the execution host before starting the job.

4.9.1 Boot Node Environment Variables

CCS additionally sets the following job specific environment variables on the boot node (M):

- **CCS**: Path to the OpenCCS installation.
CHAPTER 4. GETTING STARTED WITH OPENCCS

CCS_ARCH Defines the local architecture (e.g., LINUX32 or LINUX64). Used to find an architecture dependent executable.

CCS_ARRAY_ID For a subjob, the request-ID of the related job array.

CCS_ARRAY_INDEX For a subjob, its index in the related job array.

CCS_ISLAND The island name

CCS_MAPPING A string describing the mapping of the job.
Syntax: hostname:=chunk[+chunk..][,hostname...] and chunk is: count:name=val[:name=val]
Example: CCS_MAPPING=node01:=1:ncpus=2:mem=4g,
          node12:=1:ncpus=5:mem=180g

CCS_NODEFILE Absolute path of the OpenCCS node file.

CCS_NODES A space separated list of the node names of the allocated resources.

CCS_REQID The request-ID.

CCS_REQNAME The request name.

CCS_TMPDIR The path of the request specific, node local temporary directory. At allocation time, CCS creates a node local directory named <path>/<reqID>. The value of <path> is set by the CCS administration. This directory can be used by applications for writing temporary files during runtime. The directory will be removed automatically when releasing the partition.

CCS_UMASK Value of the current umask.

NCPUS For the MPI process with rank 0. Set to the value of ncpus requested for the last chunk. For other MPI processes, behavior depends on the MPI implementation.

OMP_NUM_THREADS For the MPI process with rank 0. Set to the value of ompthreads. For other MPI processes, behavior depends on the MPI implementation. It defaults to the value of ncpus. If you do not request ompthreads in the last chunk, then OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk.

TMPDIR Same as CCS_TMPDIR.
4.9.2 The OpenCCS Node File

OpenCCS creates a file containing the node names allocated to a job. The file name is stored in the environment variable CCS_NODEFILE. Each node appears once in a single line. The file will contain the names of the allocated nodes with each name repeated N times, where N is the number of mpiprocs specified for all chunks allocated on that node. mpiprocs is the number of MPI instances per chunk and defaults to 1. The order in which nodes appear in the node file is the reverse order in which chunks were specified in the --res=rset directive.
Chapter 5

Resources

This chapter describes the OpenCCS resource categories, the units and formats to specify them, the built-in resources, the syntax to request resources, the resource assignment, and the limit assignment and enforcement. The chapter ends with examples on how to request resources.

5.1 Resource Categories

All resources managed by OpenCCS are part of at least one of the following categories:

- **Built-In**: Is provided by the RMS as shipped. E.g.: `ncpus`, `mem`, `vmem`, `arch`, `host`, ...
- **Custom**: Is not defined in RMS as shipped. It is created by the administrator.
- **Node**: Is made available at the node level and is only usable as chunk resources.
- **Job-Wide**: Is used for requesting floating licenses or other resources not tied to specific nodes, such as `cput` and `wall time`. A job can request a job-wide resource for the entire job, but not for individual chunks.
- **Consumable**: Is reduced by being used. Examples are `ncpus`, `licenses`, or `mem`.
- **Non-Consumable**: Is not reduced through use. Examples are `walltime`, `file`, `arch`, `cput`, `nice`, or boolean resources. For non-consumable resources such as `arch` or `hostname`, OpenCCS matches the value requested by a job with the value at one or more nodes.
Matching a job this way does not change whether or not other jobs can be matched as well.

**Static**
Is managed by the RMS. Static resources have values that are fixed until you change them or until the hardware changes.

**Dynamic**
Is not under the control of the RMS, (i.e., can change independently).

### 5.2 Resource Formats

Resources can be specified using the following units and formats.

**Boolean**
A boolean value. Syntax:

- True ::= "t" | "y" | "1" | "yes" | "true"
- False ::= "f"| "n" | "0" | "no" | "false"

Values are not case sensitive.

**Cron**
Specifies a periodic time interval like in a cron job specification. Syntax: A string of five space separated tokens (a b c d e)

- a is minute: 0-59
- b is hour: 0-23
- c is day of month: 1-31
- d is month: 1-12
- e is day of week: 0-6 (0 is Sun)

Given intervals include both boundaries (e.g., hours 17-22 start at 17:00:00 and end at 22:59:59. Each token may be:

- a wildcard given as asterisk '*', which always stands for "first-last"
- a comma separated list of time points, e.g. "2,3,5"
- an interval, e.g. "3-4"
- a combination of lists and intervals, e.g. "1,2,4-6"

Not allowed are:

- step values, e.g. "/2"
- shortcuts like "/@weekly"
- weekday’s name, e.g. "sun"
5.2. RESOURCE FORMATS

**Datetime** Specifies a Date and/or a time. The following formats are recognized.

- **POSIX format**
  - CC is the first two digits of the year (the century),
  - YY is the second two digits of the year,
  - MM is the two digits for the month,
  - DD is the day of the month,
  - hh is the hour,
  - mm is the minute,
  - SS seconds.

- **OpenCCS format**
  Syntax: `<hh[:mm] | hh:mm:dd.mm[.yy]>`
  - hh hours from 00 to 23
  - mm minutes from 00 to 59
  - ss seconds from 00 to 59
  - mm months from 01 to 12
  - yy years from 00 to 99
  Units are not case sensitive.

For all Datetime formats, the following is valid: If the month is not specified, it will be set to the current month if the specified day is in the future. Otherwise, the month will be set to next month. If the day is not specified, it will be set to today if the time is in the future. Otherwise, the day will be set to tomorrow. For example: specifying at `11:15am` a time of `11:10`, will be evaluated as `11:10am` tomorrow.

**Size** Specifies a size (memory, disk, ...) or a count
Syntax: `<number[multiplier]>`

- Kilo: k is $2^{10}$ and K is $10^3$
- Mega: m is $2^{20}$ and M is $10^6$
- Giga: g is $2^{30}$ and G is $10^9$
- Terra: t is $2^{40}$ and T is $10^{12}$

Example: `1000K` denotes $1000*1000$ and `1000k` denotes $1000*1024$. Default multiplier is 1.
CHAPTER 5. RESOURCES

String
A series of alpha-numeric characters without whitespace(s), beginning with an alphabetic character.

String Array
A comma separated list of Strings. The character ‘,’ is not allowed within a String. A resource of type ‘string array’ is non-consumable. A resource request will succeed if request matches one of the values. A resource request can contain only one string. A string array resource with one value works exactly like a string resource.

Timespan
The following two formats are recognized:

- \[[\text{hours:}]\text{minutes:}]\text{seconds}\n  Example: \(120:12:13\) denotes 120 hours, 12 minutes, and 13 seconds.
- \([*\text{w}][*\text{d}[*\text{h}[*\text{m}]]]*\text{s}\n  Supported units are:
  - \(\text{w}\) (week) equals to 7 days
  - \(\text{d}\) (day) equals to 24 hours
  - \(\text{h}\) (hour) equals to 60 minutes
  - \(\text{m}\) (minute) equals to 60 seconds
  - \(\text{s}\) (second)
  Default unit ist second. The unit order is irrelevant. Example: \(14\text{d}1\text{h}12\text{m}3\text{s}3\text{w}\) denotes 3 weeks, 14 days, 1 hour, 12 minutes, and 3 seconds.

Unitary
Specifies the maximum amount of a resource which is expressed as a simple integer.

5.3 Built-In Resources

Table 5.1 shows a list of resources available on all OpenCCS systems. The shown units are corresponding to the previous section. The listed shortcuts can be used directly, instead of using the \(--\text{res}\) parameter.

5.4 Specifying Resources

This section describes the syntax to specify CCS resources if submitting or altering requests.
5.4. SPECIFYING RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Shortcut</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td></td>
<td>String</td>
<td>Architecture</td>
</tr>
<tr>
<td>cput</td>
<td></td>
<td>Timespan</td>
<td>cputime</td>
</tr>
<tr>
<td>hostname</td>
<td></td>
<td>String</td>
<td>Hostname</td>
</tr>
<tr>
<td>mem</td>
<td></td>
<td>Size</td>
<td>Physical memory, sets a limit</td>
</tr>
<tr>
<td>mpiprocs</td>
<td></td>
<td>Unitary</td>
<td>Number of MPI instances per chunk (controls hostfile entries). Defaults to 1 if ncpus &gt; 0, 0 otherwise.</td>
</tr>
<tr>
<td>ncpus</td>
<td>-c</td>
<td>Unitary</td>
<td>Number of CPUs / Cores, sets a limit</td>
</tr>
<tr>
<td>exclnodes</td>
<td>-n</td>
<td>Unitary</td>
<td>Number of exclusively used nodes</td>
</tr>
<tr>
<td>ompthreads</td>
<td></td>
<td>Unitary</td>
<td>Number of OpenMP threads (controls hostfile entries). Defaults to 1</td>
</tr>
<tr>
<td>vmem</td>
<td></td>
<td>Size</td>
<td>Virtual memory. Establishes a per-chunk limit.</td>
</tr>
<tr>
<td>walltime</td>
<td>-t</td>
<td>Timespan</td>
<td>Max. duration</td>
</tr>
</tbody>
</table>

Table 5.1: Always available resources.

5.4.1 Syntax

The user may specify resources by:

```
ccsalloc --res="resource_name[=value][,resource_name[=value],...]"
```

`resource_name` is the name of an allocatable resource (which is generic or system dependent). `ccsinfo` will show the allocatable resources.

A resource name:

- Is not case sensitive.
- May include white spaces between '=' or ','.
- May be a resource set specification, a placement specification, or a job wide resource specification.

The option `--res` may be used serveral times. The parameters will be concatenated.

5.4.2 Resource Set / Chunk Specification

A resource set (also named chunk) specifies a set of resources that have to be allocated as a unit on one node. Chunks cannot be split across nodes. Resource sets are specified using the keyword "rset".

Syntax: `rset=[N:]chunk[+[N:]chunk...]`

If N is not specified, it is set to 1. A chunk comprises one or more `res=value` statements separated by a colon. Examples:

- `ncpus=2:mem=10g:hostname=Host1`
- `ncpus=27:vmem=20g:arch=linux+4:acc=fpga`
5.4.3 Placement Specification

This specification controls how the chunks are placed on the nodes. Syntax: \texttt{place=[arrangement][:sharing ][:grouping][:ignore]}

- Arrangement is one of \texttt{free}, \texttt{pack}, or \texttt{scatter}.
- Sharing is one of \texttt{excl} or \texttt{shared}.
- Grouping can have only one instance of \texttt{group}=\texttt{resource}.
- Ignore is a ‘;’ separated list of hostnames, which should be excluded from mapping (\texttt{ignore='H1;H2;...'}). 

Default is: \texttt{free:shared} All keywords are described in the following table.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>no restriction</td>
</tr>
<tr>
<td>pack</td>
<td>all chunks must be placed on one node</td>
</tr>
<tr>
<td>scatter</td>
<td>only one chunk per node</td>
</tr>
<tr>
<td>exclusive</td>
<td>only this job may use the node</td>
</tr>
<tr>
<td>shared</td>
<td>this chunk may share the node with other chunks</td>
</tr>
<tr>
<td>group</td>
<td>group all chunks by the specified resource</td>
</tr>
<tr>
<td>ignore</td>
<td>exclude these hosts from mapping</td>
</tr>
</tbody>
</table>

Table 5.2: Placement Specification

Grouping on a Resource

One can specify that all of the chunks of a job should run on nodes that have the same value for a selected resource. To group the chunks this way, use the following format: \texttt{place=:group=<resource>}

Chunk grouping will be ignored if:

- the OpenCCS administration disabled grouping,
- it is a single host job, i.e., all chunks are mapped to a single host,
- the job is part of a reservation,
- \texttt{<resource>} is
  - a job wide resource,
  - one of \texttt{cput}, \texttt{hostname}, \texttt{mpiprocs}, \texttt{ompthreads}, or \texttt{walltime}.
5.4. SPECIFYING RESOURCES

For example, let's assume there is a resource named switch which reflects to which Infiniband switches a node is connected. The value can be “s10,s1” at one node, and “s11,s1” at another node. Then these nodes can be grouped by place=:group=switch because they share the string “s1”. All classes printed by ccsinfo -a --classes can be used for grouping (except the ones listed above).

Using the method of grouping on a resource, one cannot specify what the value of the resource should be, only that all nodes have the same value. If one needs the resource to have a specific value, specify that value in the description of the chunks.

Depending on the settings of the OpenCCS administration a job will be rejected at submit time, if no group is found which is large enough.

During mapping OpenCCS tries to fill small groups first.

5.4.4 Job-Wide Resources

Job-wide resources are assigned to the system level and may be used for requesting floating licenses or other resources, which are not tied to specific nodes, such as cput or walltime. Job-wide resources can only be requested outside of an rset statement.

Not allowed are: arch, hostname, mem, ncpus, and vmem.

Syntax: keyword=value[,keyword=value ...] Example: --res=sw=g03

5.4.5 Specifying Resource Values

- Resource values which contain commas, quotes, plus signs, equal signs, colons, or parentheses must be quoted. The string must be enclosed in quotes so that the OpenCCS CLI command will parse it correctly.

- If specifying resources via the command line, any quoted strings must be escaped or enclosed in another set of quotes. This second set of quotes must be different from the first set, meaning that double quotes must be enclosed in single quotes, and vice versa.

- If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want.

5.4.6 Examples

No resource specification

If you do not specify a resource, the CLI will request:

rset=1:ncpus=1,place=free:shared
CHAPTER 5. RESOURCES

CPUs
Free placement of 10 CPUs across nodes:
--res="rset=10:ncpus=1"
There is a ccsalloc short cut: ccsalloc -c <number of cores>

Chunks
- 4 chunks each with 1 CPU and 4GB memory, placed anywhere, but exclude the hosts smp05 and phi001:
  --res="rset=4:ncpus=1:mem=4g,place=ignore=smp05;phi001"
- 4 chunks each with 1 CPU and 4GB memory each of them in a separate node:
  --res="rset=4:ncpus=1:mem=4g, place=scatter"
- 4 chunks each with 1 CPU and 4GB memory placed in only one node:
  --res="rset=4:ncpus=1:mem=4g, place=pack"
- 4 chunks each with 1 CPU and 4GB memory placed on host FOO:
  --res="rset=4:ncpus=1:mem=4g:hostname=FOO"

Nodes
- Nodes a and b exclusively:
  --res="rset=hostname=a+hostname=b, place=free:excl"
- 4 nodes exclusively:
  --res="rset=4:ncpus=1, place=scatter:excl"
There is a ccsalloc short cut:
ccsalloc -n <number of nodes>

MPI-Jobs
- 10-way MPI-Job each with 2GB:
  --res="rset=10:ncpus=1:mem=2g"
- 4 chunks, each with 6 CPUs with 3 MPI processes, each on a separate host:
  --res="rset=4:cpus=6:mpiprocs=3, place=scatter"
- 2 chunks, each with 8 CPUs and 8 MPI tasks and four threads:
  --res="rset=2:ncpus=8:mpiprocs=8:ompthreads=4"

Networks
Nodes with Infiniband HCAs (provided this custom resources are specified):
--res="rset=5:net=IB"

Accelerators
5 CPUs and a NVIDIA card (provided this custom resources are specified):
--res="rset=1:ncpus=5:gpu=nvidia"

Licenses
- 4 chunks each with 1 CPU, 3GB of memory and 1 node-locked Fluent license. This assumes Fluent is specified as a node local resource:
  --res="rset=4:fluent=1:ncpus=1:mem=3g,place=free"
5.5. Resource Assignment to Jobs

5.5.1 Default and Force Resources

The administrator may define default and force resources for system, group, and user level. They are assigned in the following order: user, group, system. First match wins.

Default attributes are assigned if the user did not specify the resource in question. Force attributes overwrite the user given ones or act as a default value. If both a default and a force attribute have been specified, the force attribute will be taken. The command `ccsinfo --def` (section 8.3.3) shows the default and force values specified by the administrator.

**mem and vmem**

OpenCCS assigns default values for the resources `mem` and `vmem`, even if there were no defaults specified by the administrator. The amount is set to the minimum amount per core over all hosts and is adapted automatically if nodes became (un)available.

The default values are shown in the column **Default** of the command `ccsinfo -a` (section 8.2.3). OpenCCS sets the missing value in a chunk to `default*ncpus`.

Hence, assuming the following defaults: `mem=3g` and `vmem=4g`, the request `--res=rset=1:ncpus=3` will result in `ncpus=3, mem=9g, and vmem=12g`.

If the user specified only one of `mem` or `vmem`, the missing one is set equal to the specified one. Hence, `--res=rset=2:ncpus=3:mem=4g` will result in `ncpus=3, mem=4g, and vmem=4g`.

5.5.2 Matching Unset Resources

When job resource requests are being matched with available resources, unset non-consumable resources are treated as follows:

- A numerical resource that is unset on a host is treated as if it were zero

---

- 4 chunks each with 1 CPU, 3GB of memory and 4 floating Fluent licenses. This assumes Fluent is specified as a consumable, global dynamic resource:
  
  ```
  --res="fluuent=4,rset=4:ncpus=1:mem=3g,place=free"
  ```

Scratch File System

100G ccratch space and 3 chunks, each with 1 CPU and 10GB of memory. Scratch is assumed to be on a file system common to all hosts.

```
--res="scratch=100g, rset=3:ncpus=1:mem=10g"
```
• An unset Boolean resource is treated as if it were set to False.

• An unset string cannot be matched

• The resources ompthreads, mpiprocs, and nodes are ignored for unset resource matching.

Examples:

• requesting \texttt{--res=rset=smp=f} will match all hosts where the resource \texttt{smp} is unset or set to \texttt{false}.

• requesting \texttt{--res=rset=rack=0} will match all hosts where the resource \texttt{rack} is unset or set to 0.

5.6 Resources and Limits

Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources. OpenCCS provides built-in resources, and in addition, allows the administrator to define custom resources which may be consumable. A consumable resource is one that is reduced by being used, for example, \texttt{ncpus}, licenses, or \texttt{mem}. A non-consumable resource is not reduced through use, for example, \texttt{walltime} or a boolean resource.

Jobs have assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each node (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources. If a job’s job resource limit exceeds the restrictions, it will not be accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

Job limits are created from the directive for each consumable resource. For example, \texttt{ccsalloc --res=rset=2:ncpus=3:mem=4g} will have the following job limits set: \texttt{ncpus=6, mem=8g, and vmem=8g} (refer to section 5.5.1). The command \texttt{ccsinfo --limits} (section 8.3.2) will show the limits assigned to a user / group.

5.6.1 Limit Enforcement

For a job, enforcement of resource limits is per NSM. For example, if a job requests 3 chunks each of which has 1GB of memory, and all chunks are placed on one host, the memory limit for that job for that NSM is 3GB. Therefore one chunk can be using 2 GB and the other two using 0.5GB and the job can continue to run. The NSM polls for resource usage for \texttt{cput}, \texttt{walltime}, \texttt{mem}, and \texttt{ncpus} each X seconds. The value of X is specified by the administration.
5.6. RESOURCES AND LIMITS

The \texttt{ncpus} limit is checked each poll period. The job is killed if the following is true:

\[
\frac{\text{cput}}{\text{walltime}} > \text{ncpus} \times \text{cpuFactor} + \frac{\text{percentOver}}{100}
\]

Per default \texttt{cpuFactor} is 1.025 and \texttt{percentOver} is 50. The values can be changed by the administrator for each node.

A job may exceed its limit for the period between two polling cycles. Per-process limits are enforced by the operating system kernel. OpenCCS calls the kernel call \texttt{setrlimit()} to set the limit for the top process (the shell), and any process started by the shell inherits those limits.

5.6.2 Limits on Exclusively Used Nodes

If the placement directive \texttt{excl} is used, the whole node is assigned to the user. In this case, the NSM does not poll for resource limits.

5.6.3 Resource Credits

The OpenCCS administration may assign credits to a consumer (group or user) for any consumable resource (per-chunk or job-wide). A credit is given as an area (\texttt{duration\times count}), e.g., 200 CPU hours.

Resource credits apply to jobs and reservations. Before a job is allocated, OpenCCS checks if there is enough credit available. If not, CCS will (depending on the specified policy):

- Reject the job and the user will get a related message.
  
  \textbf{Example:}

  \begin{verbatim}
  Resource 'tesla':credit exceeded:'
  requested(100:00:00) > remaining(99:57:00)
  \end{verbatim}

- Set the job priority to the “background job” priority. This is the lowest priority in OpenCCS.

- Do nothing.

If enough credit available, CCS adds the product of requested maximum job duration and number of requested resources to “Used-Credit”.

When the job terminates, OpenCCS updates “Used-Credit” with the real value because the duration may be shorter than requested. If altering the duration of a running job, “Used-Credit” is also updated.

Resource reservation requests are checked at submit time and immediately added to “Used-Credit”. If the reservation is killed or terminated, “Used-Credit” is updated. Jobs running in a reservation are not changing the “Used-Credit” value. If altering a reservation (resources or duration), “Used-Credit” is also updated.
The command `ccsinfo --limits` (section 8.3.2) shows the current values for resource credits assigned to a user / group.

**Example:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Credit</th>
<th>Used-Credit</th>
<th>Remaining-Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdce</td>
<td>1000:00:00</td>
<td>0:00:18</td>
<td>999:59:42</td>
</tr>
<tr>
<td>tesla</td>
<td>100:00:00</td>
<td>0:03:00</td>
<td>99:57:00</td>
</tr>
<tr>
<td>ncpus</td>
<td>2500000:00:00</td>
<td>0:03:00</td>
<td>2499999:57:00</td>
</tr>
</tbody>
</table>
Chapter 6

Submitting Jobs

6.1 Introduction

The standard way of submitting jobs to OpenCCS is using the command ccsalloc. CCS distinguishes between batch jobs and interactive jobs. For a submission, OpenCCS needs this information:

- The resources to allocate.
- When to run the job.
- How long the job will run.
- The corresponding executable.

If you are familiar with PBSPro or Torque, you will find several similar directives.

6.2 Script Jobs

6.2.1 Submitting a Job Script

OpenCCS jobs can be submitted using ccsalloc (e.g., ccsalloc job.sh). If job.sh is a script file, it may contain directives describing the job, followed by the job itself. OpenCCS directives can be set by adding comments with the following syntax to the job script: #CCS <option> [value]. If directives are given multiple times, first match wins. The possible options in the script directives are the same options as the ccsalloc command line options. The job itself is usually called by ccsworker in the job script. A job script may be structured as shown in example 6.2.1.

The shown script runs my preproc and then starts a Gaussian job using the command ccsworker on one core of the Island HAWAII. The resources are allocated for one hour.
CHAPTER 6. SUBMITTING JOBS

Example 6.2.1: A simple job script

```
#!/usr/bin/sh
#CCS --island HAWAII
#CCS --res=rset=5:ncpus=4:mem=10g
#CCS --res=matlab=5
##CCS --res=place=scatter:excl
#CCS -t 1h
my_preproc
ccsworker g09 -- my_gaussianjob
```

Parsing Rules

An initial line in the script that begins with the characters "#!/" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first non white space character is '#'.

If directives occur on subsequent lines, they will be ignored. A line in the script file will be processed as a directive to `ccsalloc` if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix (i.e. `#CCS`).

The remainder of the directive line consists of the options to `ccsalloc` in the same syntax as they appear on the command line. The option character has to be preceded with the '-' character.

If an option is present in both a directive and the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not in the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The option `--res` may occur multiple times.

6.2.2 Changing the Job’s CCS Directive

By default, the text string '#CCS' is used by OpenCCS to determine which lines in the job file are directives. The leading # symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on UNIX systems. Because directives look like comments, the scripting language ignores them. The directive may be changed by the `ccsalloc` option -C. E.g., `ccsalloc -C '#PBS' job.sh` This may be useful if one wants to re-use a PBS or Torque script, since many directives are identical.
6.3. INTERACTIVE JOBS

6.2.3 Passing Arguments to Job Scripts
If you need to pass arguments to a job script, just add them to the `ccsalloc` call. Example: 
`ccsalloc job.sh -x 1 34`

6.2.4 Jobs Without a Job Script
Jobs can be submitted with the following syntax:
Jobs are regularly executed as batch jobs. The specification of the used Island has to be the first element of the [args] list.

Simple Jobs
Simple jobs may also be submitted without selecting a worker by calling `ccsalloc <job>`.

6.3 Interactive Jobs

By using the `ccsalloc` option `-I` one can submit interactive jobs. The `-I` option is ignored in a script directive. The streams `STDIN`, `STDOUT`, and `STDERR` are then connected to the submitting terminal. This is useful for debugging applications or for computational steering.

When the job starts to execute, all input to the job is from the terminal session in which `ccsalloc` is running.

When an interactive job is submitted, the `ccsalloc` command will not terminate when the job is submitted. It will remain running until the job terminates, or is aborted.

If connected to job, the terminal characteristics are changed in the following way:

CTRL-C Sends the signal SIGINT to the job.
CTRL-S, CTRL-Q Acts as usual.
CTRL-Z Acts as usual with one exception. If you type `bg` in your Unix-shell to put the job into the background, the output (`STDOUT`, `STDERR`) will be redirected as described in section 6.3.1: 'The Virtual Terminal'.

6.3.1 The Virtual Terminal

Interactive applications are not stopped when the connection between the user interface and the application has been broken.
CHAPTER 6. SUBMITTING JOBS

All output (STDOUT and/or STDERR) which is not explicitly redirected into files will then be buffered by the EM on the execution host and (if the buffer gets full) redirected into specified files. If you have specified a redirection into a file and the file cannot be opened, OpenCCS aborts the job.

The redirections of STDOUT and STDERR become active if the connection between the user interface (UI) on the frontend and the Execution Manager (EM) on the compute host gets lost. This may be caused by:

- Typing CTRL-Z bg in the terminal
- A crash of the UI
- A forced shutdown of the UI
- A crash of the frontend

One can reconnect the application with the ccsbind command. The applications streams (STDIN, STDOUT, and STDERR) will then be redirected to the terminal. Please note: If you redirected the STDIN stream it will be not redirect to the terminal.

6.4 Background Jobs

Background jobs are jobs which have the lowest priority (4.8). Therefore a background job may never run. They are submitted by using the group background. E.g., ccsalloc -g background -c 5 hostname.

A background job is not allowed to submit:

- reservations,
- jobs running in a reservation (ccsalloc --rsvid),
- jobs with a earliest start time (ccsalloc -a),
- jobs with a fixed start time (ccsalloc -s),
- jobs with a deadline (ccsalloc -e).

A user may inspect the used policy by calling ccsinfo -l -g background.

6.5 Job Submission Options

There are many options to the ccsalloc command. The rest of this chapter explains the important ones. For a full description, refer to the ccsalloc man page (page 95).
6.5. JOB SUBMISSION OPTIONS

6.5.1 Time Related Attributes

There are 3 time related options: Maximum runtime, start time, and end time (i.e., a deadline).

Maximum Runtime

Each job has a maximum runtime. If you do not specify a maximum runtime, a site specific value is taken. It can be shown by `ccsinfo --default`. If the maximum runtime is exceeded OpenCCS will terminate the job.

Syntax: `<d|-t|--duration=|--time=|--walltime=/> DURATION`

Start Time

The start time can be specified in 3 different ways:

1. No specification
   This means OpenCCS handles the job as a best-effort job, i.e., it may move the job on the time axis as necessary. This is the normal way to submit a job.

2. The job should not start before time T
   Syntax: `<a|--after=/> T`
   T may be given as absolute time (Format Datetime) or relative to now (Format `'+Timespan').
   Examples:
   ```
   ccsalloc -a +2h
   ccsalloc -a 17:00:24.03.2023
   ccsalloc -a 2348
   ```

3. The job should start exactly at time T or never
   Syntax: `<s|--starttime=/> T`
   T may be given as absolute time (Format Datetime).
   If the start time should be now, use `now`.
   If the start time should be relative to now, use `'+Timespan'.
   If it is not possible to schedule the job, it will be rejected.
   Examples:
   ```
   ccsalloc -s now
   ccsalloc -s 17:00:24.03.2023
   ccsalloc -s +3d
   ```

End Time

You may specify a deadline. This is the time the job should be terminated at the latest.

Syntax: `<e|--endtime=/> T`
Format: Datetime.
Example: ccsalloc -e 23
The scheduler may move the job on the time axis before as long as the deadline is met.

Combinations and Effects
Table 6.1 summarizes the combinations of the different time related options and the effects on how OpenCCS plans the job.

<table>
<thead>
<tr>
<th>CLI Option Combinations</th>
<th>Start</th>
<th>Stop</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t d</td>
<td>asap</td>
<td>start + d</td>
<td>Best-effort</td>
</tr>
<tr>
<td>-a T -t d</td>
<td>≥ T</td>
<td>T+d</td>
<td>Best-effort</td>
</tr>
<tr>
<td>-s T -t d</td>
<td>= T</td>
<td>T+d</td>
<td>Fix</td>
</tr>
<tr>
<td>-e T -t d</td>
<td>≤ T-d</td>
<td>≤ T</td>
<td>Deadline</td>
</tr>
<tr>
<td>-a T1 -e T2 -t d</td>
<td>≥ T1</td>
<td>≤ T2</td>
<td>Deadline, d must be ≤ T2-T1</td>
</tr>
<tr>
<td>-a T1 -s T2 -t d</td>
<td></td>
<td></td>
<td>Not allowed</td>
</tr>
<tr>
<td>-s T1 -e T2 -t d</td>
<td></td>
<td></td>
<td>Not allowed</td>
</tr>
<tr>
<td>-a T1 -s T2 -e T3 -t d</td>
<td></td>
<td></td>
<td>Not allowed</td>
</tr>
</tbody>
</table>

Table 6.1: Correlations of the different scheduling hints

6.5.2 Request Name
The option <-N|--name=> <name> specifies a name for the request. The name specified may be any length. If no request name has been specified, OpenCCS will set the following:

- **INT**, if an interactive shell is requested.
- **RSV**, if a reservation is requested.
- The base name of the job script file or the executable specified on the command line in all other cases.

A default value can be set. Refer to 4.5.2.

6.5.3 Email Notification
The option <-m|--notifyuser=> <EVENTS> specifies the set of conditions under which OpenCCS will send mail messages about the job. EVENTs is a string which consists of either the single character n, or one or more of the characters a, b, e, r, s, and w.

- **a** Send mail if job is aborted by OpenCCS.
6.5. JOB SUBMISSION OPTIONS

b  Send mail when job begins execution.
e  Send mail when job ends execution.
n  Do not send mails.
r  Send mail if job start has been replanned.
s  Send mail for each subjob of a job array.
w  Send warning mails. E.g.: resource could not be allocated, reservation unused, runtime expires soon, ...

If not set it defaults to: 'n'.
A default value can be set. Refer to 4.5.2.

6.5.4 Email Recipients

The option `<M|--mail=><account@domain, ...>` specifies a list of email addresses separated by ','. OpenCCS will send all emails to the stated recipient(s). Defaults to: no mail address given.
A default value can be set. Refer to 4.5.2.

6.5.5 Job Notification

The option `--notifyjob=<HOW,WHEN>` specifies how and when a job should be notified by OpenCCS.

**HOW** is either a command or a signal which is executed/ sent to the job.

**WHEN** specifies the timespan before the resource is released.

- **cmd**  Is an executable. It will have the same environment variables as the initial job started on the boot node.
- **signal**  Can be given as: [-]<digit> or [-][SIG]<signal>.

**WHEN** Format: Timespan. It must be ≥ 60 seconds.

Setting **HOW** and **WHEN** to 0 (eg. -notifyjob 0, 0) disables this feature. If not set it defaults to: 0, 0 (ie., do not notify the job.)
A default value can be set. Refer to 4.5.2. Examples:

- `--notifyjob=$HOME/bin/myScript.sh,10m`
  Executes `$HOME/bin/myScript.sh` 10 minutes before the resource is released.

- `--notifyjob=XCPU, 135`
  Sends the signal SIGXCPU 135 seconds before the resource is released.

- `--notifyjob=-9,12m`
  Sends the signal SIGKILL 12 minutes before the resource is released.
6.5.6 Input, Output and Error Files

If submitting a batch job OpenCCS, by default, sets the following redirections:

STDIN from /dev/null, which means no input.
STDOUT to a file named job-name.<reqID>.out
STDERR to a file named
verb—job-name.<reqID>.err—

<reqID> is the request-ID assigned by OpenCCS. STDOUT and STDERR files are located in the submit directory. This redirection also takes place for interactive jobs if the user-interface has lost the connection to the job (see also section 6.3.1).

Specifying Path for STDIN Redirection

The ccsalloc option --stdin= <FILE> specifies the path for the STDIN redirection. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory.

Specifying Path for STDOUT Redirection

The ccsalloc option -o|--output=|--stdout=> <FILE> specifies the path for the STDOUT redirection. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory.
A default value can be set. Refer to 4.5.2.

Specifying Path for STDERR Redirection

The ccsalloc option --stderr=<FILE> specifies the path for the STDERR redirection. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory.

Keywords

The following keywords may be used while specifying redirections:

%reqid will be replaced by the reqID.
%A will be replaced by the reqID of the related job array. If the job is no subjob, then %A and will be replaced by the job’s reqID.
%a will be replaced by the subjob index of a job array subjob. If the job is no subjob, then %a will be replaced by the job’s reqID.
6.5. JOB SUBMISSION OPTIONS

Joining STDOUT and STDERR Redirection

The ccsalloc option \(-j|--join\[=]\) \([\text{HOW}]\) will specify how OpenCCS should join the STDOUT and STDERR streams. HOW is one of the following:

- n: Do not join STDOUT and STDERR.
- oe: Join STDERR into STDOUT.
- eo: Join STDOUT into STDERR.

If HOW is not given joining is set to 'oe'. If joining is not set it defaults to 'n'.

Avoiding STDOUT and STDERR Redirection

Set the option \(-o\) or \(-e\) to /dev/null.

Examples

- ccsalloc \(--stdin=myFILE%reqid.in\)
  will read the STDIN stream from the file named myFILE<reqID>.in whereby <reqID> is the request-ID assigned by OpenCCS.

- ccsalloc \(-o\) MYLOG-%A.out.%a
  will create a file where %A is replaced by the reqID of the job array and %a is replaced by the subjob index.

- ccsalloc \(--stderr=myFILE%reqid.stderr\)
  will create a file named myFILE<reqID>.stderr whereby <reqID> is the request-ID assigned by OpenCCS.

6.5.7 Job Trace File

As mentioned in section 2.3.5, OpenCCS may write a job trace file. The option \(--tracefile=<FILE>\) specifies the path for this file. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory. The keyword '%reqid' will be replaced by the request-ID.

E.g.: ccsalloc \(--tracefile=myFILE%reqid.trace\)
will create a file named myFILE<reqID>.trace whereby <reqID> is the request-ID assigned by OpenCCS.
Chapter 7

Predicting Job Start Times

It is sometimes useful to know which resources are when available. For example how many GPUs can I get now or how long is the waiting time if requesting chunks with 5 cores and 6GB per core.

For this purposes OpenCCS provides `ccsinfo --predict`. It allows to specify resource requests together with iterators and OpenCCS will print a list with the earliest start times. The syntax is:

```
ccsinfo [-g GROUP] [--raw] -p ''<resources>[;<iterator>;...]''
```

OpenCCS will replace all found iterators in resources by their specified values and plan this request(s) including all user / group related limitations. It returns the planned start times.

Please note that the situation may change within seconds, if other users are submitting jobs in the meanwhile.

7.0.1 Resource Syntax

The syntax is like specifying resources at submit call (5.4) but without `--res=rset`.

Examples

- `%C:ncpus=%1:mem=%2g`
- `%C:tesla=%1:mem=%3g+ncpus=2:mem=5g,place=scatter:excl`

The shortcuts `-n` and `-c` are allowed to iterate over nodes or cores.

Examples: `-n %C; %C=1-10` or `-c %C; %C=100-1000:100`

Note: The only iterator recognized here is `%C`.

7.0.2 Iterator Syntax

Name=<first>[-<last>[:stepping]] All of them must be integers ≥ 0. Default stepping is 1. Three types of iterators are supported:
1. \%C iterates the number of chunks.
   - The default \%C iterator is 1-1:1.

2. \%D iterates the job duration.
   - Last character is unit if not a number (5.2).
     E.g., \%D=1-5h or \%D=1-10:2h.
   - The default unit is second.
     E.g., \%D=1-10:2 iterates: 1s, 3s, 5s, 7s, 9s.
   - The default \%D iterator is the default duration assigned to the caller’s credentials (i.e., user and group). Call `ccsinfo --def` to see the default duration.

3. \%R iterates the resources.
   - Nine iterators are available: \%1..\%9.
   - Default values are -1:-1:1.
   - One may use \%R iterators in any consumable resource (chunk or job wide).
   - An \%R iterator may be used for multiple resources.
     E.g., `ncpus=\%1:tesla=\%1;\%1=1-5`
   - \%R iterators will be evaluated in each \%C iteration, until the maximum of all \%R iterators is reached.
     E.g., \%1=1-10; \%2=1-5

Remarks

1. Loop-Nesting is: \%D, \%C, \%R.
2. \%C and \%D are not case sensitive.
3. Spaces are allowed in the resource and iterator specifications.
4. The order of iterator specifications does not matter.
5. Specifying an iterator which is not used is possible.
6. Multiple specifications of the same iterator is possible. Last match wins.
7. OpenCCS will print only valid results. If an iteration cannot be planned due to limitations or unavailable resources, it will be silently skipped.
8. Syntax errors are printed.
7.0.3 Examples

- Predict 5-10 chunks with Tesla GPUs, 5 cores and 30GiByte RAM, duration 1-2 hours.
  Refer to Example 7.0.1.

- Predict 1-10 nodes (stepping 2) exclusively, duration 1-4 hours, group benchmark.
  `ccsinfo -g benchmark -p 'n; %C=1-10:2; %D=1-4h'

- Predict 100-500 cores (stepping 100), duration 1-5 days stepping 2.
  `ccsinfo -p 'c %C; %C=100-500:100; %D=1-5:2d'

- Predict 1-16 cores with 4GB per core, duration 1 day, output in raw format.
  `ccsinfo --raw -p ncpus=%1:mem=%2G; %1=1-16; %2=4-64:4; %D=1d'

- Predict 1-16 cores with 4GB mem and 8GiByte vmem per core, duration 1 day.
  `ccsinfo -p 'ncpus=%1:mem=%2G:vmem=%3g; %1=1-16; %2=4-64:4; %3=8-128:8; %D=1d'
  Refer to Example 7.0.1.

- Predict 100-256 chunks with Matlab licenses, duration 75m.
  `ccsinfo -p 'n:ncpus=16:mem=30g,mdce=%2; %C=100-256:64; %2=4-64:4; %D=75m'
$ ccsinfo -p "%C:tesla=1:ncpus=5:mem=30g,place=scatter;%C=5-10;%D=1-2h"

<table>
<thead>
<tr>
<th>Duration Starts at</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h now</td>
<td>5:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>1h now</td>
<td>6:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>1h now</td>
<td>7:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>1h 13:40 (58m)</td>
<td>8:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>1h 13:40 (58m)</td>
<td>9:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>1h 13:40 (58m)</td>
<td>10:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h now</td>
<td>5:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h now</td>
<td>6:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h now</td>
<td>7:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h 13:40 (58m)</td>
<td>8:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h 13:40 (58m)</td>
<td>9:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
<tr>
<td>2h 13:40 (58m)</td>
<td>10:tesla=1:ncpus=5:mem=30g,place=scatter</td>
</tr>
</tbody>
</table>

$ ccsinfo -p "ncpus=%1:mem=%2G:vmem=%3g; %1=1-16; %2=4-64:4; %3=8-128:8; %D=1d"

ccsinfo: Using default group : ccsadmin

<table>
<thead>
<tr>
<th>Duration Starts at</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d now</td>
<td>ncpus=1:mem=4G:vmem=8g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=2:mem=8G:vmem=16g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=3:mem=12G:vmem=24g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=4:mem=16G:vmem=32g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=5:mem=20G:vmem=40g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=6:mem=24G:vmem=48g</td>
</tr>
<tr>
<td>1d now</td>
<td>ncpus=7:mem=28G:vmem=56g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=8:mem=32G:vmem=64g</td>
</tr>
<tr>
<td>1d 19:37 (4h22m)</td>
<td>ncpus=9:mem=36G:vmem=72g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=10:mem=40G:vmem=80g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=11:mem=44G:vmem=88g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=12:mem=48G:vmem=96g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=13:mem=52G:vmem=104g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=14:mem=56G:vmem=112g</td>
</tr>
<tr>
<td>1d 17:28 (2h13m)</td>
<td>ncpus=15:mem=60G:vmem=120g</td>
</tr>
</tbody>
</table>

**Example 7.0.1:** Example output of ccsinfo -p
Chapter 8

Checking Job and System Status

For checking the job and system status, OpenCCS offers the `ccsinfo` command line interface. The remainder of this section shows the most important options of `ccsinfo`. A complete list is offered by the man page (page 121).

8.1 Schedule Status

The schedule can be viewed via the option `<s|--schedule [sub-options]>[reqID...]`.

8.1.1 Summary

`ccsinfo `<s|--schedule> `--summary` 

Shows summarized schedule information. This sub-option is mutual exclusive to all other ones.

%ccsinfo -s --summ  
Policy: CCS  

<table>
<thead>
<tr>
<th>State</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>1023</td>
</tr>
<tr>
<td>Planned</td>
<td>457</td>
</tr>
<tr>
<td>Reservations</td>
<td>3</td>
</tr>
<tr>
<td>Allocating</td>
<td>0</td>
</tr>
<tr>
<td>Stopping</td>
<td>12</td>
</tr>
<tr>
<td>Stopped</td>
<td>378</td>
</tr>
<tr>
<td>Waiting</td>
<td>0</td>
</tr>
<tr>
<td>Hold</td>
<td>0</td>
</tr>
<tr>
<td>New</td>
<td>0</td>
</tr>
<tr>
<td>Backfilling</td>
<td>0</td>
</tr>
<tr>
<td>Replanning</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER 8. CHECKING JOB AND SYSTEM STATUS

Total 1870

8.1.2 Job Distribution

ccsinfo <-s|--schedule> --dist[=filter]
Shows information about the job distribution, i.e., how many jobs are in
which state. This sub-option is mutual exclusive to all other ones. Possible
filters are:

all Shows distribution for users and groups.
group Shows distribution for groups.
mime Shows the callers job distribution.
user Shows distribution for users.

filter is not case sensitive and may be abbreviated as long as the abbrevia-
tion is unique. Default filter is mine.

%ccsinfo -s --dist
User Total Run Planned Waiting Hold
===================================================
kel 573 214 345 0 10

%ccsinfo -s --dist=g
Group Total Run Planned Waiting Hold
===================================================
QCD 18 18 0 0 0
ROPOL 6 3 1 0 1
UBI 2 1 1 0 0
UBI2 15324 106 14896 1 321
VON 1 1 0 0 0

8.1.3 Filtering the Data

ccsinfo <-s|--schedule>[options] reqID .... scans for the given re-
qIDs. This disables all filters. Request names are not recognized.

If not scanning for specific reqID, one can use the following sub-options
to filter the output (which may be combined):

--group=<group[,....]>
Filters for the specified group(s).

--user=<account[,....]>
Filters for the specified account(s).

--mine Shows only information about current account.
8.1. SCHEDULE STATUS

```
--state=<state[,...]>
Filters for requests having a state in the given list. Possible
states: C (Completed), H (Hold), P (Planned), R (Running),
and W (Waiting)

--type=<type[,...]>
Filters for requests having a type in the given list. Possible
types: B (Batch), I (Interactive), and R (Reservations)

state and type are not case sensitive and may be abbreviated as long as
the abbreviations are unique.
```

8.1.4 Formatting the Output

The following sub-options are available to format the output (they may be
also combined):

```
--fmt=<field[ ...]>
Shows only specified fields.
Syntax of field string is: '%.NX'
.     right justification (optional)
N    sizeof field (optional)
X    the field specifier.

Example: --fmt="%.R %T %w %.10z %P %50j"
The following fields are available. Fields marked with '(*)' are
accessible only to the request owner or the administrator. If a
field is not accessible the output is 'N/A'.

C    Command line call (*)
D    Duration
E    Given Deadline
F    Submitted from (*)
G    Group
J    Job notification (*)
M    Mail address(*)
N    Request name
O    Owner
P    Planned start time
R    ReqID
S    Given start time
T    Type
CHAPTER 8. CHECKING JOB AND SYSTEM STATUS

U User interface (*)
V Event notification (*)

a Attributes
  • A: Mapping at Allocation
  • B: Background Priority
  • D: Dynamic Limit Extension
  • F: Freepool Impact
  • L: Limits are checked at runtime
  • M: Multihost
  • S: Small-Job (Mapped on “Local only” nodes)

'-' denotes that the attribute is not set.

b Command (*)

c Core Efficiency(*)
  Shows \( \frac{cput}{walltime} \) and \( \frac{cput - ncpu \times cput}{walltime} \) in percent.
  Accuracy depends on the received values from the
  nodes which sample and send the data in an admin-
 istrator defined interval (e.g., each 10s). Addition-
 ally, in some cases OpenCCS is not able to
  sample all job resource usage data if the job is us-
  ing more than one node. Hence, real values may
  be sometimes higher.

d Percent Done

e STDERR (*)
i STDIN (*)
j Job resource set
m Mapping
n Node resource set
o STDOUT (*)
p Join (*)
q Priority
r RSV-ID
t Trace file (*)
u User resource set
v Elapsed time
w State
x Sub-state
y Release time
8.2. SYSTEM STATUS

z Submission time
--lines=<#> Limits the number of found requests to the given number.
--raw Prints the result in a raw format: No headline, no field formatting. Fields are separated by ‘ ’.

8.1.5 Examples
The following examples asks for information about jobs which were submitted by the caller, which are in state Running or Planned, which are of type Batch, and which are assigned to the group foo or bar.

%ccsinfo -s --mine --state=R,p --type=b --group=foo,bar --lines=3
reqID Name Account State Start Walltime Job-Resource-Set
==============================================================================
  163 ccsHAWAI12B kel PLANNED 18:57:19.04.12 145d vmem=50g,ncpus=17
  180 ccsHAWAI16B kel PLANNED 19:11:19.04.12 37h30m mem=512m,ncpus=1
  160 kel_2 kel ALLOCATED 17:57:19.04.12 1h vmem=1t,ncpus=4

%ccsinfo -s --mine --state=r,p --type=b --group=foo,bar --lines=3 --raw
  163 ccsHAWAI12B kel PLANNED 18:57:19.04.12 145d vmem=50g,ncpus=17
  180 ccsHAWAI16B kel PLANNED 19:11:19.04.12 37h30m mem=512m,ncpus=1
  160 kel_2 kel ALLOCATED 17:57:19.04.12 1h vmem=1t,ncpus=4

8.2 System Status

8.2.1 Node Status

ccsinfo <-n, --nodeinfo> [--summary | --state=<state> | <node, ...>] shows information about the listed nodes.

Summary
Using --summary shows a summarized information about the node states.

%ccsinfo -n --summary
State   Count
================================
ok       650
offline   2
down     1
down/offline   0
unknown   0
Total     653
Nodes in Use/Exclusive  451/109
CHAPTER 8. CHECKING JOB AND SYSTEM STATUS

Detailed Information

The option `--nodeinfo` [node, ...] shows information about the listed nodes. Giving no node name, shows information about all nodes.

```
% ccsinfo -n kel123
kel123
  rectime = 18:30:23
  status = up,online
  coordinates = 0,0,0
  running jobs = 345,56,7
  uptime = 1d4h28m59s since Wed Mar,19 2014 14:21
  uname = Linux kel123 2.6.32-35-generic #78-Ubuntu SMP i686
  ncpus = 2
  totmem = 1652472k
  vmem = 1652472k
  availmem = 840896k
  physmem = 1025980k
  loadave = 0.68
  sessions = 5906 5756 2478 5815 5853 5880 5890 5899
  nsessions = 8
  nusers = 2
  idletime = 0
  only local jobs = false
```

Filtering the Data

Using `--state=<state>` shows only nodes having a specific state. Possible states are:

- **all**: Does not filter, prints them all.
- **sick**: Filters for nodes which are in trouble.
- **ok**: Filters for nodes which are not in trouble.
- **up**: Filters for nodes which are in state UP.
- **down**: Filters for nodes which are in state DOWN.
- **online**: Filters for nodes which are in state ONLINE.
- **offline**: Filters for nodes which are in state OFFLINE.
- **unknown**: Filters for nodes which are in state UNKNOWN.

**state** is not case sensitive and may be abbreviated as long as the abbreviation is unique.
### 8.2. SYSTEM STATUS

```bash
%ccsinfo -n --state=of
```

<table>
<thead>
<tr>
<th>Host</th>
<th>State</th>
<th>Running Jobs</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel245</td>
<td>up,offline</td>
<td>34,45</td>
<td>will be rebooted</td>
</tr>
</tbody>
</table>

Using `--reqid=<reqID>` shows only nodes assigned to reqID....

```bash
%ccsinfo -n --reqid=1356
```

<table>
<thead>
<tr>
<th>Host</th>
<th>State</th>
<th>Running Jobs</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel15</td>
<td>up,online</td>
<td>1356,45</td>
<td></td>
</tr>
<tr>
<td>kel78</td>
<td>up,offline</td>
<td>1356</td>
<td>defect hard disk</td>
</tr>
</tbody>
</table>

### Formatting the Output

The following sub-options are available to format the output. They may be combined and used together with `--reqid` or `--state`.

`--fmt=<field[ ...]>`

- Shows only specified fields.
- Syntax of field string is: `%.NX`
  - `.` right justification (optional)
  - `N` sizeof field (optional)
  - `X` the field specifier.

`Example: `--fmt="%.A %p %50i"

The following fields are available:

- `A` Available memory
- `C` Number of cores (ncpus)
- `H` Hostname
- `J` Running jobs
- `M` Physical memory
- `L` Load
- `N` Note
- `O` Uptime
- `S` Status
- `U` Uname
- `V` Virtual memory
- `a` Architecture
- `c` Coordinates
CHAPTER 8. CHECKING JOB AND SYSTEM STATUS

<table>
<thead>
<tr>
<th>i</th>
<th>Idletime</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Properties</td>
</tr>
<tr>
<td>m</td>
<td>Minimum Resources</td>
</tr>
<tr>
<td>r</td>
<td>Record time</td>
</tr>
<tr>
<td>s</td>
<td>Sessions</td>
</tr>
<tr>
<td>t</td>
<td>Number of sessions (nsessions)</td>
</tr>
<tr>
<td>u</td>
<td>Number of users (nusers)</td>
</tr>
</tbody>
</table>

If a field is not accessible the output is 'N/A'.

--raw

Prints the result in a raw format: No headline, no field formatting. Fields are separated by ' '.

8.2.2 Available Workers

The option --worker shows the system specific available workers:

% ccsinfo --worker -i HAWAII
ccsinfo: HAWAII provides the following workers:
ccsinfo: Refer also to the man page ccsworker(1) or
call 'ccsinfo --whelp=<worker>'
HAWAII provides the following workers:

<table>
<thead>
<tr>
<th>Worker</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>abaqus</td>
<td>starts an ABAQUS application</td>
</tr>
<tr>
<td>g03</td>
<td>starts a Gaussian-03 application</td>
</tr>
<tr>
<td>g09</td>
<td>starts a Gaussian-09 application</td>
</tr>
<tr>
<td>mpich2</td>
<td>starts an MPICH2 application</td>
</tr>
<tr>
<td>mvapich</td>
<td>starts an MVAPICH application</td>
</tr>
<tr>
<td>ompi</td>
<td>starts an OpenMPI application</td>
</tr>
<tr>
<td>starccm</td>
<td>starts a STAR-CCM+ application</td>
</tr>
<tr>
<td>turbomole</td>
<td>starts a Turbomole application</td>
</tr>
</tbody>
</table>
8.2.3 Allocatable Resources

The option `--allocatable` shows the allocatable resources. This list comprises the built-in and the customized resources. The column **Type** represents the resource format as described in 5.2.

- 'A' is String
- 'B' is Boolean
- 'D' is DateTime
- 'S' is Size
- 'T' is Timespan
- 'U' is Unitary
- 'V' is String Array.

The column **Flags** represents the resource categories as described in 5.1.

- 'C' marks a consumable resource
- 'D' marks a dynamic resource
- 'J' marks a job wide resource
- 'N' marks a non alterable resource.

The column **Amount** prints the used, online, and maximum amount of the related resource. The online amount depends on the availability of the nodes. The column **Default** prints the system default value. **N/A** means “Not Available”.

```bash
%ccsinfo -a
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type, Amount</th>
<th>Default</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncpus</td>
<td>U,C 7993/9456/9568</td>
<td>1</td>
<td>number of cores</td>
</tr>
<tr>
<td>nodes</td>
<td>U,C 294/589/614</td>
<td>1</td>
<td>number of nodes</td>
</tr>
<tr>
<td>mem</td>
<td>S,C 18.69t/40.12t/40.74t</td>
<td>3g</td>
<td>physical memory</td>
</tr>
<tr>
<td>vmem</td>
<td>S,C 22.33t/49.09t/49.81t</td>
<td>2g</td>
<td>virtual memory</td>
</tr>
<tr>
<td>cput</td>
<td>T, -</td>
<td>N/A</td>
<td>CPU time</td>
</tr>
<tr>
<td>walltime</td>
<td>T,J</td>
<td>N/A</td>
<td>walltime</td>
</tr>
<tr>
<td>hostname</td>
<td>A, -</td>
<td>N/A</td>
<td>hostname</td>
</tr>
<tr>
<td>arch</td>
<td>A, -</td>
<td>N/A</td>
<td>host architecture</td>
</tr>
<tr>
<td>mpirproc</td>
<td>U, -</td>
<td>N/A</td>
<td>number of mpi processes per chunk</td>
</tr>
<tr>
<td>ompthreads</td>
<td>U, -</td>
<td>N/A</td>
<td>number of threads per chunk</td>
</tr>
<tr>
<td>acc</td>
<td>B, -</td>
<td>N/A</td>
<td>node with accelerator card</td>
</tr>
<tr>
<td>norm</td>
<td>B, -</td>
<td>N/A</td>
<td>64GB compute node</td>
</tr>
<tr>
<td>phi</td>
<td>U,C 0/5/8</td>
<td>N/A</td>
<td>Intel Xeon Phi card</td>
</tr>
<tr>
<td>smp</td>
<td>B, -</td>
<td>N/A</td>
<td>SMP node</td>
</tr>
<tr>
<td>tesla</td>
<td>U,C 31/31/32</td>
<td>N/A</td>
<td>Tesla K20xm card</td>
</tr>
<tr>
<td>sw</td>
<td>A,CJ</td>
<td>N/A</td>
<td>Software</td>
</tr>
</tbody>
</table>

The option `--allocatable --classes` shows allocatable resource classes.
The column #Hosts prints the online and maximum number of hosts.

```bash
% ccsinfo -a --classes
Name       Class      #Hosts
           Online/Max
ncpus      16          587/594
            32          2/2
nodes      1           589/614
mem        63g         576/582
           1009g        2/2
           252g         11/12
vmem       84g         31/31
           78g         545/551
           1t          2/2
           267g        11/12
arch       SL 6.3      589/614
           CENTOS-5.2  25/614
acc        false       558/582
           true        31/32
norm       false       49/62
           true       540/552
phi        1           5/8
smp        false       587/612
           true       2/2
tesla      1           31/32
wash       false       578/594
           true       11/20
sw         g03         -
```

### 8.2.4 FreePools

The option --freepools shows the defined FreePools:

```bash
% ccsinfo --freepools
name= CPUS
resource = ncpus
quantity = 1/50%
allowed = count: 5, runtime: 2h
validity = * 10-20 * * *
name= PHYSICS
resource = ncpus
quantity = 50
allowed = users:kel || groups:+phys || count: 5, runtime: 2h
validity = always
```
For a description of the rows refer to 2.4.3

8.3 Group / User Related Infos

8.3.1 Group Membership

The option --groups shows a list of groups the caller is member of.

%ccsinfo --groups
Groups: ccsadmin,FoO,pc2guests

8.3.2 Limits and Privileges

The option --limits shows the limits and privileges. Both are assigned to a group and/or a user.
If not using the sub options -g <group> and --user=<user> the CLI takes the default values of the caller.
Using --user=ALL, shows the group data and all members of the group, having an own specification.

%ccsinfo -l -g pc2guests
Active policy for jobs exceeding their resource credits is: Reject the job.

Group-Data
==============
name : pc2guests
validity : always
privileges : alter, interactive, reserve
members : +pc2guests, arnie

Resource Limits:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Items</th>
<th>Duration</th>
<th>Area</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>unlimited</td>
<td>315d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>mdce</td>
<td>256</td>
<td>315d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>tesla</td>
<td>10</td>
<td>120d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>ncpus</td>
<td>1800</td>
<td>21d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>jobs</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arrayjobs</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alteration limits if request is in state ALLOCATED:

<table>
<thead>
<tr>
<th>What</th>
<th>Limit</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>walltime</td>
<td>10h/10%</td>
<td>always</td>
</tr>
</tbody>
</table>
Resource Credits (in hours:mm:ss)
Only resources with a specified credit are printed

<table>
<thead>
<tr>
<th>Resource</th>
<th>Credit</th>
<th>Used-Credit</th>
<th>Remaining-Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdce</td>
<td>1000:00:00</td>
<td>0:00:18</td>
<td>999:59:42</td>
</tr>
<tr>
<td>tesla</td>
<td>100:00:00</td>
<td>0:03:00</td>
<td>99:57:00</td>
</tr>
<tr>
<td>ncpus</td>
<td>2500000:00:00</td>
<td>0:03:00</td>
<td>2499999:57:00</td>
</tr>
</tbody>
</table>

For a description of the columns, refer to section 2.4.2.

The alteration limit for walltime is related only to already running jobs. It is not valid for jobs which are not yet running. In this example, it means that the user can extend the runtime of a running job at most to the maximum of 1 hour and 10% of the initial maximum duration. If for example the initial maximum duration was 10 days, then the user may extend the runtime of the running job at most to 11 days. If the initial maximum duration was 10 minutes, then the user may extend the runtime of the running job at most to 1 hour and 10 minutes.

**Privilege**

A privilege specifies which actions are allowed for a consumer. The following privileges are available:

- `a` alter jobs
- `i` submit interactive jobs
- `l` is locked
- `r` reserve resources

**8.3.3 Default and Force Values**

The option `--defaults` shows default and force values. Both are assigned to a group and/or a user.

- **Attribute** (the first column) describes the attribute.
- **Default** (the second column) describes the default value. It is taken, if the caller did not specify the attribute in question.
- **Force** (the third column) shows values which overwrite user given values or will be taken as a default.

The administrator may assign defaults to specific users, groups, or the whole system. If both Default and Force are specified, Force will be taken. If not using the sub options `-g NAME` and `--user=USER`, the CLI shows the default values valid for the caller’s default group.

**Example:**
8.4. REQUEST STATUS

%ccsinfo --def
Attribute Default Force
===================================================================
mem 128m
mdce 128
place free:shared

8.4 Request Status

ccsinfo <req_identifier> shows detailed information about the specified request(s). This data will be available for a site specific interval (often 30m) before OpenCCS removes the job completely from its runtime database. Thereafter, the command ccstracejob may be used to print job data.

%ccsinfo 29308
Request-ID : 29308
Name : kel_3
Owner : kel
Group : Foo
Type : Batch
Priority : 1
CLI call : --group=foo go9 -- Scan.com
Submitted from : /pc2/work/kel/2D
Start Time : None
Deadline : None
Submission Time : 13:18
Allocation Time : 13:18
Maximum Runtime : 2w
Release Time : 13:18:03.05 (in 1w6d18h27m)
State : ALLOCATED since 55m56s
User Resource Set : 2:ncpus=1:mem=36g,place=scatter:excl
Job Resource Set : exclnodes=2,mem=124g,vmem=157g,ncpus=32,mpiprocs=32, place=scatter:excl
Chunks : 2:mem=36g:ncpus=1
Mapping : node513:=mem=62g:ncpus=16,node45:=mem=62g:ncpus=16
Event-Notification : abe---
Emails goto : kel@hell.org
CMD : g09 -- Scan.com
Job notifying : Off
Trace file : None
STDIN : redirected from : /dev/null
STDOUT : redirected to : /pc2/work/kel/2D/Scan.log
STDERR : redirected to : /pc2/work/kel/2D/Scan.log
Stream Joining : n
Resource-Usage :
Item cput mem vmem walltime
======================================================================
Summary 22h1m56s 11.44g 25.89g 55m56s
node45 13h27m1s 5.77g 13.13g 55m46s
node513 8h34m55s 5.68g 12.77g 55m56s
Chapter 9

Working with OpenCCS Jobs

9.1 Altering Scheduled Requests

Nearly all attributes of a request (job or reservation) may be altered after submission, using the `ccsalter` command. The syntax is:

```
ccsalter <options> <req_identifier...>
```

Several requests can be altered simultaneously. If one value cannot be altered for a request, the whole alteration of this request fails. The resource definition can be done using the same shortcuts as in `ccsalloc`, e.g. `-c` or `-n`. New values have to be given as absolute values. E.g., resources must not be specified relatively (e.g., `-c+2`). Exceptions are the time related attributes:

- **start time**: `-a 0` or `-s 0` both remove the (minimum) start time.
- **runtime**: Accepts an absolute value or `<+->timespan`.
- **stop time**: `-e 0` removes the end time.

Table 9.1 depicts what can be when altered.

Table 9.1
Table 9.1: Which job attribute can be when altered.

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>–after</td>
<td>PLANNED, WAITING</td>
<td>1, 2</td>
</tr>
<tr>
<td>–allowed</td>
<td>always</td>
<td>Reservations only. Valid only for new jobs.</td>
</tr>
<tr>
<td>–cwd</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–duration</td>
<td>always</td>
<td>1, 2. A longer duration may be denied if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The sum of all requested prolongations exceeds the limit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Planned jobs, which are not best-effort jobs, would be delayed.</td>
</tr>
<tr>
<td>–endtime</td>
<td>PLANNED, WAITING</td>
<td>1, 3</td>
</tr>
<tr>
<td>–group</td>
<td>PLANNED, WAITING</td>
<td>1, 2</td>
</tr>
<tr>
<td>–hold</td>
<td>PLANNED, WAITING</td>
<td>4</td>
</tr>
<tr>
<td>–join</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–mail</td>
<td>always</td>
<td></td>
</tr>
<tr>
<td>–name</td>
<td>always</td>
<td></td>
</tr>
<tr>
<td>–notifyjob</td>
<td>always</td>
<td></td>
</tr>
<tr>
<td>–notifyuser</td>
<td>always</td>
<td></td>
</tr>
<tr>
<td>–res</td>
<td>PLANNED, WAITING</td>
<td>1, 2</td>
</tr>
<tr>
<td>–resume</td>
<td>WAITING</td>
<td>4</td>
</tr>
<tr>
<td>–rsvid</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–starttimes</td>
<td>PLANNED, WAITING</td>
<td>1, 3</td>
</tr>
<tr>
<td>–stdin</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–stderr</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–stdout</td>
<td>PLANNED, WAITING</td>
<td></td>
</tr>
<tr>
<td>–tracefile</td>
<td>always</td>
<td>Alter message is written to the new file.</td>
</tr>
</tbody>
</table>

Notes:
1: For a reservation, only if no job is running in the reservation.
2: For a job array, only if no job was already started in the job array.
3: Not allowed for job arrays.
4: Not for reservations.
9.2 Holding / Resuming Jobs

Using the command `ccsalter`, one can put a request to state hold. Hold means, the job is ignored in planning.

Syntax is: `ccsalter --hold [-m MESSAGE] request\_id ...
-m MESSAGE` will notify the user (depending on the notify flags).

`ccsalter --hold` cannot be used with other `ccsalter` options.

Holding a reservation or single job-array subjobs is not possible.

A job in state hold can be altered, killed, and resumed.

Resuming means, the job is planned again and may start running.

Resuming a job is done by `ccsalter -r|--release request\_id ...
ccsinfo --state=hold` prints jobs in state hold.

`ccsinfo --dist` prints a column for jobs in state hold.

`ccsinfo --summary` prints the total number of jobs in state hold.

9.3 Sending Signals to Jobs

Using the command `ccssignal`, one can send a signal to running jobs. The signal is sent to the session leader of the job on the boot node. The syntax is:

`ccssignal <signal> <req_identifier>[...]`

**signal** Can be given as: `[-]<digit>` or `[-][SIG]<signal>.

**req_identifier** ...

The request-ID(s) or request-name(s). They can be mixed.

Signaling a job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state or exiting.
- The requested signal is not supported.

Two special signal names, `suspend` and `resume`, (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources. Admin privilege is required to suspend or resume a job.

Examples:

- `ccssignal -9 123`
- `ccssignal 9 123`
- `ccssignal -KILL 123`
- `ccssignal SIGKILL 123`

All examples above send the signal SIGKILL to the job 123
9.4 Sending Messages to Jobs

Sending a message to a job means that OpenCCS writes a message string into one or more output files of the job. Typically, this is done to leave an informative message in the output of the job. Such messages can be written using the command `ccsmsg`. The syntax is:
```
ccsmsg [-e] [-o] <msg> <req_identifier> [...]
```

- `e` Write message to stderr (default)
- `o` Write message to stdout
- `msg` Message to send. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the jobs file.
- `req_identifier` ... The request-ID(s) or request-name(s). They can be mixed.

9.5 Deleting Requests

OpenCCS provides the command `ccskill` for deleting jobs or reservations. It deletes in the order in which the request identifiers are presented to the command.

Syntax: `ccskill [options] <req_identifier ...>`
The following options are available:

- `--all` May be used to kill all owned jobs. This also valid for group managers and Administrators. The higher privilege is ignored if using “--all”.
- `f` Interactive jobs which are connected to a user interface (e.g. `ccsalloc` or `ccsbind`), Can be killed using this force parameter.
- `m,--message <MESSAGE>`
  If given MESSAGE will be sent to the owner of the request. If a tracefile is assigned to the request, MESSAGE will also appear in that file.
- `req_identifier` ...
  The request-ID(s) or request-name(s). They can be mixed.
Chapter 10

Reservations

10.1 Submitting a Reservation

The `ccsalloc` command can also be used to submit a reservation. Reservations can be submitted using the parameter `-s`, `-e` and `-t`, at which:

- `-s` Determines the starting time of the reservation. Format: Datetime.
- `-e` Is the end time of the reservation. Format: Datetime.
- `-t` Is the runtime of the reservation. Format: Timespan.

Two of these parameters have to be set to submit a valid reservation. The optional parameter `--allowed` can be used to allow other users (or groups) to use the reservation. It expects a comma separated list of users or groups. **Note:** If jobs submitted to the reservation should be able get nodes exclusively, you have to reserve them exclusively.

Examples:

- Reserve 10 arbitrary nodes, starting at 8 pm, for 2 hours:
  ```
  ccsalloc -s 800 -t 2h -n 10
  ```

- Reserve 10 nodes (each with 32 cores), starting at 8 pm, for 2 hours:
  ```
  ccsalloc -s 800 -t 2h --res=rset=10:ncpus=32,place=scatter:excl
  ```

- Reserve 10 cores, starting in one hour, ending at 11 pm. In addition to the submitting user `user1` all members of the group `group1` may use the reserved resources:
  ```
  ccsalloc -s +1h -e 23:00 -c 10 --allowed="user1,group1"
  ```

After a successful submission, `ccsalloc` returns a reservation-id (rsvid), which has to be known for using the reserved resources.
10.2 Using a Reservation

A previously made reservation can be used by setting the --rsvid option of ccsalloc. If the job does not fit in the reservation’s duration it is rejected. The resources from one reservation can be split to different jobs.

Examples:
ccsalloc -t 2h -n 7 --rsvid=123 myjob.sh
ccsalloc -t 10m -c 5 --rsvid=123 myjob.sh

10.3 Altering a Reservation

A reservation can be altered like any other request. However, some things are different. It is not possible to alter time or resource dependent attributes, while a job is running. After altering time or resource dependent attributes, all related jobs are re-planned. If it is not possible to plan such a job it is switched to the state WAITING.

10.4 Deleting a Reservation

Use ccskill to delete a reservation. If there are related jobs planned or running, these jobs are deleted first.
Chapter 11

Job Arrays

11.1 Introduction

Job arrays may be used to group closely related work into a set so that you can submit, query, modify, and display the set as a unit. Job arrays are useful whenever you want to run the same program over and over on different input files. Each job in a job array is called a subjob.

All subjobs have the same attributes, including resource requirements, limits, and scheduling priority. The scheduler handles each subjob in a job array as a separate job. The given executable is run once for each subjob and may invoke different commands based on the subjob index.

Subjobs are scheduled and treated like normal jobs, with the exceptions noted in this chapter.

Each subjob has its own reqID. The array itself also has a reqID. A subjob may be specified by its reqID or its subjob identifier.

11.2 Glossary

Job Array is a container for a collection of similar jobs submitted under a single reqID. It can be submitted, queried, modified, or displayed as a unit. The jobs in the collection are called subjobs.

Job Array Identifier The reqID returned when submitting a job array, e.g., 1234.

Job Array Range A set of subjobs within a job array.

Subjob Individual entity within a job array.

Subjob Index The unique index which differentiates one subjob from another. This must be a non-negative integer. E.g., 1234[7], where 1234[] is the job array itself, and 7 is the index.
Subjob Identifier (SJID)

A SJID identifies one or more subjobs.

Syntax: reqID[ID] where reqID is the request-ID of the job array and ID may comprise comma separated job array ranges. E.g., 1234[5] or 1234[1-8:2,26]. The syntax is explained in 11.3

11.3 Identifier Syntax

To identify the job array itself use the reqID returned by ccsalloc. Identifying subjobs is done by the subjob index or by subjob ranges.

Syntax: <first>-<last>[<stepping factor>], ...

- first, last, and stepping factor must be integers \( \geq 0 \).
- stepping factor defaults to 1.
- <first>-<last> may be a single number to allow different indexes.
- last must be greater equal than first.
- If last is not a multiple of stepping factor above first, it will not be used as an index value, and the highest index value used will be lower than last.

Since some shells, for example csh and tcsh, read ‘[’ and ’]’ as shell metacharacters, job array names and subjob names should be enclosed in double quotes for all OpenCCS commands.

11.3.1 Examples

- 1234 or 1234[] is the job array.
- 1234[X] is the sub-job with index X.
- 1234[X-Y:Z] are the sub-jobs with indexes X to Y with stepping Z.
- 1234[X-Y:Z,A-B:C] are sub-jobs with indexes X to Y with stepping Z and indexes A to B with stepping C.
- 1234[1-8:3] results in indexes 1,4, and 7.
11.4 Environment Variables set by OpenCCS

For each subjob OpenCCS sets two environment variables on the boot node:

- **CCS_ARRAY_INDEX**
  The subjob index of the job in the array, e.g., 2.

- **CCS_ARRAY_ID**
  The request-ID of the job array, e.g., 1234.

11.5 Limits

The administrator may specify two limits relevant to job arrays:

- **Number of subjobs**
  limits the number of subjobs in a single job array.

- **Number of jobs**
  limits the total number of jobs for a user or a group (including all subjobs). This limit overrules the number of subjobs limit.

11.6 Submission

A job array is submitted like a normal job by using the `ccsalloc` command. The subjob indexes are specified by the parameter `-J <SJID>`. The subjob-identifier follows the syntax described in 11.3.

Example: `ccsalloc -J '17-100:3, 128' myJob.sh`

If the job array comprises more than 500 subjobs, `ccsalloc` prints all 500 planned subjobs a progress message while planning the subjobs. One can interrupt the `ccsalloc` command by typing `CTRL-C`, but OpenCCS will continue to plan the job array. Use `ccskill` to kill a job array.

Caveats

- Job arrays with interactive subjobs are not allowed.

- Specifying a fixed start time (`ccsalloc -s`) or a deadline (`ccsalloc -e`) is not allowed.

- Specifying an earliest start time (`ccsalloc -a`) is possible.

11.7 File Naming

The file names are built as described in 6.5.6. The default file names for subjobs are:
CHAPTER 11. JOB ARRAYS

STDIN  /dev/null, which means no input.
STDOUT  <job name>.%A.%a.out
STDERR  <job name>.%A.%a.err

%A is the reqid of the job array.
%a is the subjob index.
%A and %a may be used as placeholders while specifying the file names.
E.g., ccsalloc -o MYLOG-%A.%a.out.%a will create a file where %A is replaced
by the reqID of the job array and %a is replaced by the subjob index. If the
job is no subjob, then %A and %a will be replaced by the job’s reqID.

11.8 Tracefiles

Subjobs do not have an own tracefile. OpenCCS logs all events (including
the subjob events) in the job array tracefile. For subjobs, the field reqID is
then replaced by the subjob identifiere (SJID).

11.9 Exit Status

The exit status of a job array is determined by the status of each of the
completed subjobs. It is only available when all subjobs have completed.

0   All subjobs of the job array returned an exit status of 0. No
    OpenCCS error occurred. Deleted subjobs are not considered.

1   At least 1 subjob returned a non-zero exit status. No OpenCCS
    error occurred.

2   A OpenCCS error occurred.

11.10 Checking Status

This section describes the differences to the normal behaviour of the ccsinfo
command. For detailed information about ccsinfo refer to 8.

ccsinfo -s --summary
    Shows the number of job arrays in the system.

ccsinfo -s --dist
    Job arrays itself are not counted, only subjobs.

ccsinfo -s
    • Subjobs in state PLANNED are not shown. Only the job
      array itself is listed.
    • To see only job arrays, use ccsinfo -s --type=array
11.11. ALTERING

- To see all subjobs of job array 1234, use `ccsinfo -s 1234[]`
- To see specific subjobs of job array 1234, use `ccsinfo -s 1234[SJID]`. Where SJID is a subjob identifier as described in 11.3. Unknown indexes are ignored.
- The parameter `--fmt=%s` will show the subjob index.
- The parameter `--fmt=%d` will show the share of completed jobs of a job array in percent. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a (sub)job, it is the time used divided by the time requested.
- all ccsinfo filters are applicable.

```bash
ccsinfo <reqID>
```

The detailed information about a job array shows a summary of the states of all subjobs:

- Number of subjobs: 2000
- Completed subjobs: 137
- Running subjobs: 57
- Planned subjobs: 1806
- Waiting subjobs: 0

**Examples** `ccsalloc -J'1-10:2,36' myJob.sh` will create a job array with this subjob-IDs: 1,3,5,7,9,36. If we assume the reqID of the job array is 1234, then:

- `ccsinfo -s 1234[1-100:3,36]` will show only subjobs 1, 7, and 36.
- to show information about the subjobs of job array 1234 with indexes 1,7,9 and state ALLOCATED or PLANNED use `ccsinfo -s 1234[1,7,9] --state=r,p`.
- `ccsinfo -s 1234[] --state=r` will show all running subjobs.

### 11.11 Altering

As like normal jobs one can change nearly all characteristics of a job array. However, one cannot alter single subjobs. Only the whole job array can be altered. For detailed information about `ccsalter` refer to 9.1.

If the job array comprises more than 500 subjobs, `ccsalter` prints all 500 subjobs a progress message while altering the subjobs. One can interrupt the `ccsalter` command by typing `CTRL-C`, but OpenCCS will continue to alter the job array.
CHAPTER 11. JOB ARRAYS

Caveats

- Altering the job-indexes (-J) is not possible.
- Specifying a fixed start time (ccsalloc -s) or a deadline (ccsalloc -e) is not allowed.
- Altering the resources (--res), the group (-g), or the maximum runtime (-t) is only possible if there were no subjobs started.

11.12 Holding/ Resuming

Using ccsalter one can hold / resume whole job-arrays. Holding a job-array affects only subjobs in state PLANNED. Subjobs in state hold are counted to state WAITING if printing detailed job-array information.

11.13 Killing

Killing the whole job array is done by using ccskill <reqID>. If the job array comprises more than 500 subjobs, ccskill prints all 500 subjobs a progress message while killing the subjobs. One can interrupt the ccskill command by typing CTRL-C, but OpenCCS will continue to kill the job array. Killing subjobs can be done by using ccskill <SJID>. Unknown indexes are ignored.
Example: ccskill 1234[1-100:3, 45-90:5].

11.14 Signalling

Sending a signal to all running subjobs of a job array use ccssignal <signal> <req_identifier>[...].
Sending a signal to specific subjobs of a job array use ccssignal <signal> <SJID>[...]. Unknown indexes are ignored.

11.15 Sending Messages

Sending a message to all running subjobs of a job array use ccsmsg [-e] [-o] <msg> <req_identifier>[...].
Sending a signal to specific subjobs of a job array use ccsmsg [-e] [-o] <msg> <SJID>[...]. Unknown indexes are ignored.

11.16 User Notification

Using the ccsalloc parameter -mb will send a mail if the first subjob of the job array started.
Using the \texttt{ccsalloc} parameter \texttt{-me} will send a mail if all subjobs are finished. Please note, that if using email notification OpenCCS will \textit{not} send mails for each subjob. One can activate email notification for subjobs by using the \texttt{ccsalloc} parameter \texttt{-m} switch \texttt{s}. Refer also to \textbf{6.5.3}.

\section*{11.17 Job Arrays in Reservations}

Job arrays may be submitted / altered to a reservation like normal jobs. However, altering a job array or the reservation may lead to subjobs in state \textsc{WAITING} if not all subjobs can be planned within the reservation interval.
Appendix A

ccsalloc Man Page

A.1 SYNOPSIS

1. `ccsalloc [options] [job_file [job_file_args]]`
2. `ccsalloc [options] [ worker [worker_args] – ] [job [job_args]]`
3. `ccsalloc [options] [job [job_args]]`

A.2 DESCRIPTION

`ccsalloc` is used to submit jobs, job arrays, or reserve resources managed by CCS. You can submit batch jobs, interactive jobs, start an interactive session, or reserve resources in advance. For a job submission, CCS needs this information:

- The resources to allocate.
- How long the job will run.
- The corresponding executable.

A.2.1 Script Jobs

Jobs can be submitted using `ccsalloc` (e.g., `ccsalloc job.sh`). If job.sh is a script file, it may contain directives describing the job, followed by the job itself. Directives can be set by adding comments with the following syntax to the job script:

```
#CCS OPTION [VALUE].
```

If directives are given multiple times, first match wins. The possible options in the script directives are the same options as the `ccsalloc` command line options.
A.2.2 Jobs Without a Job Script

Jobs can be submitted with the following syntax:
\`ccsalloc [args] <worker> [worker_args][--] <job> [job_args]\`

Jobs are regularly executed as batch jobs. The specification of the used island has to be the first element of the [args] list.

A.2.3 Simple Jobs

Simple jobs may also be submitted without selecting a worker by calling \`ccsalloc <job>[job_args].\`

A.2.4 Interactive Jobs

By using the option -I one can submit interactive jobs. The -I option is ignored in a script directive. The streams STDIN, STDOUT, and STDERR are then connected to the submitting terminal. When the job starts to execute, all input to the job is from the terminal session in which ccsalloc is running. When an interactive job is submitted, the ccsalloc command will not terminate when the job is submitted. It will remain running until the job terminates, or is aborted. When connected to the job, the terminal characteristics are changed in the following way:

\texttt{Control-C} Sends the signal SIGINT to the job.

\texttt{Control-Z} Acts as usual with one exception. If you type bg in your Unix-shell to put the job into the background, the output (STDOUT, STDERR) will be redirected as described in section 'THE VIRTUAL TERMINAL'.

\texttt{^S, ^Q} will act as usual.

A.2.5 Reservations

One can reserve resources in advance and then submit jobs to the reserved resources. A reservation consists of three components. The resource specification, the time specification, and optionally a list of users and/or groups which may use the reserved resources.

A.2.6 Job Arrays

Job arrays may be submitted by:
\`ccsalloc -J '<first>-<last>[:<stepping],...]' <job>[job_args]\`

For detailed syntax refer to option -J below or the CCS 'User Manual'.
A.2.7 Specifying Resources

Syntax

The user may specify resources by:

```
ccsalloc --res="resource_name=[value],[resource_name=[value],...]
```

`resource_name` is the name of an allocatable resource (which is generic or system dependent). `ccsinfo` shows the allocatable resources. A resource name:

- Is not case sensitive.
- May include white spaces between `=` or `,`.
- May be a resource set specification, a placement specification, or a job wide resource specification.

The option `--res` may be used serveral times. The parameters will be concatenated.

Resource Set / Chunk Specification

A resource set (also named chunk) specifies a set of resources that have to be allocated as a unit on one node. Chunks cannot be split across nodes. Resource sets are specified using the keyword "rset".

**Syntax:** `rset=[N:]chunk[+[N:]chunk...]`

If N is not specified, it is set to 1. A chunk comprises one or more `res=value` statements separated by a colon.

**Examples:**

- `ncpus=2:mem=10g:hostname=Host1`
- `ncpus=27:vmem=20g:arch=linux+4:acc=fpga`

Placement Specification

This specification controls how the chunks are placed on the nodes.

**Syntax:** `place=[arrangement][:sharing][:grouping][:ignore]`

- Arrangement is one of free, pack, or scatter.
- Sharing is one of excl or shared.
- Grouping can have only one instance of group=resource.
- Ignore is a ';' separated list of hostnames, which should be excluded from mapping `(ignore='H1;H2,...')`.

Default is: `free:shared`
Job-Wide Resources

Job-wide resources are assigned to the system level (i.e., they are not tied to specific nodes) and may be used for requesting floating licenses or other resources, such as cput or walltime. Job-wide resources can only be requested outside of an `rset` statement. Not allowed are: arch, hostname, mem, ncpus, and vmem.

Syntax: `keyword=value[,keyword=value...]`

Example: `--res=sw=g03`

A.3 OPTIONS

- `-a`, `-after=WHEN` Time after which the job is eligible for execution.
  WHEN may be given as absolute time with format Datetime or, if the
time should be relative to now, as ’+’Timespan.

- `--admin` Enable admin mode if caller is a registered CCS admin).

- `--allowed=LIST, ...` LIST is a comma separated list of users and/or groups
  who are allowed to submit jobs to the reserved resources.

- `-c`, `-cores=NUMBER` Number of requested CPU cores.
  Format: Unity

- `C PREFIX` Change prefix used for directives in a job script.

- `--cwd=DIRECTORY` Use DIRECTORY as working directory for execu-
tion.
  If not given, the directory where the job was submitted is used.

- `--debug=DEBUG_LEVEL` The DEBUG_LEVEL argument is a string
  which consists of either the word ”all”, or one or more of the characters
  ”c”, ”e”, ”i”, and ”m”.
  - all enable all debug messages,
  - c enable comm-layer debug messages,
  - e enable event-layer debug messages,
  - i enable internal debug messages,
  - m enable message-layer debug messages.
  A default value can be set. Refer to section ENVIRONMENT.

- `-d`, `-t`, `--duration`, `--time`, `--walltime=DURATION` Set the maximum
duration to use the resources. Default unit is second.
  If you do not specify a value, a site specific value will be taken. It can
be shown by `ccsinfo --default`. A default value can be set. Refer to section ENVIRONMENT. If the maximum duration is exceeded, the job will be terminated.

Format: Timespan

-e, –endtime=WHEN Deadline of a job.
Format: Datetime.

-g, –group=NAME Set the group name.
A default value can be set. Refer to section ENVIRONMENT.
NAME is not case sensitive.

OPTION is specified without hyphens (`-`).

-i, –island=NAME Specify the CCS island to be used.
Must be the first argument. A default value can be set. Refer to section ENVIRONMENT.

-I, –interactive Interactive job or interactive shell.

HOW is one of the following:

n Do not join STDOUT and STDERR.
oe join STDERR into STDOUT.
eo join STDOUT into STDERR.

If HOW is not given, joining is set to ’oe’.
If joining is not set it defaults to ’n’.
Format: String

-J, –jobarray=SJID,... specifying a job array.
SJID is a subjob identifier. Syntax: `first - last [:stepping]`

first is the first index, last the last index, and stepping the stepping factor. All of them must be integers >= 0.

stepping defaults to 1.

If last is not a multiple of stepping above first, it will not be used as an index value, and the highest index value used will be lower than last.

Not allowed are interactive subjobs, a specific start time or a deadline.
Format: String
APPENDIX A. CCSALLOC MAN PAGE

-M, –mail=RECIPIENT[,...] RECIPIENT is an ACCOUNT@DOMAIN string.
CCS will send all emails to the stated recipient(s). A default value can be set. Refer to section ENVIRONMENT.
Defaults to: no mail address given.

-m, –notifyuser=EVENTS Specifies the set of conditions under which CCS will send mail messages about the job.
EVENTS is a string which consists of either the single character 'n', or one or more of the characters 'a', 'b', 'e', 'r', 's', and 'w'.

a Send mail if job is aborted by CCS.
b Send mail when job begins execution.
c Send mail when job ends execution.
n Do not send mails.
r Send mail if job start has been re-planned.
s Send mail for each subjob of a job array.
w Send warning mails. E.g.: resource could not be allocated, reservation unused, runtime expires soon, ...

A default value can be set. Refer to section ENVIRONMENT. If not set it defaults to: 'n'.

-N, –name=NAME Specifies a name for the request.
The name specified may be any length.
A default value can be set. Refer to section ENVIRONMENT.
If not specified the request name will be the base name of the job script file or the executable specified on the command line. If an interactive shell is requested, the request name will be set to INT. If a reservation is requested, the request name will be set to RSV.

-n, –nodes=NUMBER Number of exclusively requested nodes.
Format: Unity

–notifyjob=HOW,WHEN HOW specifies a command or a signal which is executed/ sent to the job before the resource is released.

  cmd Is an executable. It will have the same environment variables as the initial job started on the boot node.

  signal Can be given as: [-]<digit> or [-][SIG]<signal>.

  WHEN Is a timespan. It must be >= 60s.

Setting HOW and WHEN to 0 (eg. -notifyjob 0, 0) disables this feature. A default value can be set. Refer to section ENVIRONMENT. If not set it defaults to: 0, 0 (ie., do not notify the job.)

Examples:
A.3. OPTIONS

- --notifyjob=$HOME/bin/myScript.sh,10m
  Executes $HOME/bin/myScript.sh 10 minutes before the resource is released.

- --notifyjob=XCPU, 135
  Sends the signal SIGXCPU 135 seconds before the resource is released.

- --notifyjob -9,600
  Sends the signal SIGKILL 10 minutes before the resource is released.

-o, --output=FILE, --stdout=FILE Specifies the path for the STDOUT redirection.
   The path may be absolute or relative. In the latter case it is assumed relative to the submit directory.
   A default value can be set. Refer to section ENVIRONMENT. If not given it defaults to: job-name.<reqID>.out, whereby <reqID> is the request-ID assigned by CCS.

-q, --quiet Be quiet.
   No logging messages will be printed. If submission was successful ccsalloc prints the request-ID to stdout.

--res=RES_LIST RES_LIST is a comma separated list of 'resource=value' strings.
   Resource is the name of an allocatable resource (which is generic or system dependent). ccsinfo -a shows the allocatable resources. For more details, refer to 'Specifying Resources' or the CCS "User Manual".

--rsvid=requID Request-ID of a previously made reservation.
   Used to submit jobs to previously reserved resources.

-s, --starttime=WHEN The job should start exactly at time WHEN or never.
   WHEN may be given as absolute time. Format: Datetime or, if the start time should be now, use 'now'
   or if the time should be relative to now, use ' + 'timespan.
   If this option is not specified, CCS tries to start the job as soon as possible.

--stderr=FILE Specifies the path for the STDERR redirection.
   The path may be absolute or relative. In the latter case it is assumed relative to the submit directory. If not given it defaults to: job-name.<reqID>.err, whereby <reqID> is the request-ID assigned by CCS.
-**stdin=FILE** If given, the **STDIN** stream will be read from this file. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory. If not given it defaults to: /dev/null, which means no input.

-**tracefile=FILE** Specifies the path for the trace file. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory. The keyword ‘%reqid’ will be replaced by the request-ID. E.g.: ccsalloc --tracefile=myTRC%reqid.trace will create a file named myTRC<reqID>.trace whereby <reqID> is the request-ID assigned by CCS. If given CCS writes all state changes into this file.

-**usage** Show usage.

-**v, --verbose=NUMBER** The higher the value the verbose ccsalloc will be. Format: Unity

-**V, --version** Print version.

-**whelp=WORKER** Show worker help.

**worker worker_opts [-] cmd [cmd args]** Worker are tools to start jobs under specific run time environments (e.g., abaqus, gaussian, ompi, . . .). If you start ccsalloc without any parameter, it will show the currently available workers.

### A.4 Keywords Usable at Redirection

The following keywords may be used while specifying redirections for **STDIN**, **STDOUT**, and **STDERR**.

- **%reqid** will be replaced by the reqID.

- **%A** will be replaced by the reqID of the related job array. If the job is no subjob, then %A and will be replaced by the job’s reqID.

- **%a** will be replaced by the subjob index of a job array subjob. If the job is no subjob, then %a will be replaced by the job’s reqID.
A.5 THE VIRTUAL TERMINAL

The redirections of STDOUT and STDERR become active if the connection between the user interface (UI) on the frontend and the Execution Manager (EM) on the compute host gets lost. This may be caused by:

- Typing CTRL-Z bg in the terminal
- A crash of the UI
- A forced shutdown of the UI
- A crash of the frontend

All output which is not explicitly redirected into files will be buffered by the EM and (if the buffer gets full) redirected into the specified files. If you have specified a redirection into a file and the file cannot be opened, CCS aborts the job.

One can reconnect the application with the ccsbind command. The applications streams (STDIN, STDOUT, and STDERR) will then be redirected to the terminal. ccsbind can only be used for interactive jobs.

A.6 EXIT STATUS

Upon successful processing, the exit status will be a value of zero. If the command fails, the command exits with a value greater than zero.

A.7 ENVIRONMENT

If an option is not specified via a CLI switch, ccsalloc first looks for a corresponding environment variable. If the environment variable is not specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, the file $HOME/.ccsrc/uirc is checked. If no value has been found, a compile time default value will be taken. An example file can be copied from $CCS/examples/uirc.
ccsalloc scans for the following (in alphabetic order) default values.

CCS_UI_ADMIN <ON|OFF> Related CLI switch --admin.
Defaults to: OFF.

CCS_UI_BG_OUTPUT FILE Related CLI switch -o.
Defaults to: /dev/null.

CCS_UI_DEF_DURATION <timespan> Related CLI switch -d.
Defaults to: '10m'.
APPENDIX A. CCSALLOC MAN PAGE

CCS_UI_DEBUG DEBUG_LEVEL Related CLI switch --debug.
 Defaults to: no debug mode.

CCS_UI_DEF_EMAIL_RECIPIENTS MAIL_LIST Related CLI switch --mail.
 Defaults to: not specified.

CCS_UI_DEF_GROUP NAME Related CLI switch --group.
 Defaults to: not specified.

CCS_UI_DEF_ISLAND NAME Related CLI switch -i.
 Defaults to: not specified.

CCS_UI_DEF_NOTIFY_JOB HOW,WHEN Related CLI switch --notifyjob.
 Defaults to: no notification.

CCS_UI_DEF_NOTIFY_USER MAIL_OPTIONS Related CLI switch --notifyuser.
 Defaults to: no notification.

CCS_UI_NOHUP <ON|OFF> If set to ON prevents the user-interface to break
 the connection to a running,interactive job if catching the SIGHUP sig-
 nal. The caught signal will be sent to the job instead.
 Defaults to: no notification.

CCS_UI_REQ_NAME NAME Related CLI switch --name.
 Defaults to: the base name of the job script file or the executable spec-
 ified on the command line. If an interactive shell is requested, the
 request name will be set to INT. If a reservation is requested, the re-
 quest name will be set to RSV.

CCS_UI_RC_FILE FILE Specifies an alternative CLI rc file.
 Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be specified in
 the environment.

CCS_UI_WORKER_FILE FILE Specifies an alternative worker configuration file.
 Defaults to: $CCS/etc/<island>/worker.conf. NOTE: Needs ad-
 min privileges and can only be specified in the environment.

A.8 Job Environment

When submitting a job, OpenCCS copies the process environment and re-
 builds it on the execution host before starting the job.

A.8.1 The Node File

OpenCCS creates a file containing the node names allocated to a job. The
 file name is stored in the environment variable CCS_NODEFILE. Each node
 appears once in a single line. The file will contain the names of the allocated
nodes with each name repeated N times, where N is the number of `mpiprocs` specified for all chunks allocated on that node. `mpiprocs` is the number of MPI instances per chunk and defaults to 1. The order in which nodes appear in the node file is the reverse order in which chunks were specified in the `--res=rset` directive.

A.8.2 Execution Host Environment Variables

OpenCCS additionally sets the following environment variables on the execution host:

- **CCS** Path to the OpenCCS installation.
- **CCS_ARCH** Defines the local architecture (e.g., LINUX32 or LINUX64). Used to find an architecture dependent executable.
- **CCS_ARRAY_ID** For a subjob, the request-ID of the related job array.
- **CCS_ARRAY_INDEX** For a subjob, its index in the related job array.
- **CCS_ISLAND** The island name.
- **CCS_MAPPING** A string describing the mapping of the job.
  Syntax: `hostname:=chunk[+chunk..][,hostname...]`
  and chunk is: `count:name=val[:name=val]`
  Example: `CCS_MAPPING=node01:=1:ncpus=2:mem=4g, node35:=1:ncpus=5:mem=180g`
- **CCS_NODEFILE** Absolute path of the node file.
- **CCS_NODES** A space separated list of the node names of the allocated resources.
- **CCS_REQID** The request-ID.
- **CCS_REQNAME** The request name.
- **CCS_TMPDIR** The path of the request specific, node local temporary directory. At allocation time, CCS creates a node local directory named `<path>/<reqID>`. The value of `<path>` is set by the CCS administration. This directory can be used by applications for writing temporary files during runtime. The directory will be removed automatically when releasing the partition.
- **CCS_UMASK** Value of the current umask.
- **NCPUS** For the MPI process with rank 0. Set to the value of ncpus requested for the related chunk. For other MPI processes, behavior depends on the MPI implementation.
OMP_NUM_THREADS For the MPI process with rank 0. Set to the value of ompthreads. For other MPI processes, behavior depends on the MPI implementation.

TMPDIR Same as CCS_TMPDIR.

A.9 FILES

$HOME/.ccsrc/uirc[.ISLAND_NAME] specifies default values for the CCS commands.

A.10 EXAMPLES

Refer to the CCS 'User Manual' for more detailed examples.

1. Getting an interactive shell on the boot node.

   ccsalloc -I

   Allocates 1 core for 10 minutes and gets you an interactive login shell on the boot-node of the partition. After leaving the shell the resources are released automatically.

2. Batch jobs using a job script file

   Write a shell script and use the ccsworker wrapper.

   #!/bin/sh
   #CCS -c 64
   #CCS -t 2h
   #CCS -o MYFILE-%A.out.%a
   cp foo bar
   ccsworker mpich -- hello -F 123
   rm foo
   #use only 32 of the requested 64 cores
   ccsworker openmpi -ni 32 -- goodbye -o results
   exit 0

   Submit this job via ccsalloc myScript.sh

3. Interactive jobs

   ccsalloc -I hostname

4. Batch Job with redirection
ccsalloc -o job.%reqid.out --stderr=%reqid.err hostname

5. Use grouping

ccsalloc --res=rset=10:npcus=12:mem=20g,place=scatter:group=switch

All chunks should be mapped to the same switch.

6. Submit a job array

ccsalloc -J '12-35:5,1000-1500:100,45-68' myJob.sh

7. Make a reservation

ccsalloc -s +1h -e 23:00 -c 10 --allowed="user1,group1"

Reserves 10 cores, starting in one hour, ending at 11 pm. In addition to the submitting user user1 and all members of the group group1 may use the reserved resources.

8. Use a reservation

ccsalloc --rsvid=23 -c 4 myjob.sh

Submits the job script to the reservation with the ID 23.

A.11 SEE ALSO

ccsalter(1), ccsbind(1), ccsinfo(1), ccskill(1), ccsmask(1), ccssignal(1), ccsstracejob(1), ccsworker(1), ccs_resource_formats(7), the CCS 'User Manual'.

A.12 AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix B

ccsalter Man Page

B.1 SYNOPSIS

ccsalter [options] req_identifier ...

B.2 DESCRIPTION

Nearly all attributes of a request (job (array) or reservation) may be altered after submission, using the ccsalter command. Several requests can be altered simultaneously. If one value cannot be altered for a request, the whole alteration of this request fails. The resource definition can be done using the same shortcuts as in ccsalloc. New values have to be given as absolute values. E.g., resources must not be specified relatively (e.g., -c+2). Exceptions are the time related attributes:

**start time** -a 0 or -s 0 both remove the (minimum) start time.

**runtime** Accepts an absolute value or 't+/-t.timespan.

**stop time** -e 0 removes the end time.

B.3 OPTIONS

-a, --after=WHEN Alter the time after the job becomes eligible.
    WHEN may be given as absolute time with format Datetime.
    If the time should be altered relatively to the set use '+/-t.timespan.
    If the time should be removed, use '0'.
    If altering a reservation, all related jobs are re-planned.

-admin Enable admin mode (if caller is a registered CCS admin).

--allowed=LIST, ... LIST is a comma separated list of users and/or groups who are allowed to submit jobs to the reserved resources.
APPENDIX B. CCSALTER MAN PAGE

-`c`, `-cores=NUMBER` Number of requested CPU cores.
  Format: Unity

-`cwd=DIRECTORY` Use DIRECTORY as working directory for execution.

-`debug=DEBUG_LEVEL` The DEBUG_LEVEL argument is a string which consists of either the word "all", or one or more of the characters "c", "e", "i", and "m".
  all enable all debug messages,
  c enable comm-layer debug messages,
  e enable event-layer debug messages,
  i enable internal debug messages,
  m enable message-layer debug messages.
  A default value can be set. Refer to section ENVIRONMENT.

-`d`, `-t`, `-duration`, `-time`, `-walltime=DURATION` Alter the maximum duration to use the resources.
  If the duration should be altered relatively to the set duration, use `'+|-'Timespan'.
  Default unit is second.
  If altering a reservation, all related jobs are re-planned.

-`e`, `-endtime=WHEN` Alter the time at which the job should end (i.e., the deadline).
  WHEN may be given as absolute time with format Datetime.
  If the deadline should be altered relatively to the set one, use `'+|-'Timespan'.
  If the deadline should be removed, use '0'.
  If altering a reservation, all related jobs are re-planned.

-`g`, `-group=NAME` Set the group name.
  NAME is not case sensitive.
  If altering a reservation, all related jobs are re-planned.

-`h`, `-help=[OPTION]` Show help.
  OPTION is specified without hyphens ('-').
  Format: String

-`hold` Put jobs in state hold.
  This means the job will be ignored while planning the schedule and not started until resumed or killed.
  A job in state hold may be altered, killed, or resumed.
Reservations and single job-array subjobs cannot be altered to state hold. Using \texttt{--message} one can give a message to the user.

\texttt{-i, --island=NAME} Specify the CCS island to be used. Must be the first argument.
Format: String A default value can be set. Refer to section ENVIRONMENT.

\texttt{-j, --join=[HOW]} Joining of \texttt{STDOUT} and \texttt{STDERR} streams.
HOW is one of the following:

- \texttt{n} Do not join \texttt{STDOUT} and \texttt{STDERR}.
- \texttt{oe} join \texttt{STDERR} into \texttt{STDOUT}.
- \texttt{eo} join \texttt{STDOUT} into \texttt{STDERR}.

If \texttt{HOW} is not given joining is set to 'oe'. If joining is not set it defaults to 'n'.
Format: String

\texttt{-M, --mail=RECIPIENT[,...]} RECIPIENT is an ACCOUNT@DOMAIN string. CCS will send all emails to the stated recipient(s).

\texttt{--message[MESSAGE]} Used in conjunction with \texttt{--hold}.
MESSAGE is a text which may be used to notify the user about the reason for holding the job.
Format: String

\texttt{-m, --notifyuser=EVENTS} Specifies the set of conditions under which CCS will send mail messages about the job.
EVENTS is a string which consists of either the single character 'n', or one or more of the characters 'a', 'b', 'e', 'r', and 'w'.

- \texttt{a} Send mail if job is aborted by CCS.
- \texttt{b} Send mail when job begins execution.
- \texttt{e} Send mail when job ends execution.
- \texttt{n} Do not send mails.
- \texttt{r} Send mail if job start has been re-planned.
- \texttt{w} Send warning mails. E.g.: resource could not be allocated, reservation unused, runtime expires soon, ...

\texttt{-N, --name=NAME} Specifies a name for the request.
The name specified may be any length.
APPENDIX B. CCSALTER MAN PAGE

-n, --nodes=NUMBER  Number of exclusively requested nodes.
    Format: Unity
    If altering a reservation, all related jobs are re-planned.

-notifyjob=HOW,WHEN  HOW specifies a command or a signal which is executed/ sent to the job before the resource is released.

    cmd  Is an executable.

    signal  Can be given as: [-]<digit> or [-][SIG]<signal>.

    WHEN  Is a timespan. It must be >= 60s.

    Setting HOW and WHEN to 0 (eg. -notifyjob 0, 0) disables this feature.

-o, --output=FILE  Specifies the path for the stdout redirection. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory.

    The keyword '%reqid' will be replaced by the request-ID.

    E.g.: ccsalloc -o myFILE%reqid.out will create a file named myFILE$<$reqID>.out whereby <reqID> is the request-ID assigned by CCS.

-q, --quiet  Be quiet.
    No logging messages will be printed.

-res=RES_LIST  RES_LIST is a comma separated list of 'resource=value' strings.

    Resource is the name of an allocatable resource (which is generic or system dependent). ccsinfo -a shows the allocatable resources. For more detailed information, refer to ccsalloc(1) or the CCS 'User Manual'.

    If altering a reservation, all related jobs are re-planned.

-r, --resume  Resume jobs from state hold to state Planned.

-rsvid=req_identifier  Alter the reservation, the job should run in.

    Use '0' if request should not be assigned to any reservation.

-s, --starttime=WHEN  Alter the start time.

    WHEN may be given as absolute time. Format: Datetime.

    If the start time should bee now, use 'now'.

    If the time should be relative to the set one, '+'-='Timespan.

    If the start time should be removed, use '0'.

    If altering a reservation, all related jobs are re-planned.

-stderr=FILE  Specifies the path for the stderr redirection. The path may be absolute or relative. In the latter case it is assumed relative to the submit directory. The keyword '%reqid' will be replaced by the
request-ID.
E.g.: ccsalloc --stderr myFILE%reqid.err will create a file named
myFILE<reqID>.err whereby <reqID> is the request-ID assigned by
CCS.

`-stdin=FILE` If given, the stdin stream will be read from this file. The
path may be absolute or relative. In the latter case it is assumed
relative to the submit directory.

`-tracefile=FILE` Specifies the path for the trace file. The path may be
absolute or relative. In the latter case it is assumed relative to the
submit directory. The keyword `%reqid` will be replaced by the
request-ID. E.g.: ccsalloc -tracefile=myFILE will create a file named
myFILE<reqID>.trace whereby <reqID> is the request-ID assigned by
CCS. CCS writes all state changes into this file.

`-usage` Show usage.

`-V, --version` Print version.

`-v, --verbose=NUMBER` The higher the value the verbose CCS will be.

`req_identifier ...` A req_identifier is either a reqID, a request name, or
a subjob identifier. They can be mixed. For the syntax of a subjob
identifier refer to `ccsalloc(1)` or the CCS 'User Manual'.
B.4 WHAT CAN BE WHEN ALTERED

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>–after</td>
<td>PLANNED,WAITING</td>
<td>1, 2</td>
</tr>
<tr>
<td>–allowed</td>
<td>always</td>
<td>Reservations only. Valid only for new jobs.</td>
</tr>
<tr>
<td>–cwd</td>
<td>PLANNED,WAITING</td>
<td></td>
</tr>
<tr>
<td>–duration</td>
<td>always</td>
<td>1,2. A longer duration may be denied if:</td>
</tr>
</tbody>
</table>
|        |             | • The sum of all requested prolongations ex-
|        |             | ceeds the limit.                            |
|        |             | • Planned jobs, which are not best-effort   |
|        |             | jobs, would be delayed.                     |
| –endtime | PLANNED,WAITING | 1, 3                                        |
| –group  | PLANNED,WAITING | 1, 2                                        |
| –hold   | PLANNED,WAITING | 4                                           |
| –join   | PLANNED,WAITING |                                            |
| –mail   | always       |                                             |
| –name   | always       |                                             |
| –notifyjob | always   |                                             |
| –notifyuser | always   |                                             |
| –res    | PLANNED,WAITING | 1, 2                                        |
| –resume | WAITING      | 4                                           |
| –rsvid  | PLANNED,WAITING |                                            |
| –starttimes | PLANNED,WAITING | 1, 3                                    |
| –stdin  | PLANNED,WAITING |                                            |
| –stderr | PLANNED,WAITING |                                            |
| –stdout | PLANNED,WAITING |                                            |
| –tracefile | always | Alter message is written to the new file. |

Notes:
1: For a reservation, only if no job is running in the reservation.
2: For a job array, only if no job was already started in the job array.
3: Not allowed for job arrays.
4: Not for reservations.

B.5 EXIT STATUS

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.
B.6 ENVIRONMENT

If an option is not specified via a CLI switch, ccsalter first looks for a corresponding environment variable. If the environment variable is not specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, the file $HOME/.ccsrc/uirc is checked.

If no value has been found, a compile time default value will be taken.

An example file can be copied from $CCS/examples/uirc.

ccsalter scans for the following (in alphabetic order) default values.

CCS_UI_ADMIN <ON|OFF> Activate admin mode.
    Defaults to: OFF.

CCS_UI_DEBUG DEBUG_LEVEL Related CLI switch --debug.
    Defaults to: no debug mode.

CCS_UI_DEF_ISLAND NAME Related CLI switch -i.
    Defaults to: not specified.

CCS_UI_RC_FILE FILE Specifies an alternative CLI rc file.
    Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be specified in the environment.

B.7 FILES

$HOME/.ccsrc/uirc.ISLAND_NAME specifies default values for the CCS commands.

B.8 SEE ALSO

ccsalloc(1), ccsbind(1), ccsinfo(1), ccskill(1), ccsmsg(1), ccsignal(1), ccstracejob(1), ccsworker(1), cc_resource_formats(7), the CCS ’User Manual’.

B.9 AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
C.1 SYNOPSIS

ccsbind [options] req_identifier

C.2 DESCRIPTION

ccsbind binds the current UNIX-Shell to the given, interactive job. The UNIX shell then will act as the controlling terminal for the job indicated by req_identifier. This is useful if the connection to your interactive job was lost (e.g. due to a network failure).

C.3 OPTIONS

–admin Enable admin mode, if caller is a registered CCS admin.

–debug=DEBUG_LEVEL The DEBUG_LEVEL argument is a string which consists of either the word "all", or one or more of the characters "c", "e", "i", and "m".

  all enable all debug messages,
  c enable comm-layer debug messages,
  e enable event-layer debug messages,
  i enable internal debug messages,
  m enable message-layer debug messages.

A default value can be set. Refer to section ENVIRONMENT.

-h, –help=[OPTION] Show help. OPTION is specified without hyphens ('-').
Format: String
APPENDIX C. CCSBIND MAN PAGE

-i, –island=NAME Specify the CCS island to be used. Must be the first argument. A default value can be set. Refer to section ENVIRONMENT.
Format: String

-q, –quiet Be quiet. No logging messages will be printed.

–usage Show usage.

-V, –version Print version.

-v, –verbose=NUMBER The higher the value the verbose CCS will be.
Format: Unity

req.identifier A req.identifier is either a reqID or a request name.

C.4 EXIT STATUS

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.

C.5 ENVIRONMENT

If an option is not specified via a CLI switch, ccsbind first looks for a corresponding environment variable. If the environment variable is not specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, the file $HOME/.ccsrc/uirc is checked.

If no value has been found, a compile time default value will be taken. An example file can be copied from $CCS/examples/uirc. ccsbind scans for the following (in alphabetic order) default values.

CCS_UI_ADMIN <ON|OFF> Activate admin mode.
Defaults to: OFF.

CCS_UI_DEBUG DEBUG_LEVEL Related CLI switch --debug.
Defaults to: no debug mode.

CCS_UI_DEF_ISLAND NAME Related CLI switch -i.
Defaults to: not specified.

CCS_UI_RC_FILE FILE Specifies an alternative CLI rc file.
Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be specified in the environment.
C.6. FILES

$HOME/.ccsrc/uirc[.ISLAND_NAME] specifies default values for the CCS commands.

C.7. SEE ALSO

ccsalloc(1), ccsalter(1), ccsinfo(1), ccskill(1), ccsmsg(1), ccsignal(1), ccsstracejob(1), ccsworlker(1), ccs_resource_formats(7), the CCS 'User Manual'.

C.8. AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix D

ccsinfo Man Page

D.1 SYNOPSIS

ccsinfo [options] [req_identifier ...]

D.2 DESCRIPTION

show status about CCS schedule, system, groups and users, and requests. Refer to the accordant sections.

D.3 GENERAL OPTIONS

–admin Enable admin mode, if caller is a registered CCS admin.

–debug=DEBUG_LEVEL The DEBUG_LEVEL argument is a string which consists of either the word "all", or one or more of the characters "c", "e", "i", and "m".

  all enable all debug messages,
  c enable comm-layer debug messages,
  e enable event-layer debug messages,
  i enable internal debug messages,
  m enable message-layer debug messages.

A default value can be set. Refer to section ENVIRONMENT.

-g, –group=NAME Set the group name. NAME is not case sensitive. A default value can be set. Refer to section ENVIRONMENT.
-h, –help=[OPTION] Show help. OPTION is specified without hyphens (‘-’).
   Format: String

-i, –island=NAME Specify the CCS island to be used. Must be the first
   argument. A default value can be set. Refer to section ENVIRONMENT.
   Format: String

–islands Shows available islands.

–motd Prints the message of the day, if specified by the CCS administra-
   tion.

–usage Show usage.

-V, –version Print version.

-v, –verbose=NUMBER The higher the value the verbose CCS will be.
   Format: Unity

.req_identifier ... A req_identifier is either a reqID, a request name, or a
   subjob identifier. They can be mixed. For the syntax of a subjob
   identifier refer to ccsalloc(1) or the CCS ’User Manual’.

D.4 SCHEDULE STATUS

The schedule can be viewed via the option -s|--schedule.
It knows the following sub-options:

–dist=FILTER Shows information about the job distribution, i.e., how
   many jobs are in which state. This sub-option is mutual exclusive to
   all other ones.
   Possible filters are:

   all Shows distribution for users and groups.
   group Shows distribution for groups.
   mine Shows the callers job distribution.
   user Shows distribution for users.

   FILTER is not case sensitive and may be abbreviated as long as the
   abbreviation is unique. Default FILTER is mine.
   –dist respects –raw.

–summary Shows summarized schedule information. This sub-option is
   mutual exclusive to the other ones.
The following sub-options may be combined:

- **–group=GROUP_LIST** Filters for the specified group(s).
  GROUP_LIST is a comma separated list of group names.

- **–user=ACCOUNT_LIST** Filters for the specified account(s).
  ACCOUNT_LIST is a comma separated list of account names.

- **–mine** Shows only information about the current account.

- **–state=STATES** Filters for requests having a state in the given list.
  Possible states: C (Completed), H (Hold), P (Planned), R (Running),
  and W (Waiting)
  STATES is a comma separated list of the characters 'c', 'h', 'p', 'r',
  and 'w'.

- **–type=TYPES** Filters for requests having a type in the given list. Possible types: A (Array), B (Batch), I (Interactive), and R (Reservation)
  TYPES is comma separated list of the characters 'a', 'b', 'i', and 'r'.

- **–lines=NUMBER** Limits the number of found requests to the given number.

- **–fmt=FIELDS** Shows only specified fields.
  Syntax of FIELDS string is: ‘%.NX’
  . right justification (optional)
  N sizeof field (optional)
  X the field specifier.

  Example: --fmt="%.R %T %.10z %.50j".
  The following fields are available. Fields marked with ‘(*)’ are accessible only to the request owner or the administrator. If a field is not accessible the output is ‘N/A’.

  C Command line call (*)
  D Duration
  E Given Deadline
  F Submitted from (*)
  G Group
  J Job notification (*)
  M Mail address (*)
  N Request name
  O Owner
APPENDIX D. CCSINFO MAN PAGE

P  Planned start time
R  ReqID
S  Given start time
T  Type
U  User interface (*)
V  Event notification (*)

a  Attributes
   • A: Mapping at Allocation
   • B: Background Priority
   • D: Dynamic Limit Extension
   • F: Freepool Impact
   • L: Limits are checked at runtime
   • M: Multihost
   • S: Small-Job (Mapped on “Local only” nodes)
   ’-’ denotes that the attribute is not set.

b  Command (*)
c  Core Efficiency(*)
   Shows cput/walltime and cput/(ncpus*walltime) in percent.
   Accuracy depends on the received values from the nodes which sample and send the data in an administrator defined interval (e.g., each 10s). Additionally, in some cases OpenCCS is not able to sample all job resource usage data if the job is using more than one node. Hence, real values may be sometimes higher.

d  Percent Done
e  STDERR (*)
i  STDIN (*)
j  Job resource set
m  Mapping
n  Node resource set
o  STDOUT (*)
p  Join (*)
q  Priority
r  ReqID of related reservation (RSV-ID)
t  Trace-File (*)
u  User resource set
v  Elapsed time
D.5. SYSTEM STATUS

\[ w \] State
\[ x \] Sub-state
\[ y \] Release time
\[ z \] Submission time

–raw Prints in a raw format: No headline, no field formatting. Fields are separated by ‘ ’.

reqID ... One may give reqIDs to scan for. This disables all filters. --lines and --fmt are still active. Request names are not recognized.
Example: ccsinfo -s --fmt="%.R %T %.10z %P %50j" 1134 235.

STATES and TYPES are not case sensitive and may be abbreviated as long as the abbreviations are unique.

Examples:

%ccsinfo -s --mine --state=r,p --type=b --group=foo,bar RE--lines=3
reqID Name Account State Start Walltime Job-Resource-Set
==============================================================================
163 ccsHAWAII2B kel PLANNED 18:57:19.04.12 145d vmem=50g,ncpus=17
180 ccsHAWAII6B kel PLANNED 19:11:19.04.12 37h30m mem=512m,ncpus=1
160 kel_2 kel ALLOCATED 17:57:19.04.12 1h vmem=1t,ncpus=4

%ccsinfo -s --mine --state=r,p --type=b --group=foo,bar --line=3 --raw
163 ccsHAWAII2B kel PLANNED 18:57:19.04.12 145d vmem=50g,ncpus=17
180 ccsHAWAII6B kel PLANNED 19:11:19.04.12 37h30m mem=512m,ncpus=1
160 kel_2 kel ALLOCATED 17:57:19.04.12 1h vmem=1t,ncpus=4

D.5 SYSTEM STATUS

D.5.1 Node Status

Summarized Node Info

-n, --nodeinfo --summary

%ccsinfo -n --summary

State Count
==============================================================================
ok 650
offline 2
down 1
down/offline 0
unknown 0
Total 653
Nodes in Use/Exclusive 451/109
Detailed Node Info

-n, --nodeinfo=[NODE, ...]
shows information about the listed nodes.
Giving no node name, shows detailed information about all nodes.
Example:

```
% ccsinfo -n kel123
```

kel123
rectime = 18:30:23
status = up,online
coordinates = 0,0,0
running jobs = 345,56,7
uptime = 1d4h28m59s since Wed Mar,19 2014 14:21
uname = Linux kel123 2.6.32-35-generic #78-Ubuntu SMP i686
ncpus = 2
totmem = 1652472k
vmem = 1652472k
availmem = 840896k
physmem = 1025980k
loadave = 0.68
sessions = 5906 5756 2478 5815 5853 5880 5890 5899
nsessions = 8
nusers = 2
idletime = 0
only local jobs = false

Brief Node Info

-n, --nodeinfo -fmt=[FIELDS] --raw --reqID=[reqID] --state=[STATE]

--reqID=[reqID] shows only nodes assigned to reqID....
This option is mutual exclusive to --state.
Example:

```
% ccsinfo -n --reqid=1356
```

<table>
<thead>
<tr>
<th>Host</th>
<th>State</th>
<th>Running Jobs</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel5</td>
<td>up,online</td>
<td>1356,45</td>
<td></td>
</tr>
<tr>
<td>kel78</td>
<td>up,offline</td>
<td>1356</td>
<td>defect hard disk</td>
</tr>
</tbody>
</table>

--state=[STATE] filters for a state. The following states are available:

all  Does not filter, prints them all.
sick  Filters for nodes which are in trouble.
ok  Filters for nodes which are active (i.e., UP and ONLINE).
up  Filters for nodes which are in state UP.
down  Filters for nodes which are in state DOWN.
online  Filters for nodes which are in state ONLINE.
offline  Filters for nodes which are in state OFFLINE.
unknown  Filters for nodes which are in state UNKNOWN. STATE is not case sensitive and may be abbreviated as long as the abbreviation is unique.

Example:
```
%ccsinfo -n --state='sic'
```

<table>
<thead>
<tr>
<th>Host</th>
<th>State</th>
<th>Running Jobs</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel245</td>
<td>up,offline</td>
<td>34,45</td>
<td>will be rebooted</td>
</tr>
<tr>
<td>kel512</td>
<td>offline</td>
<td></td>
<td>Maintenance</td>
</tr>
</tbody>
</table>

--fmt=FIELDS  Shows only specified fields. Syntax of FIELDS string is:
'.NX'

.  right justification (optional)
N  sizeof field (optional)
X  the field specifier.

Example: `--fmt="%.A %p %50i"`. The following fields are available:

A  Available memory
C  Number of cores (ncpus)
H  Hostname
J  Running jobs
M  Physical Memory
L  Load
N  Note
O  Uptime
S  Status
U  uname
V  Virtual Memory
a  Architecture
c  Coordinates
i  Idletime
p  Properties
m  Minimum resources
r  Record time
s  Sessions
t  Number of sessions (nsessions)
u  Number of users (nusers)

If a field is not accessible the output is ‘N/A’.

--raw  Prints in a raw format: No headline, no field formatting. Fields are separated by ‘ ’.
--fmt and --raw may be used together with --reqid or --state.

D.5.2 Available Workers

--worker  shows the system specific available workers.
Example:

%ccsinfo --worker -i HAWAII
ccsinfo: HAWAII provides the following workers:
ccsinfo: Refer also to the man page ccsworker(1) or call ’ccsinfo --whelp=<worker>’
HAWAII provides the following workers:
<table>
<thead>
<tr>
<th>Worker</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>abaqus</td>
<td>starts an ABAQUS application</td>
</tr>
<tr>
<td>g03</td>
<td>starts a Gaussian-03 application</td>
</tr>
<tr>
<td>g09</td>
<td>starts a Gaussian-09 application</td>
</tr>
<tr>
<td>mpich2</td>
<td>starts an MPICH2 application</td>
</tr>
<tr>
<td>mvapich</td>
<td>starts an MVAPICH application</td>
</tr>
<tr>
<td>ompi</td>
<td>starts an OpenMPI application</td>
</tr>
<tr>
<td>starccm</td>
<td>starts a STAR-CCM+ application</td>
</tr>
<tr>
<td>turbomole</td>
<td>starts a Turbomole application</td>
</tr>
</tbody>
</table>

D.5.3 Allocatable Resources

-a, --allocatable  shows allocatable resources.  
The column Type represents the resource format as described in ccs_resource_formats(7).
'A' is String
'B' is Boolean
'D' is DateTime
'S' is Size
'T' is Timespan
'U' is Unitary
'V' is String Array.
The column Flags represents the resource categories as described in the 'User Manual'.
'C' marks a consumable resource
'D' marks a dynamic resource
'J' marks a job wide resource
'N' marks a non alterable resource.
The column Amount prints the used, online, and maximum amount of the related resource. The online amount depends on the availability of the nodes.
The column Default prints the system default value.

**Example:**

```bash
%ccsinfo -a
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type, Used/Online/Max</th>
<th>Default Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncpus</td>
<td>U,C 7993/9456/9568</td>
<td>number of cores</td>
</tr>
<tr>
<td>nodes</td>
<td>U,C 294/589/614</td>
<td>number of nodes</td>
</tr>
<tr>
<td>mem</td>
<td>S,C 18.69t/40.12t/40.74t</td>
<td>physical memory</td>
</tr>
<tr>
<td>vmem</td>
<td>S,C 22.33t/49.09t/49.81t</td>
<td>virtual memory</td>
</tr>
<tr>
<td>cput</td>
<td>T, N/A</td>
<td>CPU time</td>
</tr>
<tr>
<td>walltime</td>
<td>T,J N/A</td>
<td>walltime</td>
</tr>
<tr>
<td>hostname</td>
<td>A, N/A</td>
<td>hostname</td>
</tr>
<tr>
<td>arch</td>
<td>A, N/A</td>
<td>host architecture</td>
</tr>
<tr>
<td>mpiprocs</td>
<td>U, N/A</td>
<td>number of mpi processes per chunk</td>
</tr>
<tr>
<td>ompthreads</td>
<td>U, N/A</td>
<td>number of threads per chunk</td>
</tr>
<tr>
<td>acc</td>
<td>B, N/A</td>
<td>node with accelerator card</td>
</tr>
<tr>
<td>norm</td>
<td>B, N/A</td>
<td>64GB compute node</td>
</tr>
<tr>
<td>phi</td>
<td>U,C 0/5/8</td>
<td>Intel Xeon Phi card</td>
</tr>
<tr>
<td>smp</td>
<td>B, N/A</td>
<td>SMP node</td>
</tr>
<tr>
<td>tesla</td>
<td>U,C 31/31/32</td>
<td>Tesla K20xm card</td>
</tr>
<tr>
<td>sw</td>
<td>A,CJ N/A</td>
<td>Software</td>
</tr>
</tbody>
</table>

-a --classes shows allocatable resource classes

The column #Hosts prints the online and maximum number of hosts. **Example:**

```bash
%ccsinfo -a --classes
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>#Hosts Online/Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncpus</td>
<td>16</td>
<td>587/594</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>2/2</td>
</tr>
</tbody>
</table>
### D.5.4 FreePools

--freepools shows the defined FreePools.

**Example:**

```bash
% ccsinfo --freepools
name= CPUS
  resource = ncpus
  quantity = 1/50%
  allowed = count: 5, runtime: 2h
  validity = * 10-20 * * *
name= PHYSICS
  resource = ncpus
  quantity = 50
  allowed = users:kel || groups:+phys || count: 5, runtime: 2h
  validity = always
```

For a description of the columns, refer to the 'User Manual'
D.6 GROUPS / USERS

D.6.1 Group Membership

--groups shows a list of groups the caller is member of.

Example:

% ccsinfo --groups
Groups: ccsadmin,FoO,pc2guests

D.6.2 Limits and Privileges

-l, --limits shows the limits and privileges. Both are assigned to user and
or groups.

If not using the sub options -g NAME and --user=USER, the CLI takes the
default values of the caller.

Using --user=ALL, shows the group data and all members of the group,
having an own specification.

Example:

% ccsinfo -l
Active policy for jobs exceeding their resource credits is: Reject the job.

Group-Data

name : pc2guests
validity : always
privileges : alter,interactive,reserve
manager : devil
members : +pc2guests

User-Data

account : arnie
privileges : alter

Resource Limits:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Items</th>
<th>Duration</th>
<th>Area</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>unlimited</td>
<td>315d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>mdce</td>
<td>256</td>
<td>315d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>tesla</td>
<td>10</td>
<td>120d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>ncpus</td>
<td>1800</td>
<td>21d</td>
<td>unlimited</td>
<td>always</td>
</tr>
<tr>
<td>jobs</td>
<td>5000</td>
<td>-</td>
<td>-</td>
<td>from 14:32:10.12.14</td>
</tr>
<tr>
<td>arrayjobs</td>
<td>1000</td>
<td>-</td>
<td>-</td>
<td>always</td>
</tr>
</tbody>
</table>
Alteration limits if request is in state ALLOCATED:

<table>
<thead>
<tr>
<th>What</th>
<th>Limit</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>walltime</td>
<td>10s/10%</td>
<td>always</td>
</tr>
</tbody>
</table>

Resource Credits (in hours:mm:ss)

Only resources with a specified credit are printed

<table>
<thead>
<tr>
<th>Resource</th>
<th>Credit</th>
<th>Used-Credit</th>
<th>Remaining-Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdce</td>
<td>1000:00:00</td>
<td>0:00:18</td>
<td>999:59:42</td>
</tr>
<tr>
<td>tesla</td>
<td>100:00:00</td>
<td>0:03:00</td>
<td>99:57:00</td>
</tr>
<tr>
<td>ncpus</td>
<td>2500000:00:00</td>
<td>0:03:00</td>
<td>2499999:57:00</td>
</tr>
</tbody>
</table>

For a description of the columns, refer to the 'User Manual'.

D.6.3 Default Values

`--defaults` shows default values.

Attribute (the first column) describes the attribute.

Default (the second column) describes the default value. It is taken, if the caller did not specify the attribute in question.

Force (the third column) shows values which overwrite user given values or will be taken as a default.

The administrator may assign defaults to specific users, groups, or the whole system. If both Default and Force are specified, Force will be taken.

If not using the sub options `-g NAME` and `--user=USER`, the CLI shows the default values valid for the caller’s default group.

Example:

```
% ccsinfo --def
```

Attribute Default Force

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>128m</td>
<td></td>
</tr>
<tr>
<td>mdce</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>place</td>
<td>free:shared</td>
<td></td>
</tr>
</tbody>
</table>

D.7 REQUEST STATUS

`ccsinfo req_identifier ...` shows detailed information about the specified request(s). Example:

```
% ccsinfo 29308
Request-ID : 29308
Name       : kel_3
```
D.8 PREDICTING START TIMES

It is sometimes useful to know which resources are when available. For example how many GPUs can I get now or how long is the waiting time if requesting chunks with 5 cores and 6GB per core.

Using `-p, --predict '='<RESOURCES>[;ITERATOR];...''`, one can specify RESOURCES together with ITERATORS. OpenCCS will evaluate the iterators, plan the resulting resource request (related to the caller’s limits) and print the earliest start times.
NOTE: The situation may change within seconds, if other users are submitting jobs. Hence the printed start times may become invalid.

The following sub-options may be combined:

\-group=\textit{GROUP} If not using this sub-option, the CLI takes the default group of the caller.

\-raw prints in a raw format: No headline, no field formatting. Fields are separated by ’ ’.

\subsection*{D.8.1 Resource Syntax}

The syntax is like specifying resources at submit call using \textit{ccsalloc} (1) but without \texttt{--res=rset}.

\textbf{Examples}

\begin{itemize}
\item \%\texttt{C}:ncpus=\%1:mem=\%2g
\item The shortcuts \texttt{-n} and \texttt{-c} are allowed to iterate over nodes or cores. Examples: \texttt{-n \%C;\%C=1-10} or \texttt{-c \%C; \%C=100-1000:100}. Note: The only iterator recognized here is \%C.
\end{itemize}

\subsection*{D.8.2 Iterator Syntax}

\texttt{Name=<first>[<-<last>[:stepping]]}

All of them must be integers > 0. Default stepping is 1. Three types of iterators are supported:

1. \%\texttt{C} iterates the number of chunks.
   \begin{itemize}
   \item The default \%\texttt{C} iterator is 1-1:1.
   \end{itemize}

2. \%\texttt{D} iterates the job duration.
   \begin{itemize}
   \item Last character is unit if not a number (See also \textit{ccs-resource-formats}(7)). E.g., \%\texttt{D}=1-5h or \%\texttt{D}=1-10:2m).
   \item The default unit is second. E.g., \%\texttt{D}=1-10:2 iterates: 1s, 3s, 5s, 7s, 9s.
   \item The default \%\texttt{D} iterator is the default duration assigned to the caller’s credentials (i.e., user and group). Refer to \texttt{--def}.
   \end{itemize}

3. \%\texttt{R} iterates the resources.
   \begin{itemize}
   \item Nine iterators are available: \%1..\%9.
   \item Default values are -1:-1:1.
   \end{itemize}
D.8. PREDICTING START TIMES

- One may use %R iterators in any consumable resource (chunk or job wide).
- An %R iterator may be used for multiple resources.
  E.g., ncpus=%1:tesla=%1; %1=1-5
- %R iterators will be evaluated in each %C iteration, until the maximum of all %R iterators is reached.
  E.g., %1=1-10; %2=1-5;

Remarks

1. Loop-Nesting is: %D, %C, %R.
2. %C and %D are not case sensitive.
3. Spaces are allowed in the resource and iterator specifications.
4. The order of iterator specifications does not matter.
5. Specifying an iterator which is not used is possible.
6. Multiple specifications of the same iterator is possible. Last match wins.
7. OpenCCS will print only valid results. If an iteration cannot be planned due to limitations or unavailable resources, it will be silently skipped.
8. Syntax errors are printed.

D.8.3 Examples

- Predict 5-10 chunks with Tesla GPUs, 5 cores and 30GiByte RAM, duration 1-2 hours.
  ccsinfo -p '%C:tesla=1:ncpus=5:mem=30g,place=scatter;%C=5-10;%D=1-2h'

- Predict 1-10 nodes (stepping 2) exclusively, duration 1-4 hours, group benchmark.
  ccsinfo -g benchmark -p '-n %C; %C=1-10:2; %D=1-4h'

- Predict 100-500 cores (stepping 100), duration 1-5 days stepping 2.
  ccsinfo -p '-c %C; %C=100-500:100; %D=1-5:2d'

- Predict 1-16 cores with 4GB per core, duration 1 day, output in raw format.
  ccsinfo --raw -p 'ncpus=%1:mem=%2G; %1=1-16; %2=4-64:4; %D=1d'
• Predict 1-16 cores with 4GB mem and 8GiByte vmem per core, duration 1 day.
  ccsinfo -p 'ncpus=%1:mem=%2G:vmem=%3g; %1=1-16; %2=4-64:4; %3=8-128:8; %D=1d'

• Predict 100-256 chunks with Matlab licenses, duration 75m.
  ccsinfo -p '%C:ncpus=16:mem=30g,mdce=%2; %C=100-256:64; %2=4-64:4; %D=75m'

D.9 EXIT STATUS

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.

D.10 ENVIRONMENT

If an option is not specified via a CLI switch, ccsinfo first looks for a corresponding environment variable. If the environment variable is not specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, the file $HOME/.ccsrc/uirc is checked.

If no value has been found, a compile time default value will be taken.
An example file can be copied from $CCS/examples/uirc.
ccsinfo scans for the following (in alphabetic order) default values.

CCS_UI_ADMIN <ON|OFF> Activate admin mode.
  Defaults to: OFF.

CCS_UI_DEBUG DEBUG_LEVEL Related CLI switch --debug.
  Defaults to: no debug mode.

CCS_UI_DEF_GROUP NAME Related CLI switch --group.
  Defaults to: not specified.

CCS_UI_DEF_ISLAND NAME Related CLI switch -i.
  Defaults to: not specified.

CCS_UI_DEF_NODE_FMT Related CLI switch ccsinfo -n --fmt.
  Defaults to: not specified.

CCS_UI_DEF_SCHED_FMT Related CLI switch ccsinfo -s --fmt.
  Defaults to: not specified.

CCS_UI_RC_FILE FILE Specifies an alternative CLI rc file.
  Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be specified in the environment.
D.11 FILES

$HOME/.ccsrc/uirc.[ISLAND_NAME] specifies default values for the CCS commands.

D.12 SEE ALSO

ccsalloc(1), ccsalter(1), ccsbind(1), ccskill(1), ccmsg(1), ccsignal(1), cctracejob(1), ccsworker(1), ccs_resource_formats(7), the CCS 'User Manual'.

D.13 AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix E

ccskill Man Page

E.1 SYNOPSIS

ccskill [options] req_identifier ...
ccskill [options] --all

E.2 DESCRIPTION

ccskill kills the given requests. Running jobs will be aborted. It deletes in the order in which the request identifiers are presented to the command.

E.3 OPTIONS

--admin Enable admin mode, if caller is a registered CCS admin.

--all Concerns all owned requests, even if bound by another user interface. This also valid for group managers and Administrators. The higher privilege is ignored if using this option.

--debug=DEBUG_LEVEL The DEBUG_LEVEL argument is a string which consists of either the word "all", or one or more of the characters "c", "e", "i", and "m".

  all enable all debug messages,
  c  enable comm-layer debug messages,
  e  enable event-layer debug messages,
  i  enable internal debug messages,
  m  enable message-layer debug messages.

A default value can be set. Refer to section ENVIRONMENT.
-f  Releases the given request(s), even if bound by another user interface.

-\(h\), –\(help=\[OPTION\]  Show help. OPTION is specified without hyphens (\(\_\_\_\). Format: String

-i, –island=\(NAME\) Specify the CCS island to be used. Must be the first argument. A default value can be set. Refer to section ENVIRONMENT. Format: String

-m, –message=\(MESSAGE\) If given MESSAGE will be sent to the owner of the request. If it is a batch job, MESSAGE will be written to the error file. If a tracefile is assigned to the request, MESSAGE will also appear in that file.

-q, –quiet  Be quiet. No logging messages will be printed.

-usage  Show usage.

-V, –version  Print version.

-v, –verbose=\(NUMBER\) The higher the value the verbose CCS will be. Format: Unity

\(\text{req}\_\text{identifier} \ldots\) A req_identifier is either a reqID, a request name, or a subjob identifier. They can be mixed. For the syntax of a subjob identifier refer to ccsalloc(1) or the CCS 'User Manual'.

E.4 EXIT STATUS

Upon successful processing, the exit status will be a value of zero. If the command fails, the command exits with a value greater than zero.

E.5 ENVIRONMENT

If an option is not specified via a CLI switch, ccskill first looks for a corresponding environment variable. If the environment variable is not specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND. If such a file does not exist, the file $HOME/.ccsrc/uirc is checked.

If no value has been found, a compile time default value will be taken. An example file can be copied from $CCS/examples/uirc. ccskill scans for the following (in alphabetic order) default values.

\(\text{CCS}\_\text{UI}\_\text{ADMIN} <\text{ON}|\text{OFF}>\)  Activate admin mode. Defaults to: OFF.
E.6. FILES

`CCS_UI_DEBUG` `DEBUG_LEVEL`  Related CLI switch `--debug`.
  Defaults to: no debug mode.

`CCS_UI_DEF_ISLAND NAME`  Related CLI switch `-i`.
  Defaults to: not specified.

`CCS_UI_RC_FILE` `FILE`  Specifies an alternative CLI rc file.
  Defaults to: `$HOME/.ccsrc/uirc`. NOTE: Can only be specified in
  the environment.

E.6  FILES

`$HOME/.ccsrc/uirc[.ISLAND_NAME]` specifies default values for the CCS
commands.

E.7  SEE ALSO

`ccsalloc(1), ccsalter(1), ccsbind(1), ccsinfo(1), ccsmgr(1), ccssignal(1), ccsworker(1),
ccs_resource_formats(7)`, the CCS 'User Manual'.

E.8  AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix F

ccsmmsg Man Page

F.1 SYNOPSIS

ccsmmsg [options] message req_identifier ...

F.2 DESCRIPTION

Sending a message to a job means that CCS writes a message string into one
or more output files of the job. Typically, this is done to leave an informative
message in the output of the job. The syntax is: ccsmmsg TARGET MESSAGE req_identifier[...]

F.3 OPTIONS

–admin Enable admin mode, if caller is a registered CCS admin.
–all Send message to all jobs.
–debug=DEBUG_LEVEL The DEBUG_LEVEL argument is a string
which consists of either the word "all", or one or more of the characters
"c", "e", "i", and "m".

  all enable all debug messages,
  c enable comm-layer debug messages,
  e enable event-layer debug messages,
  i enable internal debug messages,
  m enable message-layer debug messages.

  A default value can be set. Refer to section ENVIRONMENT.

-e Write message to stderr (default).
-h, --help=[OPTION]  Show help. OPTION is specified without hyphens ('-').
   Format: String

-i, --island=NAME  Specify the CCS island to be used. Must be the first
   argument. A default value can be set. Refer to section ENVIRONMENT.
   Format: String

-o  Write message to stdout.

-q, --quiet  Be quiet. No logging messages will be printed.

--usage  Show usage.

-V, --version  Print version.

-v, --verbose=NUMBER  The higher the value the verbose CCS will be.
   Format: Unity

MESSAGE  Message to send. If the MESSAGE contains blanks, the string
   must be quoted. If the final character of the string is not a newline, a
   newline character will be added when written to the job's file.

req_identifier ...  A req_identifier is either a reqID, a request name, or a
   subjob identifier. They can be mixed. For the syntax of a subjob
   identifier refer to ccsalloc(1) or the CCS 'User Manual'.

F.4 EXIT STATUS

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.

F.5 ENVIRONMENT

If an option is not specified via a CLI switch, ccsmsg first looks for a
corresponding environment variable. If the environment variable is not
specified, the file $HOME/.ccsrc/uirc.ISLAND_NAME will be checked, where
ISLAND_NAME is derived from the environment variable CCS_UI_DEF_ISLAND.
If such a file does not exist, the file $HOME/.ccsrc/uirc is checked.

If no value has been found, a compile time default value will be taken.
An example file can be copied from $CCS/examples/uirc.
ccsmsg scans for the following (in alphabetic order) default values:

CCS_UI_ADMIN <ON|OFF>  Activate admin mode.
   Defaults to: OFF.
F.6. FILES

CCS_UI_DEBUG DEBUG_LEVEL Related CLI switch --debug.
  Defaults to: no debug mode.

CCS_UI_DEF_ISLAND_NAME Related CLI switch -i.
  Defaults to: not specified.

CCS_API_RC_FILE FILE Specifies an alternative CLI rc file.
  Defaults to: $HOME/.ccsrc/uirc. NOTE: Can only be specified in the environment.

F.6 FILES

$HOME/.ccsrc/uirc[.ISLAND_NAME] specifies default values for the CCS commands.

F.7 SEE ALSO

ccsalloc(1), ccsalter(1), ccsbind(1), ccsinfo(1), ccskill(1), ccssignal(1), ccsstracejob(1), ccesworker(1), ccs_resource_formats(7), the CCS ’User Manual’.

F.8 AUTHOR

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix G

ccssignal Man Page

G.1 SYNOPSIS

ccssignal [options] signal req_identifier ...

G.2 DESCRIPTION

ccssignal is used to send a signal to a running job. The signal is sent to the
session leader of the job.

G.3 OPTIONS

–admin Enable admin mode, if caller is a registered CCS admin.

–all Send message to all jobs.

–debug=DEBUG_LEVEL The DEBUG_LEVEL argument is a string
which consists of either the word "all", or one or more of the characters
"c", "e", "i", and "m".

  all enable all debug messages,
  c enable comm-layer debug messages,
  e enable event-layer debug messages,
  i enable internal debug messages,
  m enable message-layer debug messages.

A default value can be set. Refer to section ENVIRONMENT.

-h, –help=[OPTION] Show help. OPTION is specified without hyphens
(‘-’).

Format: String
-i, --island=\texttt{NAME}  Specify the CCS island to be used. Must be the first argument. A default value can be set. Refer to section \texttt{ENVIRONMENT}.
  Format: String

-q, --quiet  Be quiet. No logging messages will be printed.

--usage  Show usage.

-V, --version  Print version.

-v, --verbose=\texttt{NUMBER}  The higher the value the verbose CCS will be.
  Format: Unity

\texttt{signal}  Signal to send. Can be given as: [-]<digit> or [-][SIG]<signal>.

\texttt{req.identifier} ...  A req.identifier is either a reqID, a request name, or a subjob identifier. They can be mixed. For the syntax of a subjob identifier refer to \texttt{ccsalloc(1)} or the CCS 'User Manual'.

\textbf{G.4 \small EXIT STATUS}

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.

\textbf{G.5 \small ENVIRONMENT}

If an option is not specified via a CLI switch, \texttt{ccsbind} first looks for a corresponding environment variable. If the environment variable is not specified, the file \texttt{$HOME/.ccsrc/uirc.ISLAND\_NAME} will be checked, where \texttt{ISLAND\_NAME} is derived from the environment variable \texttt{CCS\_UI\_DEF\_ISLAND}.
If such a file does not exist, the file \texttt{$HOME/.ccsrc/uirc} is checked.
If no value has been found, a compile time default value will be taken.
An example file can be copied from \texttt{$CCS/examples/uirc}.
\texttt{ccsbind} scans for the following (in alphabetic order) default values.

\texttt{CCS\_UI\_ADMIN} \texttt{<ON|OFF>}  Activate admin mode.
  Defaults to: \texttt{OFF}.

\texttt{CCS\_UI\_DEBUG} \texttt{DEBUG\_LEVEL}  Related CLI switch \texttt{--debug}.
  Defaults to: no debug mode.

\texttt{CCS\_UI\_DEF\_ISLAND} \texttt{NAME}  Related CLI switch \texttt{-i}.
  Defaults to: not specified.

\texttt{CCS\_UI\_RC\_FILE} \texttt{FILE}  Specifies an alternative CLI rc file.
  Defaults to: \texttt{$HOME/.ccsrc/uirc}. NOTE: Can only be specified in the environment.
G.6. FILES

$HOME/.ccsrc/uirc[.ISLAND_NAME] specifies default values for the CCS commands.

G.7. EXAMPLES

- ccssignal -9 123
- ccssignal 9 123
- ccssignal -KILL 123
- ccssignal SIGKILL 123

All examples above send the signal SIGKILL to the job 123.

G.8. SEE ALSO

ccsalloc(1), ccsalter(1), ccsbind(1), ccsm.info(1), ccskill(1), ccsm.msg(1), ccstrace-
job(1), ccsworker(1), cc.s_resource_formats(7), the CCS ‘User Manual’.

G.9. AUTHORS

Paderborn Center for Parallel Computing
ccs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix H

cctestracejob Man Page

H.1 SYNOPSIS

cctestracejob [options] reqID ...

H.2 DESCRIPTION

cctestracejob prints (in chronological order) log and accounting data for the
given requests. The data is printed if the caller is the owner of the given
request(s), or a group manager of the related group(s), or member of the
OpenCCS admin group.

H.3 OPTIONS

-\(f\), \(–filter=FILTER\) Skip log messages of type type (E)rror, (I)nformation,
(L)og, or (W)arning. Given as, for example, 'EW'. Default is no filter.
Format: String

-\(h\), \(–help=[OPTION]\) Show help. OPTION is specified without hyphens
('`').
Format: String

-\(n\) \(days\) \(back\) Report information from up to days days in the past. Default
is 1.
Format: Unitary.

-\(q\), \(–quiet\) Be quiet. No logging messages will be printed.

-\(r\), \(–raw\) Print data in raw format.

–\(usage\) Show usage.

\(reqID\) ... cctestracejob accepts maximal 10 reqIDs.
H.4 EXIT STATUS

Upon successful processing, the exit status will be a value of zero.
If the command fails, the command exits with a value greater than zero.

H.5 SEE ALSO

ccsalloc(1), ccsalter(1), ccsbind(1), ccsinfo(1), ccskill(1), ccsmsg(1), ccssignal(1), ccsworker(1), ccs_resource_formats(7), the CCS 'User Manual'.

H.6 AUTHORS

Paderborn Center for Parallel Computing
cs-team@uni-paderborn.de
http://pc2.uni-paderborn.de
http://openccs.eu
Appendix I

ccsworker Man Page

I.1 SYNOPSIS

ccsworker [options] [ [worker] [–] [worker_args] ] [ [cmd] [cmd_args] ]

I.2 DESCRIPTION

CCS provides so called workers to start jobs under specific run time environments. They conceal system specific options and provide a convenient way to start programs. If you start ccsalloc without any parameter, it will show the currently available workers. ccsalloc –whelp=WORKER prints a worker specific help (e.g. ccsalloc –whelp=mpich). You can call CCS workers from within shell scripts by using this wrapper ccsworker.

I.3 OPTIONS

- h shows usage message
- -v enables verbose mode

I.4 EXIT STATUS

ccsworker will normally return the exit code of the worker, which in turn returns the exit code of the called command. If ccsworker or the worker fails, the exit status will be a value greater than zero.

I.5 EXAMPLES

#! /bin/sh
APPENDIX I. CCSWORKER MAN PAGE

    cp foo bar
    ccsworker mpich -nn 64 -- hello -nn 123
    rm foo
    ccsworker openmpi -nn 32 -- goodbye -o results
    exit 0

I.6 SEE ALSO

    ccsalloc(1), ccsalter(1), ccsbind(1), ccsinfo(1), ccskill(1), ccsmgs(1), ccssignal(1), cctracejob(1), ccs_resource_formats(7), the CCS 'User Manual'.

I.7 AUTHORS

    Paderborn Center for Parallel Computing
    ccs-team@uni-paderborn.de
    http://pc2.uni-paderborn.de
    http://openccs.eu
Appendix J

ccs_resource_formats Man Page

J.1 DESCRIPTION

To specify resource and time dependent values, CCS provides the resource formats Boolean, Cron, Datetime, Size, String, Timespan, and Unity. This man page describes their syntax.

J.2 Boolean

A boolean value. Syntax:

- True ::= "t" |"y" |"1" |"yes" |"true"
- False ::= "f"|"n" |"0" |"no" |"false"

Values are not case sensitive.

J.3 Cron

Specifies a periodic time interval like in a cron job specification. Syntax: A string of five space separated tokens (a b c d e)

- a is minute: 0-59
- b is hour: 0-23
- c is day of month: 1-31
- d is month: 1-12
- e is day of week: 0-6 (0 is Sun)
Each token may be:

- a wildcard given as asterisk ‘*’, which always stands for ”first-last”
- a comma separated list of time points, e.g. ”2,3,5”
- an interval, e.g. ”3-4”
- a combination of lists and intervals, e.g. ”1,2,4-6”

Not allowed are:

- step values, e.g. ”/2”
- shortcuts like ”@weekly”
- weekday’s name, e.g. ”sun”

### J.4 Datetime

- **POSIX format**
  
  - CC is the first two digits of the year (the century),
  - YY is the second two digits of the year,
  - MM is the two digits for the month,
  - DD is the day of the month,
  - hh is the hour,
  - mm is the minute,
  - SS seconds.


- **CCS format**
  Syntax: `<hh[:mm] | hh:mm:dd.mm[.yy]>`
  
  - hh hours from 00 to 23
  - mm minutes from 00 to 59
  - ss seconds from 00 to 59
  - mm months from 01 to 12
  - yy years from 00 to 99

  Units are not case sensitive.

For all Datetime formats, the following is valid: If the month is not specified, it will be set to the current month if the specified day is in the future. Otherwise, the month will be set to next month. If the day is not specified, it will be set to today if the time is in the future. Otherwise, the day will be set to tomorrow. For example: specifying at 11:15am a time of 11:10, will be evaluated as 11:10am tomorrow.

J.5 Size

Specifies a size (memory, disk, ....) or a count
Syntax: <number[multiplier]>
- Kilo: \(k\) is \(2^{10}\) and \(K\) is \(10^3\)
- Mega: \(m\) is \(2^{20}\) and \(M\) is \(10^6\)
- Giga: \(g\) is \(2^{30}\) and \(G\) is \(10^9\)
- Terra: \(t\) is \(2^{40}\) and \(T\) is \(10^{12}\)

Example: 1000K denotes 1000*1000 and 1000k denotes 1024*1024. Default multiplier is 1.

J.6 String

A series of alpha-numeric characters without whitespace(s), beginning with an alphabetic character.

J.7 String Array

A comma separated list of Strings. The character ',' is not allowed within a String. A resource of type 'string array' is non-consumable. A resource request will succeed if request matches one of the values. A resource request can contain only one string. A string array resource with one value works exactly like a string resource.

J.8 Timespan

- \([\text{hours:}]\text{minutes:}]\text{seconds}\)
  Example: 120:12:13 denotes 120 hours, 12 minutes, and 13 seconds.

- \([*w][*d[*h[*m]]]]*s\)
  Supported units are:
  - \(w\) (week) equals to 7 days
APPENDIX J. **CCS_RESOURCE_FORMATS MAN PAGE**

- d (day) equals to 24 hours
- h (hour) equals to 60 minutes
- m (minute) equals to 60 seconds
- s (second)

Default unit ist second. The unit order is irrelevant. Example: 14d1h12m3s3w denotes 3 weeks, 14 days, 1 hour, 12 minutes, and 3 seconds.

J.9 Unitary

Specifies the maximum amount of a resource which is expressed as a simple integer.

J.10 SEE ALSO

ccsalloc(1), ccsalter(1), ccsbind(1), ccsinfo(1), ccskill(1), ccsmsg(1), ccsignal(1), ccstracejob(1), ccsworker(1), the CCS ’User Manual’.

J.11 AUTHORS

Paderborn Center for Parallel Computing
cs-team@uni-paderborn.de
http://www.uni-paderborn.de/pc2
http://www.openccs.eu
Appendix K

Node States

The following node states (listed in alphabetically order) are used in OpenCCS:

- **Down**: This state is set automatically by OpenCCS if the node is failing to report, is detecting local failures with node configuration or resources, or it has been shutdown by the administrator.

- **Job Exclusive**: The whole node is assigned to a single job. This may be because the job requested the node exclusively or the node has the attribute **Space Shared**.

- **Offline**: The node has been instructed by an administrator to no longer accept work. Running jobs are not affected. This may be due to a defect or maintenance. Often administrators add a comment why the node is offline.

- **Online**: The node accepts work.

- **Sick**: Node is not up and **online**. This is a “meta” state, often used as a filter.

- **Unknown**: The node has been specified by the administrator but has not yet been connected to OpenCCS.

- **Up**: The node is up and running and the **NSM** is also up and running and is connected to the **MM**.
Appendix L

Node Properties

The following built-in node properties are used in OpenCCS:

Access Control List (ACL)

The administrator may specify a list of consumers which may request the node. This is useful to “reserve” nodes for specific accounts and/or groups. Additionally, a maximum duration may be specified to allow jobs which do not match the ACL. The ACL overrules the maximum duration limit, i.e., if the caller is member of the ACL, OpenCCS does not check the jobs maximum duration against the node specific maximum duration. Note: All other restrictions (e.g., minimum resources) are still checked.

Users may inspect the node specific ACL by:
```
ccsinfo -n --sta=ok --fmt=%H%p --raw | grep -i acl
```
or by
```
ccsinfo -n <NODENAME>
```

LocalOnly

If set to true only jobs which run completely on this node are mapped to this node. This attribute is also set if this node is part of the dynamic partitioning policy (Appendix:M:SMJ).

Minimum Resources

The administrator may specify a set of resource amounts and/or a maximum duration which have to be requested at least by a job to be mapped on that node. Example:
```
vmem=512g or mem=256g or ncpus=17 or duration <= 2h
```
This means a chunk is mapped to this node if it requires either at least 512GiByte virtual memory, 256GiByte RAM, or 17 cores, or it runs at most 2 hours.

ncpus Freepool

The administrator may specify a node specific free pool. This is mostly used on nodes hosting accelerator cards like a GPU. Using the GPU in offload mode often also needs at least one
core on the host. To avoid that a job which does not need the GPU will block all cores on the host, the administrator may specify how many cores are kept free for jobs which request a GPU.

**Space Shared**

The node will be always assigned exclusively to a job. The state is set by the administrator.

Additionally all non consumable resources set by the local administration are part of the properties.

One may inspect the node properties by using `ccsinfo -n --state=%H%P%M` or `ccsinfo -n`. Refer to section 8.2.1 for more details.
Appendix M

Glossary

This glossary defines OpenCCS specific items.

Advance Reservation
A reservation for a set of resources for a specified time. The reservation is only available to the specified users and groups.

AM
The Access Manager (AM) manages the user interfaces and is responsible for authentication, authorization, and accounting.

Boot Node
At allocating a partition one of the nodes becomes the boot node. This node controls the jobs started by the user (e.g. starts the job, holds the connection to the UI). The boot node is the first node in the environment variable CCS NODES and in all mapping infos. OpenCCS assigns the boot node to one of the nodes which satisfy the last specified chunk. Example: --res=rset=ncpus=240:phi=1:arch=MIC+4:ncpus=8:mem=32g. The boot node will be one of (4:ncpus=8:mem=32g)

Chunk
Specifies a set of resources that have to be allocated as a unit on one node. Chunks cannot be split across nodes.

CLI
Command Line Interface

Client
A OpenCCS module connected to the IM.

CoreModule
AM, EM, IM, MM, NSM, OS, PM.

Daemon
An operating system process, which may be single- or multi-threaded. Runs in the background.

Host
A host is any computer. It may consist of at least one node.

Execution Host
See Boot Node.
APPENDIX M. GLOSSARY

IM  The Island Manager (IM) provides name services and watchdog facilities to keep the island in a stable condition.

Job  A command running on an allocated partition. A job is a collection of related processes which is managed as a whole. A job can often be thought of as a shell script running in a POSIX session. A non-singleton job consists of multiple tasks of which each is a POSIX session. One task will run the job shell script.

Job Array  A container for a collection of similar jobs submitted under a single reqID. It can be submitted, queried, modified, or displayed as a unit.

MM  The Machine Manager (MM) provides an interface to machine specific features like node management or job controlling.

Module  Logical entity like the Machine-Manager. It may comprise more than one daemon.

N/A  Means “Not available”. Used if a value is not specified or not accessible.

Node  A node has at least 1 Processing Element (PE) (e.g., CPU, core, GPU, ...) and may have have an operating system.

OS  The Operator Shell (OS) is the main interface for system administrators to control a OpenCCS island, e.g. by connecting to the core modules.

PM  The Planning Manager (PM) schedules the user requests onto the machine.

Request  A reservation, a job array, or a job.

reqID or request-ID  After accepting a submitted request OpenCCS assigns a unique numerical identifier to the request. The so called request-ID or reqID.

Req_identifier  Identifies a submitted request. May be a reqID, a subjob identifier, or a request name.

SMJ  Small Job. The administrator may specify dynamic partitioning of the system, to enforce mapping of “small jobs” to specific nodes. A “small job” is defined by the administrator. The system automatically adapts the number of used nodes by the ratio of requested ncpus for small and large nodes. Once can
inspect the job attribute using `ccsinfo --fmt=%a`. Refer to section 8.1.4 for more details.

**Standing Reservation**

An advance reservation, which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

**Subjob Identifier (SJID)**

Identifies one or more subjobs. Syntax: `reqID[ID]` where `reqID` is the requestID of the job array and `ID` may comprise comma separated job array ranges. E.g., `1234[5]` or `1234[1-8:2,26]`. The syntax is explained in 11.3

**Symbol**

OpenCCS uses special files to specify runtime information like filenames, timeouts, or limits. Symbols are evaluated by all OpenCCS scripts and executables.

**Task**

One or more session(s) belonging to a job, running on one of the nodes assigned to the job.

**UI**

The *User Interface (UI)* provides a single access point to one or more systems.

**WLM**

Workload Management System

**Worker**

OpenCCS provides so called workers to start jobs under specific run time environments (e.g. MPICH, Gaussian, etc.). They conceal system specific options and provide a convenient way to start programs.
List of Figures

2.1 The OpenCCS modules (left) and event type handling (right) 16
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>OpenCCS CLI User Commands</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Always available resources</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Placement Specification</td>
<td>46</td>
</tr>
<tr>
<td>6.1</td>
<td>Correlations of the different scheduling hints</td>
<td>58</td>
</tr>
<tr>
<td>9.1</td>
<td>Which job attribute can be when altered</td>
<td>82</td>
</tr>
</tbody>
</table>
Index

–admin, 30
/dev/null, 34, 61
/etc/profile.d, 35

ALLOCATED, 17, 31
ALLOCATING, 17, 31
AM, 15

batch job, 30

CCS, 37
cs, 17
ccs, 17
CCS_ARCH, 38
CCS_ARRAY_ID, 38
CCS_ARRAY_INDEX, 38
CCS_ISLAND, 38
CCS_MAPPING, 38
CCS_NODEFILE, 38
CCS_NODES, 38, 163
CCS_REQID, 38
CCS_REQNAME, 38
CCS_TMPDIR, 38
CCS_UI_ADMIN, 34
CCS_UI_BG_OUTPUT, 34
CCS_UI_DEBUG, 34
CCS_UI_DEF_DURATION, 34
CCS_UI_DEF_EMAIL_-RECIPIENTS, 34
CCS_UI_DEF_GROUP, 34
CCS_UI_DEF_ISLAND, 34
CCS_UI_DEF_NODE_FMT, 34
CCS_UI_DEF_NOTIFY_JOB, 34
CCS_UI_DEF_NOTIFY_USER, 34
CCS_UI_DEF_SCHED_FMT, 35
CCS_UI_NOHUP, 35
CCS_UI_RC_FILE, 35
CCS_UI_REQ_NAME, 35
CCS_UI_WORKER_FILE, 35
CCS_UMASK, 38
ccs_x, 17
ccsaloc, 53–56, 60, 61, 81, 84–86, 88, 89, 92, 93
ccsalter, 18, 30, 32, 81, 83, 91
ccsbind, 30, 56, 84
ccsgenrcfiles, 35
ccsinfo, 26, 30, 37, 45, 67, 90
ccskill, 30, 89, 92
ccsmsg, 30, 84
ccssignal, 30, 83
ccstracejob, 17, 20, 30, 32, 79
ccsworker, 53
ccsx, 17
chunk, 45
CLI, 32
credit, 20, 51
down, 159

EM, 16, 56
exclusive, 159

FOO, 13
group, 19
Hold, 83

IM, 16
ISLAND_AAL_FILE, 20

last, 88

MM, 16
INDEX

my_preproc, 53
NCPUS, 38
NSM, 16

offline, 159
OMP_NUM_THREADS, 38
online, 159
OS, 16

PLANNED, 17, 31
PLANNING, 17, 31
PM, 15
Privilege, 78
profile.d, 35

reqID, 31
Resume, 83
rsvid, 85

sick, 159
SJID, 88
STOPPED, 17, 32
STOPPING, 17, 31

TMPDIR, 38

UI, 15
uirc, 33
unknown, 159
up, 159

WAITING, 17, 32
WLM, 15
Worker, 22