Gold User's Guide

Scott Jackson
Pacific Northwest National Laboratory

Gold User's Guide
by Scott Jackson

Table of Contents

Notice **9**

1. Overview **11**

- Background 11
- Features 11
- Interfaces 12
  - Command Line Clients 13
  - Interactive Control Program 13
  - Web-based Graphical User Interface 13
  - Perl API 14
  - Java API 14
  - SSSRMAP Wire Protocol 14

2. Getting Started **17**

- Define Users 17
- Define Machines 17
- Define Projects 18
- Add Users and Machines to the Projects 18
- Create Accounts 19
- Define Time Periods 19
- Make Deposits 20
- Check The Balance 20
- Define Charge Rates 21
- Integrate Gold with your Resource Management System 21
- Obtain A Job Quote 21
- Make A Job Reservation 22
- Charge for a Job 23
- Refund a Job 23
- List Transactions 24
- List Jobs 25
- List Usage 25
- Examine Account Statement 25

3. Managing Users **27**

- Creating Users 27
- Querying Users 27
- Modifying Users 28
- Deleting Users 28

4. Managing Machines **29**

- Creating Machines 29
- Querying Machines 29
- Modifying Machines 29
- Deleting Machines 30

5. Managing Projects **31**

- Creating Projects 31
- Querying Projects 31
- Modifying Projects 32
- Deleting Projects 32

6. Managing Accounts **33**

- Creating Accounts 33
- Querying Accounts 34
- Modifying Accounts 35
- Making Deposits 35
- Querying The Balance 35
- Making Withdrawals 36

Making Transfers....36
Obtaining an Account Statement....37
Deleting Accounts....38
7. Managing Jobs....39
Creating Jobs....39
Querying Jobs....39
Modifying Jobs....39
Deleting Jobs....40
Obtaining Job Quotes....40
Making Job Reservations....41
Charging Jobs....41
Issuing Job Refunds....42
8. Managing Reservations....43
Creating Reservations....43
Querying Reservations....43
Modifying Reservations....44
Deleting Reservations....44
9. Managing Quotations....47
Creating Quotations....47
Querying Quotations....47
Modifying Quotations....47
Deleting Quotations....47
10. Managing Charge Rates....49
Creating ChargeRates....49
Querying ChargeRates....49
Modifying Charge Rates....50
Deleting Charge Rates....50
11. Managing Time Periods....53
Creating Time Periods....53
Querying Time Periods....53
Modifying Time Periods....53
Deleting Time Periods....54
12. Managing Usage Records....55
Querying Usage Records....55
13. Managing Transactions....57
Querying Transactions....57
14. Integration with the Resource Management System....59
Dynamic versus Delayed Accounting....59
Delayed Accounting....59
Dynamic Accounting....59
Interaction Points....59
Job Quotation @ Job Submission Time [Optional — Recommended]....59
Job Reservation @ Job Start Time [Optional — Highly Recommended]....59
Job Charge @ Job End Time [Required]....60
Methods of interacting with Gold....60
Configuring an application that already has hooks for Gold....60
Using the appropriate command-line client....61
Using the Gold control program....61
Use the Perl API....61
Use the Java API....62
Communicating via the SSSRMAP Protocol....62
15. Configuration Files....65

Server Configuration 65
Client Configuration 67

Notice

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

Chapter 1. Overview

Gold is an open source accounting system that tracks resource usage on High Performance Computers. It acts much like a bank in which resource credits are deposited into accounts with access controls designating which users, projects and machines may access the account. As jobs complete or as resources are utilized, accounts are charged and resource usage recorded. Gold supports familiar operations such as deposits, withdrawals, transfers and refunds. It provides balance and usage feedback to users, managers, and system administrators.

Since accounting needs vary widely from organization to organization, Gold has been designed to be extremely flexible, featuring customizable accounting and supporting a variety of accounting models. Attention has been given to scalability, security, and fault tolerance. Gold facilitates the sharing of resources between organizations or within a Grid by providing distributed accounting while preserving local site autonomy.

Background

Gold is being developed at Pacific Northwest National Laboratory (PNNL) as open source software under the Scalable Systems Software (SSS) SciDAC project. Gold is currently in alpha release and is beginning alpha testing at a number of DOE and university sites.

Gold was designed to meet the accounting needs of computing centers that share resources in multi-project environments. In order for an organization to use its high performance computers most effectively, it must be able to allocate resources to the users and projects that need them 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.

Gold was also designed to facilitate the sharing of resources between organizations or within a Grid to take advantage of the tremendous utilization gains afforded by meta-scheduling.

Features

- *Dynamic Charging* — Rather than post-processing resource usage records on a periodic basis to rectify project balances, acounts 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 charge at the moment the job completes, thereby preventing projects from using more resources than were allocated to them.
- *Flexible Accounts* — A uniquely flexible account design allows resource credits to be allocated to specific projects, users and machines.
- *Expiring Allocations* — Resource credits may be restricted for use within a designated time period allowing sites to implement a use-it-or-lose-it policy to prevent year-end resource exhaustion and establishing a project cycle.

- *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 Accounts* — Accounts feature an optional credit limit allowing support for both debit and credit models. This feature can also be used to enable overdraft protection for specific accounts.
- *Nested Projects* — A hierarchical relationship may be created between accounts. This allows for the delegation of management responsibilities, the establishment of automatic rules for the distribution of downstream resource credits, and the option of making higher level credits available to lower level accounts.
- *Powerful Querying* — Gold supports a powerful querying and update mechanism that facilitates flexible reporting and streamlines administrative tasks.
- *Transparency* — Gold allows the establishment of default projects, machines and users. Additionally Gold can allow user, machines and projects to be automatically created the first time they are seen by the resource management system. These features allow 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 all 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 allows 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 dynamic 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 2. Getting Started

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

It is assumed that you have already installed and bootstrapped Gold as an allocation and accounting manager and started the gold server before performing the steps suggested in this section.

> **Important:** You will need to be a Gold System Adminstrator 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 dave.

```
$  gmkuser -n "Wilkes, Amy" -E "amy@western.edu" amy
Successfully created 1 User

$  gmkuser -n "Smith, Robert F." -E "bob@western.edu" bob
Successfully created 1 User

$  gmkuser -n "Miller, David" -E "dave@western.edu" dave
Successfully created 1 User

$  glsuser
Name Active CommonName       PhoneNumber EmailAddress     Organization De-
faultProject Description
---- ------ ---------------- ----------- ---------------- ------------
-------------- -----------
gold True                                                                G
amy  True   Wilkes, Amy                  amy@western.edu
bob  True   Smith, Robert F.             bob@western.edu
dave True   Miller, David                dave@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 Architecture OperatingSystem Organization Description
------ ------ ------------ --------------- ------------ -------------

colony True                                             Linux Cluster
blue   True                                             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 "Biology Department" biology
Successfully created 1 Project

$  gmkproject -d "Chemistry Department" chemistry
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 --addUsers amy,bob --addMachine colony biology
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser
Successfully created 1 ProjectMachine

$  gchproject --addUsers amy,bob,dave chemistry
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser

$  glsproject
Name      Active Users        Machines Organization Description
```

```
--------- ------ ----------- -------- ------------ -----------------
--
biology   True   amy,bob      colony                Biology Department
chemistry True   amy,dave,bob                      Chemistry Department
```

Create Accounts

Next, you can create your accounts (see Creating Accounts).

Example 2-5. We will create accounts for use by the biology and chemistry departments.

```
$  gmkaccount -p biology -u MEMBER -m blue -n Biology
Successfully created 1 Account
Successfully created 1 AccountProject
Successfully created 1 AccountUser
Successfully created 1 AccountMachine

$  gmkaccount -p chemistry -u MEMBER -m ANY -n Chemistry
Successfully created 1 Account
Successfully created 1 AccountProject
Successfully created 1 AccountUser
Successfully created 1 AccountMachine

$  glsaccount
Id Name      Allocations CreditLimit Projects  Users  Machines Descrip-
tion
-- --------- ----------- ----------- --------- ------ -------- ------
-----
1  Biology              0           biology   MEMBER blue
2  Chemistry            0           chemistry MEMBER ANY
```

Define Time Periods

It will be useful to define a new time period (see Creating Time Periods).

Example 2-6. Let's create a time period for this fiscal year.

```
$  gold TimePeriod Create Name=FY2004 StartTime="2003-10-01" EndTime="2004-
10-01" Description="Fiscal Year 2004"
Successfully created 1 TimePeriod

$  gold TimePeriod Query
Name     StartTime              EndTime                Active Descrip-
tion
-------- ---------------------- ---------------------- ------ --------
--------
Eternity -infinity              infinity               True   Always Active
FY2004   2003-10-01 00:00:00-07 2004-10-01 00:00:00-07 True   Fiscal Year 2004
```

Make Deposits

Now you can make some deposits (see Making Deposits).

Example 2-7. Let's add 360000000 credits to each account. We will cause both to expire at the end of the year.

```
$  gdeposit -t FY2004 -z 360000000 -a 1
Successfully deposited 3600000 credits into account 1

$  gdeposit -t FY2004 -z 360000000 -a 2
Successfully deposited 3600000 credits into account 2

$  glsaccount
Id Name      Allocations        CreditLimit Projects  Users  Machines De-
scription
-- --------- ------------------ ----------- --------- ------ --------
-----------
1  Biology   360000000 [FY2004] 0           biology   MEMBER blue
2  Chemistry 360000000 [FY2004] 0           chemistry MEMBER ANY
```

Check The Balance

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

Example 2-8. Let's look at amy's balance

```
$  gbalance -u amy
Balance
---------
720000000
The account balance is 720000000 credits
```

Example 2-9. It is often useful to get more detail on the balance composition

```
$  gbalance -u amy --detail
Id Name      Amount    Reservations CreditLimit Projects  Users  Machines
-- --------- --------- ------------ ----------- --------- ------ ----
----
1  Biology   360000000              0           biology   MEMBER blue
2  Chemistry 360000000              0           chemistry MEMBER ANY
```

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

Example 2-10. We may want to get more specific

```
$  gbalance -u amy -p chemistry -m colony --quiet
360000000
```

Define Charge Rates

Finally, you should define how much you will charge for your resources (see Creating Charge Rates).

Example 2-11. 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 Description
-------- ---------- ---- -----------
Resource Processors 1
```

Note: Not defining any charge rates will result in zero-credit charges for all jobs.

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 invoked automatically by your resource management system, it is useful to understand their effects by invoking them manually.

Let's simulate the lifecycle of a job.

Example 2-12. We'll assume our job has the following characteristics:

```
Job Id:               PBS.1234.0
Job Name:             heavywater
User Name:            amy
Project Name:         chemistry
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 2-13. Let's see how much it will cost to run our job.

```
$  gquote -p chemistry -u amy -m colony -P 16 -t 3600
Successfully quoted 57600 credits with quote id 1

$  glsquote
Id Project   User Machine Amount ExpirationTime         WallDuration Used Charg-
eRates           Description
-- --------- ---- ------- ------ -------------------- ------------ --
-- -------------------- -----------
1  chemistry amy  colony  57600  2004-08-10 15:27:07-07 3600         0    Resource:Proc
```

Make A Job Reservation

When a job starts, the resource management system 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 2-14. Make a reservation for our job.

```
$  greserve -J PBS.1234.0 -p chemistry -u amy -m colony -P 16 -
t 3600
Successfully reserved 57600 credits for job PBS.1234.0

$  glsres
Id Account Amount Name       User Project   Machine ExpirationTime         De-
scription
-- ------- ------ ---------- ---- --------- ------- ------------------
---- -----------
1  2       57600  PBS.1234.0 amy  chemistry colony  2004-08-03 15:29:30-
07
```

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

```
$  gbalance -p chemistry --quiet
359942400
```

As illustrated by the detailed balance listing:

```
$  gbalance -p chemistry --detail
Id Name      Amount    Reservations CreditLimit Projects  Users  Machines
-- --------- --------- ------------ ----------- --------- ------ ----
----
```

```
2  Chemistry 360000000 -57600       0           chemistry MEMBER ANY
```

Although our allocation has not changed.

```
$  glsaccount -p chemistry
Id Name      Allocations        CreditLimit Projects  Users  Machines De-
scription
-- --------- ------------------ ----------- --------- ------ --------
-----------
2  Chemistry 360000000 [FY2004] 0           chemistry MEMBER ANY
```

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 2-15. Issue the charge for our job.

```
$  gcharge -J PBS.1234.0 -u amy -p chemistry -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.

```
$  glsaccount -p chemistry
Id Name      Allocations        CreditLimit Projects  Users  Machines De-
scription
-- --------- ------------------ ----------- --------- ------ --------
-----------
2  Chemistry 359980256 [FY2004] 0           chemistry MEMBER ANY
```

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

```
$  gbalance -p chemistry
Balance
---------
359980256
The account balance is 359980256 credits
```

Refund a Job

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

Example 2-16. Let's isse a refund for our job.

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

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

```
$  gbalance -p chemistry
Balance
---------
360000000
The account balance is 360000000 credits
```

The allocation, of course, is likewise restored.

```
$  glsaccount -p chemistry
Id Name      Allocations        CreditLimit Projects  Users  Machines De-
scription
-- --------- ------------------ ----------- --------- ------ --------
-----------
2  Chemistry 360000000 [FY2004] 0           chemistry MEMBER ANY
```

List Transactions

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

Example 2-17. Let's list all the job transactions

```
$  glstxn -O Job --show="RequestId,TransactionId,Object,Action,JobId,Project,U

RequestId TransactionId Object Action  JobId                Project   User Ma-
chine Amount
--------- ------------- ------ ------- -------------------- --------
- ---- ------- ------
634       456           Job    Quote                        chemistry amy  colony  576
637       459           Job    Reserve PBS.1234.0           chemistry amy  colony  576
655       463           Job    Create
655       465           Job    Charge  PBS.1234.0           chemistry amy  colony  197
655       467           Job    Modify
662       469           Job    Refund  PBS.1234.0,PBS.1234.0
662       470           Job    Modify
```

Example 2-18. It may also be illustrative to examine what transactions actually composed our charge request...

```
$  glstxn -R 655 --show="Id,Object,Action,Name,JobId,Amount,Account,Delta"

Id  Object            Action Name       JobId      Amount Account Delta
--- ----------------- ------ ---------- ---------- ------ ------- ----
--
462 Usage             Create
463 Job               Create
464 AccountTimePeriod Modify 1
```

```
465 Job              Charge 1          PBS.1234.0 19744  1       -19744
466 Reservation      Delete PBS.1234.0
467 Job              Modify 1
```

List Jobs

A job record was created for the job as a side-effect of the charge (see Querying Jobs).

Example 2-19. We'll list all the jobs

```
$ glsjob
Id JobId      User Project   Machine Charge Class Type QualityOfService Nodes Pro-
cessors Executable Application StartTime EndTime WallDuration QuoteId De-
scription
-- ---------- ---- --------- ------- ------ ----- ---- --------------
-- ----- ---------- ---------- ---------- --------- ------- --------
---- ------- -----------
1  PBS.1234.0 amy  chemistry colony  0                                    16
```

Notice that the charge is zero because the job has been fully refunded.

List Usage

Additionally a usage record was created for each resource used by the job (see Querying Usage).

Example 2-20. Let's look at the usage

```
$ glsusage
Id JobId      Resource   Amount Machine WallDuration ConsumptionRate Charg-
eRate Multiplier Charge Description
-- ---------- ---------- ------ ------- ------------ --------------- --
-------- ---------- ------ -----------
1  PBS.1234.0 Processors 16     colony  1234                         1         1
```

Examine Account Statement

Finally, you can examine the account statement for our activities (see Obtaining an Account Statement).

Example 2-21. We can request a detailed account statement over all time for the Chemistry account (account 2)

```
$ gstatement 2 --detail
################################################################################
#
# Statement for account 2 generated on Tue Aug  3 16:06:15 2004.
#
# Reporting account activity from -infinity to now.
```

```
#
################################################################################

Beginning Balance:                    0
------------------ --------------------
Total Credits:                360019744
Total Debits:                    -19744
------------------ --------------------
Ending Balance:               360000000

################################# Credit Detail ###################################

Object  Action  Child      Delta     CreationTime           Description
------- ------- ---------- --------- ---------------------- ----------
-
Account Deposit FY2004     360000000 2004-08-03 16:01:15-07
Job     Refund  PBS.1234.0 19744     2004-08-03 16:04:02-07

################################# Debit Detail ####################################

Object Action Child      Delta  CreationTime           Description
------ ------ ---------- ------ ---------------------- -----------
Job    Charge PBS.1234.0 -19744 2004-08-03 16:03:39-07

################################# End of Report ###################################
```

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, organization, 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*] [-o *organization_name*] [-p *default_project*] [-d *description*] [—debug] [-? | —help] [–man] [—quiet] [-v | —verbose] {[-u] *user_name*}

Example 3-1. Creating a user

```
$  gmkuser -n "Smith, Robert F." -E "bob@western.edu" -F "(509)
555-1234" bob
Successfully created 1 User
```

> **Note:** It is possible to have users be created automatically when first encountered in a job function (charge, reserve or quote) by setting the user.autogen configuration parameter to True. It is also possible to establish a system default user to be used in job functions when the user is unspecified (see Server Configuration).

Querying Users

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

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

Example 3-2. Listing all info about active users

```
$  glsuser -A
Name Active CommonName       PhoneNumber    EmailAddress     Organiza-
tion DefaultProject Description
---- ------ ---------------- -------------- ---------------- --------
---- -------------- -----------
amy  True   Wilkes, Amy      (509) 555-8765 amy@western.edu
bob  True   Smith, Robert F. (509) 555-1234 bob@western.edu
```

Example 3-3. Displaying bob's phone number

```
$  glsuser --show PhoneNumber bob --quiet
(509) 555-1234
```

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

```
$  glsuser --show Name --quiet
amy
bob
```

Modifying Users

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

gchuser [-A | -I] [-n *common_name*] [-F *phone_number*] [-E *email_address*] [-o *organization_name*] [-p *default_project*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-u] *user_name*}

Example 3-5. Activating a user

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

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

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

Deleting Users

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

grmuser [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-u] *user_name*}

Example 3-7. Deleting a user

```
$  grmuser bob
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] [—arch *architecture*] [—opsys *operating_system*] [-o *organization_name*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-m] *machine_name*}

Example 4-1. Creating a machine

```
$  gmkmachine -d "Linux Cluster" colony
Successfully created 1 Machine
```

> **Note:** It is possible to have machines be created automatically when first encountered in a job function (charge, reserve or quote) by setting the machine.autogen configuration parameter to True. It is also possible to establish a system default machine to be used in job functions when the machine is unspecified (see Server Configuration).

Querying Machines

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

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

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

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

Modifying Machines

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

gchmachine [-A | -I] [—arch *architecture*] [—opsys *operating_system*] [-o *organization_name*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-m] *machine_name*}

Example 4-3. Deactivating a machine

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

Deleting Machines

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

grmmachine [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-m] *machine_name*}

Example 4-4. Deleting a machine

```
$  grmmachine colony
Successfully deleted 1 Machine
```

Chapter 5. Managing Projects

A project is a research interest or activity requiring the use of computational resources for a common purpose. Users may be designated as members of a project and allowed to share its allocations. Machines may also be designated as members of a project as a default resource pool.

Creating Projects

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

gmkproject [-A | -I] [-u [+ | -]*user_name* [, [+ | -]*user_name*...]] [-m [+ | -]*machine_name* [, [+ | -]*machine_name*...]] [-o *organization_name*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-p] *project_name*}

Example 5-1. Creating a project

```
$  gmkproject -d "Chemistry Department" chemistry
Successfully created 1 Project
```

> **Note:** It is possible to have projects be created automatically when first encountered in a job function (charge, reserve or quote) by setting the project.autogen configuration parameter to True. It is also possible to establish a system default project to be used in job functions when the project is unspecified (see Server Configuration).

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] [—quiet] [[-p] *project_pattern*]

Example 5-2. Listing all info about all projects

```
$  glsproject
Name      Active Users        Machines Organization Description
--------- ------ ------------ -------- ------------ ------------------
--
biology   True   amy,bob      colony                Biology Department
chemistry True   amy,dave,bob                      Chemistry Department
```

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

```
$  glsproject --show Name,Users -l chemistry
Name      Users
--------- -----
```

```
chemistry bob
          dave
          amy
```

Example 5-4. Listing all project names

```
$  glsproject --show Name --quiet
biology
chemistry
```

Modifying Projects

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

gchproject [-A | -I] [-o *organization_name*] [-d *description*] [—addUser(s) [+ | -]*user_name* [, [+ | -]*user_name*...]] [—addMachines(s) [+ | -]*machine_name* [, [+ | -]*machine_name*...]] [—delUser(s) *user_name* [,*user_name*...]] [—delMachines(s) *machine_name* [,*machine_name*...]] [—actUser(s) *user_name* [,*user_name*...]] [—actMachines(s) *machine_name* [,*machine_name*...]] [—deactUser(s) *user_name* [,*user_name*...]] [—deactMachines(s) *machine_name* [,*machine_name*...]] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-p] *project_name*}

Example 5-5. Deactivating a project

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

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

```
$  gchproject --addUsers jsmith,barney chemistry
Successfully created 2 ProjectUsers
```

Deleting Projects

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

grmproject [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-p] *project_name*}

Example 5-7. Deleting a project

```
$  grmproject chemistry
Successfully deleted 1 Project
```

Chapter 6. Managing Accounts

An account is a container for time-bounded resource credits valid toward a specific set of projects, users and machines. Much like with a bank, accounts hold resource credits. Each account has a set of access control lists designating which users, projects, and machines may access the account. An account may restrict the projects that can charge to it. Normally an account will be tied to a single project but it may be tied to an arbitrary set of projects or ANY project. An account 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 an arbitrary set of users or ANY user. An account may restrict the machines that can charge to it. It may be tied to an arbitrary set of machines, just the machine MEMBERs of the associated project(s) or ANY machine.

When resource credits are deposited into an account, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of resource credits associated with an account for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish a project cycle.

Accounts may be nested. Hierarchically nested accounts may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level accounts. Additionally, an optional overflow feature allows charges against lower level accounts to trickle up the hierarchy.

Operations include creating, querying, modifying and deleting accounts as well as making deposits, withdrawals, transfers and balance queries.

Creating Accounts

gmkaccount is used to create a new account. A new id is automatically generated for the account.

gmkaccount [-n *account_name*] [-p [+ | -]*project_name* [, [+ | -]*project_name*...]] [-u [+ | -]*user_name* [, [+ | -]*user_name*...]] [-m [+ | -]*machine_name* [, [+ | -]*machine_name*...]] [-L *credit_limit*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

> **Important:** When creating an account, it is important to specify at least one user, machine and project designation. If omitted, it will default to ANY.

Example 6-1. Creating an account

```
$  gmkaccount -p chemistry -u MEMBER -m ANY -n "Chemistry"
Successfully created 1 Account
Successfully created 1 AccountProject
Successfully created 1 AccountUser
Successfully created 1 AccountMachine
```

Example 6-2. Creating a wide-open credit account

```
$  gmkaccount -p ANY -u ANY -m ANY -L 1000000000000 -n "Cornucopia"

Successfully created 1 Account
Successfully created 1 AccountProject
Successfully created 1 AccountUser
Successfully created 1 AccountMachine
```

Example 6-3. Creating an account valid toward all biology project members except for dave and all machines except for blue

```
$  gmkaccount -p biology -u MEMBER,-dave -m ANY,-blue -n "Not Dave"

Successfully created 1 Account
Successfully created 1 AccountProject
Successfully created 1 AccountUser
Successfully created 1 AccountUser
Successfully created 1 AccountMachine
Successfully created 1 AccountMachine
```

Querying Accounts

To display account information, use the command **glsaccount**:

glsaccount [-A | -I] [-n *account_name*] [-p *project_name*] [-u *user_name*] [-m *machine_name*] [-t *time_period_name*] [—show *attribute_name* [,*attribute_name*...]...] [—showl den] [-l | —long] [-w | —wide] [—raw] [—debug] [-? | —help] [—man] [—quiet] [[-a] *account_id*]

Example 6-4. Listing all info about all accounts with multi-valued fields displayed in a multi-line format

```
$  glsaccount -long
Id Name       Allocations        CreditLimit   Projects  Users  Machines De-
scription
-- ---------- ------------------ ------------- --------- ------ ------
-- -----------
1  Chemistry  360000000 [FY2005] 0             chemistry MEMBER ANY
              360000000 [FY2004]
2  Cornucopia 0 [Eternity]       1000000000000 ANY       ANY    ANY
3  Not Dave   250000 [4Q04]      0             biology   -dave  -blue
              250000 [3Q04]                               MEMBER ANY
              250000 [2Q04]
              250000 [1Q04]
```

Example 6-5. Listing all info about all accounts useable by dave

```
$  glsaccount -u dave -long
Id Name       Allocations        CreditLimit   Projects  Users  Machines De-
scription
-- ---------- ------------------ ------------ --------- ------ ------
-- -----------
1  Chemistry  360000000 [FY2005] 0            chemistry MEMBER ANY
              360000000 [FY2004]
              1000 [Eternity]
2  Cornucopia -1000 [Eternity]   1000000000000 ANY       ANY    ANY
```

Modifying Accounts

To modify an account, use the command **gchaccount**:

gchaccount [-n *account_name*] [-L *credit_limit*] [-d *description*] [—addProject(s) [+ | -]*project_name* [, [+ | -]*project_name*...]] [—addUser(s) [+ | -]*user_name* [, [+ | -]*user_name*...]] [—addMachine(s) [+ | -]*machine_name* [, [+ | -]*machine_name*...]] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-a] *account_id*}

Example 6-6. Changing the credit limit for an account

```
$  gchaccount -L 500000000000 -a 2
Successfully modified 1 Account
```

Example 6-7. Adding a user to the list of users that share the account

```
$  gchaccount --addUser dave 1
Successfully created 1 AccountUser
```

Making Deposits

gdeposit is used to deposit time-bounded resource credits into accounts. (See Time Periods for managing time periods). The time period will default to Eternity (always valid) if not specified. Accounts must first be created using **gmkaccount**.

gdeposit [-t *time_period_name*] [-d *description*] {-z *amount*} [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-a] *account_id*}

Example 6-8. Making a deposit

```
$  gdeposit -t FY2004 -z 360000000 -a 1
Successfully deposited 360000000 credits into account 1
```

Querying The Balance

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

gbalance [-p *project_name*] [-u *user_name*] [-m *machine_name*] [—available] [—detail] [-l | —long] [-w | —wide] [—raw] [—debug] [-? | —help] [—man] [—quiet]

Example 6-9. Querying the balance for a particular user in a particular project on a particular machine

```
$  gbalance -u bob -m colony -p chemistry
Balance
---------
360000000
The account balance is 360000000 credits
```

Example 6-10. Querying the simple amount available for charging including available credit for a particular user in a particular project on a particular machine

```
$  gbalance -u bob -m colony -p chemistry --available --quiet
1000360000000
```

Example 6-11. Querying the project balance detail broken down by account

```
$  gbalance -p chemistry --detail
Id Name       Amount    Reservations CreditLimit   Projects  Users  Ma-
chines
-- ---------- --------- ------------ ------------- --------- ------ --
------
1  Chemistry  360000000             0             chemistry MEMBER ANY
2  Cornucopia 0                     1000000000000 ANY       ANY    ANY
```

Making Withdrawals

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

gwithdraw {[-z] *amount*} [-t *time_period_name*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-a] *account_id*}

Example 6-12. Making a withdrawal

```
$  gwithdraw -z 12800 -a 1 -d "Grid Tax"
Successfully withdrew 12800 credits from account 1
```

Making Transfers

To issue a transfer between accounts, use the command **gtransfer**. If the time period is specified, then only credits associated with the specified time period will be transferred, otherwise, only active credits will be transferred. Account transfers preserve the time periods associated with the resource credits from the source to the destination accounts.

gtransfer {–fromId *source_account_id*} {–toId *destination_account_id*} [-t *time_period_name*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-z] *amount*}

Example 6-13. Transferring credits between two accounts

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

Obtaining an Account Statement

To generate an account statement, use the command **gstatement**. For a specified time frame it displays the beginning and ending balances as well as the total credits and debits to the account over that period. A detailed report of the debits and credits may be obtained by using the –detail option.

gstatement [-s *start_time*] [-e *end_time*] [—detail] [—debug] [-? | —help] [—man] {[-a] *account_id*}

Example 6-14. Generating an account statement

```
$  gstatement -detail -a 2
################################################################################
#
# Statement for account 2 generated on Tue Aug  3 16:06:15 2004.
#
# Reporting account activity from -infinity to now.
#
################################################################################

Beginning Balance:                    0
------------------ --------------------
Total Credits:                360019744
Total Debits:                    -19744
------------------ --------------------
Ending Balance:               360000000

################################# Credit Detail ##################################

Object  Action  Child      Delta     CreationTime          Description
------- ------- ---------- --------- --------------------- ----------
-
Account Deposit FY2004     360000000 2004-08-03 16:01:15-07
Job     Refund  PBS.1234.0 19744     2004-08-03 16:04:02-07

################################## Debit Detail ##################################
```

```
Object Action Child      Delta  CreationTime           Description
------ ------ ---------- ------ -------------------- -----------
Job    Charge PBS.1234.0 -19744 2004-08-03 16:03:39-07

################################ End of Report ###################################
```

Deleting Accounts

To delete an account, use the command **grmaccount**:

grmaccount [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-a] *account_id*}

Example 6-15. Deleting an account

```
$  grmaccount 2
Successfully deleted 1 Account
```

Chapter 7. 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

In most cases, jobs will be created by the resource management system with the gcharge command (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=<*Qual*
Of Service>] [Nodes=<*Number Of Nodes*>] [Processors=<*Number Of Processors*>] [State=<*Jo*
State>] [Executable=<*Executable*>] [Application=<*Application*>] [StartTime=<*Start Time*>] [EndTime=<*End Time*>] [WallDuration=<*Wallclock Time in seconds*>] [QuoteId=<*Quo*
Id>] [Description=<*Description*>] [ShowUsage:=true]

Example 7-1. 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*] [-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] [—quiet] [[-j] *gold_job_id*]

Example 7-2. Show specific info about jobs run by amy

```
$  glsjob --show=JobId,Project,Machine,Charge -u amy
JobId      Project   Machine Charge
---------- --------- ------- ------
PBS.1234.0 chemistry colony  0
```

Modifying Jobs

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

gold Job Modify [JobId==<*Job Id*> | Id==<*Gold Job Id*>] [User=<*User Name*>] [Project=<*Proje*
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*>] [Wal
Duration=<*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 inadvertant modification of all jobs.

Example 7-3. 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 inadvertant deletion of all jobs.

Example 7-4. 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*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 7-5. Requesting a quotation

```
$  gquote -p chemistry -u amy -m colony -P 2 -t 3600
Successfully quoted 7200 credits for quote 2
```

Note: It is possible to establish a system default machine, project or user to be used in job functions (charge, reserve or quote) when left unspecified (see Server Configuration).

Making Job Reservations

A job reservation can be used to place a hold on the user's account 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*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-J] *job_id*}

Example 7-6. Creating a reservation

```
$  greserve -J PBS.1234.0 -p chemistry -u amy -m colony -P 2 -t
3600
Successfully reserved 7200 credits for job PBS.1234.0
```

Note: It is possible to establish a system default machine, project or user to be used in job functions (charge, reserve or quote) when left unspecified (see Server Configuration).

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 7-7. Issuing a job charge

```
$  gcharge -J PBS.1234.0 -p chemistry -u amy -m colony -P 2 -t 1234

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

Note: It is possible to establish a system default machine, project or user to be used in job functions (charge, reserve or quote) when left unspecified (see Server Configuration).

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*] [[-j] *gold_job_id*] [-z *amount*] [-a *account_id*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 7-8. Issuing a job refund

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

Chapter 8. Managing Reservations

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

Associated with a reservation is the name of the reservation (often the job id requiring the reservation), the user, project, and machine as applicable, an expiration time, and an amount. Operations include creating, querying, modifying and deleting reservations.

Creating Reservations

Most reservations are normally created by the resource management system with the greserve command (See Making Job Reservations).

However, reservations can also be manually created using the command **gmkres**:

gmkres {-a *account_id*} {-z *amount*} [-n *reservation_name*] [-u *user_name*] [-p *project_name*] [-m *machine_name*] [-e *expiration_time*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 8-1. Placing a hold against an account

```
$  gmkres -a 1 -z 3600 -n "Interactive.789654" -u bob -p chemistry
-m blue -e "2004-08-07"
Successfully created 1 Reservation
```

Querying Reservations

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

glsres [-A | -I] [-n *reservation_name | 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] [—quiet] [[-r] *reservation_id*]

Example 8-2. Listing all info about all reservations for bob

```
$  glsres -u bob
Id Account Amount Name               User Project   Machine ExpirationTime         De-
scription
-- ------- ------ ------------------ ---- --------- ------- ----------
------------ -----------
1  1       3600   Interactive.789654 bob  chemistry blue    2004-08-07 00:00:00-
07
```

Example 8-3. Listing all info about all reservations that impinge against amy's balance

```
$  glsres -u amy --option name=UseRules value=True
Id Account Amount Name              User Project   Machine ExpirationTime        De-
scription
-- ------- ------ ----------------- ---- --------- ------- ----------
------------ -----------
1  1       3600   Interactive.789654 bob  chemistry blue    2004-08-07 00:00:00-
07
2  1       7200   PBS.1234.0         amy  chemistry colony  2004-08-02 17:59:09-
07
```

Modifying Reservations

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

gchres [-e *expiration_time*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-r] *reservation_id*}

Example 8-4. Changing the expiration time of a reservation

```
$  gchres -e "2004-08-07 14:43:02" 1
Successfully modified 1 Reservation
```

Deleting Reservations

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

grmres [—debug] [-? | —help] [—man] [-q | —quiet] [-v | —verbose] {-I | -n *reservation_name | job_id* | [-r] *reservation_id*}

Example 8-5. Deleting a reservation by name (JobId)

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

Example 8-6. Deleting a reservation by ReservationId

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

Example 8-7. Purging stale reservations

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

Chapter 9. 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 chargerates 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

Quotations are normally created by the resource management system with the gquote command (See Making Job Quotations).

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] [—quiet] [[-q] *quote_id*]

Example 9-1. Listing all info about all quotes for user amy on machine colony

```
$  glsquote -u amy -m colony
Id Project   User Machine Amount ExpirationTime         WallDuration Used Charg-
eRates           Description
-- --------- ---- ------- ------ -------------------- ------------ --
-- -------------------- -----------
1  chemistry amy  colony  57600  2004-08-10 17:02:44-07 3600         0    Resource:Proc
```

Modifying Quotations

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

gchquote [-e *expiration_time*] [-d *description*] [—debug] [-? | —help] [–man] [—quiet] [-v | —verbose] {[-q] *quote_id*}

Example 9-2. 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] [—quiet] [-v | —verbose] {-I | [-q] *quote_id*}

Example 9-3. Deleting a quotation

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

Example 9-4. Purging stale quotations

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

Chapter 10. 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 10-1. Creating a resource charge rate

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

Example 10-2. Creating another resource charge rate

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

Example 10-3. Creating a quality-based charge rate

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

Example 10-4. Creating another quality-based charge rate

```
$  gold ChargeRate Create Type=QualityOfService Name=Premium 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 10-5. Listing all charge rates

```
$  gold ChargeRate Query
Type             Name         Rate  Description
---------------- ------------ ----- -----------
Resource         Processors   1
QualityOfService BottomFeeder 0.5
QualityOfService Normal       1
QualityOfService Premium      2
Resource         Memory       0.001
```

Modifying Charge Rates

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

gold ChargeRate Modify [Rate=<*Floating Point Multiplier*>] [Description=<*Description*>] [T
Rate Type>] [Name==<*Charge Rate Name*>] [Rate==<*Floating Point Multiplier*>] [ShowU

> **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 modification of all charge rates.

Example 10-6. Changing a charge rate

```
$  gold ChargeRate Modify Type==Resource Name==Memory Rate=0.05

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 10-7. Deleting a charge rate

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

Chapter 11. Managing Time Periods

A Time Period represents a named time frame with a specific start and end time. An active flag is maintained that indicates whether the current time is within the time period. Time Periods are associated with resource credits (as allocations) to define the period during which the charges may be made against the credits.

Creating Time Periods

To create a new time period, use the command **gold TimePeriod Create**:

gold TimePeriod Create Name=<*Time Period Name*> [StartTime=*YYYY-MM-DD [hh:mm:ss]|-infinity|infinity (-infinity)*] [EndTime=*YYYY-MM-DD [hh:mm:ss]|-infinity|infinity (-infinity)*] [Description=<*Description*>] [ShowUsage:=True]

Example 11-1. Creating a time period

```
$  TimePeriod Create Name=FY2005 StartTime="2004-10-01" EndTime="2005-
10-01" Description="Fiscal Year 2005"
Successfully created 1 TimePeriod
```

Querying Time Periods

To display time period information, use the command **gold TimePeriod Query**:

gold TimePeriod Query [show:=<*"Field1,Field2,..."*>] [Name==<*Time Period Name*>] [Active==*True|False*] [ShowUsage:=True]

Example 11-2. Listing all time periods

```
$  gold TimePeriod Query
Name     StartTime              EndTime                Active Descrip-
tion
-------- ---------------------- ---------------------- ------ --------
------------------------
Eternity -infinity              infinity               True   Always Active
1Q04     2004-01-01 00:00:00-08 2004-04-01 00:00:00-08 False  First Quar-
ter Calendar Year 2004
2Q04     2004-04-01 00:00:00-08 2004-07-01 00:00:00-07 False  Second Quar-
ter Calendar Year 2004
3Q04     2004-07-01 00:00:00-07 2004-10-01 00:00:00-07 True   Third Quar-
ter Calendar Year 2004
4Q04     2004-10-01 00:00:00-07 2005-01-01 00:00:00-08 False  Fourth Quar-
ter Calendar Year 2004
FY2004   2003-10-01 00:00:00-07 2004-10-01 00:00:00-07 True   Fiscal Year 2004
FY2005   2004-10-01 00:00:00-07 2005-10-01 00:00:00-07 False  Fiscal Year 2005
```

Modifying Time Periods

To modify a time period, use the command **gold TimePeriod Modify**:

gold TimePeriod Modify [StartTime=*YYYY-MM-DD [hh:mm:ss]*|*-infinity*|*infinity*] [EndTime=*YYYY-MM-DD [hh:mm:ss]*|*-infinity*|*infinity*] [Description=<*Description*>] Name==<*Period Name*> [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 inadvertant modification of all time periods.

Example 11-3. Changing a time period

```
$  gold TimePeriod Modify Name==FY2005 StartTime="2004-10-01" EndTime="2005-10-01"
Successfully modified 1 TimePeriods
```

Deleting Time Periods

To delete a time period, use the command **gold TimePeriod Delete**:

gold TimePeriod Delete [Name==<*Time Period Name*>] [Active==*True*|*False*]

> **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 time periods (including all associated allocations).

Example 11-4. Deleting a time period

```
$  gold TimePeriod Delete Name==FY2005
Successfully deleted 1 TimePeriods
```

Chapter 12. Managing Usage Records

Usage records are generated as a side-effect of a Job Charge. At the end of a job, a usage record is created for each resource used by the job.

Querying Usage Records

To display usage information, use the command **glsusage**:

glsusage [-J *job_id_pattern*] [-m *machine_name*] [-T *resource_type*] [-s *start_time*] [-e *end_time*] [—show *attribute_name*[,*attribute_name*...]...] [—showHidden] [—raw] [—debug] [-? | —help] [—man] [—quiet] [*usage_id*]

Example 12-1. Listing all usage on machine colony

```
$  glsusage -m colony
Id JobId      Resource   Amount Machine WallDuration ConsumptionRate Charg-
eRate Multiplier Charge Description
-- ---------- ---------- ------ ------- ------------ --------------- --
-------- ---------- ------ -----------
1  PBS.1234.0 Processors 16     colony  1234                         1         1
```

Chapter 13. 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*] [-U *actor*] [-u *user_name*] [-p *project_name*] [-m *machine_name*] [-J *job_id*] [-s *start_time*] [-e *end_time*] [-T *transaction_id*] [-R *request_id*] [—show *attribute_name*[,*attribute_name*...]...] [—showHidden] [—raw] [—debug] [-? | —help] [—man] [—quiet]

Example 13-1. List all deposits made in 2004

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

Example 13-2. List everything done by amy since the beginning of 2004

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

Example 13-3. List all transactions affecting Job Id PBS.1234.0

```
$  glstxn -J PBS.1234.0
```

Example 13-4. List all transactions affecting charge rates

```
$  glstxn -O ChargeRate
```

Chapter 14. 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 project 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 access 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 manager 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 account(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 ommitted, 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 manager 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 account(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 manager 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 manament system supports some kind of feedback or notification machanism 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 programatically 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 14-1. 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 14-2. To issue a charge at the completion of a job, you would use gcharge:

```
gcharge -J PBS.1234.0 -p chemistry -u amy -m colony -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 14-3. To issue a charge you must invoke the Charge action on the Job object:

```
gold Data:="<Job><JobId>PBS.1234.0</JobId><ProjectId>chemistry</ProjectId>
<UserId>amy</UserId><MachineName>colony</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 14-4. 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("JobId", "PBS.1234.0");
$job->setValue("ProjectId", "chemistry");
$job->setValue("UserId", "amy");
$job->setValue("MachineName", "colony");
$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 14-5. 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("JobId", "PBS.1234.0");
    job.setValue("ProjectId", "chemistry");
    job.setValue("UserId", "amy");
    job.setValue("MachineName", "colony");
    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 14-6. 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>
```

```
      <JobId>PBS.1234.0</JobId>
      <ProjectId>chemistry</ProjectId>
      <UserId>amyh</UserId>
      <MachineName>colony</MachineName>
      <Processors>2</Processors>
      <WallDuration>1234</WallDuration>
     </Job>
    </Data>
   </Request>
  <///Body>
  <Signature>
   <DigestValue>azu4obZswzBt89OgATukBeLyt6Y=</DigestValue>
   <SignatureValue>YXE/C08XX3RX4PMU1bWju+5/E5M=</SignatureValue>
   <SecurityToken type="Symmetric"></SecurityToken>
  </Signature>
</Envelope>
0
```

Notes

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

Chapter 15. Configuration Files

Gold uses two configuration files: one for the server (goldd.conf) and one for the clients (gold.conf). For configuration parameters that have hard-coded defaults, the default value is specified within brackets.

Server Configuration

The following configuration parameters may be set in the server configuration file (goldd.conf).

- *database.datasource* [DBI:Pg:dbname=gold;host=localhost] — The Perl DBI data source name for the database you wish to connect to.

- *database.password* — The password to be used for the database connection (if any).

- *database.user* — The username to be used for the database connection (if any).

- *log4perl.appender.Log.filename* — Used by log4perl to set the base name of the log file.

- *log4perl.appender.Log.max* — Used by log4perl to set the number of rolling backup logs.

- *log4perl.appender.Log.size* — Used by log4perl to set the size the log will grow to before it is rotated.

- *log4perl.appender.Log.Threshold* — Used by log4perl to set the debug level written to the log. The logging threshold can be one of TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

- *log4perl.appender.Screen.Threshold* — Used by log4perl to set the debug level written to the screen. The logging threshold can be one of TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

- *machine.autogen* [false] — If set to true, Gold will automatically create new machines when they are first encountered in a job function (charge, reserve, or quote).

- *machine.default* [NONE] — If not set to NONE, Gold will use the specified default for the machine in a job function (charge, reserve, or quote) in which a machine was not specified.

- *project.autogen* [false] — If set to true, Gold will automatically create new projects when they are first encountered in a job function (charge, reserve, or quote).

- *project.default* [NONE] — If not set to NONE, Gold will use the specified default for the project in a job function (charge, reserve, or quote) in which a project was not specified and no default project can be found for the user.

- *security.authentication* [true] — Indicates whether incoming message authentication is required.

- *security.encryption* [false] — Indicates whether incoming message encryption is required.

- *server.host* [localhost] — The hostname on which the gold server runs.

- *server.port* [7112] — The port the gold server listens on.

- *super.user* [root] — The primary gold system admin which by default can perform all actions on all objects. The super user is sometimes used as the actor in cases where an action is invoked from within another action.

- *user.autogen* [false] — If set to true, Gold will automatically create new users when they are first encountered in a job function (charge, reserve, or quote).

- *user.default* [NONE] — If not set to NONE, Gold will use the specified default for the user in a job function (charge, reserve, or quote) in which a user was not specified.

Client Configuration

The following configuration parameters may be set in the client configuration file (gold.conf).

- *log4perl.appender.Log.filename* — Used by log4perl to set the base name of the log file.

- *log4perl.appender.Log.max* — Used by log4perl to set the number of rolling backup logs.

- *log4perl.appender.Log.size* — Used by log4perl to set the size the log will grow to before it is rotated.

- *log4perl.appender.Log.Threshold* — Used by log4perl to set the debug level written to the log. The logging threshold can be one of TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

- *log4perl.appender.Screen.Threshold* — Used by log4perl to set the debug level written to the screen. The logging threshold can be one of TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

- *response.chunking* [true] — Indicates whether large responses should be segmented.

- *response.chunkSize* [1000] — Indicates the line length in the data response that will trigger message segmentation.

- *security.authentication* [true] — Indicates whether outgoing message are signed.

- *security.encryption* [false] — Indicates whether outgoing messages are encrypted.

- *security.token.type* [Symmetric] — Indicates the default security token type to be used in both authentication and encryption.

- *server.host* [localhost] — The hostname on which the gold server runs.

- *server.port* [7112] — The port the gold server listens on.