Gold User's Guide

Scott Jackson
Pacific Northwest National Laboratory

Gold User's Guide
by Scott Jackson

Table of Contents

Notice

Important: This is the second beta release of the User's Guide. Other information may be found by browsing the FAQ (<http://sss.scl.ameslab.gov/cgi-bin/faq?file=3&keywords=file>) posting to the gold users list (<gold-users@lyris.pnl.gov>) submitting bug reports or change requests (<gold-support@sss.scl.ameslab.gov>) or contacting the author (<Scott.Jackson@pnl.gov>).

Chapter 1. Overview

Gold is an open source accounting system that tracks and manages 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 Accounts* — 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 goldsh 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

```
goldsh User Query
```

> **Caution**
>
> The goldsh 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. Installation

Gold uses the standard configure, make and make install steps that we all know and love. However, there are a number of preparation, prerequisite, setup and customization steps that need to be performed. This document provides general installation guidance and provides a number of sample steps referenced to a particular installation on a Linux platform using the bash shell. These steps indicate the userid in brackets performing the step. The exact commands to be performed and the user that issues them will vary based on the platform, shell, installation preferences, etc.

Preparation

To build and install Gold, you first need to unpack the archive and change directory into the top directory of the distribution. For security reasons, it is recommended that you install and run Gold under its own non-root userid.

```
[root]#  useradd gold
[root]#  passwd gold
[gold]$  mkdir ~/src
[gold]$  cd ~/src
[gold]$  gzip -cd gold-2.b2.7.0.tar.gz | tar xvf -
[gold]$  cd gold-2.b2.7.0
```

Install Prerequisites

You will first need to build, test and install the following prerequisites:

Perl 5.6.1 or higher (with suidperl) [REQUIRED]

The gold server and clients are written in Perl. Perl 5.6.1 or higher is required. The perl installation must include suidperl for proper client authentication. Use 'perl -v' to see what level of Perl is installed and 'suidperl -v' to see if suidperl is installed. Perl is available at: <http://www.perl.com/>

```
[root]#  cd /usr/local/src
[root]#  wget http://www.cpan.org/src/stable.tar.gz
[root]#  gzip -cd stable.tar.gz | tar xvf -
[root]#  cd perl-5.8.5
[root]#  sh Configure -Dd_dosuid -de
[root]#  make
[root]#  make test
[root]#  make install
[root]#  (cd /usr/include && /usr/local/bin/h2ph *.h sys/*.h)
```

Or if you are using rpms, you will need the perl and the perl-suidperl rpms appropriate for your architecture and operating system:

```
[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/per
5.8.0-88.3.i386.rpm
```

```
[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/per
suidperl-5.8.0-88.3.i386.rpm
[root]#  rpm -Uvh perl-5.8.0-88.3.i386.rpm perl-suidperl-5.8.0-
88.3.i386.rpm
```

libxml2 2.4.25 or higher [REQUIRED]

LibXML2 is needed by the XML::LibXML perl module to communicate via the SSSRMAP message format. LibXML2 is available at: <http://www.xmlsoft.org/>

```
[root]#  cd /usr/local/src
[root]#  wget --passive-ftp ftp://xmlsoft.org/libxml2-2.6.17.tar.gz

[root]#  gzip -cd libxml2-2.6.17.tar.gz | tar xvf -
[root]#  cd libxml2-2.6.17
[root]#  ./configure
[root]#  make
[root]#  make install
```

PostgreSQL database 7.2 or higher [REQUIRED]

Gold makes use of a database for transactions and data persistence. Two databases have been tested for use with Gold thus far: PostgreSQL and SQLite. Postgres is an external database that must be separately installed configured and started, while SQLite is an embedded database bundled with the Gold source code. If you intend to use the PostgreSQL database, you will need to install it. PostgreSQL is available at: <http://www.postgresql.org/>

```
[root]#  cd /usr/local/src
[root]#  wget ftp://ftp3.us.postgresql.org/pub/postgresql/source/v7.4.5/postgre
7.4.5.tar.gz
[root]#  gzip -cd postgresql-7.4.5.tar.gz | tar xvf -
[root]#  cd postgresql-7.4.5
[root]#  ./configure
[root]#  make
[root]#  make install
[root]#  adduser postgres
[root]#  mkdir /usr/local/pgsql/data
[root]#  chown postgres /usr/local/pgsql/data
[root]#  touch /var/log/pgsql
[root]#  chown postgres /var/log/pgsql
```

Or if you are using rpms, you will need the postgresql, postgresql-libs, postgresql-server, and postgresql-devel rpms appropriate for your architecture and operating system:

```
[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/pos
7.3.2-3.i386.rpm
```

```
[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/pos
libs-7.3.2-3.i386.rpm

[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/pos
server-7.3.2-3.i386.rpm

[root]#  wget ftp://rpmfind.speakeasy.net/linux/redhat/updates/9/en/os/i386/pos
devel-7.3.2-3.i386.rpm

[root]#  rpm -Uvh postgresql-7.3.2-3.i386.rpm postgresql-libs-7.3.2-
3.i386.rpm postgresql-server-7.3.2-3.i386.rpm postgresql-devel-
7.3.2-3.i386.rpm
```

Gnu readline 2.0 or higher [OPTIONAL]

The interactive control program (goldsh) can support command-line-editing capabilities if readline support is enabled. Most recent linux distributions come with the appropriate readline support. Gnu readline is available at: <http://www.gnu.org/>

```
[root]#  cd /usr/local/src

[root]#  wget http://ftp.gnu.org/gnu/readline/readline-5.0.tar.gz

[root]#  gzip -cd readline-5.0.tar.gz | tar xvf -

[root]#  cd readline-5.0

[root]#  ./configure

[root]#  make

[root]#  make install
```

Java 2 SDK 1.4 or higher [OPTIONAL]

Gold provides a web based gui so that managers, users and administrators can interact with the accounting and allocation system. This alpha gui is still under active development and has not been widely tested, however, it is made available at this stage in the hopes that it may be useful to you as is, and so that we can receive feedback, improvements, etc. The web interface utilizes Java Server Pages and needs to have a servlet container installed. A low-level java client also exists but most clients are written in Perl. Java 2 SDK is available at: <http://java.sun.com/j2se>

```
[root]#  cd /usr/local/src
```

Download j2sdk-1_4_2_01-linux-i586.bin (or j2sdk-1_4_2_01-linux-i586-rpm.bin) from http://java.sun.com/j2se/1.4/download.html

```
[root]#  chmod +x j2sdk-1_4_2_01-linux-i586.bin

[root]#  cd /usr/local

[root]#  src/j2sdk-1_4_2_01-linux-i586.bin
```

Apache Tomcat Server 4 or higher [OPTIONAL]

Gold provides a web based gui so that managers, users and administrators can interact with the accounting and allocation system. The web interface utilizes Java Server Pages and needs to have a servlet container installed. Tomcat is available at: <http://jakarta.apache.org/tomcat/>

```
[root]#  cd /usr/local/src
```

wget ftp://ftp.tux.org/pub/net/apache/dist/jakarta/tomcat-4/v4.1.29/bin/jakarta-tomcat-4.1.29.tar.gz

```
[root]#  cd /usr/local
[root]#  gzip -cd src/jakarta-tomcat-4.1.29.tar.gz | tar xvf -
```

Configuration

To configure Gold, run the "configure" script provided with the distribution.

To see the list of options:

-h, —help display the list of options

Use prefix to tell it where Gold should be installed (defaults to /usr/local):

—prefix=PREFIX install architecture-independent files in PREFIX

Use with-db to specify the database you intend to use with Gold. Currently only PostgreSQL (Pg) and SQLit
gres is an external database which runs in a distinct (possibly remote) process and communicates over sockets while SQLite is an embedded database bundled with Gold with SQL queries being performed within the goldd process itself through library calls. Initial testing has shown SQLite to be at l
greSQL for small installations. The default is to use PostgreSQL.

—with-db=DATABASE database to be used { Pg, SQLite } [Pg]

Use without-readline if you do not want to use the gnu readline library

—without-readline Don’t use readline in interactive control program

Use with-user to specify the userid that gold will run under (defaults to the user running the configure command).

—with-user=USER user id under which the gold server will run

Use with-log-dir to specify the directory to which logs will be written (defaults to PREFIX/log).

—with-log-dir=PATH directory for log files [PREFIX/log]

Use with-perl-libs to indicate whether you want to install the required perl modules in a local gold directory (PREFIX/lib) or in the default system site-perl directory (triggered by running make deps).

—with-perl-libs=local | site install policy for prerequisite perl libs [local]

Use with-gold-libs to indicate whether you want to install the Gold modules in a local gold directory (PREFIX/lib) or in the default system site-perl directory (defaults to local).

—with-gold-libs=local | site install policy for Gold perl libs [local]

The PERL environment variable helps the install process find the desired (5.6) perl interpreter if it is not in your path or not found first in a path search.

PERL full pathname of the Perl interpreter

The JAVA_HOME environment variable helps the install process locate the java, javac and jar executables. JAVA_HOME can be calculated by configure if the desired java executable is found in a path searc

JAVA_HOME Java 2 SDK home directory (contains bin, lib, jre, ...)

The CATALINA_HOME environment variable helps the install process locate the Tomcat home directory. This variable must be set if you want to use the web GUI.

CATALINA_HOME
Tomcat home directory (contains common, conf, logs, webapps ...)

Some other influential environment variables are:
CC C compiler command
CFLAGS C compiler flags
LDFLAGS linker flags, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir>
CPPFLAGS C/C++ preprocessor flags, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir>

So, as an example you might use something like:

```
[gold]$ cd gold-2.b2.7.0
[gold]$ ./configure --prefix=/usr/local/gold CATALINA_HOME=/usr/local/jakarta-tomcat-4.1.29
```

Compilation

To compile the program, type make:

```
[gold]$ make
```

If you would like to try out the alpha web-gui, type make gui:

```
[gold]$ make gui
```

Perl Module Dependencies

Gold requires the use of a number of Perl modules. These modules are included in tarball form in the Gold distribution and they can be installed by typing 'make deps':

```
[root]# make deps
```

This will install the following Perl modules as necessary. By default, these will be installed under gold's lib/perl5 directory. To install these in the system site-perl directory, use the configure parameter with-perl-libs as described in the configuration section.

Compress::Zlib
Crypt::CBC
Crypt::DES
Crypt::DES_EDE3
Data::Properties

Date::Manip
DBI
DBD::Pg or DBD::SQLite
Digest
Digest::HMAC
Digest::MD5
Digest::SHA1
Error
Log::Dispatch
Log::Dispatch::FileRotate
Log::Log4perl
MIME::Base64
Module::Build
Params::Validate
Term::ReadLine::Gnu
Time::HiRes
XML::SAX
XML::LibXML::Common
XML::LibXML
XML::NamespaceSupport

If you would prefer to do so, you could install these modules via other sources, such as from rpm, or from CPAN using 'perl -MCPAN -e shell'.

Installation

Use 'make install' to install Gold. You may need to do this as root if any of the installation or log directories do not already have write permission as the gold admin user.

```
[root]#  make install
```

If you would like to try out the alpha web-gui, type make install-gui.

```
[root]#  make install-gui
```

The standard installation process will copy the binaries and perl scripts to /usr/local/bin, install the server in /usr/local/sbin, put the libs in /usr/local/lib, the config files in /usr/local/etc and the man pages in /usr/local/man. You can customize the directories either through the configuration process or by making the necessary changes in the Makefile.

To delete the files created by the Gold installation, you can use 'make uninstall'.

You will also need to generate a secret key which enables secure communication between clients and server. This key is a pass-phrase consisting of up to 80 characters and can include spaces and the regular visible ASCII characters. Note that if you are using Gold with the Maui Scheduler, they will need both need to use a shared secret key.

```
[root]#  make auth_key

Enter your secret key (up to 80 characters and can include spaces): sss
```

General Setup

Edit the Gold configuration files.

```
[gold]$  vi /usr/local/gold/etc/goldd.conf
[gold]$  vi /usr/local/gold/etc/gold.conf
```

Database Setup

If you have chosen to use PostgreSQL, you will need to configure the database to support Gold connections and schema. No setup is needed if you are using SQLite.

Initialize the database (if you installed from tarball).

```
[postgres]$  /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

Add the IP ADDRESS of the host where the Gold server will run (even if it is the same host as the database server).

```
[postgres]$  echo "host all all 192.168.1.1 255.255.255.255 trust"
»/usr/local/pgsql/data/pg_hba.conf
```

Startup postgres with the -i option to allow internet domain sockets

```
[postgres]$  /usr/local/pgsql/bin/postmaster -i -D /usr/local/pgsql/data
>/var/log/pgsql 2>&1 &
```

Add the "gold" user as a database administrator

```
[postgres]$  /usr/local/pgsql/bin/createuser gold

Shall the new user be allowed to create databases? y
Shall the new user be allowed to create more new users? n
```

Create the gold database

```
[gold]$  /usr/local/pgsql/bin/createdb gold
```

Edit the Gold configuration files.

```
[gold]$  vi /usr/local/gold/etc/goldd.conf
[gold]$  vi /usr/local/gold/etc/gold.conf
```

Web Server Setup

If you want to use the Gold web GUI, you will need to make some modifications to the tomcat configuration to support the use of SSL connections.

Configure your environment with the following environment variables:

```
[root]#  export CATALINA_HOME=/usr/local/jakarta-tomcat-4.1.29
[root]#  export JAVA_HOME=/usr/local/j2sdk1.4.2_01
```

Edit the server.xml file under $CATALINA_HOME/conf:

```
[root]#  vi $CATALINA_HOME/conf/server.xml
```

Uncomment the SSL connector. This will allow secure connections. You can specify the port, timeout, etc. according to your preferences. It might look something like this:

```
  <Connector className="org.apache.catalina.connector.http.HttpConnector" port="8443" m
Processors="5" maxProcessors="75" enableLookups="true" redirectPort="8443" ac-
ceptCount="10" debug="0" scheme="https" secure="true" connectionTimeout="60000">
    <Factory className="org.apache.catalina.net.SSLServerSocketFactory" clien-
tAuth="false" protocol="TLS"/>
  </Connector>
```

Create a certificate keystore by executing the following command and specify a password value of "changeit". For more help, see <http://jakarta.apache.org/tomcat/tomcat-4.1-doc/ssl-howto.html>.

```
[root]#  $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
```

Startup Tomcat. The make gui-install step should have copied the gold.war file into the $CATALINA_HOME/webapps directory. When tomcat is started up, it should unpack the jar into a gold directory here after a few seconds.

```
[root]#  $CATALINA_HOME/bin/startup.sh
```

Edit the servlet container's gold.conf file

```
[root]#  vi $CATALINA_HOME/webapps/gold/etc/gold.conf
```
At a minimum you will want to specify the server.host and log4j.appender.Log.File

Restart Tomcat

```
[root]#  $CATALINA_HOME/bin/shutdown.sh
[root]#  $CATALINA_HOME/bin/startup.sh
```

Bootstrap

You will need to populate the gold database with an sql dump that defines the objects, actions and attributes necessary to function as an Accounting and Allocation Manager.

```
[gold]$  /usr/local/pgsql/bin/psql gold <bank.sql
```

Startup

Start the gold server daemon. It is located in the PREFIX/sbin directory.

```
[gold]$  /usr/local/gold/sbin/goldd
```

Alternatively, if you are on linux system that supports init.d scripts, you can add an add gold as a system startup service by copying etc/gold.d to /etc/init.d/gold, giving it execute permission, and then start gold by issuing:

```
[root]#  service gold start
```

Initialization

You are now ready to define users, projects, machines, accounts etc. as necessary for your site. The next chapter (Getting Started) provides a useful primer for this phase of the Gold setup.

Chapter 3. 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, make deposits, 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 3-1. 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     DefaultPro-
ject 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 will also need to add the names of the machines that provide resources (see Creating Machines).

Example 3-2. 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 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).

> **Note:** In these examples we assume that the account.autogen configuration parameter is set to automatically create a default account for each project (see Server Configuration).

Example 3-3. We will define the projects biology and chemistry.

```
$  gmkproject -d "Biology Department" biology
Successfully created 1 Project
Auto-generated Account 1

$  gmkproject -d "Chemistry Department" chemistry
Successfully created 1 Project
Auto-generated Account 2

$  glsproject
Name      Active Users Machines Description
--------- ------ ----- -------- --------------------
biology   True                  Biology Department
chemistry True                  Chemistry Department
```

Add Users 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).

Example 3-4. Adding users to our projects.

```
$  gchproject --addUsers amy,bob biology
Successfully created 1 ProjectUser
Successfully created 1 ProjectUser

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

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

Make Deposits

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

Example 3-5. Let's add 360000000 credits to each project. We will cause them both to be valid just for the fiscal year 2005.

```
$  gdeposit -s 2005-01-01 -e 2006-01-01 -z 360000000 -p biology

Successfully deposited 3600000 credits into account 1

$  gdeposit -s 2005-01-01 -e 2006-01-01 -z 360000000 -p chemistry

Successfully deposited 3600000 credits into account 2
```

Let's examine the allocations we just created

```
$  glsalloc
Id Account StartTime  EndTime    Amount    CreditLimit Deposited Descrip-
tion
-- ------- ---------- ---------- --------- ----------- --------- ----
-------
1  1       2005-01-01 2006-01-01 360000000           0 360000000
2  2       2005-01-01 2006-01-01 360000000           0 360000000
```

Check The Balance

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

Example 3-6. Let's look at amy's balance

```
$  gbalance -u amy
Id Name      Amount    Reserved Balance   CreditLimit Available
-- --------- --------- -------- --------- ----------- ---------
1  biology   360000000 0        360000000 0           360000000
2  chemistry 360000000 0        360000000 0           360000000
```

Example 3-7. You may just want the total balance for a certain project and machine

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

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 3-8. 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 3-9. 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
```

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 3-10. 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       Job User Project   Machine EndTime               Type   D
scription
-- ------- ------ ---------- --- ---- --------- ------- --------------
-------- ------- -----------
1  2       57600  PBS.1234.0 1   amy  chemistry colony  2005-08-03 15:29:30-
07 Normal
```

This reservation will decrease our balance by the amount reserved.

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

Although our allocation has not changed.

```
$  glsalloc -p chemistry
Id Account StartTime  EndTime    Amount    CreditLimit Deposited Descrip-
tion
-- ------- ---------- ---------- --------- ----------- --------- ----
-------
2  2       2005-01-01 2006-01-01 360000000           0 360000000
```

This is best illustrated by the detailed balance listing:

```
$  gbalance -p chemistry
Id Name      Amount    Reserved Balance   CreditLimit Available
-- --------- --------- -------- --------- ----------- ---------
2  chemistry 360000000 57600    359942400 0           359942400
```

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 3-11. 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.

```
$ glsalloc -p chemistry
Id Account StartTime  EndTime    Amount    CreditLimit Deposited Descrip-
tion
-- ------- ---------- ---------- --------- ----------- --------- ----
-------
2  2       2005-01-01 2006-01-01 359980256           0 360000000
```

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

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

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

```
$ glsjob
Id JobId      User Project   Machine Charge Class Type   Stage  Qual-
ityOfService Nodes Processors Executable Application StartTime EndTime Wall-
Duration QuoteId Description
-- ---------- ---- --------- ------- ------ ----- ------ ------ ------
---------- ----- ---------- ---------- ----------- --------- ------- --
---------- ------- -----------
1  PBS.1234.0 amy  chemistry colony  19744        Normal Charge
```

Refund a Job

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

Example 3-12. 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 --total
Balance
---------
360000000
The account balance is 360000000 credits
```

The allocation, of course, is likewise restored.

```
$  glsalloc -p chemistry
Id Account StartTime  EndTime    Amount    CreditLimit Deposited Descrip-
tion
-- ------- ---------- ---------- --------- ----------- --------- ----
-------
2  2       2005-01-01 2006-01-01 360000000           0 360000000
```

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

```
$  glsjob
Id JobId      User Project   Machine Charge Class Type   Stage  Qual-
ityOfService Nodes Processors Executable Application StartTime EndTime Wall-
Duration QuoteId Description
-- ---------- ---- --------- ------- ------ ----- ------ ------ ------
---------- ----- ---------- ---------- ----------- --------- ------- --
---------- ------- -----------
1  PBS.1234.0 amy  chemistry colony  0            Normal Charge
```

List Transactions

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

Example 3-13. 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 Machine Amount
--------- ------------- ------ ------- ---------- --------- ---- ----
--- ------
298       299           Job    Create
298       303           Job    Quote              chemistry amy  colony  57600
299       304           Job    Modify
299       307           Job    Reserve PBS.1234.0 chemistry amy  colony  57600
300       311           Job    Charge  PBS.1234.0 chemistry amy  colony  19744
300       312           Job    Modify
301       314           Job    Refund  PBS.1234.0
301       315           Job    Modify
```

Example 3-14. 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
--- ----------- ------ ---------- ---------- ------ ------- ------
```

```
308 Usage       Create
309 Reservation Delete PBS.1234.0
310 Allocation  Modify 2
311 Job         Charge 1          PBS.1234.0 19744  2       -19744
312 Job         Modify 1
```

Examine Account Statement

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

Example 3-15. We can request an itemized account statement over all time for the chemistry project (account 2)

```
$  gstatement -p chemistry
################################################################################
#
# Statement for account 2 (chemistry) generated on Tue Aug  3 16:06:15 2005.
#
# Reporting account activity from -infinity to now.
#
################################################################################

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

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

Object  Action  JobId      Amount    Time
------- ------- ---------- --------- ---------------------
Account Deposit            360000000 2005-08-03 16:01:15-07
Job     Refund  PBS.1234.0 19744     2005-08-03 16:04:02-07

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

Object     Action     JobId      Project   User Machine Amount Time
---------- ---------- ---------- --------- ---- ------- ------ ------
-------------
Job        Charge     PBS.1234.0 chemistry amy  colony  -19744 2005-08-
03 16:03:39-07


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

Examine Project Usage

An additional report examines the charge totals for each user that completed jobs (see Project Usage Summary).

Example 3-16. Display usage by user for the chemistry project

```
$  gusage -p chemistry
################################################################################
#
# Usage Summary for project chemistry
# Generated on Tue Feb  8 11:05:06 2005.
# Reporting user charges from 2006-07-01 to 2006-10-01
#
################################################################################
User Amount
---- ------
amy   19744
```

Chapter 4. Getting More Advanced

In the previous chapter, a view of the system was presented that largely ignored the presence of accounts and other advanced features in Gold. This chapter will touch on the additional versatility derived from explicit use of accounts and other advanced features.

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

Define Projects

Let's assume that we have created users and machines as before in the Getting Started chapter (see Define Users and Define Machines). Again we will create some projects.

Note: In these examples we assume that the account.autogen configuration parameter is NOT set to automatically create a default account for each project (see Server Configuration).

Example 4-1. Now we will define the projects. This time we will define the project members at the same time.

For the biology project we will define a set of users and a default set of machines for the project. The specified default machine will be honored within accounts associated with this project that specify MEMBERS in the machine list.

```
$  gmkproject -d "Biology Department" -u amy,bob -m blue biology

Successfully created 1 Project
```

For the chemistry projects we will just define a set of member users.

```
$  gmkproject -d "Chemistry Department" -u amy,bob,dave chemistry

Successfully created 1 Project
```

Let's see what we've got so far in terms of projects.

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

Note: Note that accounts were not auto-generated this time because the account.autogen feature is set to false.

Define Accounts

Next, you can create your accounts (see Creating Accounts). Think of your accounts as bank accounts to which you can associate the users, projects and machines that can use them.

Example 4-2. We will create some accounts for use by the biology and chemistry projects.

```
$  gmkaccount -p biology -u MEMBERS -m MEMBERS -n "biology"
Successfully created Account 1

$  gmkaccount -p chemistry -u MEMBERS -m colony -n "chemistry on
colony"
Successfully created Account 2

$  gmkaccount -p chemistry -u amy -n "chemistry for amy"
Successfully created Account 3

$  gmkaccount -p chemistry -u MEMBERS,-amy -n "chemistry not amy"

Successfully created Account 4

$  glsaccount
Id Name                Amount Projects  Users        Machines Descrip-
tion
-- ------------------- ------ --------- ------------ -------- --------
------
1  biology                    biology   MEMBERS      MEMBERS
2  chemistry on colony        chemistry MEMBERS      colony
3  chemistry for amy          chemistry amy          ANY
4  chemistry not amy          chemistry MEMBERS,-amy ANY
```

So what we have here is: 1) a single account for biology available to all of its defined members and able to be used only on the blue machine (since blue is its only member machine) 2) an account usable toward the chemistry project on the colony machine only 3) an account usable anywhere for chemistry by amy only 4) an account usable anywhere for chemistry by any member except for amy

Make Deposits

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

Example 4-3. Let's deposit 100 million credits for use by the biology project. We are going to establish a use-it-or-lose-it policy here in which one fourth of the credits expire each quarter. Since there is only one account for the biology project, we can specify the project name in the deposit.

```
$  gdeposit -s 2005-01-01 -e 2005-04-01 -z 25000000 -p biology
Successfully deposited 25000000 credits into account 1
```

```
$  gdeposit -s 2005-04-01 -e 2005-07-01 -z 25000000 -p biology
Successfully deposited 25000000 credits into account 1

$  gdeposit -s 2005-07-01 -e 2005-10-01 -z 25000000 -p biology
Successfully deposited 25000000 credits into account 1

$  gdeposit -s 2005-10-01 -e 2006-01-01 -z 25000000 -p biology
Successfully deposited 25000000 credits into account 1
```

Example 4-4. Next we will make some deposits valid toward the chemistry project for the entire year. Since there are multiple accounts for the chemistry project, we must specify the appropriate account id in the deposit.

First, we'll dedicate 50 million credits for use on colony.

```
$  gdeposit -s 2005-01-01 -e 2006-01-01 -z 50000000 -a 2
Successfully deposited 50000000 credits into account 2
```

Then we'll give amy special access to 10 million credits that she can use anywhere — with 9 million credits prepaid, and a million credits of overdraft.

```
$  gdeposit -s 2005-01-01 -e 2006-01-01 -z 9000000 -L 1000000 -
a 3
Successfully deposited 9000000 credits into account 3
```

Finally, we'll give all the other members except amy access to the remaining 40 million credits.

```
$  gdeposit -s 2005-01-01 -e 2006-01-01 -z 40000000 -a 4
Successfully deposited 40000000 credits into account 4
```

Example 4-5. We can now take a closer look at the accounts and the allocations that we have created.

```
$  glsaccount
Id Name                Amount   Projects  Users        Machines Descrip-
tion
-- ------------------- -------- --------- ------------ -------- ------
-----
1  biology             25000000 biology   MEMBERS      MEMBERS
2  chemistry on colony 50000000 chemistry MEMBERS      colony
3  chemistry for amy    9000000 chemistry amy          ANY
4  chemistry not amy   40000000 chemistry MEMBERS,-amy ANY
```

Let's examine the allocations we just created with the time period information.

```
$  glsalloc
Id Account StartTime  EndTime    Amount   CreditLimit Deposited Descrip-
tion
```

```
-- ------- ---------- ---------- -------- ----------- --------- ------
-----
1  1       2005-01-01 2005-04-01 25000000           0  25000000
2  1       2005-04-01 2005-07-01 25000000           0  25000000
3  1       2005-07-01 2005-10-01 25000000           0  25000000
4  1       2005-10-01 2006-01-01 25000000           0  25000000
5  2       2005-01-01 2006-01-01 50000000           0  50000000
6  3       2005-01-01 2006-01-01  9000000     1000000   9000000
7  4       2005-01-01 2006-01-01 40000000           0  40000000
```

Check The Balance

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

Example 4-6. Let's look at amy's balance

```
$  gbalance -u amy
Id Name                Amount   Reserved Balance  CreditLimit Available
-- ------------------- -------- -------- -------- ----------- --------
-
1  biology             25000000        0 25000000           0  25000000
2  chemistry on colony 50000000        0 50000000           0  50000000
3  chemistry for amy    9000000        0  9000000     1000000  10000000
```

We see that amy's total balance is composed of some 25000000 credits useable toward the biology project, 50000000 for chemistry on colony and another 10000000 which can be used for chemistry on any machine. Notice that the 10000000 credits available for use in account 3 is composed of a 9000000 balance plus an overdraft limit of 1000000 (meaning your account can go negative by that amount).

Example 4-7. Let's just get amy's balance for chemistry on colony.

```
$  gbalance -u amy -p chemistry -m colony --total
Balance
-----
59000000
The account balance is 60000000 credits
```

Example 4-8. Now let's just get the total that can be used by amy for chemistry on colony. This includes amy's available credit.

```
$  gbalance -u amy -p chemistry -m colony --total --available
Balance
-----
60000000
The account balance is 60000000 credits
```

Define Charge Rates

Gold allows you to define how much you will charge for your resources (see Creating Charge Rates).

In the Getting Started chapter, we relied on the fact that the default Gold installation predefines a Processors charge rate for you. This means that the total charge for a job will be calculated by taking the number of processors used in the job multiplied by the Processors charge rate which is then multiplied by the wallclock limit. For example: ((16 [Processors] * 1 [ChargeRate{Resource}{Processors}])) * 1234 [WallDuration] = 19744.

Example 4-9. Let's examine the predefined charge rates.

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

Now let's create a few of our own.

Example 4-10. Let's say we want to charge for memory used

```
$  goldsh ChargeRate Create Type=Resource Name=Memory Rate=0.001

Successfully created 1 ChargeRate
```

Example 4-11. We also want a quality of service multiplier

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

Example 4-12. Creating another quality-based charge multiplier

```
$  goldsh ChargeRate Create Type=QualityOfService Name=Premium Rate=2

Successfully created 1 ChargeRate
```

Example 4-13. Let's take a look at the current charge rates.

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

Obtain A Guaranteed Job Quote

This time, we will use the job quote to guarantee our charge rates (this may be useful in the case of fluxuating rates like market based rates).

Example 4-14. Let's request a guaranteed charge quote that reflects the memory and quality of service we expect to use.

```
$  gquote -p chemistry -u amy -m colony -P 16 -M 2048 -t 3600 -
Q Premium -guarantee
Successfully quoted 129946 credits with quote id 1
```

This time it actually created a persistent quote ...

```
$  glsquote 1
Id Amount Job Project   User Machine StartTime           EndTime             Wall-
Duration CallType Used ChargeRates
scription
-- ------ --- --------- ---- ------- ------------------- ------------
------- ------------ -------- ---- ---------------------------------
----------------------------------- -----------
1  129946 1   chemistry amy  colony  2005-02-16 12:06:25 2005-02-23 13:06:25 3600
mal   0    QualityOfService:Premium:2,Resource:Processors:1,Resource:Memory:0.001
```

... and created a job entry.

```
$  glsjob -j 1
Id JobId User Project   Machine Queue QualityOfService Stage Charge Pro-
cessors Nodes WallDuration StartTime EndTime Description
-- ----- ---- --------- ------- ----- ---------------- ----- ------ --
-------- ----- ------------ --------- ------- -----------
1        amy  chemistry colony        Premium          Quote        16              36
```

Make A Quoted Job Reservation

If the quote id is specified when we make the reservation, the reservation will use the quoted amounts in calculating the amount to reserve and it will connect to the existing job entry.

Example 4-15. Make a reservation for our job that reflects our resource and quality preferences while specifying the quote id.

```
$  greserve -J PBS.1234.0 -p chemistry -u amy -m colony -P 16 -
M 2048 -t 3600 -Q Premium -q 1
Successfully reserved 129946 credits for job PBS.1234.0
```

```
$  glsres
Id Name       Amount StartTime           EndTime             Job User Project   Ma-
chine Accounts Description
-- ---------- ------ ------------------- ------------------- --- ----
--------- ------- -------- -----------
1  PBS.1234.0 129946 2005-02-16 12:35:13 2005-02-16 13:35:13 3   amy  chem-
istry colony  3
```

The reservation modifies the job entry to take on the new JobId and to change its stage from Quote to Reserve.

```
$  glsjob -j 1
Id JobId      User Project   Machine Queue QualityOfService Stage   Charge Pro-
cessors Nodes WallDuration StartTime EndTime Description
-- ---------- ---- --------- ------- ----- ---------------- ------- --
---- ---------- ----- ------------ --------- ------- -----------
1  PBS.1234.0 amy  chemistry colony        Premium          Reserve        16
```

As before, the reservation will decrease our balance by the amount reserved.

```
$  gbalance -u amy -p chemistry -m colony
Id Name                Amount   Reserved Balance  CreditLimit Available
-- ------------------- -------- -------- -------- ----------- --------
-
2  chemistry on colony 50000000        0 50000000           0  50000000
3  chemistry for amy    8960512   129946  8830566     1000000   9830566
```

Gold has two accounts to choose from. Gold will debit allocations in the order of earliest expiring and most specific first. Specifically, precedence is considered in the following order of highest to lowest: hierarchical relation, expiration time, generality of the project, generality of the user, and generality of the machine. Here we see that Gold considers the account that is exclusively for amy to be more specific (and of hence of higher precedence) than the account that is exclusively for the colony machine. This ordering will ensure that allocations that will expire the soonest will be used up first and that accounts with more specific access restrictions will be used in favor of accounts that have more general access (for example - amy will use up an account just for amy before the she begins using a shared account).

Charge for a Quoted Job

Even if the charge rates change between submission and completion of a job, a job tied to a quote will use the quoted charge rates in a prorated manner.

Example 4-16. Let's change a charge rate and issue the charge for our job. We will request that the quote be honored.

```
$  goldsh ChargeRate Modify Type==Resource Name==Memory Rate=.002

Successfully modified 1 ChargeRate

$  gcharge -J PBS.1234.0 -u amy -p chemistry -m colony -P 16 -M
2048 -t 1234 -Q Premium -q 1
Successfully charged job PBS.1234.0 for 44542 credits
1 reservations were removed
```

The charge modifies the job entry with the actual usage, charges and wallduration while changint its stage from Reserve to Charge.

```
$  glsjob -j 1
```

```
Id JobId      User Project   Machine Queue QualityOfService Stage  Charge Pro-
cessors Nodes WallDuration StartTime EndTime Description
-- ---------- ---- --------- ------- ----- ---------------- ------ --
---- ---------- ----- ------------ --------- ------- -----------
3  PBS.1234.0 amy  chemistry colony        Premium          Charge  44542 16
```

The detail charge information for the job can be extracted from the transaction log.

```
$  glstxn -A Charge -J PBS.1234.0 -show Details
Details
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
---
WallDuration=1234,QuoteId=1,QualityOfService=Premium,Processors=16,ItemizedCharges:=( (
cessors] * 1 [ChargeRate{Resource}{Processors}] ) + ( 2048 [Memory] * 0.001 [Charg-
eRate{Resource}{Memory}] ) ) * 1234 [WallDuration] * 2 [ChargeRate{QualityOfService}{Pr
```

Notice from the Itemized Charges above that the quoted memory charge rate of .001 was used instead of the current rate of .002. Notice also that the amounts have been prorated according to actual resources used and actual wallclock duration.

Partially Refund a Job

Example 4-17. Suppose you want to issue a partial refund.

```
$  grefund -j 1 -z 10000
Successfully refunded 10000 credits for job PBS.1234.0
```

Notice that the Job Charge is now 10000 credits lower as a result. Gold will not let your refunds total more than the total charge for the job.

```
$  glsjob 1
Id JobId      User Project   Machine Queue QualityOfService Stage  Charge Pro-
cessors Nodes WallDuration StartTime EndTime Description
-- ---------- ---- --------- ------- ----- ---------------- ------ --
---- ---------- ----- ------------ --------- ------- -----------
3  PBS.1234.0 amy  chemistry colony        Premium          Charge  34542 16
```

Examine Account Statement

You can get request account statement for our activites as they apply to a particular account.

Example 4-18. We can request an itemized account statement over all time for account 3 (chemistry for amy)

```
$  gstatement -a 3
```

```
################################################################################
#
# Statement for account 3 (chemistry for amy)
# Generated on Wed Feb 16 15:16:04 2005.
# Reporting account activity from -infinity to now.
#
################################################################################

Beginning Balance:                    0
------------------ --------------------
Total Credits:                  9010000
Total Debits:                    -44542
------------------ --------------------
Ending Balance:                 8965458

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

Object  Action  JobId Amount  Time
------- ------- ----- ------- -------------------
Account Deposit       9000000 2005-02-16 15:10:44
Job     Refund          10000 2005-02-16 15:15:36

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

Object Action JobId      Project   User Machine Amount Time
------ ------ ---------- --------- ---- ------- ------ --------------
-----
Job    Charge PBS.1234.0 chemistry amy  colony  -44542 2005-02-16 15:14:39

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

Chapter 5. 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*] [-p *default_project*] [-d *description*] [—debug] [-? | —help] [–man] [—quiet] [-v | —verbose] {[-u] *user_name*}

> **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 (see Server Configuration). However, bear in mind that users must be defined in order to assign them as members of a project. It is also possible to establish a system default user to be used in job functions (charge, reserve, quote) when the user is unspecified (user.default parameter).

Example 5-1. Creating a user

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

Querying Users

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

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

Example 5-2. Listing all info about active users

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

Example 5-3. Displaying bob's phone number

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

Example 5-4. Listing all user names without the header

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

Example 5-5. Listing a user's projects

```
$  glsuser --show Projects amy -l
Projects
---------
chemistry
biology
```

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] [—quiet] [-v | —verbose] {[-u] *user_name*}

Example 5-6. Activating a user

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

Example 5-7. 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 5-8. Deleting a user

```
$  grmuser bob
Successfully deleted 1 User
```

Chapter 6. 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*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-m] *machine_name*}

> **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 (see Server Configuration). However, bear in mind that machines must be defined in order to assign them as members of a project. It is also possible to establish a system default machine to be used in job functions (charge reserve, quote) when the machine is unspecified (machine.default parameter).

Example 6-1. Creating a machine

```
$  gmkmachine -d "Linux Cluster" colony
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] [—quiet] [[-m] *machine_pattern*]

Example 6-2. 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*] [-d *description*] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-m] *machine_name*}

Example 6-3. 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 6-4. Deleting a machine

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

Chapter 7. 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. The project user list will be honored within accounts including the project that specify MEMBERS in the user list. Machines may also be designated as members of a project as a default resource pool. The project machine list will be honored within accounts including the project that specify MEMBERS in the machine list.

Creating Projects

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

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

Note: If the account.autogen configuration parameter is set to true (see Server Configuration), an account will be automatically created for the project (unless overridden with the —createAccount option). The auto-generated account will be associated with the new project, the user MEMBERS of the project and ANY machine.

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 (see Server Configuration). It is also possible to establish a system default project (project.default) to be used in job functions (charge, reserve, quote) when the project is unspecified and the user does not have a default project.

Example 7-1. Creating a project

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

Example 7-2. Creating a project and specifying user members at the same time

```
$  gmkproject -d "Chemistry Department" -u amy,bob,dave chemistry

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] [—quiet] [[-p] *project_pattern*]

Example 7-3. Listing all info about all projects

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

Example 7-4. 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 7-5. Listing all project names

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

Modifying Projects

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

gchproject [-A | -I] [-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 7-6. Deactivating a project

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

Example 7-7. Adding users as members of a project

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

Example 7-8. Adding machines as members of a project

```
$  gchproject --addMachines colony chemistry
Successfully created 1 ProjectMachines
```

Deleting Projects

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

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

Example 7-9. Deleting a project

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

Project Usage Summary

To generate a project usage summary broken down by user, use the command **gusage**. This report lists the total charges by each of the active users during the specified time frame.

gusage [-s *start_time*] [-e *end_time*] [-h | —hours] [—debug] [-? | —help] [—man] {[-p] *project_name*}

Example 7-10. Displaying a usage summary for the chemistry project during the third quarter of 2006

```
$  gusage -p chemistry -s 2006-07-01 -e 2006-10-01
################################################################################
#
# Usage for project chemistry
# Generated on Tue Feb  8 11:05:06 2005.
# Reporting user charges from 2006-07-01 to 2006-10-01
#
################################################################################

User Amount
---- ------
amy   19744
bob   36078
```

Chapter 8. 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, an account is a repository for 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*...]] [-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, these will default to ANY.

> **Note:** It is possible to have accounts be created automatically when projects are created by setting the account.autogen configuration parameter to true (see Server Configuration). The auto-generated account will be associated with the new project, the user MEMBERS of the project and ANY machine.

Example 8-1. Creating an account

```
$  gmkaccount -p chemistry -u MEMBERS -m ANY -n "Chemistry"
Successfully created 1 Account
```

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

Example 8-2. Creating a wide-open account

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

Example 8-3. Creating an account valid toward all biology project members except for dave and all machines except for blue

```
$  gmkaccount -p biology -u MEMBERS,-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*] [-s *start_time*] [-e *end_time*] [—exact-match] [—show *attribute_name* [,*attribute_name*...]...] [—showHidden] [-l | —long] [-w | —wide] [—raw] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [[-a] *account_id*]

Example 8-4. Listing all info about all accounts with multi-valued fields displayed in a multi-line format

```
$  glsaccount --long
Id Name       Amount    Projects  Users   Machines Description
-- ---------- --------- --------- ------- -------- -----------
1  Biology    360000000 biology   MEMBERS blue
2  Chemistry  360000000 chemistry MEMBERS ANY
3  Cornucopia 0         ANY       ANY     ANY
4  Not Dave   250000    biology   -dave   -blue
```

Example 8-5. Listing all info about all accounts useable by dave

```
$  glsaccount -u dave --long
Id Name       Amount    Projects  Users   Machines Description
-- ---------- --------- --------- ------- -------- -----------
2  Chemistry  360000000 chemistry MEMBERS ANY
3  Cornucopia 0         ANY       ANY     ANY
```

Modifying Accounts

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

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

Example 8-6. 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 resulting in the creation or enlargement of an allocation. (See Allocations for managing allocations). The start time will default to -infinity and the end time will default to infinity if not specified. Accounts must first be created using **gmkaccount** (unless auto-generated).

gdeposit {-a *account_id* | -p *project_name*} [-i *allocation_id*] [-s *start_time*] [-e *end_time*] [[-z] *amount*] [-L *credit_limit*] [-d *description*] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 8-7. Making a deposit

```
$  gdeposit -s 2003-10-01 -e 2004-10-01 -z 360000000 -a 1
Successfully deposited 360000000 credits into account 1
```

Example 8-8. Making a deposit "into" a project

If a project has a single account then a deposit can be made against the project.

```
$  gdeposit -s 2003-10-01 -e 2004-10-01 -z 360000000 -p chemistry

Successfully deposited 360000000 credits into account 2
```

Example 8-9. Creating a credit allocation

```
$  gdeposit -L 10000000000 -a 3
Successfully deposited 0 credits into account 3
```

Querying The Balance

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

gbalance [-p *project_name*] [-u *user_name*] [-m *machine_name*] [—total] [—available] [—raw] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet]

Example 8-10. Querying the project balance detail broken down by account

```
$  gbalance -p chemistry
Id Name       Amount    Reserved Balance   CreditLimit   Available
-- ---------- --------- -------- --------- ------------- ------------
-
1  Chemistry  360000000 0        360000000 0             360000000
2  Cornucopia 0         0        0         1000000000000 1000000000000
```

Example 8-11. Querying the total balance for a particular user in a particular project on a particular machine

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

Example 8-12. List the projects and available balance amy can charge to

```
$  gbalance -u amy --show Project,Balance
Project   Balance
--------- ---------
biology   360000000
chemistry 360000000
```

Personal Balance

The **mybalance** has been provided as a wrapper script to show users their personal balance. It provides a list of balances for the projects that they can charge to:

gbalance [-h | —hours] [-? | —help] [—man]

Example 8-13. List my (project) balances

```
$  mybalance
Project   Balance
--------- -------------
biology       324817276
chemistry 9999979350400
```

Example 8-14. List my balance in (Processor) hours

```
$  mybalance -h
Project   Balance
--------- -------------
biology        90227.02
chemistry 2777772041.77
```

Making Withdrawals

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

gwithdraw {-a *account_id* | -p *project_name*} [-i *allocation_id*] {[-z] *amount*} [-d *description*] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 8-15. Making a withdrawal

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

Example 8-16. Making a withdrawal "from" a project

If a project has a single account then a withdrawal can be made against the project.

```
$  gwithdraw -z 12800 -p chemistry
Successfully withdrew 12800 credits from account 2
```

Making Transfers

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

the destination accounts. If a one-to-one mapping exists between project and account, then the fromProject/toProject options may be used in place of the fromAccount/toAccount options.

gtransfer {—fromAccount *source_account_id* | —fromProject *source_project_name* | -i *allocation_id*} {—toAccount *destination_account_id* | —toProject *destination_project_name*} [-d *description*] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-z] *amount*}

Example 8-17. Transferring credits between two accounts

```
$  gtransfer -fromAccount 1 -toAccount 2 10000
Successfully transferred 10000 credits from account 1 to account 2
```

Example 8-18. Transferring credits between two single-account projects

```
$  gtransfer -fromProject biology -toProject chemistry 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. This is followed by an itemized report of the debits and credits. Summaries of the debits and credits will be displayed instead of the itemized report if the —summarize option is specified. If a project, user or machine is specified instead of an account, then the statement will consist of information merged from all accounts valid toward the specified entities.

gstatement [[-a] *account_id*] [-p *project_name*] [-u *user_name*] [-m *machine_name*] [-s *start_time*] [-e *end_time*] [—summarize] [-h | —hours] [—debug] [-? | —help] [—man]

Example 8-19. Generating an account statement for the third quarter of 2006

```
$  gstatement -a 2 -s 2006-07-01 -e 2006-10-01
################################################################################
#
# Statement for account 2 (chemistry) generated on Tue Aug  3 16:06:15 2005.
#
# Reporting account activity from -infinity to now.
#
################################################################################

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

```
################################ Credit Detail ###################################

Object  Action  JobId      Amount    Time
------- ------- ---------- --------- ---------------------
Account Deposit            360000000 2005-08-03 16:01:15-07
Job     Refund  PBS.1234.0 19744     2005-08-03 16:04:02-07

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

Object     Action     JobId      Project   User Machine Amount Time
---------- ---------- ---------- --------- ---- ------- ------ ------
------------
Job        Charge     PBS.1234.0 chemistry amy  colony  -19744 2005-08-
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 8-20. Deleting an account

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

Chapter 9. Managing Allocations

An allocation is a time-bounded pool of resource credits associated with an account. An account may have multiple allocations, each for use during a different time period. An allocation may also have a credit limit representing the amount by which it can go negative.

Operations include querying, modifying and deleting allocations.

Creating Allocations

Allocations are created by making account deposits via the gdeposit command (See Making Deposits).

Querying Allocations

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

glsalloc [-A | -I] [-a *account_id*] [-p *project_name*] [—show *attribute_name* [,*attribute_name*...] [—hidden] [—raw] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [[-i] *allocation_id*]

Example 9-1. Listing allocations for account 4

```
$  glsalloc -a 4
Id Account StartTime  EndTime    Amount CreditLimit Deposited Active De-
scription
-- ------- ---------- ---------- ------ ----------- --------- ------ --
---------
4  4       2005-01-01 2005-04-01 250000           0    250000 False
5  4       2005-04-01 2005-07-01 250000           0    250000 False
6  4       2005-07-01 2005-10-01 250000           0    250000 True
7  4       2005-10-01 2006-01-01 250000           0    250000 False
```

Modifying Allocations

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

gchalloc [-s *start_time*] [-e *end_time*] [-L *credit_limit*] [-d *description*] [-h | —hours] [—debug] [-? | —help] [–man] [—quiet] [-v | —verbose] {[-i] *allocation_id*}

Example 9-2. Changing the end time for an allocation

```
$  gchalloc -e "2005-01-01" 4
Successfully modified 1 Allocation
```

Example 9-3. Changing the credit limit for an allocation

```
$  gchalloc -L 500000000000 -i 2
Successfully modified 1 Allocation
```

Deleting Allocations

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

grmalloc [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {-I | [-i] *allocation_id*}

Example 9-4. Deleting an allocation

```
$  grmalloc 4
Successfully deleted 1 Allocation
```

Example 9-5. Purging inactive allocations

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

Chapter 10. 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

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

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] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [[-r] *reservation_id*]

Example 10-1. Listing all info about all reservations for bob

```
$  glsres -u bob
Id Name              Amount StartTime           EndTime             Job User Project
chine Accounts Description
-- ----------------- ------ ------------------- ------------------- --
- ---- --------- ------- -------- -----------
1  Interactive.789654   3600 2005-01-13 16:48:15 2005-01-13 17:48:15 1   bob  chem-
istry blue    1
```

Example 10-2. Listing all info about all reservations that impinge against amy's balance

```
$  glsres -u amy --option name=UseRules value=True
Id Name              Amount StartTime           EndTime             Job User Project
chine Accounts Description
-- ----------------- ------ ------------------- ------------------- --
- ---- --------- ------- -------- -----------
1  Interactive.789654   3600 2005-01-13 16:48:15 2005-01-13 17:48:15 1   bob  chem-
istry blue    1
2  PBS.1234.0           7200 2005-01-13 17:59:09 2005-01-14 02:28:41 2   amy  chem-
istry colony  2
```

Modifying Reservations

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

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

Example 10-3. 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 10-4. Deleting a reservation by name (JobId)

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

Example 10-5. Deleting a reservation by ReservationId

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

Example 10-6. Purging stale reservations

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

Chapter 11. 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 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] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [[-q] *quote_id*]

Example 11-1. Listing all info about all quotes for user amy on machine colony

```
$  glsquote -u amy -m colony
Id Amount Job Project   User Machine StartTime           EndTime             Wall-
Duration Type   Used ChargeRates           Description
-- ------ --- --------- ---- ------- ------------------- ------------
------- ------------ ------ ---- -------------------- -----------
1  57600  1   chemistry amy  colony  2005-01-14 10:09:58 2005-09-10 15:27:07 3600
mal 0    Resource:Processors:1
```

Modifying Quotations

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

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

Example 11-2. 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 11-3. Deleting a quotation

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

Example 11-4. Purging stale quotations

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

Chapter 12. 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**:

goldsh Job Create JobId=<*Job Id*> [User=<*User Name*>] [Project=<*Project Name*>] [Machine=<*Machine Name*>] [Charge=<*Charge*>] [Queue=<*Class or Queue*>] [Type=<*Job Type*> *(Normal)*] [Stage=<*Last Job Stage*>] [QOS=<*Quality Of Service*>] [Nodes=<*Number Of Nodes*>] [Processors=<*Number Of Processors*>] [State=<*Job State*>] [Executable=<*Executable*>] [Application=<*Application*>] [StartTime=<*Start Time*>] [EndTime=<*End Time*>] [WallDuration=<*Wallclock Time in seconds*>] [QuoteId=<*Quote Id*>] [Description=<*Description*>] [ShowUsage:=true]

Example 12-1. Creating a job record

```
$  goldsh 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*] [—stage *stage*] [-s *start_time*] [-e *end_time*] [—show *attribute_name*[,*attribute_name*...]...] [—showHidden] [—raw] [—debug] [-? | —help] [—man] [—quiet] [[-j] *gold_job_id*]

Example 12-2. 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 **goldsh Job Modify**:

goldsh Job Modify [JobId==<*Job Id*> | Id==<*Gold Job Id*>] [User=<*User Name*>] [Project=<*Project Name*>] [Machine=<*Machine Name*>] [Charge=<*Charge*>] [Queue=<*Cla or Queue*>] [Type=<*Job Type*>] [Stage=<*Last Job Stage*>] [QOS=<*Quality Of Service*>] [Nodes=<*Number Of Nodes*>] [Processors=<*Number Of Processors*>] [State=<*Jo State*>] [Executable=<*Executable*>] [Application=<*Application*>] [StartTime=<*StartTime*>] [En Time=<*EndTime*>] [WallDuration=<*Wallclock Time in seconds*>] [QuoteId=<*Quote Id*>] [Description=<*Description*>] [ShowUsage:=true]

> **Caution**
>
> The goldsh 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 12-3. Changing a job

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

Successfully modified 1 Job
```

Deleting Jobs

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

goldsh Job Delete [JobId==<*Job Id*> | Id==<*Id*>]

> **Caution**
>
> The goldsh 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 12-4. Deleting a job

```
$  goldsh 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. This step verifies that the submitter has sufficient funds for, and meets all the allocation policy requirements for running the job and 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. If a guaranteed quote is requested, 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. A guaranteed quote has the side effect of creating a quotation record and a permanent job record.

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*] [-s *start_time*] [-e *end_time*] [-d *description*] [—guarantee] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 12-5. Requesting a quotation

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

Example 12-6. Requesting a guaranteed quote

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

Successfully quoted 57600 credits with quote id 1

$  glsquote
Id Amount Job Project   User Machine StartTime           EndTime             Wall-
Duration Type   Used ChargeRates           Description
-- ------ --- --------- ---- ------- ------------------- ------------
------- ------------ ------- ---- -------------------- -----------
1  57600  1   chemistry amy  colony  2005-01-14 10:09:58 2005-08-10 15:27:07 3600
mal 0    Resource:Processors:1
```

> **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*] [-s *start_time*] [-e *end_time*] [-q *quote_id*] [-d *description*] [—replace] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose] {[-J] *job_id*}

Example 12-7. 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 12-8. 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*] [-h | —hours] [—debug] [-? | —help] [—man] [—quiet] [-v | —verbose]

Example 12-9. Issuing a job refund

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

Chapter 13. 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 **goldsh ChargeRate Create**:

goldsh ChargeRate Create Type=<*Charge Rate Type*> Name=<*Charge Rate Name*> Rate=<*Floating Point Multiplier*> [Description=<*Description*>] [ShowUsage:=True]

Example 13-1. Creating a resource charge rate

```
$  goldsh ChargeRate Create Type=Resource Name=Processors Rate=1

Successfully created 1 ChargeRate
```

Example 13-2. Creating another resource charge rate

```
$  goldsh ChargeRate Create Type=Resource Name=Memory Rate=0.001

Successfully created 1 ChargeRate
```

Example 13-3. Creating a quality-based charge rate

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

Example 13-4. Creating another quality-based charge rate

```
$  goldsh ChargeRate Create Type=QualityOfService Name=Premium Rate=2

Successfully created 1 ChargeRate
```

Querying ChargeRates

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

goldsh ChargeRate Query [Show:=<*"Field1,Field2,..."*>] [Type==<*Charge Rate Type*>] [Name==<*Charge Rate Name*>] [Rate==<*Floating Point Multiplier*>] [Description==<*Description*>] [ShowUsage:=True]

Example 13-5. Listing all charge rates

```
$  goldsh 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 **goldsh ChargeRate Modify**:

goldsh ChargeRate Modify [Rate=<*Floating Point Multiplier*>] [Description=<*Description*>] [Type==<*Charge Rate Type*>] [Name==<*Charge Rate Name*>] [Rate==<*Floating Point Multiplier*>] [ShowUsage:=True]

> **Caution**
>
> The goldsh 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 13-6. Changing a charge rate

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

Successfully modified 1 ChargeRate
```

Deleting Charge Rates

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

goldsh ChargeRate Delete [Name==<*Charge Rate Name*>] [Rate==<*Floating Point Multiplier*>]

> **Caution**
>
> The goldsh 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 13-7. Deleting a charge rate

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

Chapter 14. 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*] [-a *account_id*] [-i *allocation_id*] [-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 14-1. List all deposits made in 2004

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

Example 14-2. List everything done by amy since the beginning of 2004

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

Example 14-3. List all transactions affecting Job Id PBS.1234.0

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

Example 14-4. List all transactions affecting charge rates

```
$  glstxn -O ChargeRate
```

Chapter 15. Managing Roles

Gold uses instance-level role based access controls to determine what users can perform what functions. Named roles are created, privileges are associated with the roles, and users are assigned to these roles.

Querying Roles

To display the currently defined roles, use the command **goldsh Role Query**:

goldsh Role Query [Show:=<*"Field1,Field2,..."*>] [Name==<*Role Name*>] [Description==<*Description*>] [ShowUsage:=True]

Example 15-1. Listing all roles

```
$  goldsh Role Query
Name         Description
------------ ------------------------------------------------
SystemAdmin  Can update or view any object
Anonymous    Things that can be done by anybody
OVERRIDE     A custom authorization method will be invoked
ProjectAdmin Can update or view a project they are admin for
UserServices User Services
Scheduler    Scheduler relevant Transactions
```

Querying Role Users

To list what users can perform what roles, use the command **goldsh RoleUser Query**:

goldsh RoleUser Query [Show:=<*"Field1,Field2,..."*>] [Role==<*Role Name*>] [Name==<*User Name*>] [ShowUsage:=True]

Example 15-2. Listing all role users

```
$  goldsh RoleUser Query
Role         Name
------------ ----
SystemAdmin  gold
Anonymous    ANY
OVERRIDE     ANY
Scheduler    maui
SystemAdmin  root
UserServices amy
```

Querying Role Actions

To list what actions can be performed by what roles, use the command **goldsh RoleAction Query**:

goldsh RoleAction Query [Show:=<*"Field1,Field2,..."*>] [Role==<*Role Name*>] [Object==<*Object Name*>] [Name==<*Action Name*>] [Instance==<*Instance Name*>] [ShowUsage:=Tr

Example 15-3. Listing all role actions

```
$  goldsh RoleAction Query
Role         Object         Name    Instance
------------ -------------- ------- --------
Anonymous    ANY            Query   ANY
Anonymous    Account        Balance ANY
Anonymous    Password       ANY     SELF
OVERRIDE     Account        Balance ANY
ProjectAdmin Project        ANY     ADMIN
Scheduler    Job            Charge  ANY
Scheduler    Job            Quote   ANY
Scheduler    Job            Reserve ANY
SystemAdmin  ANY            ANY     ANY
UserServices Job            Refund  ANY
UserServices Machine        ANY     ANY
UserServices Project        ANY     ANY
UserServices ProjectMachine ANY     ANY
UserServices ProjectUser    ANY     ANY
UserServices User           ANY     ANY
```

Creating Roles

To create a new role, use the command **goldsh Role Create**:

goldsh Role Create Name=<*Role Name*> [Description=<*Description*>] [ShowUsage:=True]

Example 15-4. Creating a Manager role

```
$  goldsh Role Create Name=Manager Description="Manages Roles and
Responsibilities"
Name    Description
------- ---------------------------------
Manager Manages Roles and Responsibilities
Successfully created 1 Role
```

Associating an Action with a Role

To add an action to a role, use the command **goldsh RoleAction Create**:

goldsh RoleAction Create Role=<*Role Name*> Object=<*Object Name*> Name=<*Action Name*> [Instance=<*Instance Name*>] [ShowUsage:=True]

The Instance indicates which specific instances of the object the action(s) can be performed on. Instances are interpreted as the value of the solitary primary key for an object. Unless otherwise specified, the instance will default to a value of ANY.

Valid values for Instance include:

ANY Any or all of the object instances
NONE No object instances
SELF Only objects identified with myself (like my own username)
ADMIN Only object instances that I am an admin for
<specific> A specific named instance

For example, the Role Action:

```
Role           Object         Name    Instance
-------------- -------------- ------- ---------
ChemistryAdmin Project        Modify  Chemistry
```

allows users having the ChemistryAdmin role to modify the Chemistry Project.

Example 15-5. Allow the Manager to change role responsibilities

```
$  goldsh RoleAction Create Role=Manager Object=RoleAction Name=ANY

Role    Object     Name Instance
------- ---------- ---- --------
Manager RoleAction ANY  ANY
Successfully created 1 RoleAction
```

Adding a Role to a User

To associate a user with a role, use the command **goldsh RoleUser Create**:

goldsh RoleUser Create Role=<*Role Name*> Name=<*User Name*> [ShowUsage:=True]

Example 15-6. Adding a user to the Manager role

```
$  goldsh RoleUser Create Role=Manager Name=dave
Role    Name
------- ----
Manager dave
Successfully created 1 RoleUser
```

Removing an Action from a Role

To disassociate an action from a role, use the command **goldsh RoleAction Delete**:

goldsh RoleAction Delete [Role==<*Role Name*>] [Object==<*Object Name*>] [Name==<*Action Name*>] [Instance==<*Instance Name*>] [ShowUsage:=True]

> **Caution**
>
> The goldsh 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 role actions.

Example 15-7. Don't let UserServices Create or Update Projects

```
$  goldsh RoleAction Delete Role==UserServices Object==Project Name==ANY

Role         Object  Name Instance
------------ ------- ---- --------
UserServices Project ANY  ANY
Successfully deleted 1 RoleActions
```

Removing a Role from a User

To disassociate a user and a role, use the command **goldsh RoleUser Delete**:

goldsh RoleUser Delete [Role==<*Role Name*>] [Name==<*User Name*>] [ShowUsage:=True]

> **Caution**
>
> The goldsh 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 role users.

Example 15-8. Removing dave as a Manager

```
$  goldsh RoleUser Delete Role==Manager Name==dave
Role    Name
------- ----
Manager dave
Successfully deleted 1 RoleUser
```

Deleting Roles

To delete a role, use the command **goldsh Role Delete**:

goldsh Role Delete [Name==<*Role Name*>] [Description==<*Description*>] [ShowUsage:=True]

Caution

The goldsh 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 roles.

Example 15-9. Deleting the Manager role

```
$  goldsh Role Delete Name==Manager
Name    Description
------- ---------------------------------
Manager Manages Roles and Responsibilities
Successfully deleted 1 Roles and 3 associations
```

Chapter 16. Managing Passwords

Passwords must be established for each user who wishes to use the web-based GUI. Passwords must be at least eight characters and are stored in encrypted form. Valid operations on passwords include creating, modifying and deleting passwords.

Creating Passwords

To create a new password, use the command **goldsh Password Create**:

goldsh Password Create User=<*User Name*> Password=<*Encrypted Password*> [ShowUsage:=True]

Example 16-1. Creating a password

```
$  goldsh Password Create User=amy Password=mysecret
User Password
---- --------------------------------
amy  Nn0NaSpwELQ+FKa36og9l6EczO+kUEoN
Successfully created 1 Password
```

Querying Passwords

To display password information, use the command **goldsh Password Query**:

goldsh Password Query [Show:=<*"Field1,Field2,..."*>] [User==<*User Name*>] [ShowUsage:=Tru

Example 16-2. List the users who have set passwords

```
$  goldsh Password Query Show:=User
User
----
amy
gold
```

Modifying Passwords

To change a password, use the command **goldsh Password Modify**:

goldsh Password Modify [Password=<*Encrypted Password*>] [Name==<*User Name*>] [ShowUsage:=True]

> **Caution**
>
> The goldsh 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 passwords.

Example 16-3. Changing amy's password

```
$  goldsh Password Modify User==amy Password=changeme
User Password
---- --------------------------------
amy  HZYzwD20o1XIE/gxRYyFKP2sumkCluHm
Successfully modified 1 Passwords
```

Deleting Passwords

To delete a password, use the command **goldsh Password Delete**:

goldsh Password Delete [Name==<*User Name*>]

> **Caution**
>
> The goldsh 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 passwords.

Example 16-4. Deleting a password

```
$  goldsh Password Delete User==amy
User Password
---- --------------------------------
amy  HZYzwD20o1XIE/gxRYyFKP2sumkCluHm
Successfully deleted 1 Passwords
```

Chapter 17. Using the Gold Shell (goldsh)

goldsh is an interactive control program that can access all of the advanced functionality in Gold.

> **Caution**
>
> The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. Inadvertant mistakes could result in potentially irreversible modifications.

Usage

Gold commands can be invoked directly from the command line as arguments, or read from stdin (interactively or redirected from a file).

goldsh [—debug] [-? | —help] [–man] [—raw] [—quiet] [-v | —verbose] [<*Command*>]

Example 17-1. Specifying the command as direct arguments

```
$ goldsh System Query
Name Version  Organization Description
---- -------- ------------ ------------
Gold 2.0.b1.0              Beta Release
```

Example 17-2. Using the interactive prompt

```
$ goldsh
gold> System Query
Name Version  Organization Description
---- -------- ------------ ------------
Gold 2.0.b1.0              Beta Release

gold> quit
```

Example 17-3. Reading commands from a file

```
$ cat >commands.gold <<EOF
System Query
quit
EOF

$ goldsh <commands.gold
Name Version  Organization Description
---- -------- ------------ ------------
Gold 2.0.b1.0              Beta Release
```

Command Syntax

Gold commands are of the form:

<*Object*> [,<*Object*>...] <*Action*> [[<*Conjunction*>] [<*Open_Parenthesis*>...] [<*Object*>.]<*Name*> <*Operator*> [<*Object*>.]<*Value*> [<*Close_Parenthesis*>...] ...]

The basic form of a command is <Object> <Action> [<Name><Operator><Value>]*. When an action is performed on more than one object, such as in a multi-object query, the objects are specified in a comma-separated list. Commands may accept zero or more predicates which may function as fields to return, conditions, update values, processing options, etc. Predicates, in their simplest form, are expressed as Name, Operator, Value tuples. Predicates may be combined via conjunctions with grouping specified with parentheses. When performing multi-object queries, names and values may need to be associated with their respective objects.

Valid conjunctions include:

&&
: and

||
: or

&!
: and not

|!
: or not

Open parentheses may be any number of literal open parentheses '('.

Name is the name of the condition, assignment, or option. When performing a multi-object query, a name may need to be prepended by its associated object separated by a period.

Valid operators include:

==
: equals

<
: less than

>
: greater than

<=
: less than or equal to

>=
: greater than or equal to

!=
: not equal to

~
: matches

=
: is assigned

+=
: is incremented by

-=
: is decremented by

:=
: option

:!
: not option

Value is the value of the selection list, condition, assignment, or option. When performing a multi-object query, a value may need to be prepended by its associated object (called the subject) separated by a period.

Close parentheses may be any number of literal closing parentheses ')'.

Valid Objects

To list the objects available for use in Gold commands, issue the gold command: Object Query

Example 17-4. Listing all objects

```
gold> Object Query Show:="sort(Name)"
Name
--------------------
ANY
Account
AccountAccount
AccountMachine
AccountOrganization
AccountProject
AccountUser
Action
Allocation
Attribute
ChargeRate
Job
Machine
NONE
Object
Organization
Password
Project
```

```
ProjectMachine
ProjectUser
Quotation
QuotationChargeRate
Reservation
Role
RoleAction
RoleUser
System
Transaction
Usage
User
```

Valid Actions for an Object

To list the actions that can be performed on an object, use the gold command: Action Query

Example 17-5. Listing all actions associated with the Account object

```
gold> Action Query Object==Account Show:=sort(Name)
Name
--------
Balance
Create
Delete
Deposit
Modify
Query
Transfer
Undelete
Withdraw
```

Valid Predicates for an Object and Action

By appending the option "ShowUsage:=True" to a command, the syntax of the command is returned, expressed in SSSRMAP XML Message Format.

Example 17-6. Show the usage for Allocation Query

```
gold> Allocation Query ShowUsage:=True
<Request action="Query">
    <Object>Allocation<Object>
    [<Get name="Id" [op="sort|tros|count|groupby|max|min"]></Get>]
    [<Get name="Account" [op="sort|tros|count|groupby|max|min"]></Get>]
    [<Get name="StartTime" [op="sort|tros|count|groupby|max|min"]></Get>]
    [<Get name="EndTime" [op="sort|tros|count|groupby|max|min"]></Get>]
    [<Get name="Amount" [op="sort|tros|count|groupby|max|min|sum|average"]></Get>]
    [<Get name="Deposited" [op="sort|tros|count|groupby|max|min|sum|average"]></Get>]
    [<Get name="Active" [op="sort|tros|count|groupby"]></Get>]
    [<Get name="Description" [op="sort|tros|count|groupby|max|min"]></Get>]
    [<Where name="Id" [op="eq|ne|gt|ge|lt|le (eq)"] [conj="and|or (and)"] [group="<Int
    [<Where name="Account" [op="eq|ne|gt|ge|lt|le|match (eq)"] [conj="and|or (and)"] [g
```

```
    [<Where name="StartTime" [op="eq|ne|gt|ge|lt|le (eq)"] [conj="and|or (and)"] [grou
MM-DD [hh:mm:ss]|-infinity|infinity|now</Where>]
    [<Where name="EndTime" [op="eq|ne|gt|ge|lt|le (eq)"] [conj="and|or (and)"] [group="
MM-DD [hh:mm:ss]|-infinity|infinity|now</Where>]
    [<Where name="Amount" [op="eq|ne|gt|ge|lt|le (eq)"] [conj="and|or (and)"] [group="<
    [<Where name="Deposited" [op="eq|ne|gt|ge|lt|le (eq)"] [conj="and|or (and)"] [grou
    [<Where name="Active" [op="eq|ne (eq)"] [conj="and|or (and)"] [group="<Integer Num
    [<Where name="Description" [op="eq|ne|gt|ge|lt|le|match (eq)"] [conj="and|or (and)"
    [<Option name="ShowHidden">True|False (False)</Option>]
    [<Option name="ShowUsage">True|False (False)</Option>]
    [<Option name="Time">YYYY-MM-DD [hh:mm:ss]</Option>]
    [<Option name="Unique">True|False (False)</Option>]
    [<Option name="Limit">Integer Number}</Option>]
<Request>
```

Common Options

There are a number of options that may be specified for all commands. These options include: ShowUsage

ShowUsage

This option may be included with any command to cause the command to return a usage message in SSSRMAP XML Message Format.

Common Actions Available for most Objects

There are a number of actions that are available for most objects. These actions include Query, Create, Modify, Delete and Undelete. Commands involving these actions inherit some common structure unique to the action type.

Query Action

The Query action is used to query objects. It accept predicates that describe the attributes (fields) to return (including aggregation operations on those attributes), conditions that select which objects to return the attributes for, and other options unique to queries.

Selections

Selections use the Show option to specify a list of the attributes to return for the selected object. If selections are not specified, a default set of attributes (those not marked as hidden) will be returned.

Name = Show
Op = :=
Value = "attribute1,attribute2,attribute3,..."

Aggregation operators may be applied to attributes by enclosing the target attribute in parenthesis and prepending the name of the desired operator. The aggregation operators that can be applied depend on the datatype of the attribute.

Valid selection operators include:

sort Ascending sort
tros Descending sort
count Count
max Maximum value
min Minimum value
average Average value
sum Sum
groupby Group other aggregations by this attribute

For example: Allocation Query Show:="sum(Amount),groupby(Account)"

Conditions

Conditions are used to select which objects the action is to be performed on.

Name = Name of the attribute to be tested
Op = conditional operator
Value = The object or value against which the attribute is tested

Valid condition operators include:

== Equal to
!= Not equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
~ Matches

Matching uses the wildcards * and ? (equivalent to SQL % and _ respectively) in a manner similar to file globbing. * matches zero or more unspecified characters and ? matches exactly one unspecified character. For example mscf* matches objects having the specified attributes whose values start with the letters mscf, while mscf? matches objects having the specified attributes whose values start with mscf and have a total of exactly five characters.

Options

Options indicate processing options that affect the result.

Name = Name of the option
Op = :=
Value = Value of the option

Valid options for query actions include:

ShowHidden:=True | False (False) Includes hidden attributes in the result
Time:="YYYY-MM-DD [hh:mm:ss]" Run the command as if it were the specified time
Unique:=True | False (False) Display only unique results (like DISTINCT in SQL)
Limit:={Integer Number} Limit the results to the number of objects specified

Example 17-7. Return the number of inactive reservations

```
gold> Reservation Query EndTime<now Show:="count(Id)"
Id
--
8
```

Create Action

The Create action is used to create a new object. It accepts predicates that describe the values of the attributes to be set.

Assignments

Assignments specify values to be assigned to attributes in the new object.

Name = Name of the attribute being assigned a value
Op = = (is assigned)
Value = The new value being assigned to the attribute

Example 17-8. Add a new project member

```
gold> ProjectUser Create Project=chemistry Name=scottmo
Project   Name    Active Admin
--------- ------- ------ -----
chemistry scottmo True   False
Successfully created 1 ProjectUser
```

Modify Action

The Modify action is used to modify existing objects. It accepts predicates that select which objects will be modified and predicates that describe the values of the attributes to be set.

Assignments

Assignments specify values to be assigned to attributes in the selected objects.

Name = Name of the attribute being assigned a value
Op = assignment operators {=, +=, -=}
Value = The value being assigned to the attribute

Valid assignment operators include:

= is assigned
+= is incremented by
-= is decremented by

Conditions

Conditions are used to select which objects the action is to be performed on.

Name = Name of the attribute to be tested
Op = conditional operator
Value = The object or value against which the attribute is tested

Valid condition operators include:

== Equal to
!= Not equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
~ Matches

Matching uses the wildcards * and ? (equivalent to SQL % and _ respectively) in a manner similar to file globbing. * matches zero or more unspecified characters and ? matches exactly one unspecified character. For example mscf* matches objects having the specified attributes whose values start with the letters mscf, while mscf? matches objects having the specified attributes whose values start with mscf and have a total of exactly five characters.

Example 17-9. Change/set scottmo phone number and email address

```
gold> User Modify Name==scottmo PhoneNumber="(509) 376-2204" EmailAd-
dress="Scott.Jackson@pnl.gov"
Name    Active CommonName        PhoneNumber    EmailAddress          De-
faultProject Description
------- ------ ----------------- -------------- ---------------------
-------------- -----------
scottmo True   Jackson, Scott M. (509) 376-2204 Scott.Jackson@pnl.gov

Successfully modified 1 Users
```

Example 17-10. Extend all reservations against project chemistry by 10 days

```
gold> Reservation Modify EndTime+="10 days" Project==chemistry
Id Account Amount Name       Job User Project   Machine EndTime            De-
scription
-- ------- ------ ---------- --- ---- --------- ------- --------------
----- -----------
1  2       57600  PBS.1234.0 1   amy  chemistry colony  2004-11-06 10:47:30
Successfully modified 1 Reservations
```

Delete Action

The Delete action is used to delete objects. It accepts predicates that select which objects are to be deleted.

Conditions

Conditions are used to select which objects the action is to be performed on.

Name = Name of the attribute to be tested
Op = conditional operator
Value = The object or value against which the attribute is tested

Valid condition operators include:

== Equal to
!= Not equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
~ Matches

Matching uses the wildcards * and ? (equivalent to SQL % and _ respectively) in a manner similar to file globbing. * matches zero or more unspecified characters and ? matches exactly one unspecified character. For example mscf* matches objects having the specified attributes whose values start with the letters mscf, while mscf? matches objects having the specified attributes whose values start with mscf and have a total of exactly five characters.

Example 17-11. Get rid of the pesky Jacksons

```
gold> User Delete CommonName~"Jackson*"
Name    Active CommonName        PhoneNumber    EmailAddress          De-
faultProject Description
------- ------ ----------------- -------------- ---------------------
-------------- -----------
scottmo True   Jackson, Scott M. (509) 376-2204 Scott.Jackson@pnl.gov

Successfully deleted 1 Users and 1 associations
```

Undelete Action

The Delete action is used to restore deleted objects. It accepts predicates that select which objects are to be undeleted.

Conditions

Conditions are used to select which objects the action is to be performed on.

Name = Name of the attribute to be tested
Op = conditional operator
Value = The object or value against which the attribute is tested

Valid condition operators include:

== Equal to
!= Not equal to
< Less than
> Greater than

<= Less than or equal to
>= Greater than or equal to
~ Matches

Matching uses the wildcards * and ? (equivalent to SQL % and _ respectively) in a manner similar to file globbing. * matches zero or more unspecified characters and ? matches exactly one unspecified character. For example mscf* matches objects having the specified attributes whose values start with the letters mscf, while mscf? matches objects having the specified attributes whose values start with mscf and have a total of exactly five characters.

Example 17-12. Let's resurrect the deleted users that were active

```
gold> User Undelete Active==True
Name    Active CommonName        PhoneNumber    EmailAddress          De-
faultProject Description
------- ------ ----------------- -------------- --------------------
-------------- -----------
scottmo True   Jackson, Scott M. (509) 376-2204 Scott.Jackson@pnl.gov

Successfully undeleted 1 Users and 1 associations
```

Multi-Object Queries

Gold supports multi-object queries (table joins). Multiple objects are specified via a comma-separated list and attributes need to be prefixed by the associated object.

Example 17-13. Print the current and total allocation summed by project

```
gold> Allocation,AccountProject Query Show:="groupby(AccountProject.Name),sum(A
Allocation.Account==AccountProject.Account Allocation.Active==True

Name      Amount    Deposited
--------- --------- ---------
biology   193651124 360000000
chemistry 296167659 360000000
```

Example 17-14. Show all active projects for amy or bob

```
gold> Project,ProjectUser Query Show:="Project.Name" ( ProjectUser.Name==bob
|| ProjectUser.Name==amy ) && Project.Name==ProjectUser.Project
&& Project.Active==True Unique:=True
Name
---------
biology
chemistry
```

Chapter 18. 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 their config files.

Example 18-1. Configuring Maui to use Gold

Add an appropriate AMCFG line into maui.cfg to tell Maui how to talk to Gold

```
$  vi /usr/local/maui/maui.cfg
```

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

Add a CLIENTCFG line into maui-private.cfg to specify the shared secret key. This secret key will be the same secret key specified in the "make auth_key" step.

```
$  vi /usr/local/maui/maui-private.cfg
```

```
CLIENTCFG[AM:bank] CSKEY=sss CSALGO=HMAC
```

Gold will need to allow the the user id that maui runs under to perform scheduler related commands (Job Charge, Reserve, Quote, etc).

```
$  gmkuser -d "Maui Scheduler" maui
```

```
Successfully created 1 User
```

```
$  goldsh RoleUser Create Role=Scheduler Name=maui
```

```
Role      Name
---------- ----
Scheduler maui
Successfully created 1 RoleUser
```

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 18-2. 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, goldsh, will issue a charge for a job expressed in xml (SSS Job Object).

Example 18-3. To issue a charge you must invoke the Charge action on the Job object:

```
goldsh 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 18-4. 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 18-5. 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 18-6. 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 19. 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).

- *account.autogen* [true] — If set to true, when a new project is created Gold will automatically create an associated default account.

- *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).

- *response.chunksize* [0] — Indicates the line length in the data response that will trigger message segmentation (or truncation). A value of 0 (zero) means unlimited, i.e. that the server will not truncate or segment large responses unless overriden by a chunksize specification in a client request. The response chunksize will be taken to be the smaller of the client and server chunksize settings.

- *currency.precision* [0] — Indicates the number of decimal places in the resource credit currency. For example, if you are will be dealing with processor-seconds of an integer resource unit, use 0 (which is the default). If you will be charging dollars and cents, then use 2. This parameter should be the same in the goldd.conf and gold.conf files.

- *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 chunked (segmented). If set to false, large responses will be truncated.

- *response.chunksize* [0] — Indicates the line length in the data response that will trigger message segmentation (or truncation). A value of 0 (zero) means unlimited, i.e. that the client will accept the chunksize set by the server. The response chunksize will be taken to be the smaller of the client and server chunksize settings.

- *currency.precision* [0] — Indicates the number of decimal places in the resource credit currency. For example, if you are will be dealing with processor-seconds of an integer resource unit, use 0 (which is the default). If you will be charging dollars and cents, then use 2. This parameter should be the same in the goldd.conf and gold.conf files.

- *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.