

Gold User's Guide

Scott Jackson
Pacific Northwest National Laboratory

Gold User's Guide

by Scott Jackson

Copyright © 2004 by Pacific Northwest National Laboratory, Battelle Memorial Institute.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Battelle nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

Notice	7
1. Overview	9
Background	9
Features.....	9
Interfaces.....	10
Command Line Clients	10
Interactive Control Program	11
Web-based Graphical User Interface	11
Perl API	12
Java API.....	12
SSSRMAP Wire Protocol.....	12
2. Getting Started	15
Define Users	15
Define Machines	15
Define Projects	16
Add Users and Machines to the Projects	16
Make Deposits	16
Check The Balance	17
Define Charge Rates.....	18
Integrate Gold with your Resource Management System	18
Obtain A Job Quote.....	18
Make A Job Reservation.....	19
Charge for a Job	19
Refund a Job.....	20
List Transactions.....	21
3. Managing Users	23
Creating Users	23
Querying Users.....	23
Modifying Users.....	23
Deleting Users.....	24
4. Managing Machines.....	25
Creating Machines	25
Querying Machines.....	25
Modifying Machines.....	25
Deleting Machines.....	25
5. Managing Projects	27
Creating Projects.....	27
Querying Projects	27
Modifying Projects	28
Deleting Projects.....	28
6. Managing Allocations.....	31
Making Deposits.....	31
Creating Allocations	31
Querying Allocations.....	32
Querying The Balance	33
Modifying Allocations.....	33
Making Withdrawals	34
Making Transfers.....	34
Deleting Allocations.....	35
7. Managing Reservations	37
Creating Reservations.....	37
Querying Reservations	37

Modifying Reservations	37
Deleting Reservations	37
8. Managing Quotations	39
Creating Quotations	39
Querying Quotations	39
Modifying Quotations	39
Deleting Quotations	39
9. Managing Charge Rates.....	41
Creating ChargeRates	41
Querying ChargeRates	41
Modifying Charge Rates	42
Deleting Charge Rates	42
10. Managing Jobs.....	45
Creating Jobs	45
Querying Jobs	45
Modifying Jobs	45
Deleting Jobs	46
Obtaining Job Quotes	46
Making Job Reservations.....	47
Charging Jobs.....	47
Issuing Job Refunds	48
11. Managing Transactions	49
Querying Transactions.....	49
12. Integration with the Resource Management System.....	51
Dynamic versus Delayed Accounting.....	51
Delayed Accounting.....	51
Dynamic Accounting	51
Interaction Points	51
Job Quotation @ Job Submission Time [Optional — Recommended]	51
Job Reservation @ Job Start Time [Optional — Highly Recommended] ..	51
Job Charge @ Job End Time [Required].....	52
Methods of interacting with Gold	52
Configuring an application that already has hooks for Gold	52
Using the appropriate command-line client.....	53
Using the Gold control program	53
Use the Perl API.....	53
Use the Java API	54
Communicating via the SSSRMAP Protocol	54

Notice

Important: This User's Guide is in an early alpha release and is incomplete. Additional documentation will be forthcoming in future releases.

Notice

Chapter 1. Overview

Gold is a unique open source dynamically customizable information service. This manual describes Gold as used in the context of a dynamic accounting and allocation management system. The Gold Accounting and Allocation Manager manages the utilization of computational resources in a multi-project environment. It is used in conjunction with a resource management (batch) system allowing an organization to guarantee greater fairness and enforce mission priorities. It does this by associating a charge with the use of computational resources and allocating resource credits which limit how much of the resources may be used at what time and by whom. It tracks resource utilization and allows for insightful planning.

Background

In order for an organization to efficiently use its high performance computers, it must be able to allocate resources to the users and projects that need them most in a manner that is fair and according to mission objectives. Tracking the historical resource usage allows for insightful capacity planning and in making decisions on how to best mete out these resources. It allows the funding sources that have invested heavily in a supercomputing resource a means to show that it is being utilized efficiently. Additionally, accounting and allocation management are critical to being able to take advantage of the tremendous utilization gains afforded by meta-scheduling.

The Gold Accounting and Allocation Manager tracks and manages job and resource usage. Much like a bank, an allocation manager associates a cost to computing resources and allows resource credits to be allocated to users and projects and meted out in a fair and judicious manner. As jobs complete or as resources are utilized, projects are dynamically charged and resource usage recorded.

Gold is being developed at PNNL as open source software under the Scalable Systems Software (SSS) SciDAC project. A standard communication protocol has been designed to facilitate communication between resource management and accounting components and to allow component substitution. Gold is currently in alpha release and will soon begin alpha testing at a number of DOE and university sites. A flexible GUI is being developed to simplify use and the management of project and accounting data.

Features

- *Dynamic Charging* — Rather than post-processing resource usage records on a periodic basis to rectify project accounts, allocations are updated immediately at job completion.
- *Reservations* — A hold is placed against the account for the estimated number of resource credits before the job runs, followed by an appropriate withdrawal at the moment the job completes, thereby preventing projects from using more resources than were allocated to them.
- *Allocations* — A uniquely flexible allocation design allows the attribution of resource credits to any combination of users, projects, machines and time.
- *Expiration Cycles* — Allocations support activation and expiration times allowing sites to implement a use-it-or-lose-it policy to prevent year-end resource exhaustion and allow for insightful capacity planning.

- *Flexible Charging* — The system can track and charge for composite resource usage (memory, disk, CPU, etc) and custom charge multipliers can be applied (Quality of Service, Node Type, Time of Day, etc).
- *Guaranteed Quotes* — Users and resource brokers can determine ahead of time the cost of using resources.
- *Credit and Debit Allocations* — Sites can base allocations on credit or debit models and even enable overdraft protection for specific projects.
- *Nested Projects* — Hierarchical project support allows the management of nested subprojects and the automatic trickle-down of deposited resource credits.
- *Powerful Querying* — Gold supports a powerful querying and update mechanism that facilitates flexible reporting and streamlines administrative tasks.
- *Transparency* — Support for default projects and generic user accounts allows job submitters to use the system without even knowing it.
- *Security* — Gold supports multiple security mechanisms for strong authentication and encryption.
- *Role Based Authorization* — Gold provides fine-grained (instance-level) Role Based Access Control for its operations.
- *Dynamic Customization* — Sites can create or modify record types on the fly enabling them to meet their custom accounting needs. Dynamic object creation allows sites to customize the types of accounting data they collect without modifying the code. This capability turns this system into a generalized information service. This capability is extremely powerful and can be used to manage all varieties of custom configuration data, to provide meta-scheduling resource mapping, or to function as a persistence interface for other components.
- *Multi-Site Exchange* — A traceback mechanism will allow all parties of a transaction (resource requestor and provider) to have a first-hand record of the resource utilization and to have a say as to whether or not the job should be permitted to run, based on their independent policies and priorities. A job will only run if all parties are agreeable to the idea that the target resources can be used in the manner and amount requested. Support for traceback debits will facilitate the establishment of trust and exchange relationships between administrative domains.
- *Web Interface* — Gold will implement a powerful web-based GUI for easy remote access for users, managers and administrators.
- *Journaling* — Gold implements a journaling mechanism that preserves the indefinite historical state of all objects and records. This powerful mechanism allows historical bank statements to be generated, provides an undo/redo capability and allows commands to be run as if it were any arbitrary time in the past.
- *Open Source* — Being open source allows for site self-sufficiency, customizability and promotes community development and interoperability.

Interfaces

Gold provides a variety of means of interaction, including command-line interfaces, graphical user interfaces, application programming interfaces and communication protocols.

Command Line Clients

The command-line clients provided feature rich argument sets and built-in documentation. These commands allow scripting and are the preferred way to interact with Gold for basic usage and administration. Use the `-help` option for usage information or the `-man` option for a manual page on any command.

Example 1-1. Listing Users

```
glsuser
```

Interactive Control Program

The gold command uses a control language to issue object-oriented requests to the server and display the results. The commands may be included directly as command-line arguments or read from stdin. Use the "ShowUsage:=True" option after a valid Object Action combination for usage information on the command.

Example 1-2. Listing Users

```
gold User Query
```

Caution

The gold control program allows you to make powerful and sweeping modifications to gold objects. Do not use this command unless you understand the syntax and the potential for unintended results.

Web-based Graphical User Interface

A powerful and easy-to-use web-based GUI is being developed for use by users, managers and administrators. It sports two interface types:

- *Management Interface* — The management interface supports an interface that makes administration and interaction very safe and easy. It approaches things from a functional standpoint, aggregating results and protecting against accidental modifications.
- *Object Interface* — The object interface exposes you to the full power of the actions the server can perform on the objects. This interface allows actions to be performed on many objects in a single command and can impose arbitrary field conditions, field updates and field selections to the query.

Example 1-3. Listing Users

Click on "Manage Users" -> "List Users"

Note: The gold web gui is still in an early development phase and although it is included, it is not yet ready for general use.

Perl API

You can access the full Gold functionality via the Perl API. Use perldoc to obtain usage information for the Perl Gold modules.

Example 1-4. Listing Users

```
use Gold;

my $request = new Gold::Request(object => "User", action => "Query");
my $response = $request->getResponse();
foreach my $datum ($response->getData())
{
    print $datum->toString(), "\n";
}
```

Java API

You can also access Gold operations via a Java API. This is used by the web GUI which uses Java Server Pages. The javadoc command can be run on the src/gold directory to generate documentation for the gold java classes.

Example 1-5. Listing Users

```
import java.util.*;
import gold.*;

public class Test
{
    public static void main(String [] args) throws Exception
    {
        Gold.initialize();
        Request request = new Request("User", "Query");
        Response response = request.getResponse();
        Iterator dataItr = response.getData().iterator();
        while (dataItr.hasNext())
        {
            System.out.println(((Datum)dataItr.next()).toString());
        }
    }
}
```

SSSRMAP Wire Protocol

It is also possible to interact with Gold by directly using the SSSRMAP Wire Protocol and Message Format over the network. Documentation for these protocols can be found at *SSS Resource Management and Accounting Documentation*¹.

Example 1-6. Listing Users

```
POST /SSSRMAP HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Transfer-Encoding: chunked

190
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body actor="scottmo" chunking="True">
    <Request action="Query" object="User"></Request>
  </Body>
  <Signature>
    <DigestValue>azu4obZswzBt89OgATukBeLyt6Y=</DigestValue>
    <SignatureValue>YXE/C08XX3RX4PMU1bWju+5/E5M=</SignatureValue>
    <SecurityToken type="Symmetric" name="scottmo"></SecurityToken>
  </Signature>
</Envelope>
0
```

Notes

1. <http://sss.scl.ameslab.gov/docs.shtml>

Chapter 1. Overview

Chapter 2. Getting Started

In order to prepare Gold for use as an allocation manager, you will need to perform some initial steps to define users, machines and projects, create allocations, establish charge rates, etc. This chapter proceeds by offering a number of examples in performing these steps. Use these steps as a guide, substituting values and options appropriate for your system.

Important: You will need to be a Gold System Administrator to perform the tasks in this chapter!

Define Users

First, you will need to define the users that will use, manage or administer the resources (see Creating Users).

Example 2-1. Let's add the users amy, bob and charlie.

```
$ gmkuser -n "Wilkes, Amy" -E "amy@western.edu" amy
```

```
Successfully created 1 User
```

```
$ gmkuser -n "Novak, Bob T." -E "bob@western.edu" bob
```

```
Successfully created 1 User
```

```
$ gmkuser -n "Brown, Charles Winston (III)" -E "trey@western.edu" charlie
```

```
Successfully created 1 User
```

```
$ glsuser
```

Name	Active	CommonName	PhoneNumber	EmailAddress	De-
faultProject	Description				
-----	-----	-----	-----	-----	-----
amy	t	Wilkes, Amy		amy@western.edu	
bob	t	Novak, Bob T.		bob@western.edu	
charlie	t	Brown, Charles Winston (III)		trey@western.edu	

Define Machines

You may want to add the names of the machines that provide resources (see Creating Machines).

Example 2-2. Let's define machines called colony and blue.

```
$ gmkmachine -d "Linux Cluster" colony
```

```
Successfully created 1 Machine
```

```
$ gmkmachine -d "IBM SP2" blue
```

```
Successfully created 1 Machine
```

```
$ glsmachine
Name   Active Description
-----
colony t       Linux Cluster
blue   t       IBM SP2
```

Define Projects

Next you should create the projects that will use the resources (see Creating Projects).

Example 2-3. We will define the projects weather and genome.

```
$ gmkproject -d "Weather Modelling" weather
Successfully created 1 Project

$ gmkproject -d "Human Genome Sequencing" genome
Successfully created 1 Project
```

Add Users and Machines to the Projects

Although this could have been done at the project creation step, you can now assign users to be members of your projects (see Modifying Projects). Additionally, you can assign a default set of machines that may be used by the projects.

Example 2-4. Adding users and default machines to our projects.

```
$ gchproject --add Users=amy,bob,charlie Machine=colony weather

Successfully created 1 ProjectMachine
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser

$ gchproject --add User=amy genome
Successfully created 1 ProjectUser

$ glsproject
Name   Active Users           Projects Machines Description
-----
weather t       amy,bob,charlie        colony   Weather Modelling
genome t       amy,charlie            Human Genome Sequencing
```

Make Deposits

Now, you need to make some deposits (see Making Deposits).

Example 2-5. Let's add 3600000 credits each to weather and genome. We will cause both to expire at the end of the year.

```
$ gdeposit -p weather -m MEMBER -e 2005-01-01 3600000
```

```
Successfully deposited 3600000 credits
```

```
$ gdeposit -p genome -m ANY -e 2005-01-01 3600000
```

```
Successfully deposited 3600000 credits
```

```
$ gsalloc
```

Id	Type	Amount	CreditLimit	Active	ActivationTime	ExpirationTime	Projects	Users	Ma-
		Description						chines	
1	Debit	3600000	0	t	-infinity	2005-01-01	weather	MEM-	
	BER							MEM-	
2	Debit	3600000	0	t	-infinity	2005-01-01	genome	MEM-	
	BER							MEM-	
									ANY

Check The Balance

You can verify the resulting balance (see Querying The Balance).

Example 2-6. Let's look at amy's balance

```
$ gbalance -u amy
```

```
Balance
```

```
-----
```

```
7200000
```

```
The allocation balance is 7200000 credits
```

Example 2-7. It is often useful to get more detail on the balance composition

```
$ gbalance -u amy -x
```

Amount	Projects	Users	Machines	ActivationTime	ExpirationTime
3600000	weather	MEMBER	MEMBER	-infinity	2005-01-01
3600000	genome	MEMBER	ANY	-infinity	2005-01-01

Here, we notice that not all of the 7200000 credits are valid toward a single project (or machine).

Example 2-8. We may want to get more specific

```
$ gbalance -u amy -p weather -m colony -q
3600000
```

Define Charge Rates

Finally, you should define how much you will charge for your resources (see [Creating ChargeRates](#)).

Example 2-9. Let's just charge for the number of processors used.

```
$ gold ChargeRate Create Type=Resource Name=Processors Rate=1
Successfully created 1 ChargeRate
```

```
$ gold ChargeRate Query
Type      Name      Rate Machine Description
-----
Resource Processors 1
```

Integrate Gold with your Resource Management System

Now you are ready to run some jobs. Before doing so you will need to integrate Gold with your Resource Management System (see [Integrating with the Resource Management System](#)).

Although the quotation, reservation and charge steps will most likely be taken care of by your resource management system, it is useful to understand their effects.

Let's simulate the lifecycle of a job.

Example 2-10. We'll assume our job has the following characteristics:

```
Job Id:          PBS.1234.0
Job Name:        seattle
User Name:       amy
Project Name:    weather
Machine Name:    colony
Requested Processors: 16
Estimated WallClock: 3600 seconds
Actual WallClock: 1234 seconds
```

Obtain A Job Quote

When a job is submitted, it is useful to check that the user's account has enough funds to run the job. This will be verified when the job starts, but by that point the job may have waited some time in the queue only to find out it never could have run in the first place. The job quotation step (see [Obtaining Job Quotes](#)) can fill this function.

Additionally, the quote can be used to determine the cheapest place to run, and to guarantee the current rates will be used when the job is charged.

Example 2-11. Let's see how much it will cost to run our job.

```
$ gquote -p weather -u amy -m colony -P 16 -t 3600
```

```
Successfully quoted 57600 credits for quote 1
```

```
$ glsquote
```

Id	Project	User	Machine	Amount	ExpirationTime	WallDuration	Used	ChargeRates	Description
1	weather	amy	colony	57600	2004-04-04 10:07:18	3600	0		Resource:Processor

Make A Job Reservation

When a job starts, it creates a reservation (or pending charge) against the appropriate allocations based on the estimated wallclock limit specified for the job (see Making a Job Reservation).

Example 2-12. Make a reservation for our job.

```
$ greserve -j PBS.1234.0 -p weather -u amy -m colony -P 16 -t 3600
```

```
Successfully reserved 57600 credits for job PBS.1234.0
```

```
$ glsres
```

Id	JobId	Amount	ExpirationTime	Allocations	Description
1	PBS.1234.0	57600	2004-04-01 10:13:07	1:57600	

This reservation will decrease our available balance by the amount reserved.

```
$ gbalance -p weather -x
```

Amount	Projects	Users	Machines	ActivationTime	ExpirationTime
3542400	weather	MEMBER	MEMBER	-infinity	2005-01-01

Although our allocation has not changed.

```
$ gsalloc -p weather
```

Id	Type	Amount	CreditLimit	Active	ActivationTime	ExpirationTime	Projects	Users	Machines	Description
1	Debit	3600000	0	t	-infinity	2005-01-01	weather	MEMBER	MEMBER	

Charge for a Job

After a job completes, any associated reservations are removed and a charge is issued against the appropriate allocations based on the actual wallclock time used by the job (see Charging Jobs).

Example 2-13. Issue the charge for our job.

```
$ gcharge -j PBS.1234.0 -p weather -u amy -m colony -P 16 -t 1234
```

```
Successfully charged job PBS.1234.0 for 19744 credits
1 reservations were removed
```

Your allocation will now have gone down by the amount of the charge.

```
$ glsalloc -p weather
```

Id	Type	Amount	CreditLimit	Active	ActivationTime	ExpirationTime	Projects	Users	Ma-
		Description							
1	Debit	3580256	0	t	-infinity	2005-01-01	weather	MEM-	
		BER MEMBER							

However, your available balance actually goes up (because the reservation that was removed was larger than the actual charge).

```
$ gbalance -p weather -x
```

Amount	Projects	Users	Machines	ActivationTime	ExpirationTime
3580256	weather	MEMBER	MEMBER	-infinity	2005-01-01

Refund a Job

Now, since this was an imaginary job, you had better refund the user's account (see Issuing Job Refunds).

Example 2-14. Let's issue a refund for our job.

```
$ grefund -j PBS.1234.0
```

```
Successfully refunded job PBS.1234.0 for 19744 credits
```

Our balance is back as it was before the job ran.

```
$ gbalance -p weather -x
```

Amount	Projects	Users	Machines	ActivationTime	ExpirationTime
3600000	weather	MEMBER	MEMBER	-infinity	2005-01-01

The allocation, of course, is likewise restored.

```
$ glsalloc -p weather
```

Id	Type	Amount	CreditLimit	Active	ActivationTime	ExpirationTime	Projects	Users	Ma-
		Description							
1	Debit	3580256	0	t	-infinity	2005-01-01	weather	MEM-	
		BER MEMBER							

```

-----
1 Debit 3600000 0          t          -infinity      2005-01-01    weather MEM-
BER MEMBER

```

List Transactions

You can now check the resulting transaction records (see Querying Transactions).

Example 2-15. Let's list all the job transactions

```
$ glstxn -o Job -show="RequestId,TransactionId,Object,Action,JobId,Project,User,Machine,Amount"
```

RequestId	TransactionId	Object	Action	JobId	Project	User	Machine	Amount
829	421	Job	Quote					57600
830	426	Job	Reserve	PBS.1234.0				57600
833	442	Job	Create	PBS.1234.0	weather	amy	colony	
833	443	Job	Modify					
833	444	Job	Charge	PBS.1234.0				19744
860	454	Job	Modify					
860	455	Job	Refund	PBS.1234.0	weather	amy	colony	19744

Example 2-16. It may also be useful to examine what transactions actually composed our three job requests (quote, reserve and charge) ...

```
$ glstxn -R 829 -show="Id,Object,Action,Name,Amount,Debit,Credit,Delta"
```

Id	Object	Action	Name	Amount	Debit	Credit	Delta
418	Quotation	Create			f	f	0
419	QuotationChargeRate	Create	1		f	f	0
420	Quotation	Modify	1	57600	f	f	0
421	Job	Quote		57600	f	f	0

```
$ glstxn -R 830 -show="Id,Object,Action,Name,Amount,Debit,Credit,Delta"
```

Id	Object	Action	Name	Amount	Debit	Credit	Delta
422	Reservation	Create		0	f	f	0
423	ReservationAllocation	Create	1	57600	f	f	0
424	Reservation	Modify	1	57600	f	f	0
425	Reservation	Modify	1		f	f	0
426	Job	Reserve	PBS.1234.0	57600	f	f	0

```
$ glstxn -R 833 -show="Id,Object,Action,Name,Amount,Debit,Credit,Delta"
```

Id	Object	Action	Name	Amount	Debit	Credit	Delta
436	Usage	Post		16	f	f	0
437	Reservation	Delete			f	f	1
438	ReservationAllocation	Delete	1		f	f	1

Chapter 2. Getting Started

439	Reservation	Modify	1	57600	f	f	0
440	Allocation	Modify	1	19744	f	f	0
441	Allocation	Withdraw	1	19744	t	f	19744
442	Job	Create			f	f	0
443	Job	Modify	1		f	f	0
444	Job	Charge	PBS.1234.0	19744	f	f	0

\$ glstxn -R 860 -show="Id,Object,Action,Name,Amount,Debit,Credit,Delta"

Id	Object	Action	Name	Amount	Debit	Credit	Delta
452	Allocation	Modify	1	19744	f	f	0
453	Allocation	Deposit	1	19744	f	t	19744
454	Job	Modify	1		f	f	0
455	Job	Refund	1	19744	f	f	0

Chapter 3. Managing Users

A user is a person authorized to submit jobs to run on a high performance computing resource. User properties include the common name, phone number, email, and default project for that person. A user can be created, queried, modified and deleted.

Creating Users

To create a new user, use the command **gmkuser**:

```
gmkuser [-A | -I] [-n common_name] [-F phone_number] [-E email_address] [-p default_project] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] user_name
```

Example 3-1. Creating a user

```
$ gmkuser -n "Smith, Joe F." -E "Joe.Smith@lab.gov" -F "(509) 555-1234" jsmith  
Successfully created 1 User
```

Querying Users

To display user information, use the command **glsuser**:

```
glsuser [-A | -I] [-show attribute_name[,attribute_name...]] [-showHidden] [-showSpecial] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [user_pattern]
```

Example 3-2. Listing all info about active users

```
$ glsuser -A  
Name      Active CommonName      PhoneNumber      EmailAddress      Default-  
Project  Description  
-----  
jsmith t      Smith, Joe F. (509) 555-1234 Joe.Smith@lab.gov
```

Example 3-3. Displaying joe's phone number

```
$ glsuser --show PhoneNumber jsmith -q  
(509) 555-1234
```

Example 3-4. Listing all user names without the header

```
$ glsuser --show Name -q  
jsmith
```

Modifying Users

To modify a user, use the command **gchuser**:

```
gchuser [-A | -I] [-n common_name] [-F phone_number] [-E email_address] [-p  
default_project] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-  
v | -verbose] user_name
```

Example 3-5. Activating a user

```
$ gchuser -A jsmith  
Successfully modified 1 User
```

Example 3-6. Changing a user's email address

```
$ gchuser -E "jsmith@cs.univ.edu" jsmith  
Successfully modified 1 User
```

Deleting Users

To delete a user, use the command **grmuser**:

```
grmuser [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] user_name
```

Example 3-7. Deleting a user

```
$ grmuser jsmith  
Successfully deleted 1 User
```

Chapter 4. Managing Machines

A machine is a resource that can run jobs such as a cluster or an SMP box. Machine properties include the description and whether it is active. A machine can be created, queried, modified and deleted.

Creating Machines

To create a new machine, use the command **gmkmachine**:

```
gmkmachine [-A | -I] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] machine_name
```

Example 4-1. Creating a machine

```
$ gmkmachine -d "Alpha Cluster" cluster  
Successfully created 1 Machine
```

Querying Machines

To display machine information, use the command **glsmachine**:

```
glsmachine [-A | -I] [-show attribute_name,attribute_name...]... [-showHidden] [-showSpecial] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [machine_pattern]
```

Example 4-2. Listing all inactive machine names and descriptions

```
$ glsmachine -I --show Name,Description  
Name Description  
-----  
inert This machine is inactive
```

Modifying Machines

To modify a machine, use the command **gchmachine**:

```
gchmachine [-A | -I] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] machine_name
```

Example 4-3. Deactivating a machine

```
$ gchmachine -I cluster  
Successfully modified 1 Machine
```

Deleting Machines

To delete a machine, use the command **grmmachine**:

```
grmmachine [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] machine_name
```

Example 4-4. Deleting a machine

```
$ grmmachine cluster  
Successfully deleted 1 Machine
```

Chapter 5. Managing Projects

A project is a named account, usually with a common resource usage purpose, against which resources are allocated and tracked. Users may be designated as project members and allowed to share its allocations. Machines may be designated as members of a project as a default resource pool. Additionally, since projects can be nested, other projects can be designated as members of a project.

Creating Projects

To create a new project, use the command **gmkproject**:

```
gmkproject [-A | -I] [-d description] [-members [Users= [+ | -]user_name [, [+ | -]user_name...]] [Projects= [+ | -]project_name [, [+ | -]project_name...]] [Machines= [+ | -]machine_name [, [+ | -]machine_name...]]... [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] project_name
```

Example 5-1. Creating a project

```
$ gmkproject chem  
Successfully created 1 Project
```

Querying Projects

To display project information, use the command **glsproject**:

```
glsproject [-A | -I] [-show attribute_name [,attribute_name...]] [-showHidden] [-showSpecial] [-l | -long] [-w | -wide] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [project_pattern]
```

Example 5-2. Listing all info about all projects

```
$ glsproject  
Name Active Users           Projects Machines Description  
---- -
```

chem	t	amy, jsmith, scott	alpha	
math	t	larry, jsmith		

Example 5-3. Displaying the name and user members of a project in long format

```
$ glsproject --show Name,Users -l chem  
Name Users  
---- -  
chem amy  
      jsmith  
      scott
```

Example 5-4. Listing all project names

```
$ glsproject --show Name -q
chem
math
```

Modifying Projects

To modify a project, use the command **gchproject**:

```
gchproject [-A | -I] [-d description] [-addMembers [Users=[+ | -]user_name [, [+ | -]user_name...]] [Projects=[+ | -]project_name [, [+ | -]project_name...]] [Machines=[+ | -]machine_name [, [+ | -]machine_name...]]... [-delMembers [Users=[+ | -]user_name [, [+ | -]user_name...]] [Projects=[+ | -]project_name [, [+ | -]project_name...]] [Machines=[+ | -]machine_name [, [+ | -]machine_name...]]... [-activateMembers [Users=user_name [, user_name...]] [Projects=project_name [, project_name...]] [Machines=machine_name [, machine_name...]]...] [-deactivateMembers [Users=user_name [, user_name...]] [Projects=project_name [, project_name...]] [Machines=machine_name [, machine_name...]]...] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] project_name
```

Example 5-5. Deactivating a project

```
$ gchproject -I chem
Successfully modified 1 Project
```

Example 5-6. Adding multiple users as members of a project

```
$ gchproject --addMembers Users=jsmith,barney chem
Successfully created 2 ProjectUsers
```

Tip: Flags can be shortened if they can be resolved unambiguously.

Example 5-7. Deactivating a user member of a project

```
$ gchproject --deactivate User=barney chem
Successfully modified 1 ProjectUser
```

Example 5-8. Making a project a subproject of another

```
$ gchproject --add Project=sulfates chem
Successfully created 1 ProjectProject
```

Deleting Projects

To delete a project, use the command **grmproject**:

```
grmproject [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] project_name
```

Example 5-9. Deleting a project

```
$ grmproject chem  
Successfully deleted 1 Project
```


Chapter 6. Managing Allocations

An allocation is a pool of resource credits with a particular set of expenditure requirements. An allocation may restrict the projects that charge to it. Normally an allocation will be tied to a single project but it may be tied to an arbitrary set of projects or even ANY project. An allocation may restrict the users that can charge to it. It will frequently be tied to the the user MEMBERS of the associated project(s) but it may be tied to any arbitrary set of users or ANY user. An allocation may restrict the machines that can charge to it. It may be tied to any arbitrary set of machines, just the machine MEMBERS of the associated project(s) or ANY machine. An allocation may be restricted to a particular timeframe. It may designate an activationTime representing the time at which it becomes active as well as an expirationTime representing the time at which it ceases to be active.

An allocation may be of type debit or credit. By default, allocations are debit-based where credits are deposited in advance and used until they are gone. These credits may be issued as a grant by an allocation review committee or on a pay first, use later basis. Credit-based allocations support a negative balance up to some limit and are used on a use first, pay later basis. They are established with a credit limit and are generally started with a zero balance. Credit-based allocations may also be initialized with a positive balance with the creditlimit serving as an overdraft buffer.

Making Deposits

gdeposit is the recommended way to create and replenish allocations since it takes into account automatic allocation merging, uses deposit share rules to reduce parameter specifications, and trickles deposits down hierarchical project trees. See also **gmkalloc** which can be used for hand-crafting new allocations. It is intended for use by administrators to explicitly create the allocations in a very controlled manner. For example, member exclusions can only be specified in **gmkalloc**.

```
gdeposit [-s activation_time] [-e expiration_time] [-T type] [-L credit_limit] [-p project_name...] [-u user_name...] [-m machine_name...] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] [{-z} amount]
```

Example 6-1. Making a deposit

```
$ gdeposit -p chem -m cluster 3600000  
Successfully deposited 3600000 credits
```

Creating Allocations

gmkalloc is used for hand-crafting new allocations. It is intended for use by administrators to explicitly create the allocations in a very controlled manner. See also **gdeposit** which is the recommended way to create allocations but takes into account automatic allocation merging, uses deposit share rules to simplify repetitive tasks, and trickles deposits down a hierarchical project tree.

```
gmkalloc [-s activation_time] [-e expiration_time] [-T type] [-L credit_limit] [-d description] [-p project_name...] [-u user_name...] [-m machine_name...] [-members [Users=[+ | -]user_name [, [+ | -]user_name...]] [Projects=[+ | -]project_name [, [+ |
```

```
-]project_name...]] [Machines=[+ | -]machine_name [, [+ | -]machine_name...]]... [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] -z amount
```

Important: When creating an allocation, it is important to explicitly specify at least one each of Users, Machines and Projects as members.

Example 6-2. Creating an allocation

```
$ gmkalloc --members Projects=chem Users=jsmith,barney Machines=ANY
-z 3600000

Successfully created 1 Allocation
Successfully created 1 AllocationUser
Successfully created 1 AllocationUser
Successfully created 1 AllocationProject
Successfully created 1 AllocationMachine
```

Querying Allocations

To display allocation information, use the command **glsalloc**:

```
glsalloc [-A | -I] [-p project_name] [-u user_name] [-m machine_name] [-s activation_time] [-e expiration_time] [-T type] [-subtractReservationsFromAmount] [-subtractCreditLimitFromAmount] [-useRules] [-show attribute_name [,attribute_name...]] [-showHidden] [-l | -long] [-w | -wide] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [allocation_id_pattern]
```

Example 6-3. Listing all info about all debit allocations useable by the chem project on machine cluster

```
$ glsalloc -T Debit -p chem -m cluster --useRules

Id Type Amount CreditLimit Active ActivationTime ExpirationTime Projects Users
chines Description
-----
-----
3 Debit 3600100 0 t -infinity infinity chem jsmith,barney
2 Debit 3597332 0 t -infinity infinity chem -
barney,MEMBER ANY
```

Example 6-4. Listing all info about all allocations useable by joe, with the amount field showing the amount available to joe for charging

```
$ glsalloc -u jsmith --useRules --subtractReservationsFromAmount

Id Type Amount CreditLimit Active ActivationTime ExpirationTime Projects Users
chines Description
-----
-----
3 Debit 3600100 0 t -infinity infinity chem jsmith,barney
```

```
2 Debit 3597332 0          t          -infinity          infinity          chem          -
barney, MEMBER ANY
```

Example 6-5. Listing all info about all allocations useable by joe, with the amount field showing joe's accountable balance

```
$ glsalloc -u jsmith --useRules --subtractCreditLimitFromAmount
```

Querying The Balance

To display balance information, use the command **gbalance**:

```
gbalance [-p project_name] [-u user_name] [-m machine_name] [-available] [-show attribute_name [,attribute_name...]] [-x | -expand] [-l | -long] [-w | -wide] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet]
```

Example 6-6. Querying the balance for a particular user in a particular project on a particular machine

```
$ gbalance -u jsmith -m cluster -p chem
```

Example 6-7. Querying the amount available for charging including available credit for a particular user in a particular project on a particular machine

```
$ gbalance -u jsmith -m cluster -p chem --available
```

Example 6-8. Querying the project balance, with all allocations contributing to this balance enumerated

```
$ gbalance -p chem -x
```

Example 6-9. Querying the balance available on a particular machine, with all allocations contributing to this balance enumerated and multi-valued fields displayed in a multi-line format

```
$ gbalance -m cluster -x -l
```

Modifying Allocations

To modify an allocation, use the command **gchalloc**:

```
gchalloc [-z amount] [-s activation_time] [-e expiration_time] [-L credit_limit] [-d description] [-addMembers [Users=[+ | -]user_name [, [+ | -]user_name...]] [Projects=[+ | -]project_name [, [+ | -]project_name...]] [Machines=[+ | -]machine_name [, [+ | -]machine_name...]]... [-delMembers [Users=[+ | -]user_name [, [+ | -]user_name...]] [Projects=[+ | -]project_name [, [+ | -]project_name...]] [Machines=[+ | -]machine_name [, [+ | -]machine_name...]]... [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] allocation_id
```

Example 6-10. Changing the expiration time of an allocation

```
$ gchalloc -e "2005-03-01" 1  
Successfully modified 1 Allocation
```

Example 6-11. Adding a user to the list of users that share the allocation

```
$ gchalloc --addMember User=barney 1  
Successfully created 1 AllocationUser
```

Making Withdrawals

To issue a withdrawal, use the command **gwithdraw**:

```
gwithdraw { [-p project_name] [-u user_name] [-m machine_name] | -n allocation_id} [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] [-z amount
```

Example 6-12. Changing the expiration time of an allocation

```
$ gwithdraw -u jsmith -p chem -m cluster 1280  
Successfully withdrew 1280 credits
```

Example 6-13. Making a withdrawal against a particular allocation

```
$ gwithdraw -n 1 1280  
Successfully withdrew 1280 credits
```

Making Transfers

To issue a transfer between allocations, use the command **gtransfer**. The credits will be transferred equally in a round-robin fashion from all of the allocations matching the from-constraints to all of the allocations matching the to-constraints.

```
gtransfer {-fromId allocation_id | [-fromProject project_name] [-fromUser user_name] [-fromMachine machine_name]} {-toId allocation_id | [-toProject project_name] [-toUser user_name] [-toMachine machine_name]} [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] [-z] amount
```

Example 6-14. Transferring credits between two allocations

```
$ gtransfer -fromId 1 -toId 2 10000
Successfully transferred 10000 credits
```

Example 6-15. Transferring credits from one project to another

```
$ gtransfer --fromProject chem --toProject math 10000
Successfully transferred 10000 credits
```

Deleting Allocations

To delete an allocation, use the command **grmalloc**:

```
grmalloc [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] {allocation_id | -I}
```

Example 6-16. Deleting an allocation

```
$ grmalloc 1
Successfully deleted 1 Allocation
```

Example 6-17. Purging inactive allocations

```
$ grmalloc -I
Successfully deleted 2 Allocations
```


Chapter 7. Managing Reservations

A reservation is a hold placed against allocated credits. Before a job runs, a reservation (or hold) is made against one or more of the requesting user's applicable allocation(s). Subsequent jobs will also post reservations while the available balance (balance minus reservations) allows. When a job completes, the reservation is removed and the actual charge is made to the allocation(s). This procedure ensures that jobs will only run so long as they have sufficient reserves.

Associated with a reservation is the jobId of the job requiring the reservation, an expiration time, and an amount. Operations include creating, querying, modifying and deleting reservations.

Creating Reservations

A new reservation is created when a Job Reserve occurs.

See Making Job Reservations

Querying Reservations

To display reservation information, use the command `glsres`:

```
glsres [-A | -I] [-j job_id_pattern] [-p project_name] [-u user_name] [-m machine_name] [-show attribute_name [attribute_name...]] [-showHidden] [-l | -long] [-w | -wide] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [-n reservation_id_pattern]
```

Example 7-1. Listing all info about all reservations for joe on machine cluster

```
$ glsres -u jsmith -m cluster
```

Modifying Reservations

To modify a reservation, use the command `gchres`:

```
gchres [-e expiration_time] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] reservation_id
```

Example 7-2. Changing the expiration time of a reservation

```
$ gchres -e "2005-03-01" 1  
Successfully modified 1 Reservation
```

Deleting Reservations

To delete a reservation, use the command **grmres**:

```
grmres [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] {-I | -j job_id |  
reservation_id}
```

Example 7-3. Deleting a reservation by JobId

```
$ grmres -j PBS.1234.0  
Successfully deleted 1 Reservation
```

Example 7-4. Deleting a reservation by ReservationId

```
$ grmres 1  
Successfully deleted 1 Reservation
```

Example 7-5. Purging stale reservations

```
$ grmres -I  
Successfully deleted 2 Reservations
```

Chapter 8. Managing Quotations

A quotation provides a way to determine beforehand how much would be charged for a job. When a quotation is requested, the charge rates applicable to the job requesting the quote are saved and a quote id is returned. When the job makes a reservation and the final charge, the quote can be referenced to ensure that the saved charge rates are used instead of current values. A quotation has an expiration time after which it cannot be used. A quotation may also be used to verify that the given job has sufficient funds and meets the policies necessary for the charge to succeed.

Operations include creating, querying, modifying and deleting quotations.

Creating Quotations

A new quote is created when a Job Quote occurs.

See Obtaining Job Quotes

Querying Quotations

To display quotation information, use the command **glsquote**:

```
glsquote [-A | -I] [-p project_name] [-u user_name] [-m machine_name] [-show attribute_name [,attribute_name...]] [-showHidden] [-l | -long] [-w | -wide] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet] [quotation_id_pattern]
```

Example 8-1. Listing all info about all quotes for user joe on machine cluster

```
$ glsquote -u jsmith -m cluster
```

Modifying Quotations

To modify a quotation, use the command **gchquote**:

```
gchquote [-e expiration_time] [-d description] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] quotation_id
```

Example 8-2. Changing the expiration time of a quotation

```
$ gchquote -e "2005-03-01" 1  
Successfully modified 1 Quotation
```

Deleting Quotations

To delete a quotation, use the command **grmquote**:

```
grmquote [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose] {-I | quotation_id}
```

Example 8-3. Deleting a quotation

```
$ grmquote 1  
Successfully deleted 1 Quotation
```

Example 8-4. Purging stale quotations

```
$ grmquote -I  
Successfully deleted 2 Quotations
```

Chapter 9. Managing Charge Rates

Charge Rates establish how much it costs to use your resources. There are two main categories of charge rates, consumable resources and quality-based charge rates. Resource charge rates define how much it costs per unit of time to use a consumable resource like processors, memory, telescope time, etc. Quality-based charge rates apply a multiplicative charge factor related to the quality or class of service obtained such as QOS, nodetype, backlog, primetime, etc.

By default, charges are calculated according to the following formula: For each consumable resource used, a resource charge is calculated by multiplying the amount of the resource used by the amount of time it was used, multiplied by the charge rate for that resource. These resource charges are added together. Then, for each quality-based charge rate, a charge factor is looked-up based on the type and name of the charge rate. The sum of the resource charges is multiplied by each of the applicable charge factors.

Creating ChargeRates

To create a new charge rate, use the command **gold ChargeRate Create**:

```
gold ChargeRate Create Type=<Charge Rate Type> Name=<Charge Rate Name>  
Rate=<Floating Point Multiplier> [Description=<Description>] [ShowUsage:=true]
```

Example 9-1. Creating a resource charge rate

```
$ gold ChargeRate Create Type=Resource Name=Processors Rate=1  
Successfully created 1 ChargeRate
```

Example 9-2. Creating another resource charge rate

```
$ gold ChargeRate Create Type=Resource Name=Memory Rate=0.001  
Successfully created 1 ChargeRate
```

Example 9-3. Creating a quality-based charge rate

```
$ gold ChargeRate Create Type=QOS Name=BottomFeeder Rate=0.5  
Successfully created 1 ChargeRate
```

Example 9-4. Creating another quality-based charge rate

```
$ gold ChargeRate Create Type=QOS Name=Urgent Rate=2  
Successfully created 1 ChargeRate
```

Querying ChargeRates

To display charge rate information, use the command **gold ChargeRate Query**:

```
gold ChargeRate Query [show:=<"Field1,Field2,...">] [Type==<Charge Rate Type>] [Name==<Charge Rate Name>] [Rate==<Floating Point Multiplier>] [Description==<Description>] [ShowUsage:=true]
```

Example 9-5. Listing all charge rates

```
$ gold ChargeRate Query
```

Type	Name	Rate	Machine	Description
Resource	Processors	1		
Resource	Memory	0.001		
QOS	BottomFeeder	0.5		
QOS	Urgent	2		

Modifying Charge Rates

To modify a charge rate, use the command **gold ChargeRate Modify**:

```
gold ChargeRate Modify [Rate=<Floating Point Multiplier>] [Description=<Description>] [Rate Type=<Charge Rate Name>] [Name==<Charge Rate Name>] [Rate==<Floating Point Multiplier>] [ShowUsage:=true]
```

Caution

The gold control program allows you to make powerful and sweeping modifications to gold objects. Misuse of this command could result in the inadvertent modification of all charge rates.

Example 9-6. Changing a charge rate

```
$ gold ChargeRate Modify Type==Resource Name==Processors Rate=0.75
```

```
Successfully modified 1 ChargeRate
```

Deleting Charge Rates

To delete a charge rate, use the command **gold ChargeRate Delete**:

```
gold ChargeRate Delete [Name==<Charge Rate Name>] [Rate==<Floating Point Multiplier>]
```

Caution

The gold control program allows you to make powerful and sweeping modifications to gold objects. Misuse of this command could result in the inadvertant deletion of all charge rates.

Example 9-7. Deleting a charge rate

```
$ gold ChargeRate Delete Type==Resource Name==Memory  
Successfully deleted 1 ChargeRate
```


Chapter 10. Managing Jobs

Gold can track the jobs that run on your system, recording the charges and resources used for each job. Typically, a job record is created when the resource manager charges for a job. Job quotes, reservations, charges and refunds can be issued.

Creating Jobs

A new job is created when a Job Charge occurs.

See Charging Jobs

However, it is also possible to create job records by hand using the command **gold Job Create**:

```
gold Job Create JobId=<Job Id> [User=<User Name>] [Project=<Project Name>] [Machine=<Machine Name>] [Charge=<Charge>] [Class=<Class>] [Type=<Job Type>] [QOS=<Quality Of Service>] [Nodes=<Number Of Nodes>] [Processors=<Number Of Processors>] [State=<Job State>] [Executable=<Executable>] [Application=<Application>] [StartTime=<StartTime>] [CompletionTime=<CompletionTime>] [WallDuration=<Wallclock Time in seconds>] [QuoteId=<Quote Id>] [Description=<Description>] [ShowUsage:=true]
```

Example 10-1. Creating a job record

```
$ gold Job Create JobId=PBS.1234.0 User=jsmith Project=chem Machine=cluster Charge=2468 Processors=2 WallDuration=1234  
Successfully created 1 Job
```

Querying Jobs

To display job information, use the command **glsjob**:

```
glsjob [-j] job_id_pattern] [-n id] [-p project_name] [-u user_name] [-m machine_name] [-C queue] [-T type] [-application application] [-s start_time] [-e end_time] [-show attribute_name[,attribute_name...]] [-showHidden] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet]
```

Example 10-2. Show specific info about jobs run by joe

```
$ glsjob -show=JobId,Project,Machine,Charge -u jsmith  
JobId      Project Machine Charge  
-----  
PBS.1234.0 chem      cluster 2468
```

Modifying Jobs

It is possible to modify a job by using the command **gold Job Modify**:

```
gold Job Modify [JobId==<Job Id> | Id==<Id>] [User=<User Name>] [Project=<Project Name>] [Machine=<Machine Name>] [Charge=<Charge>] [Class=<Class>] [Type=<Job Type>] [QOS=<Quality Of Service>] [Nodes=<Number Of Nodes>] [Processors=<Number Of Processors>] [State=<Job State>] [Executable=<Executable>] [Application=<Application>] [StartTime=<StartTime>] [CompletionTime=<CompletionTime>] [WallDuration=<Wallclock Time in seconds>] [QuoteId=<Quote Id>] [Description=<Description>] [ShowUsage=:true]
```

Caution

The gold control program allows you to make powerful and sweeping modifications to gold objects. Misuse of this command could result in the inadvertent modification of all jobs.

Example 10-3. Changing a job

```
$ gold Job Modify JobId==PBS.1234.0 Charge=1234 Description="Benchmark"

Successfully modified 1 Job
```

Deleting Jobs

To delete a job, use the command **gold Job Delete**:

```
gold Job Delete [JobId==<Job Id> | Id==<Id>]
```

Caution

The gold control program allows you to make powerful and sweeping modifications to gold objects. Misuse of this command could result in the inadvertent deletion of all jobs.

Example 10-4. Deleting a job

```
$ gold Job Delete JobId==PBS.1234.0

Successfully deleted 1 Job
```

Obtaining Job Quotes

Job quotes can be used to determine how much it will cost to run a job. A quote id is returned and can be used in the subsequent charge to guarantee the rates that were used to form the original quote. Since this step also verifies that the submitter has sufficient funds for, and meets all the allocation policy requirements for running a job, it can be used at job submission as an early filter to prevent jobs from getting in and waiting in the job queue just to be blocked from running later.

To request a job quote, use the command **gquote**:

```
gquote [-p project_name] [-u user_name] [-m machine_name] [-P processors] [-M memory] [-D disk] [-Q QOS] [-t wallclock_time] [-debug] [-? | -help] [-man] [-q | -quiet] [-v | -verbose]
```

Example 10-5. Requesting a quotation

```
$ gquote -p chem -u jsmith -m cluster -P 2 -t 3600
Successfully quoted 7200 credits for quote 1
```

Making Job Reservations

A job reservation can be used to place a hold on the user's allocation before a job starts to ensure that the credits will be there when it completes.

To create a job reservation use the command **greserve**:

```
greserve [-p project_name] [-u user_name] [-m machine_name] [-P processors] [-M memory] [-D disk] [-Q QOS] [-t wallclock_time] [-q quote_id] [-debug] [-? | -help] [-man] [-quiet] [-v | -verbose] -j job_id
```

Example 10-6. Creating a reservation

```
$ greserve -j PBS.1234.0 -p chem -u jsmith -m cluster -P 2 -t 3600

Successfully created 1 Reservation
Successfully created 1 ReservationAllocation
```

Charging Jobs

A job charge debits the appropriate allocations based on the user, project and machine associated with the job. The charge is calculated based on factors including the resources used, the job run time, and other quality-based factors (See Managing Charge Rates).

To charge for a job use the command **gcharge**:

```
gcharge [-p project_name] [-u user_name] [-m machine_name] [-P processors] [-N nodes] [-M memory] [-D disk] [-Q QOS] [-t wallclock_time] [-S job_state] [-T job_type] [-application application] [-executable executable] [-C queue] [-s start_time] [-e end_time] [-q quote_id] [-debug] [-? | -help] [-man] [-quiet] [-v | -verbose] -j job_id
```

Example 10-7. Issuing a job charge

```
$ gcharge -j PBS.1234.0 -p chem -u jsmith -m cluster -P 2 -t 1234

Successfully charged job PBS.1234.0 for 2468 credits
1 reservations were removed
```

Issuing Job Refunds

A job can be refunded in part or in whole by issuing a job refund. This action attempts to lookup the referenced job to ensure that the refund does not exceed the original charge and so that the charge entry can be updated. If multiple matches are found (such as the case when job ids are non-unique), this command will return the list of matched jobs with unique ids so that the correct job can be specified for the refund.

To issue a refund for a job, use the command **grefund**:

```
grefund [[-j] job_id | -n id] [-z amount] [-debug] [-? | -help] [-man] [-quiet] [-v |  
-verbose]
```

Example 10-8. Issuing a job refund

```
$ grefund -j PBS.1234.0  
Successfully refunded job PBS.1234.0 for 19744 credits
```

Chapter 11. Managing Transactions

Gold logs all modifying transactions in a detailed transaction journal (queries are not recorded). Previous transactions can be queried but not modified or deleted.

Querying Transactions

To display transaction information, use the command `glstxn`:

```
glstxn [-o object] [-a action] [-n name_or_id] [-c child] [-U actor] [-u user_name] [-p project_name] [-m machine_name] [-j job_id] [-T type] [-s start_time] [-e end_time] [-r transaction_id] [-R request_id] [-show attribute_name[,attribute_name...]] [-showHidden] [-raw] [-debug] [-? | -help] [-man] [-q | -quiet]
```

Example 11-1. List all deposits made in 2004

```
$ glstxn -a Deposit -s 2004-01-01 -e 2005-01-01
```

Example 11-2. List everything done by amy since the beginning of 2004

```
$ glstxn -U amy -s 2004-01-01
```

Example 11-3. List all transactions affecting Job Id PBS.1234.0

```
$ glstxn -j PBS.1234.0
```

Example 11-4. List all transactions affecting charge rates

```
$ glstxn -o ChargeRate
```


Chapter 12. Integration with the Resource Management System

Dynamic versus Delayed Accounting

Delayed Accounting

In the absence of a dynamic system, some sites enforce allocations by periodically (weekly or nightly) parsing resource manager job logs and then applying debits against the appropriate user accounts. Although Gold can easily support this type of system by the use of the `qcharge` command in post-processing scripts, this approach will allow a user or project to use resources significantly beyond their designated allocation and generally suffers from stale accounting information.

Dynamic Accounting

Gold's design allows it to interact dynamically with your resource management system. Charges for resource utilization can be made immediately when the job finishes (or even incrementally throughout the job). Additionally, reservations can be issued at the start of a job to place a hold against the user's account, thereby ensuring that a job will only start if it has sufficient reserves to complete. The remainder of this document will describe the interactions for dynamic accounting.

Interaction Points

Job Quotation @ Job Submission Time [Optional — Recommended]

When a job is submitted to a grid scheduler or resource broker, it may be useful to determine how much it will cost to run on a particular resource by requesting a job quote. If the quote succeeds, it will return a quote id along with the quoted amount for the job. This quote id may be used later to guarantee that the same charge rates used to form the quote will also be used in the final job charge calculation.

Even when a job is exclusively scheduled locally, it is useful to obtain a quote at the time of submission to the local resource manager to ensure the user has sufficient funds to run the job and that it meets the allocation policies necessary for the charge to succeed. A warning can be issued if funds are low or the job might be rejected with an informative message in the case of insufficient funds or any other problems with the account. Without this interaction, the job might wait in the queue for days only to fail when it tries to start.

To make a job quotation with Gold at this phase requires that:

- the grid scheduler has built-in Gold allocation support {Silver}, or
- the resource manager supports a submit filter {LoadLeveler(SUBMIT_FILTER), LSF(esub)}, or
- a wrapper could be created for the submit command {PBS(qsub)}.

Job Reservation @ Job Start Time [Optional — Highly Recommended]

Just before a job starts, a hold (reservation) is made against the appropriate allocation(s), temporarily reducing the user's available balance by an amount based on the resources requested and the estimated wallclock limit. If this step is omitted, it would be possible for users to start more jobs than they have funds to support.

If the reservation succeeds, it will return a message indicating the amount reserved for the job. In the case where there are insufficient resources to run the job or some other problem with the reservation, the command will fail with an informative message. Depending on site policy, this may or may not prevent the job from starting.

To make a job reservation with Gold at this phase requires that:

- the scheduler or resource manager has built-in Gold allocation support {Maui(AMCFG)}, or
- the resource manager is able to run a script at job start time {LoadLeveler(prolog), PBS(prologue), LSF(pre_exec)}.

Job Charge @ Job End Time [Required]

When a job ends, a charge is made to the user's allocation(s). Any associated reservations are automatically removed as a side-effect. Depending on site policy, a charge can be elicited only in the case of a successful completion, or for all or specific failure cases as well. Ideally, this step will occur immediately after the job completes (dynamic accounting). This has the added benefit that job run times can often be reconstructed from Gold job reservation and charge timestamps in case the resource management job accounting data becomes corrupt.

If the charge succeeds, it will return a message indicating the amount charged for the job.

To make a job charge with Gold at this phase requires that:

- the scheduler or resource manager has built-in Gold allocation support {Maui(AMCFG)}, or
- the resource manager is able to run a script at job start time {LoadLeveler(epilog), PBS(epilogue), LSF(post_exec)}, or
- the resource management system supports some kind of feedback or notification mechanism occurring at the end of a job (an email can be parsed by a mail filter).

Methods of interacting with Gold

There are essentially six ways of programmatically interacting with Gold. Let's consider a simple job charge in each of the different ways.

Configuring an application that already has hooks for Gold

The easiest way to use Gold is to use a resource management system with built-in support for Gold. For example, the Maui Scheduler and Silver Grid Scheduler can

be configured to directly interact with Gold to perform the quotes, reservations and charges by setting the appropriate parameters in the config file.

Example 12-1. Configuring maui.cfg to use Gold

```
AMCFG[bank] TYPE=GOLD HOST=control_node1 PORT=7112 SOCKETPROTOCOL=HTTP WIRE-  
PROTOCOL=XML CHARGEPOLICY=DEBITALLWC JOBFAILUREACTION=NONE TIMEOUT=15
```

Using the appropriate command-line client

From inside a script, or by invoking a system command, you can use a command line client (one of the "g" commands in gold's bin directory).

Example 12-2. To issue a charge at the completion of a job, you would use gcharge:

```
gcharge -j PBS.1234.0 -p chem -u jsmith -m cluster -P 2 -t 1234
```

Using the Gold control program

The Gold control program, gold, will issue a charge for a job expressed in xml (SSS Job Object).

Example 12-3. To issue a charge you must invoke the Charge action on the Job object:

```
gold Data:="<Job><StepId>PBS.1234.0</StepId><ProjectId>chem</ProjectId>  
<UserId>jsmith</UserId><MachineName>cluster</MachineName>  
<Processors>2</Processors><WallDuration>1234</WallDuration>"
```

Use the Perl API

If your resource management system is written in Perl or if it can invoke a Perl script, you can access the full Gold functionality via the Perl API.

Example 12-4. To make a charge via this interface you might do something like:

```
use Gold;  
  
my $request = new Gold::Request(object => "Job", action => "Charge");  
my $job = new Gold::Datum("Job");  
$job->setValue("StepId", "PBS.1234.0");  
$job->setValue("ProjectId", "chem");  
$job->setValue("UserId", "jsmith");  
$job->setValue("MachineName", "cluster");  
$job->setValue("Processors", "2");  
$job->setValue("WallDuration", "1234");  
$request->setDatum($job);  
my $response = $request->getResponse();
```

```
print $response->getStatus(), ": ", $response->getMessage(), "\n";
```

Use the Java API

If your resource management system is written in Java or if it can invoke a Java executable, you can access the full Gold functionality via the Java API.

Example 12-5. To make a charge via this interface you might do something like:

```
import java.util.*;
import gold.*;

public class Test
{
    public static void main(String [] args) throws Exception
    {
        Gold.initialize();
        Request request = new Request("Job", "Charge");
        Datum job = new Datum("Job");
        job.setValue("StepId", "PBS.1234.0");
        job.setValue("ProjectId", "chem");
        job.setValue("UserId", "jsmith");
        job.setValue("MachineName", "cluster");
        job.setValue("Processors", "2");
        job.setValue("WallDuration", "1234");
        request.setDatum(job);
        Response response = request.getResponse();
        System.out.println(response.getStatus() + ": " + response.getMessage() + "\n");
    }
}
```

Communicating via the SSSRMAP Protocol

Finally, it is possible to interact with Gold by directly using the SSSRMAP Wire Protocol and Message Format over the network (see *SSS Resource Management and Accounting Documentation*¹). This will entail building the request body in XML, appending an XML digital signature, combining these in an XML envelope framed in an HTTP POST, sending it to the server, and parsing the similarly formed response. The Maui Scheduler communicates with Gold via this method.

Example 12-6. The message might look something like:

```
POST /SSSRMAP HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Transfer-Encoding: chunked

190
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body actor="scottmo" chunking="True">
    <Request action="Charge" object="Job">
      <Data>
        <Job>
```

```
<StepId>PBS.1234.0</StepId>
<ProjectId>chem</ProjectId>
<UserId>jsmith</UserId>
<MachineName>cluster</MachineName>
<Processors>2</Processors>
<WallDuration>1234</WallDuration>
</Job>
</Data>
</Request>
</Body>
<Signature>
  <DigestValue>azu4obZswzBt89OgATukBeLyt6Y=</DigestValue>
  <SignatureValue>YXE/C08XX3RX4PMU1bWju+5/E5M=</SignatureValue>
  <SecurityToken type="Symmetric" name="scottmo"></SecurityToken>
</Signature>
</Envelope>
0
```

Notes

1. <http://sss.scl.ameslab.gov/docs.shtml>

