




Moab Accounting Manager[®]

Administrator Guide

Version 7.1.0

Notice

 This is the minor release of the Moab Accounting Manager Administrator Guide. Other information may be found by subscribing and posting to the Moab Accounting Manager users list (gold-users@supercluster.org) or by submitting bug reports or change requests to Adaptive Computing Enterprises, Inc. (gold-support@adaptivecomputing.com).

Moab Accounting Manager®	1
Notice	2
1.0 Overview	11
1.1 Background	12
1.2 Conceptual Overview	13
1.3 Features	15
1.4 Interfaces	17
1.4.1 Command-Line Clients	17
1.4.2 Interactive Control Program	17
1.4.3 Web-based Graphical User Interface	17
1.4.4 Perl API	17
1.4.5 Java API	18
1.4.6 SSSRMAP Wire Protocol	19
1.5 Documentation	20
1.6 License	21
1.6.1 Moab Accounting Manager License	21
1.6.2 BSD License	21
2.0 Installation	23
2.1 Select a Database	24
2.2 Install Prerequisites	26
2.2.1 Perl 5.8 or higher [REQUIRED]	26
2.2.2 Suidperl 5.8 or higher [OPTIONAL]	26
2.2.3 PostgreSQL database 7.2 or higher (or other tested database) [OPTIONAL]	26
2.2.4 libxml2 2.4.25 or higher [REQUIRED]	27
2.2.5 gnu readline 2.0 or higher [OPTIONAL]	28
2.2.6 Apache Httpd Server 2.0 or higher with mod_ssl [OPTIONAL]	28
2.2.7 OpenSSL 0.9.5a or higher [REQUIRED]	29
2.2.8 Disable SELinux	29
2.3 Preparation	30
2.4 Configuration	31
2.5 Compilation	34

2.6 Perl Module Dependencies	35
2.7 Installation	37
2.8 Database Setup	38
2.8.1 Initialize the database	38
2.8.2 Configure trusted connections	38
2.8.3 Enable support for transactions	38
2.8.4 Start the database	39
2.8.5 Create the database	39
2.8.6 Bootstrap	39
2.9 General Setup	41
2.10 Startup	42
2.11 Web Server Setup	43
2.12 Accessing the GUI	46
2.13 Initialization	47
3.0 Upgrading	48
3.1 Preparation	49
3.2 Configuration	50
3.3 Compilation	51
3.4 Server Shutdown	52
3.5 Installation	53
3.6 Server Startup	54
4.0 Migrating	55
4.1 Database Backup	56
4.2 Create Database	57
4.3 Preparation	58
4.4 Configuration	59
4.5 Compilation	60
4.6 Installation	61
4.7 General Setup	62
4.8 Server Startup	63
4.9 Database Migration	64
5.0 Getting Started	65

5.1 HPC Usage Tracking	66
5.1.1 Usage Record Customization (Optional)	67
5.1.2 Record the Usage	68
5.1.3 List Usage Records	69
5.2 HPC Charge Accounting	70
5.2.1 Usage Record Customization (Optional)	71
5.2.2 Decide on a Currency and Set the Currency Precision	72
5.2.3 Define Charge Rates	73
5.2.4 Create a Single NonLimiting Account	74
5.2.5 Create an Unlimited Allocation	75
5.2.6 Issue a Refund	77
5.2.7 Examine Account Statement	78
5.3 HPC Allocation Enforcement	80
5.3.1 Usage Record Customization (Optional)	81
5.3.2 Decide on a Currency and Set the Currency Precision	82
5.3.3 Define Charge Rates	83
5.3.4 Define Accountable Entities	84
5.3.5 Create Accounts	86
5.3.6 Make Deposits	88
5.3.7 Check The Balance	90
5.3.8 Integrate Moab Accounting Manager with Your Brokering System	91
5.3.9 Obtain a Usage Quote	92
5.3.10 Make a Usage Reservation	93
5.3.11 Charge for the Usage	95
5.3.12 Usage Refund	97
5.3.13 Examine Account Statement	99
6.0 Managing Users	101
6.1 Creating Users	102
6.2 Querying Users	103
6.3 Modifying Users	105
6.4 Deleting Users	106
6.5 User Auto-Generation	107

6.6 Default User	108
7.0 Managing Projects	109
7.1 Creating Projects	110
7.2 Querying Projects	111
7.3 Modifying Projects	113
7.4 Deleting Projects	114
7.5 Project Auto-Generation	115
7.6 Default Project	116
8.0 Managing Machines	117
8.1 Creating Machines	118
8.2 Querying Machines	119
8.3 Modifying Machines	120
8.4 Deleting Machines	121
8.5 Machine Auto-Generation	122
8.6 Default Machine	123
9.0 Managing Accounts	124
9.1 Creating Accounts	126
9.2 Querying Accounts	128
9.3 Modifying Accounts	130
9.4 Making Deposits	131
9.5 Querying The Balance	134
9.6 Personal Balance	136
9.7 Making Withdrawals	137
9.8 Making Transfers	138
9.9 Obtaining an Account Statement	139
9.10 Deleting Accounts	142
9.11 Account Auto-Generation	143
9.12 Hierarchical Accounts	144
9.13 Account Priority	146
10.0 Managing Allocations	147
10.1 Creating Allocations	149
10.2 Querying Allocations	150

10.3 Modifying Allocations	152
10.4 Delete Allocations	153
10.5 Allocation Auto-Generation	154
10.6 Allocation Precedence	155
11.0 Managing Reservations	156
11.1 Creating Reservations	158
11.2 Querying Reservations	159
11.3 Modifying Reservations	161
11.4 Deleting Reservations	162
12.0 Managing Quotes	163
12.1 Creating Quotes	165
12.2 Creating Quote Templates	166
12.3 Querying Quotes	167
12.4 Modifying Quotes	168
12.5 Deleting Quotes	169
13.0 Managing Usage Records	170
13.1 Creating a Usage Record	171
13.2 Querying Usage Records	172
13.3 Modifying a Usage Record	173
13.4 Deleting a Usage Record	174
13.5 Obtaining Usage Quotes	175
13.6 Making Usage Reservations	177
13.7 Charging for Usage	178
13.8 Issuing Usage Refunds	180
13.9 Customizing the Usage Record Object	181
13.10 Usage Record Property Verification	184
13.11 Usage Record Property Defaults	185
13.12 Usage Record Property Auto-Generation	186
13.13 Usage Record Property Instantiators	187
14.0 Managing Itemized Charges	190
14.1 Querying Itemized Charges	191
14.2 Displaying Itemized Charges for a Transaction	193

15.0 Managing Charge Rates	194
15.1 Creating Charge Rates	198
15.2 Querying Charge Rates	202
15.3 Modifying Charge Rates	204
15.4 Deleting Charge Rates	205
16.0 Managing Transactions	206
16.1 Querying Transactions	207
16.2 Customizing the Transaction Object	208
17.0 Managing Events	209
17.1 Querying Events	211
17.2 Deleting Events	213
18.0 Managing Notifications	214
18.1 Querying Notifications	215
18.2 Deleting Notifications	217
19.0 Managing Roles	220
19.1 Creating Roles	221
19.2 Querying Roles	222
19.3 Modifying Roles	224
19.4 Deleting Roles	225
20.0 Managing Passwords	226
20.1 Setting Passwords	227
20.2 Querying Passwords	228
20.3 Deleting Passwords	229
21.0 Using the Gold Shell (goldsh)	230
21.1 Usage	231
21.2 Command Syntax	233
21.3 Valid Objects	235
21.4 Valid Actions for an Object	236
21.5 Valid Predicates for an Object and Action	237
21.6 Common Options	240
21.7 Common Actions Available for most Objects	241
21.7.1 Query Action	241

21.7.2 Create Action	243
21.7.3 Modify Action	243
21.7.4 Delete Action	245
21.7.5 Undelete Action	246
21.8 Multi-Object Queries	249
22.0 Customizing Objects	251
22.1 Managing Objects	252
22.1.1 Creating a Custom Object	253
22.1.2 Querying Objects	254
22.1.3 Modifying an Object	255
22.1.4 Deleting an Object	256
22.1.5 Object Auto-Generation	257
22.1.6 Global Object-Based Defaults	258
22.2 Managing Attributes	259
22.2.1 Adding an Attribute to an Object	261
22.2.2 Querying Attributes	262
22.2.3 Modifying an Attribute	264
22.2.4 Removing an Attribute from an Object	265
22.2.5 Local Attribute-Based Defaults	266
22.3 Managing Actions	267
22.3.1 Adding an Action to an Object	268
22.3.2 Querying Actions	269
22.3.3 Modifying an Action	270
22.3.4 Removing an Action from an Object	271
22.4 Examples Creating Custom Objects	272
23.0 Integration with the Resource Management System	275
23.1 Dynamic versus Delayed Accounting	275
23.1.1 Delayed Accounting	275
23.1.2 Dynamic Accounting	275
23.2 Interaction Points	276
23.2.1 Usage Quote @ Usage Creation Time [Optional – Recommended] ..	276

23.2.2 Usage Reservation @ Usage Start Time [Optional – Highly Recommended]	276
23.2.3 Usage Charge @ Usage End Time [Required]	277
23.3 Methods of interacting with Moab Accounting Manager	278
23.3.1 Configuring an application that already has hooks for Moab Accounting Manager	278
23.3.2 Using the appropriate command-line client	279
23.3.3 Using the interactive control program	279
23.3.4 Use the Perl API	279
23.3.5 Use the Java API	280
23.3.6 Communicating via the SSSRMAP Protocol	281
24.0 Configuration Files	283
24.1 Site Configuration	284
24.2 Server Configuration	285
24.3 Client Configuration	287
24.4 GUI Configuration	290

1.0 Overview

Moab Accounting Manager is an accounting management system that allows for usage tracking and charging for resource or service usage in cloud and technical computing environments. It acts much like a bank in which credits are deposited into accounts with constraints designating which entities may access the account. As resources or services are utilized, accounts are charged and usage recorded. It supports familiar operations such as deposits, withdrawals, transfers, and refunds. It provides balance and usage feedback to users, managers, and system administrators.

Since the accounting and billing models vary widely from organization to organization, Moab Accounting Manager has been designed to be extremely flexible, featuring customizable usage and account definitions, and supporting a variety of tracking, charging and allocation configurations. Attention has been given to scalability, security, and fault tolerance.

1.1 Background

Moab Accounting Manager was originally developed as open source software called the Gold Allocation Manager at Pacific Northwest National Laboratory (PNNL) under the Department of Energy (DOE) Scalable Systems Software (SSS) SciDAC project. It has been extended and enhanced by Adaptive Computing Enterprises, Inc. (formerly Cluster Resources, Inc.) and is in production use at many commercial, government and educational sites.

1.2 Conceptual Overview

Moab Accounting Manager was designed to be used in cloud and technical computing environments for usage tracking, charge accounting and allocation enforcement. Usage tracking involves simply recording resource or service usage in customizable usage records. Charge accounting involves calculating and recording charges for usage for invoicing or cost tracking. Charge accounting may be enabled with the establishment of a bottomless account and the defining of charge rates. Allocation enforcement involves establishing limits on the use of system resources. Allocation enforcement can be enabled by defining separate accounts having limited debit or credit balances.

In this overview, we will assume that you want to track or charge for the usage of some salable or usable item(s). An item may be a resource such as computer cycles used within a job or virtual machine, or it may be a service or something else. The use of an item will result in a usage record. The usage record will track how much and what aspects of the item were used, to whom and what the usage was attributed, and (optionally) how much the usage cost.

With Accounting Manager, it is possible to allocate how much of the resources or services can be used by different entities. This is done by associating a cost for the usage by deciding on a currency unit (referred to generically as credits), whether based on a real currency such as dollars, or a reference currency such as billing units or processor seconds. Next, you will define charge rates in this currency for the components of your usage (resource or service costs, multipliers, fees, etc.).

Pools of funds called allocations may be created via deposits and can be debit or credit based, finite or infinite, and may be limited to a time frame in which they can be used. These allocations are deposited into logical containers called accounts which have constraints that distinguish who or what can use the funds and for what purposes.

A resource or service manager interacts with Moab Accounting Manager to ensure sufficient funds and to track and charge for usage. A typical usage pattern might be as follows. Before you use a resource or service, a quote is obtained to see how much it will cost and to verify that you have sufficient funds and access to the item. If you agree to the quoted price, you can commit your request for the usage. When it is time for you to start using the resource or service, a hold (called a reservation) is placed against your account for the quoted amount (in part or in whole). As you use the item, an appropriate account will be charged and the reservation adjusted. When the final charge is made for actual usage, the remainder of the reservation is removed. A usage record is updated and the transaction and charge history is recorded throughout this process. The actual sequence of interactions is very flexible and will be defined by the architecture between the resource or service manager and the accounting system (Moab Accounting Manager).

To recap: Accounts, which are containers for a reference currency referred to as credits, are differentiated by constraints that define the entities (such as users, projects, machines, classes, organizations, etc.) that can use the credits. Deposits of time-bounded credits are made into accounts creating allocations. Charge rates are created which define how much it will cost to use certain resources or services. Use of a resource or service results in a usage record, and will normally involve a quote detailing the cost of the item before it is used, a reservation against your account while it is being used, and a charge against your account after usage has ended. Other bank-like operations that can be performed on accounts include withdrawals, transfers, refunds, balance checks, statement reports, etc. All modifying actions against accounts or other objects are recorded in the transaction history. The current or past state of any object in Moab Accounting Manager can be queried to produce reports.

1.3 Features

- **Dynamic Charging** – Rather than post-processing resource usage records on a periodic basis to rectify project balances, charging can occur incrementally throughout usage or at usage completion.
- **Reservations** – A hold is placed against the account for the estimated amount of credits before the usage begins, followed by appropriate charges during and/or at the end of the usage, thereby preventing projects from using more resources or services than were allocated to them.
- **Customizable Usage Records** – Usage record fields can be configured by the site to track custom usage properties.
- **Flexible Accounts** – A uniquely flexible account design allows resource or service credits to be allocated to particular entities and purposes.
- **Expiring Allocations** – 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 an allocation cycle.
- **Flexible Charging** – The billing system can track and charge for composite time-based or non-time-based resource or service usage, and apply flexible charge multipliers and fees.
- **Guaranteed Quotes** – Users and resource brokers can determine ahead of time the cost of using resources or services.
- **Credit and Debit Accounts** – Allocations 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.
- **Infinite Allocations** – Deposits can be made with infinite amounts or infinite credit limits when used with a supporting database.
- **Powerful Querying** – a powerful querying and update mechanism (based on SQL queries) that facilitates flexible reporting and streamlines administrative tasks.
- **Nonintrusiveness** – object-level, attribute-level and correlated defaults may be established for arbitrary objects such as users, projects and organizations. Additionally, these objects may be configured to be automatically created the first time they are seen by the resource management system. These features allow the accounting system to be used with less impact and involvement from users and administrators.
- **Consistency** – Moab Accounting Manager has been engineered for robustness, consistency and resiliency. Complex operations are atomic and are automatically rolled back on failure.
- **Security** – multiple security mechanisms for strong authentication and

encryption.

- **Role-Based Authorization** – fine-grained (instance-level) Role Based Access Controls are provided for all operations which allows users to view and manipulate only those objects permitted to them.
- **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, or to function as a persistence interface for other components.
- **Web Interface** – a powerful dynamic web-based GUI is provided for easy remote access for users, managers and administrators which displays only the actions allowed by their role.
- **Journaling** – a journaling mechanism 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.
- **Event Scheduler** – an event engine can be used to schedule arbitrary Gold commands to run periodically or at a designated time in the future.

1.4 Interfaces

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

1.4.1 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 Moab Accounting Manager 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 - Listing Users Using a Command-Line Client

```
glsuser
```

1.4.2 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 2 - Listing Users Using the goldsh Control Program

```
goldsh User Query
```



The `goldsh` control program allows you to make powerful and sweeping modifications to many objects with a single command. Do not use this command unless you understand the syntax and the potential for unintended results.

1.4.3 Web-based Graphical User Interface

A powerful and easy-to-use web-based GUI permits browser access by users, managers and administrators according to their role definitions.

Example 3 - Listing Users via the Web GUI

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

1.4.4 Perl API

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

Example 4 - Listing Users Using the Perl API

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

1.4.5 Java API

Although deprecated, the Java API may still be usable to interact with Moab Accounting Manager. The javadoc command can be run on the contrib/java/gold directory to generate documentation for the Gold java classes.

Example 5 - Listing Users Using the Java API

```
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());
    }
}
}
```

1.4.6 SSSRMAP Wire Protocol

It is also possible to interact with Moab Accounting Manager 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](#).

Example 6 - Listing Users via the SSSRMAP Wire Protocol

```
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>

<Digest-
Value>azu4obZswzBt8-
9OgATukBeLyt6Y=</DigestValue>

<Sig-
nature-
Value>YXE/C08XX3RX4PMU1bWju+5/E5M=</SignatureValue>
  <SecurityToken
type="Symmetric"
name="scottmo"></SecurityToken>
  </Signature>
</Envelope>
0
```

1.5 Documentation

The documentation for Moab Accounting Manager includes this Administrator Guide, release notes, built-in man pages, module documentation and online documentation.

- **Moab Accounting Manager Administrator Guide** – The Moab Accounting Manager Administrator Guide is a comprehensive manual for users and administrators of Moab Accounting Manager and includes information about features, interfaces, installation, getting started, usage, configuration and customization. The Administrator Guide can be found under the `$PREFIX/doc` directory in .pdf and .html formats. These documents are also available online.
- **Release Notes** – The Release Notes describe the primary features and fixes included in the release, along with notes to aid in migration from previous versions and can be found under the doc directory in the distribution tarball.
- **Command Line built-in Man Pages and Usage Synopsis** – All command-line clients support a `--man` option that provides full documentation of the command options and a `--help` option that provides a brief usage synopsis.
- **Module Perl Pod Documentation** – Documentation for Moab Accounting Manager Perl modules can be viewed by changing directory to the `$PREFIX/lib` directory and running `perldoc <modulename>`, e.g. `perldoc Gold::Request`.
- **Online Documentation** – The Moab Accounting Manager Administrator Guide can be found online at <http://www.adaptivecomputing.com/documentation>. The Gold project web page at <http://www.adaptivecomputing.com/resources/docs/gold/files/index.php> and includes the original Gold project documentation.

1.6 License

The Moab Accounting Manager software and associated documentation, data and information include parts which are copyrighted by Adaptive Computing Enterprises, Inc., and parts which are copyrighted by Battelle Memorial Institute. The terms and conditions for the use and redistribution of these parts are governed by the Moab Accounting Manager License and the BSD License respectively. Refer to the LICENSE file for details.

1.6.1 Moab Accounting Manager License

Copyright (C) 2006 - 2012 Pacific Northwest National Laboratory, Battelle Memorial Institute. All rights reserved.

The Moab Accounting Manager License specifies the terms and conditions for use and redistribution.

The Moab Accounting Manager License applies to the Moab Accounting Manager software offered by Adaptive Computing Enterprises, Inc. By installing or using this software, Licensee accepts a non-exclusive license from Adaptive Computing Enterprises, Inc. and is bound to accept acknowledgement of and abide by the notices and conditions of the Moab Accounting Manager License.

1.6.2 BSD License

Copyright (C) 2003 - 2005 Pacific Northwest National Laboratory, Battelle Memorial Institute. All rights reserved.

The BSD license specifies the terms and conditions for use and redistribution.

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

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.0 Installation

If you are performing a fresh installation of Moab Accounting Manager, follow the instructions in this chapter. If you are upgrading an existing version of Moab Accounting Manager to a new maintenance or fix version where there are no database schema changes, follow the instructions contained in [3.0 Upgrading](#). If you are upgrading an existing version of Gold Allocation Manager or Moab Accounting Manager to a new major or minor release where there are database schema changes, follow the instructions in [4.0 Migrating](#).

Moab Accounting Manager uses the standard configure, make, and make install steps for installation. 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.

2.1 Select a Database

Moab Accounting Manager makes use of a database for transactions and data persistence. Three databases have been tested for use with Moab Accounting Manager thus far: PostgreSQL, MySQL, and SQLite. Postgres and MySQL are external databases which run in a distinct (possibly remote) process and communicate over sockets. These databases must be separately installed, configured, and started. SQLite is an embedded database bundled with SQL queries being performed within the gold process itself through library calls. The following information may help you make a choice of databases to use.

- **PostgreSQL** – PostgreSQL is an open source database. PostgreSQL version 7.2 or higher is required. The PostgreSQL database has been thoroughly tested in production with Moab Accounting Manager and all product functionality is available since it was developed using the PostgreSQL database. Postgres supports multiple connections so Moab Accounting Manager is configured to be a forking server when using PostgreSQL.

PostgreSQL is recommended since it is an excellent database, has been more thoroughly tested than the others, and supports all Moab Accounting Manager features.

- **MySQL** – MySQL is an open source database. MySQL version 4.0.6 or higher is required. Prior versions did not support UNION which is used by Moab Accounting Manager in time travel. It is possible to use 4.0 with a minor code tweak to the OFFSET line in Database.pm.

MySQL 4.1 is required in order to have support for the (undocumented) Transaction Undo and Redo functionality since subqueries were not supported until this version.

Infinite Allocations are not supported with MySQL as it does not implement the IEEE Standard 754 for Floating Point Arithmetic.

- **SQLite** – SQLite is an open source embedded database bundled with Moab Accounting Manager. It does not require any configuration and reads and writes from a file. Initial testing has shown Moab Accounting Manager to perform at least as fast as PostgreSQL for small databases.

SQLite 3.2.8 is required in order to be able to customize objects after installation. Previous versions did not support the "ALTER TABLE ADD COLUMN" functionality.

Due to the lack of "ALTER TABLE DROP COLUMN" functionality, migration of Moab Accounting Manager data to newer schema versions cannot be supported. Hence, when upgrading from one major version to another, a fresh database bootstrap is required.

Since SQLite supports only a single connection, Moab Accounting Manager is not configured to be a forking server when using SQLite. This should probably not be an issue for small to medium sized clusters.

Due to a lack of support for multi-column IN clauses, the (undocumented) Transaction Undo and Redo functions are not available.

Infinite Allocations are not supported with SQLite as it does not implement the IEEE Standard 754 for Floating Point Arithmetic.

2.2 Install Prerequisites

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

2.2.1 Perl 5.8 or higher [REQUIRED]

The Moab Accounting Manager server and clients are written in Perl. Perl 5.8 or higher is required. Use 'perl -v' to see what level of Perl is installed.

For RedHat-based systems:

```
[root]# yum install perl
```

2.2.2 Suidperl 5.8 or higher [OPTIONAL]

Command-line clients and Perl API scripts use a security promotion method (gauth or suidperl) to authenticate and encrypt communications with the server. It is recommended that you install and use setuid perl as the security promotion method if it is available for your system. Otherwise, configure will compile and use gauth as the security promotion method. Use 'suidperl -v' to see if suidperl is installed. See the description for the security.promotion configuration parameter in the [Client Configuration](#) section for more information about the two security promotion methods.

For RedHat-based systems:

```
[root]# yum install perl-  
suidperl
```

For SUSE-based systems

```
[root]# chmod 4755  
/usr/bin/sperl*
```

For Debian-based systems:

```
[root]# apt-get install perl-  
suid
```

2.2.3 PostgreSQL database 7.2 or higher (or other tested database) [OPTIONAL]

If you intend to use the PostgreSQL, the MySQL or other external database, you will need to install it. PostgreSQL is recommended since it is an excellent database supporting all necessary features and has been more thoroughly tested than

the others. The only thing needed for SQLite is the sqlite3 client for bootstrapping.

For PostgreSQL on Redhat-based systems:

```
[root]# yum install postgresql
postgresql-libs postgresql-
server
postgresql-devel
```

For PostgreSQL on SUSE-based systems:

```
[root]# zypper install
postgresql postgresql-libs
postgresql-server
postgresql-devel
```

For PostgreSQL on Debian-based systems:

```
[root]# apt-get install
postgresql postgresql-common
postgresql-client
postgresql-server-dev-9.1
```

For MySQL on Redhat-based systems:

```
[root]# yum install mysql
mysql-devel mysql-server
```

For MySQL on SUSE-based systems:

```
[root]# zypper install mysql
mysql-server
```

For MySQL on Debian-based systems:

```
[root]# apt-get install mysql-
common mysql-server
libmysqlclient-dev
```

For SQLite on Debian-based systems:

```
[root]# apt-get install sqlite3
```

2.2.4 libxml2 2.4.25 or higher [REQUIRED]

LibXML2 is needed by the XML::LibXML perl module to communicate via the SSSRMAP message format. The libxml2 development package is needed for the XML::LibXML perl module to install properly.

For RedHat-based systems:

```
[root]# yum install libxml2
libxml2-devel
```

For SUSE-based systems:

```
[root]# zypper install libxml2
libxml2-devel
```

For Debian-based systems:

```
[root]# apt-get install libxml2
libxml2-dev
```

2.2.5 gnu readline 2.0 or higher [OPTIONAL]

The interactive control program (goldsh) can support command-line-editing capabilities if readline support is enabled.

For RedHat-based systems:

```
[root]# yum install ncurses-
devel readline-devel
```

For SUSE-based systems:

```
[root]# zypper install ncurses-
devel readline-devel
```

For Debian-based systems:

```
[root]# apt-get install
ncurses-dev libreadline.dev
```

2.2.6 Apache Httpd Server 2.0 or higher with mod_ssl [OPTIONAL]

Moab Accounting Manager provides a web-based GUI so that managers, users, and administrators can interact with the accounting and allocation system. The web interface utilizes Perl CGI and SSL and needs to have an httpd server (preferably apache) installed. mod_ssl is also needed and is often bundled as part of the apache2 server.

For RedHat-based systems:

```
[root]# yum install httpd mod_
ssl
```

For SUSE-based systems:

```
[root]# zypper install apache2
```

For Debian-based systems:

```
[root]# apt-get install apache2
```

2.2.7 OpenSSL 0.9.5a or higher [REQUIRED]

OpenSSL is used to encode the secret key and is used in the web interface to encrypt communications with the server.

For RedHat-based systems:

```
[root]# yum install openssl
```

For Debian-based systems:

```
[root]# apt-get install openssl
```

2.2.8 Disable SELinux

In some distributions (e.g. redHat-based systems), Security-Enhanced Linus (SELinux) blocks the use of `setuid perl` (used in client authentication). If you are using `setuid perl` as the security promotion method (this is the default if available), you will need to disable SELinux.

For RedHat-based systems:

```
[root]# vi
/etc/sysconfig/selinux
SELINUX=disabled

[root]# setenforce 0
```

2.3 Preparation

To build and install Moab Accounting Manager, you first need to unpack the tar archive and change directory into the top directory of the distribution. For maximum security, it is recommended that you install and run Moab Accounting Manager under its own non-root userid. This user will be referred to as the accounting admin user.

```
[root]# useradd -m moab
[root]# passwd moab
[moab]$ mkdir ~/src
[moab]$ cd ~/src
[moab]$ tar -zxvf mam-
7.1.tar.gz
[moab]$ cd mam-7.1
```

2.4 Configuration

To configure Moab Accounting Manager, run the "configure" script provided with the distribution.

The following is a list of configure options:

- `-h, --help` display the list of options
Run `./configure --help` to see the list of configure options.
- `--prefix=PREFIX` install architecture-independent files in PREFIX [/`opt/mam`]
Base installation directory where all subdirectories will be installed unless otherwise designated (defaults to `/opt/mam`).
- `--exec-prefix=EPREFIX` install architecture-dependent files in EPREFIX [PREFIX]
Directory where architecture-depended subdirectories (such as `bin`, `sbin`, `lib`) will be installed (defaults to PREFIX).
- `--bindir=DIR` user executables [EPREFIX/`bin`]
Client scripts will be installed to this subdirectory (defaults to EPREFIX/`bin`).
- `--sbindir=DIR` system admin executables [EPREFIX/`sbin`]
System scripts and binaries (including `gold`, `goldsh`, `gauth`) will be installed to this subdirectory (defaults to EPREFIX/`sbin`).
- `--libdir=DIR` object code libraries [EPREFIX/`lib`]
Gold Perl modules will be installed in this subdirectory (defaults to [EPREFIX/`lib`]).
- `--localstatedir=DIR` modifiable single-machine data [PREFIX]
Home directory where per-configuration subdirectories (such as `etc`, `log`, `data`) will be installed (defaults to PREFIX).
- `--sysconfdir=DIR` read-only single-machine data [LOCALSTATEDIR/`etc`]
Subdirectory where configuration and stateful files reside (defaults to LOCALSTATEDIR/`etc`).
- `--datarootdir=DIR` read-only arch.-independent data root [PREFIX/`share`]
Directory where documentation subdirectories (such as `doc`, `man`) reside (defaults to PREFIX/`share`)

- `--docdir=DIR` documentation root [DATAROOTDIR/doc/mam]
Directory where application documentation (pdf, html) resides (defaults to DATAROOTDIR/doc/mam).
- `--with-db-name=NAME` database name [mam]
Name of the SQL database that the server will sync with (defaults to mam).
- `--with-db-type=DATABASE` database type { Pg, mysql, SQLite } [Pg]
Use `with-db-type` to specify the database server type you intend to use with Moab Accounting Manager. Currently only PostgreSQL (Pg), MySQL (mysql) and SQLite (SQLite) have been tested for use with Moab Accounting Manager. Postgres and MYSQL are external databases which runs in a distinct (possibly remote) process and communicates over sockets while SQLite is an embedded database with SQL queries being performed within the gold process itself through library calls. Initial testing has shown SQLite to be at least as fast as PostgreSQL for small installations. The default is to use PostgreSQL.
- `--with-user=USER` accounting admin user id under which the server will run
Use `--with-user` to specify the accounting admin userid that the server will run under and who will have full administrative privileges (defaults to the user running the configure command). It is recommended that this be a non-privileged user for the highest security.
- `--with-promotion=gauth|suidperl` method used to promote privileges to read site.conf
Command-line clients and scripts using the API need to use a security promotion method to authenticate and encrypt the communication using the symmetric key. The default is `suidperl` if it is installed on the system, otherwise the default is `gauth`. See the description for the `security.promotion` configuration parameter in the [Client Configuration](#) section for more information about the two security promotion methods.
- `--with-gold-libs=local|site` install policy for Gold perl libs [local]
Use `with-gold-libs` to indicate whether you want to install the Gold modules in a local gold directory (`${exec_prefix}/lib`) or in the default system site-perl directory (defaults to local).
- `--with-cgi-bin=DIR` directory to install cgi-bin files if using web gui [/var/www/cgi-bin/mam]
If you intend to use the web GUI, use `with-cgi-bin` to specify the directory where you want the Moab Accounting Manager CGI files to reside (defaults to

/var/www/cgi-bin/mam).

- `--with-context=CONTEXT` specifies the accounting context (hpc or cloud) [hpc]
By specifying the accounting context (either hpc or cloud) some client commands can be adjusted to show the proper fields for that context. The default is hpc.
- `--with-skin=SKIN` specifies the skin for the web-based GUI (viewpoint or legacy) [viewpoint]
If you intend to use the web GUI, you can specify whether you want to use a skin design more compatible with Viewpoint (if you intend to access the web GUI from within Moab Viewpoint) or whether you want to use the legacy GUI styles (normally used standalone). The default is viewpoint.

To assume the defaults, just run `configure`.

```
[moab]$ ./configure
```

2.5 Compilation

To compile the program, type `make`.

```
[moab]$ make
```

If you would like to install the web GUI, type `make gui`.

```
[moab]$ make gui
```

2.6 Perl Module Dependencies

Moab Accounting Manager requires the use of a number of Perl modules. These modules can be installed from CPAN by typing 'make deps'.

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



After running make deps initially, it is useful to run it again to see if all of the dependencies were installed cleanly. If not, you will need to intercede in the dependency installation. You can verify that this step is complete when make deps shows all modules as being up to date.



On CentOS 5, any CPAN errors you may encounter can be overcome by running:

```
perl -MCPAN -e force install  
Log::Dispatch::File::Rotate
```

This step should install the following Perl modules from CPAN:

- CGI::Session
- Compress::Zlib
- Config::Tiny
- Crypt::CBC
- Crypt::DES
- Crypt::DES_EDE3
- Date::Manip
- DBI
- DBD::Pg, DBD::MySQL or DBD::SQLite
- Digest::HMAC
- Digest::SHA1
- Error
- Log::Dispatch
- Log::Dispatch::FileRotate
- Log::Log4perl
- Module::Build
- Module::Implementation
- Params::Validate
- SOAP
- Term::ReadLine::Gnu
- XML::SAX
- XML::LibXML::Common
- XML::LibXML
- XML::NamespaceSupport

If you would prefer to do so, you can install these modules via other sources, such as from rpm, or manually from CPAN.

2.7 Installation

Use `make install` to install Moab Accounting Manager. You may need to do this as root if any of the installation or log directories do not already have write permission as the accounting admin user.

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

If you would like to install the web GUI, type `make install-gui` (as root).

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

To delete the files created by the product installation, you can use `make uninstall`.

2.8 Database Setup

If you have chosen to use PostgreSQL or MySQL, you will need to define a database user, create the Moab Accounting Manager database, and configure the database server to support transactions and connections from the server host. No setup is needed if you are using SQLite.

2.8.1 Initialize the database

For PostgreSQL, the database must be initialized before it can be configured.

```
[root]# service postgresql
initdb
```



On some operating systems, this is achieved by starting the database service.

```
[root]# service postgresql
start
```

2.8.2 Configure trusted connections

For PostgreSQL, add the host-based client authentication as appropriate. Edit or add a line in the `pg_hba.conf` file for the interface from which the Moab Accounting Manager server will be connecting to the database.

```
[postgres]$ vi
/var/lib/pgsql/data/pg_hba.conf
host all all 127.0.0.1/32 md5
host all all ::1/128 md5
```

For PostgreSQL, configure postgres to accept connections from your host.

```
[postgres]$ vi
/var/lib/pgsql/data/postgresql.conf
listen_addresses = 'localhost'
# what IP address(es) to listen
on;
```

2.8.3 Enable support for transactions

If you are using the MySQL database you will need to configure the server to support transactions (MySQL 5.5.5 and later supports transactions by default).

```
[root]$ vi /etc/my.cnf
default-storage-engine = INNODB
# Place under the [mysqld]
section
```

2.8.4 Start the database

Start (or restart) the database server with the new configurations in effect.

For PostgreSQL database:

```
[root]# service postgresql
restart
```

For MySQL database:

```
[root]# service mysqld start
```

2.8.5 Create the database

Create the Moab Accounting Manager database and add the accounting admin user as a database administrator. This must be performed as the database user (postgres or mysql).

For PostgreSQL database:

```
[postgres]$ psql

create database mam;
create user moab with password
'changeme';
```

For MySQL database:

```
[root]# mysql

create database mam;
grant all on *.* to
'moab'@'localhost' identified
by 'changeme';
exit
```

2.8.6 Bootstrap

You will need to populate the Moab Accounting Manager database with an SQL dump that defines the objects, actions, and attributes necessary to function as an

Accounting and Allocation Manager. Use cloud.sql if you are in a cloud context or hpc.sql if you are in an HPC context.

For PostgreSQL database:

```
[moab]$ psql mam < hpc.sql # or  
cloud.sql
```

For MySQL database:

```
[moab]$ mysql mam < hpc.sql #  
or cloud.sql
```

For SQLite database:

```
[moab]$ sqlite3  
/opt/mam/data/mam.db < hpc.sql  
# or cloud.sql
```


2.9 General Setup

Edit the configuration files as necessary.

```
[moab]$ vi
/opt/mam/etc/goldd.conf
database.password = changeme
[moab]$ vi
/opt/mam/etc/gold.conf
```

Edit your startup files as appropriate to configure the PATH. Then add the PATH in your current environment.

```
[root]$ cp etc/profile.d/*sh
/etc/profile.d
[moab]$ . /etc/profile.d/mam.sh
```

2.10 Startup

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

```
[moab]$ goldd
```

Alternatively, if you are on a Linux system that supports init.d scripts, you can create a system startup service for Moab Accounting Manager. Sample scripts are provided in contrib/init.d/ that can be edited for your distribution and installed into /etc/init.d. After adding execute permissions, Moab Accounting Manager can then be started by issuing:

```
[root]# service mam start
```

2.11 Web Server Setup

If you want to use the web GUI, you will need to configure your Apache HTTP server to use SSL. The following shows some sample steps to configure the web GUI. The actual steps you will need to use will vary according to your distribution and environment. The web server configuration must be modified to support the invocation of cgi-bin scripts over an SSL connection using a private key and a signed certificate.

Configure apache to use SSL

Edit the apache configuration files to use SSL, CGI and to define aliases.

For SUSE-based systems:

```
[root]# vi
/etc/sysconfig/apache2

APACHE_SERVER_FLAGS="-DSSL"
```

Configure the SSL virtual host definition.

For RedHat-based systems:

```
[root]# vi
/etc/httpd/conf.d/ssl.conf
```

For SUSE-based systems:

```
[root]# cp
/etc/apache2/vhosts.d/vhost-
ssl.te-
mplate/etc/apache2/vhosts.d/mam-
m-ssl.conf
[root]# vi
/etc/apache2/vhosts.d/mam-
ssl.conf
```

Add or edit the SSL virtual host definition as appropriate for your environment:

```
<VirtualHost _default_:443>
...

# Configure your cgi-bin
directory
<Directory "/var/www/cgi-
bin">
    Options ExecCGI
```

```

    AddHandler cgi-script .cgi
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>

# Create an alias for /cgi-
bin pointing to your cgi-bin
directory
# If you chose to install to
a cgi-bin subdir, you may want
to create
# an alias for that as well.
Alias /cgi-bin/ /var/www/cgi-
bin/
Alias /mam/ /var/www/cgi-
bin/mam
# Add index.cgi to the
DirectoryIndex so you can use
the shorter dir name
DirectoryIndex index.cgi

...
</VirtualHost>

```

Install a Signed Certificate

For the highest security, it is recommended that you install a public key certificate that has been signed by a certificate authority. The exact steps to do this will be specific to your distribution and the chosen certificate authority. An overview of this process for CentOS is documented at http://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-httpd-secure-server.html.

Alternatively, if your network domain can be secured from man-in-the-middle attacks, you could use a self-signed certificate. Often this does not require any additional steps since in many distributions (e.g., RedHat-based), the Apache SSL configuration provides self-signed certificates by default.

The following steps assume you are using self-signed certificates:

Create self-signed SSL certificate and key files. Some distributions (e.g. RedHat) ship with ready-made certificates.

For SUSE-based systems:

```
[root]# cd /etc/apache2
```

```
[root]# openssl genrsa -out
ssl.key/server.key 1024
[root]# openssl req -new -key
ssl.key/server.key -x509 -out
ssl.crt/server.crt
```

Restart the HTTP Server

Startup or restart the Apache HTTP server.

For RedHat-based systems:

```
[root]# service httpd restart
```

For SUSE-based or Debian-based systems:

```
[root]# service apache2 restart
```

2.12 Accessing the GUI



In order to use the Web GUI, users will have to generate passwords for themselves using the **gchpasswd** client command. Moab Accounting Manager may have to be restarted in order for role privileges to be reflected in the GUI.

```
[moab]# gchpasswd
```

To access the web gui, open a browser with URL: <https://localhost/mam>.

```
[moab]$ firefox  
https://localhost/mam
```

2.13 Initialization

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

You can run one of the initialization scripts in the test directory to set up a sample initial environment (with some dummy users, accounts, charge rates, etc.) for your desired accounting mode and context.

For HPC allocation enforcement:

```
[moab]$ test/hpc-allocation-  
enforcement.sh
```

For Cloud allocation enforcement:

```
[moab]$ test/cloud-allocation-  
enforcement.sh
```

3.0 Upgrading

Moab Accounting Manager uses the standard configure, make and make install steps for upgrades. This document assumes that you are updating Moab Accounting Manager to a new maintenance or fix level within the same major and minor release. Instructions for migrating to a new minor or major release can be found in the [Migrating chapter](#). This document 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.

3.1 Preparation

To build and update Moab Accounting Manager, you first need to unpack the tar archive and change directory into the top directory of the distribution.

```
[moab]$ cd ~/src
[moab]$ tar -zxvf mam-
7.1.0.tar.gz
[moab]$ cd mam-7.1.0
```

3.2 Configuration

To configure Moab Accounting Manager, run the "configure" script provided with the distribution with the desired options.

```
[moab]$ ./configure
```

3.3 Compilation

To compile the program, type `make`.

```
[moab]$ make
```

If you would like to install the web GUI, type `make gui`.

```
[moab]$ make gui
```

3.4 Server Shutdown

Stop the server daemon.

```
[moab]$ goldd -k
```

3.5 Installation

Use `make install` to install Moab Accounting Manager. You may need to do this as root if any of the installation or log directories do not already have write permission as the accounting admin user (moab).

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

If you would like to install the web GUI, type `make install-gui` (as root).

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

3.6 Server Startup

Start the server daemon back up.

```
[moab]$ goldd
```

4.0 Migrating

This procedure assumes that you are updating Moab Accounting Manager to a new major or minor release where there are database schema changes. Instructions for upgrading to a new maintenance or fix release where there are no database schema changes can be found in [3.0 Upgrading](#).

The instructions in this chapter demonstrate the process of migrating a database schema from Moab Accounting Manager 7.0 to 7.1 or from Gold Allocation Manager 2.2 to Moab Accounting Manager 7.1. By default, this version will unpack into a separate directory (*/opt/mam*) which, after modifying the database and port of the prior installation, will allow us to run or access both versions simultaneously. You can determine your current database version by running 'goldsh System Query'.



If you are migrating from a version of Gold Allocation manager prior to version 2.2, you will first need to perform a migration install to Gold Allocation Manager 2.2 using the Gold Allocation Manager tarball and the associated migration script and procedures.



SQLite databases cannot yet be migrated in most circumstances since current versions do not support the `ALTER TABLE ADD COLUMN` syntax.

This chapter 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. These steps are very similar to the steps for performing a maintenance or fix upgrade install with the exception that the migration scripts should be run after restarting the server after the make install.

4.1 Database Backup

Quiesce the former Moab Accounting Manager server daemon.

```
[adaptive]$  
/opt/gold/sbin/goldd -k
```

Back up your old database to a file.

- Migrating from Gold Allocation Manager 2.2

- For a PostgreSQL database:

```
[adaptive]$ pg_dump gold >  
~/gold-2.2.sql
```

- For a MySQL database:

```
[adaptive]$ mysqldump gold  
> ~/gold-2.2.sql
```

- Migrating from Moab Accounting Manager 7.0

- For a PostgreSQL database:

```
[adaptive]$ pg_dump gold >  
~/mam-7.0.sql
```

- For a MySQL database:

```
[adaptive]$ mysqldump gold  
> ~/mam-7.0.sql
```


4.2 Create Database

Create a new database so that you can preserve and reference the old data separately. Use the instructions that match your current version of Gold or MAM and your database type.

- Migrating from Gold Allocation Manager 2.2
 - For a PostgreSQL database:

```
[adaptive]$ psql
create database mam;
[adaptive]$ psql mam <
~/gold-2.2.sql
```

- For a MySQL database:

```
[adaptive]$ mysql
create database mam;
[adaptive]$ mysql mam <
~/gold-2.2.sql
```

- Migrating from Moab Accounting Manager 7.0
 - For a PostgreSQL database:

```
[adaptive]$ psql
create database mam;
[adaptive]$ psql mam <
~/mam-7.0.sql
```

- For a MySQL database:

```
[adaptive]$ mysql
create database mam;
[adaptive]$ mysql mam <
~/mam-7.0.sql
```

4.3 Preparation

To build and update Moab Accounting Manager, unpack the tar archive and change directory into the top directory of the distribution.

```
[adaptive]$ cd ~/src
[adaptive]$ tar -zxvf mam-
7.1.0.tar.gz
[adaptive]$ cd mam-7.1.0
```

4.4 Configuration

To configure MAM, run the `configure` script provided with the distribution with the desired options.

```
[adaptive]$ ./configure
```

4.5 Compilation

To compile the program, type `make`:

```
[adaptive]$ make
```

If you would like to install the web GUI, type `make gui`:

```
[adaptive]$ make gui
```

4.6 Installation

Use `make install` to install Moab Accounting Manager. You will need to do this as root.

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

If you would like to install the web GUI, type `make install-gui` as root.

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

You may need to rerun `make deps` if you are installing to a new directory and you chose to install the former perl libs in a local directory.

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

4.7 General Setup

Edit the configuration files as necessary for your new installation. You will very likely want to merge your previous configuration settings into your new configuration files.

```
[adaptive]$ vi
/opt/mam/etc/goldd.conf
[adaptive]$ vi
/opt/mam/etc/gold.conf
```

Configure your PATH environment variable to find the newly installed directories.

```
[adaptive]$ export
PATH=/opt/mam/bin:/opt/mam/sbin-
:$PATH
```

4.8 Server Startup

Start up the new server daemon.

```
[adaptive]$ goldd
```

4.9 Database Migration

Now you will need to migrate your database to the new schema. This script is designed to be rereunnable, so if you encounter a failure, resolve the failure and rerun the migration script. If you are unable to resolve the issue and complete the migration, contact support.

- Migrating from Gold Allocation Manager 2.2

```
[adaptive]$ sbin/migrate_2.2_  
to_7.1.pl
```

Edit the prior configuration files to change the port. This will allow you to run and access both versions simultaneously.

```
[adaptive]$ vi  
/opt/gold/etc/goldd.conf  
server.port = 7111
```

- Migrating from Moab Accounting Manager 7.0

```
[adaptive]$ sbin/migrate_7.0_  
to_7.1.pl
```

Edit the prior configuration files to change the port. This will allow you to run and access both versions simultaneously.

```
[adaptive]$ vi  
/opt/gold/etc/goldd.conf  
server.port = 7111
```


5.0 Getting Started

Moab Accounting Manager can be configured in a myriad of use cases. It can be used in different contexts such as cloud or High Performance Computing (HPC). It can be used in different accounting modes such as for usage tracking, charge accounting or allocation enforcement. This chapter will outline a few basic examples of setting up Moab Accounting Manager for use in a High Performance Computing environment to track and charge projects for job resource usage.

If you want to use Moab Accounting Manager solely for recording resource usage but not for charging, then review the section on HPC Usage Tracking. If you want to calculate and record charges, but not restrict any workload from being serviced, then review the section on HPC Charge Accounting. If you want charge and establish limits on the use of system resources, then review the section on HPC Allocation Enforcement.



You will need to be an Moab Accounting Manager System Administrator to perform the tasks in this chapter. It is assumed that you have already installed and bootstrapped Moab Accounting Manager and started the server before performing the steps suggested in this chapter.

5.1 HPC Usage Tracking

When used solely for usage tracking, Moab Accounting Manager logs resource usage in usage records. This usage can be queried to report what resources were used when and by whom. In this case, there is no need for charge rates, accounts, allocations, reservations or quotes. There is no need to define project membership. The only real consideration is whether you want to customize the usage record to display usage properties unique to your site.

5.1.1 Usage Record Customization (Optional)

As an example, we will add a usage record property to track GPU usage. See the section on [Customizing the Usage Record Object](#) for additional examples.

Example 1 - Adding a GPU Field to the Usage Record

```
$ goldsh Attribute Create
Object=UsageRecord Name=GPUs
DataType=Integer
Successfully created 1
attribute
```

Example 2 - Selecting the Usage Record fields we would like to see via glsusage

We can select the usage records fields that show up in glsusage by editing the usagerecord.show attribute in the client configuration file (gold.conf). This is the same parameter that would have to be edited for the new GPU attribute to show up in glsusage. The web GUI will automatically display the new attribute.

```
$ vi /var/gold/etc/gold.conf
usagerecord.show = Id,Instance,
User,Project,Machine,Stage,
Processors,GPUs,Nodes,Duration,
StartTime,EndTime
```

5.1.2 Record the Usage

After a job completes, the usage is recorded (see [Creating a Usage Record](#)). This step is normally performed automatically by Moab Accounting Manager via the NAMI API but we can use the command line interface for the purpose of illustration.

Example 3 - Record resource usage for our job

```
$ gmkusage -J job1 -u amy -p  
chemistry -m colony -P 16 -X  
GPUs=8 -N 4 -t 720  
Successfully created 1 usage  
record with id 1
```

5.1.3 List Usage Records

Let's examine the usage record that was created (see [Querying Usage Records](#)).

Example 4 - List Usage Records

```
$ glsusage
Id  Type  Instance  User
Project  Machine  Processors
GPUs  Nodes  Duration  EndTime
---  ---  -
1    Job    job1      amy
chemistry  colony  16
8    4      720
```

5.2 HPC Charge Accounting

Some sites may want to use Moab Accounting Manager to calculate and record charges, but not to restrict or prevent any workload from being serviced. In this case, we need only define a single account with bottomless funds. Moab Accounting Manager will ascribe a charge for resource utilization and attribute it to the entities using it. Reservations, balance queries and quotes are not needed. The main task is to define charge rates.

5.2.1 Usage Record Customization (Optional)

It may be desirable to customize the usage record to display usage properties unique to your site. See the section on [Customizing the Usage Record Object](#) for examples.

5.2.2 Decide on a Currency and Set the Currency Precision

Since we will be calculating charges, we will need to decide on a currency unit and set the currency precision. For this example we will define a currency in which one credit represents the value of using one processor core for one second. We will assume for simplicity that one processor second on one machine will have the same value as a processor second on another machine. Charges for other resource types will be given an appropriate value relative to this currency unit. All allocations and charges will be specified in terms of this currency. The only action to take here would be to set the currency precision to be the number of decimal places you want Moab Accounting Manager to display when reporting currency amounts. Since processor seconds can easily be represented as an integer with no decimal places and the default currency precision is zero, there is no action to take here. If instead we were to have chosen dollars as the currency base, we would want to set the `currency.precision` value in `goldd.conf`, `gold.conf` and `goldg.conf` to 2.

5.2.3 Define Charge Rates

Since we are charging for usage, we must establish the charge rates for the usage. In our example, we will establish a charging scheme that charges 1 credit for each processor second utilized by the job as well as 1 credit for every GigaByte of memory utilized by the job per second.

We will add Processors and Memory as consumable resource charge rates (with a Type of Value Based Resource) so their values will be multiplied by the amount of time they are used. We will define a processor charge rate of 1 currency unit that will charge one credit per processor second used and we will set the memory charge rate to be .001 since we will assume that the memory will be reported in MegaBytes and we want to charge 1 currency unit for every GigaByte second of memory used. See the chapter on [Managing Charge Rates](#) for more detailed information on setting up charge rates.

Example 5 - Define Charge Rates for Processors and Memory

```
$ gmkrate -n Processors -T VBR
-z 1
Successfully created 1 charge
rate
```

```
$ gmkrate -T VBR -n Memory -z
.001
Successfully created 1 charge
rate
```

```
$ glsrate
Name      Value  Type  Rate
Description
-----  -----
Memory                VBR   0.001
Processors            VBR    1
```

5.2.4 Create a Single NonLimiting Account

Since we do not want to limit usage in any way, it is probably not necessary to create individual accounts. It may be sufficient to create a single unconstrained account with unlimited credits. This section will demonstrate this approach. Usage charges associated with various projects, users, machines, etc. can be extracted with usage record queries by applying appropriate filters. If you do wish to track usage via separate distinct accounts (which will additionally allow you to produce separate account statements), you may want to follow the steps outlined in the HPC Allocation Enforcement section with the exception that you will make very large or infinite deposits into the accounts. See the chapter on [Managing Accounts](#) for more detailed information on setting up accounts.

Example 6 - Create a single unconstrained account

```
$ gmkaccount -n "Common  
Account"  
Successfully created 1 account  
with id 1
```

```
$ glsaccount  
Id Name          Amount  
Constraints Description  
-----  
-----  
1  Common Account      0
```

5.2.5 Create an Unlimited Allocation

Since we do not wish to limit usage, we need to create a large or an unlimited allocation. We can do this by depositing infinite credits or by establishing an infinite credit limit (which will allow the account to have an unlimited negative balance). See the section on [Making Deposits](#) for additional information.

⚠ the use of infinite allocations requires the use of a database that supports the IEEE Standard 754 for Floating-Point Arithmetic (e.g. PostgreSQL). If you are not using a supporting database type, you can deposit a very large amount (e.g. 1000000000) instead.

Example 7 - Creating a single unlimited allocation via an infinite deposit

```
$ gdedeposit -z Infinity
Successfully deposited inf
credits into account 1
```

Let's examine the allocated we just created.

```
$ glsalloc
Id Account Active StartTime
EndTime Amount CreditLimit
Deposited Description
-- -----
-----
1 1 True -Infinity
Infinity Infinity 0
Infinity
```

```
$ glsaccount
Id Name Amount
Constraints Description
-- -----
1 Common Account Infinity
```

Since the account has infinite funds, it will not be necessary to check the balance regularly because it is not going to change, but let's look at it to see how we have it set up.

```
$ gbalance
Id Name Available
Allocated PercentUsed
-- -----
```

```
-----  
1 Common Account Infinity  
Infinity 0.00
```

5.2.6 Issue a Refund

Since this was an imaginary job, refund the account (see [Issuing Usage Refunds](#)).

Example 10 - Issue a refund for our job


```
$ grefund -J job1
Successfully refunded 12960
credits to usage record 1
instance job1
```

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

```
$ glsusage -u amy --show
Instance,Charge,User,Project,
Machine,Processors,Memory,
Duration
Instance Charge  User Project
Machine Processors Memory
Duration
-----
-----
-----
-----
job1          0  amy  chemistry
colony  16          2000  720
```

5.2.7 Examine Account Statement

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

 If you want to be able to issue separate account statements for different projects, users, etc. then you will need to establish separate accounts by following the initial steps outlined in the HPC Allocation Enforcement section with the exception that you will make very large or infinite deposits into the accounts.

Example 11 - We can request an itemized account statement to see the debits and credits for the common account

```
$ gstatement
++++++ ++++++ ++++++ ++++++
++++++ ++++++ ++++++ ++++++
++++++ ++++++ ++++++ ++++++
++++++ +
+
+ Includes account 1 (Common
Account)
+ Generated on Thu Dec 22
18:26:55 2011.
+ Reporting account activity
from -Infinity to Now.
++++++ ++++++ ++++++ ++++++
++++++ ++++++ ++++++ ++++++
++++++ ++++++ ++++++ ++++++
++++++ +

Beginning Balance:
      0
-----
-----
Total Credits:
Infinity
Total Debits:
-12960
-----
-----
Ending Balance:
Infinity
```

```

++++++ ++++++ ++++++ ++++++
++++++ Credit Detail ++ ++++++
++++++ ++++++ ++++++ ++++++ +
Object      Action      Instance
Amount      Time
-----
-----
Account      Deposit
Infinity 2011-12-22 17:50:24
UsageRecord Refund      job1
      12960 2011-12-22 18:13:30
++++++ ++++++ ++++++ ++++++
++++++ + Debit Detail +++
++++++ ++++++ ++++++ ++++++
++++++ +

```

5.3 HPC Allocation Enforcement

With Moab Accounting Manager, one can establish limits on the use of system resources. Rates are established for the use of resources and resource credits can be apportioned to different parties or purposes. Some sites establish an allocation cycle where proposals for resource usage are periodically reviewed and suitable candidates are granted an allocation on the computing system. Other sites limit consumers to what they "pay" for. In either case, multiple accounts are needed; with rosters, allocation limits, balance and usage feedback, reservations, and possibly quotes.

5.3.1 Usage Record Customization (Optional)

It may be desirable to customize the usage record to display usage properties unique to your site. See the section on [Customizing the Usage Record Object](#) for examples.

5.3.2 Decide on a Currency and Set the Currency Precision

Since we will be calculating charges, we will need to decide on a currency unit and set the currency precision. For this example we will define the currency to be in dollars (and cents). Deposits into accounts will be made in this currency. Resource charges will be calculated from charge rates based on this currency. Since dollars and cents are represented as a floating point number with two decimal places we must specify a currency precision of two.

Example 12 - Setting the currency precision to two

The currency precision value must be set in the server and client configuration files (`goldd.conf` and `gold.conf`). It must also be set in the GUI configuration file (`goldg.conf`) if you will be using the web GUI.

```
$ vi /var/gold/etc/goldd.conf
currency.precision = 2

$ vi /var/gold/etc/gold.conf
currency.precision = 2
```

5.3.3 Define Charge Rates

Since we are charging, we must establish the charge rates for the usage. In our example, we will establish a charging scheme that charges 1 dollar for each processor hour utilized by the job.

Since we want to charge 1 dollar per hour of usage per processor and because time-based charge rates are multiplied by the duration in seconds, we need create a charge rate for processors that charges 1/3600th of a dollar per second. See the chapter on [Managing Charge Rates](#) for more detailed information on setting up charge rates.

Example 13 - Modify the Processors Charge Rate

```
$ gchrate -n Processors -z
.00027778 -T VBR -d "1 dollar
per processor-hour
Successfully modified 1 charge
rate

$ glsrate
Name          Value Type Rate
Description
-----
Processors          VBR
0.00027778 1 dollar per
processor-hour
```

5.3.4 Define Accountable Entities

Next we must decide to which entities we want to entitle our allocations. In this example, we will distribute the funds among different projects. Each project will be assigned a set of user members that can charge to that project. Moab Accounting Manager can be customized to associate funds with any number of arbitrary entities such as Users, Groups, Projects, Organizations, Classes, Machines, etc. We will start by defining some projects and the associated user members of the projects. We will also associate each project with an organization so that usage reports can be generated for the organization level as well as the project and user level.

We will create projects for biology, chemistry and film and assign them some users. The biology and chemistry project will be associated with the sciences organization while the film project will be associated with the arts organization. See the chapter on [Managing Projects](#) for more information on setting up projects.

Example 14 - Define the biology, chemistry and film projects

```
$ gmkproject -p biology -o
sciences -u amy, bob -d
"Biology Department"
Successfully created 1 project

$ gmkproject -p chemistry -o
sciences -u amy, dave -d
"Chemistry Department"
Successfully created 1 project

$ gmkproject -p film -o arts -u
bob, dave -d "Film Department"
Successfully created 1 project

$ glsproject
Successfully created 1 project

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

```
sciences      Chemistry  
Department
```

```
film          True    bob, dave  
arts          Film Department
```

5.3.5 Create Accounts

The next task will be to create the accounts which will hold the allocated credits. An account is much like a numbered bank account, where credits can be deposited and are defined by constraints that distinguish who or what can use the funds and for what purposes. In this example, we will create an account for each of the three projects. Had we enabled account auto-generation or used the `--create-account=True` option with the `gmkproject` command, an account would have been created automatically with the creation of each project. See [Managing Accounts](#) for more detailed information on setting up accounts.

Note that in most cases, referenced objects will be auto-generated, such as the users that were auto-generated as they were added to the projects. Undefined projects, would have likewise been auto-generated as they were newly associated with accounts, but had we taken that ordering, we would still have needed to go back and associate the appropriate users, organization, etc. to these projects. We could have also created the users explicitly and provided additional detail about each user, before adding them to the projects. Of course, even though they were auto-generated, we can still go back and add detailed information to each user as desired. See [Managing Users](#) for more detailed information on setting up users.

Example 15. Create four project-based accounts

```
$ gmkaccount -p biology -n
"biology"
Successfully created 1 account
with id 1 and 1 constraint

$ gmkaccount -p chemistry -n
"chemistry"
Successfully created 1 account
with id 2 and 1 constraints

$ gmkaccount -p film -n "film"
Successfully created 1 account
with id 3 and 1 constraints

$ glsaccount
ID Name Amount
  Constraints
Description
--  -----  -----
  -----
-----
```

```
1  biology          0
   Project=biology

2  chemistry        0
   Project=chemistry

3  film             0
   Project=film
```

5.3.6 Make Deposits

Now we need to allocate funds to these accounts by making deposits to them. An allocation has a start and end time associated with it declaring the time frame in which it can be used (defaulting to negative and positive infinity). It can also have a credit limit which defines the extent to which the allocation is allowed to go negative. Multiple allocations (usually with different expenditure time frames) can be associated with an account. Judicial use of allocation time frames can be helpful to establish an allocation cycle and set expectations for credit expenditure. See [Making Deposits](#) for additional information.

In this example, we will allocate 5000 and 3000 dollars to the biology and chemistry projects respectively. The film project will be given a credit limit of 2000 dollars which allows them to charge up to 2000 dollars before rectifying their account. When making a deposit we must specify the account we are depositing into unless the account can be unambiguously determined by its constraint references (i.e. there is only a single account associated with the project biology). We will create allocations that must be used within the current year.

Example 16. Making Deposits

```
$ gdeposit -s 2012-01-01 -e
2013-01-01 -z 5000 -p biology
Successfully deposited 5000.00
credits into account 1

$ gdeposit -s 2012-01-01 -e
2013-01-01 -z 3000 -p chemistry
Successfully deposited 3000.00
credits into account 2

$ gdeposit -s 2012-01-01 -e
2013-01-01 -L 2000 -p film
Successfully deposited 0.00
credits into account 3
```

Let's examine the allocations we just created:

```
$ glsalloc

Id Account Active   StartTime
EndTime      Amount  CreditLimit
  Deposited  Description
--  -
-----  -
-----  -
```



```

-----
1  1      True    2012-01-01
2013-01-01  5000.00
00.0    5000.00
2  2      True    2012-01-01
2013-01-01  3000.00
0.00    3000.00
3  3      True    2012-01-01
2013-01-01    0.00
2000.00    0.00

```

```
$ glsaccount
```

```

Id Name          Amount
Constraints      Description
---  -
-----
1  biology        5000.00
Project=biology
2  chemistry       3000.00
Project=chemistry
3  film            0.00
Project=film

```

5.3.7 Check The Balance

We can verify the resulting balance (see [Querying the Balance](#)).

Example 17. Let's look at amy's balance

```
$ gbalance -u amy

Id Name          Available
Allocated PercentUsed
-- -----
- -----
1  biology        5000.00
5000.00 0.00
2  chemistry      3000.00
3000.00 0.00
```

5.3.8 Integrate Moab Accounting Manager with Your Brokering System

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

In practice, the billing actions (quote, reservation and charge) will be invoked automatically by your brokering system (i.e. initiated by Moab or by the resource manager). However, we will demonstrate these steps manually to illustrate their effects.

Let's simulate the lifecycle of a job.

Example 18. *We'll assume our job has the following characteristics:*

```
Job Id:                moab.1
Job Name:              heavywater
User Name:             amy
Project Name:          chemistry
Machine Name:          colony
Requested Processors: 16
Estimated WallClock:  3600
seconds
Actual WallClock:     1234
seconds
```

5.3.9 Obtain a Usage Quote

When a job is submitted, it is useful to check that the user's account has enough funds enough funds for the requested usage. 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 usage quote step (see [Obtaining Usage 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 usage is charged.

Example 19. Let's see how much it will cost to use the resources.

```
$ gquote -u amy -p chemistry -c  
batch -m colony -P 16 -t 3600  
Successfully quoted 16.00  
credits
```

5.3.10 Make a Usage Reservation

When a job starts or usage begins, the workload manager typically creates a reservation (or hold) against the appropriate allocations based on the estimated duration of the job (see [Making Usage Reservations](#)).

Example 20. Make a reservation for the estimated usage of the job.

```
$ greserve -J moab.1 -p
chemistry -u amy -m colony -P
16 -t 3600
Successfully reserved 16.00
credits with reservation id 1
for instance moab.1 and created
usage record 1

$ glsres
Id Instance Amount
StartTime EndTime
Duration UsageRecord
Accounts Description
-----
-----
-----
-
-----
1 moab.1 16.00 2012-05-
29 15:20:45 2012-05-29 16:20:45
3600 1 2
```

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

```
$ gbalance -p chemistry -total
-quiet
2984.00
```

The actual allocation has not changed.

```
$ glsalloc -p chemistry
Id Account Active StartTime
EndTime Amount CreditLimit
Deposited Description
-----
-----
-----
-
-----
-----
```

```
2 2 True 2012-01-
01 2013-01-01 3000.00
0.00 3000.00
```

This is best illustrated by the detailed balance listing:

```
$ gbalance -u amy -p chemistry
--show=Id,Name,Amount,Reserved,
Balance,CreditLimit,Available
Id Name Amount
Reserved Balance CreditLimit
Available
-----
-----
-----
2 chemistry 3000.00
16.00 2984.00 0.00
2984.00
```

Note that the reservation resulted in the initial creation of a usage record for the job.

```
$ glsusage -u amy -p chemistry
Id Type Instance Charge Stage
User Group Project
Organization Class
QualityOfService Machine Nodes
Processors Memory Desk Network
Duration Starttime EndTime
Description
-----
-----
-----
-----
-----
-----
-----
-----
-----
-----
1 Job moab.1 0.00 Reserve
amy chemistry sciences
colony
16
0
```

5.3.11 Charge for the Usage

After a job completes, any associated reservations are removed and a charge is issued against the appropriate allocations based on the resources and wallclock time actually used by the job (see [Charging for Usage](#)).

Example 21. Issue the charge for the job.

```
$ gcharge -J moab.1 -u amy -p
chemistry -m colony -P 16 -t
1234
Successfully charged 5.48
credits for instance moab.1
1 reservation was removed
```

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

```
$ glsalloc -u amy -p chemistry
Id  Account  Active  StartTime
   EndTime      Amount
CreditLimit Deposited
Description
---  -
-
-----
--
2   2        True    2012-01-
01   2013-01-01  2994.52
0.00   3000.00
```

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

5.3.12 Usage Refund

Since this was an imaginary job, refund the user's account (see [Issuing Usage Refunds](#)).

Example 22. Issue a refund for the job.

```
$ grefund -J moab.1
Successfully refunded 5.48
credits to usage record 1 for
instance moab.1
```

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

```
$ gbalance -u amy -p chemistry
--show=Id,Name,Amount,Reserved,
Balance,CreditLimit,Available

Id Name      Amount Reserved
Balance CreditLimit Available
--  -
-----
2  chemistry 3000.00    0.00
3000.00      0.00    3000.00
```

The allocation, of course, is likewise restored.

```
$ glsalloc -u amy -p chemistry
Id Account  Active  StartTime
EndTime    Amount  CreditLimit
Deposited  Description
--  -
-----
2  2          True    2012-01-01
2013-01-01 3000.00    0.00
3000.00
```

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

```
$ glsusage
Id Type Instance Charge
Stage Quote User Group
Project Organization Class
QualityOfService Machine
Nodes Processors Memory Desk
```

```
Network  Duration  Starttime
EndTime  Description
-----
-----
-----
-----
-----
1  Job  moab.1  0.00
Reserve      amy
chemistry  sciences
              colony
16
```

5.3.13 Examine Account Statement

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

Example 23. You can request an itemized account statement over all time for use amy and the chemistry project (account 2)

```
$ gstatement -u amy -p
chemistry

#####-
#####-
#####
#
# Includes account 2 (chemistry
for amy)
# Generated on Tue May 29
15:48:22 2012
#
# Reporting account activity
from -infinity to now.
#

#####-
#####-
#####
Beginning Balance:
0.00
-----
-----
Total Credits:
3005.48
Total Debits:
-5.48
-----
-----
Ending Balance:
30000.00

#####
Credit Detail
```

```

#####-
###
Object      Action
Instance    Amount      Time
-----
-----
-----
Account      Deposit
      3000.00    2012-05-29
14:52:15
UsageRecord  Refund      moab.1
      5.48      2012-05-29
15:41:20

#####
Debit Detail
#####-
###
Object      Action      Instance
Project     User       Machine
Amount      Time
-----
-----
-----
-----
UsageRecord  Charge      moab.1
chemistry   amy        colony
-5.48 2012-05-29 15:37:02

#####
End of Report
#####-
###

```

6.0 Managing Users

A user is a person authorized to use a resource or service. Default user properties include the common name, phone number, email address, default project, and description for that person. A user can be created, queried, modified, and deleted. By default, a standard user may only query their own user record.

User queries allow the specification of filter options which narrow down the users that will be returned to those belonging to the specified project.

6.1 Creating Users

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

```
gmkuser [-A | -l] [-n common_name] [-F phone_number] [-E email_address]  
[-p default_project] [-d description] [-X, --extension property_name=pro-  
perty_value[,property_name=property_value...]] [--debug] [--site site_name] [-?,  
--help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-u] user_name}
```



Additional detail for this command can be found in the man page by issuing **gmkuser --man** at the command line.

Example 1. Creating a user

```
$ gmkuser -n "Smith, Robert F."  
-E "bob@bank.com" -F "(509)  
555-1234" bob  
Successfully created 1 user
```

6.2 Querying Users

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

```
glsuser [-A | -l] [-p project_name] [-X, --extension property_name=property_value [,property_name=property_value...]] [--full] [--show attribute_name [,attribute_name...]] [-l, --long] [-w, --wide] [--raw] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --version] [[-u] user_pattern]
```



The fields which are displayed by default by this command can be customized by setting the `user.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsuser --man` at the command line.

Example 2. Listing standard info about active users

```
$ glsuser -A
Name      Active      CommonName
          PhoneNumber
EmailAddress
DefaultProject
Description
-----
-
-----
---
amy      True      Wilkes, Amy
          (509) 555-8765
amy@bank.com

bob      True      Smith, Robert
F.      (509) 555-1234
bob@bank.com
```

Example 3. Displaying bob's phone number

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

Example 4. Listing amy's projects

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

Example 5. Listing all users belonging to the chemistry project

```
$ glsuser -show Name -p
chemistry -q
-----
amy
dave
```


6.3 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] [-X, --extension property_name=property_value [,property_name=property_value...]] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --Version] {[-u] user_name}
```



Additional detail for this command can be found in the man page by issuing **gchuser --man** at the command line.

Example 6. Activating a user

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

Example 7. Changing a user's email address

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

6.4 Deleting Users

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

```
grmuser [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose]
[-V, --version] {[-u] user_name}
```



Additional detail for this command can be found in the man page by issuing `grmuser --man` at the command line.

Example 8. Deleting a user

```
$ grmuser bob
Successfully deleted 1 user
```

6.5 User Auto-Generation

By default, users will automatically be created when first added as a member to a project or role. It is also possible to have users be created automatically when first encountered in a usage function (charge, reserve or quote). In order for user auto-generation to occur, the AutoGen property for the User object must be set to 'True'. This is the default. Additionally, for user auto-generation to occur when a user is added as a member of another object (such as Project) via an association table (e.g. ProjectUser), the Values property for the user attribute of the Association (e.g. Name) must be set to '@User', indicating that that value should be constrained to be a valid instance of the User object. For user auto-generation to occur when initially encountered in a usage function, the Values property of the user attribute of the UsageRecord object must be similarly set to '@User'. The auto-creation of users can be completely disabled by setting the AutoGen property for the User object to 'False'.

Example 9. Enable auto-generation of users when initially seen in a charge

```
$ goldsh Attribute Modify
Object==UsageRecord Name==User
Values=@User
Successfully modified 1
attribute
```

Example 10. Disable all auto-generation of users

```
$ goldsh Object Modify
Name==User AutoGen=False
Successfully modified 1 object
```

See [Object Auto-Generation](#) for more information about the auto-generation of objects.

6.6 Default User

It is possible to set a global default user to which usage would be ascribed in quotes, reservations or charges where no user is specified. This can be accomplished by setting the DefaultValue property for the User object to the desired user. It is also possible to set a user default for a specific object, which will result in usage being ascribed to the specified user when the object is attributed to the usage. This is done by creating a default usage override modifier. For example, to specify that acmeuser be the default user for usage associated with the acme organization, you might first create an attribute called DefaultUser for the Organization Object with the Values property of @?=User. Then you would populate the new DefaultUser property for the acme organization with the value of acmeuser. See the chapter on [Customizing Objects](#) for more information on default and other usage override modifiers.

Example 11. Assign a global default user

```
$ goldsh Object Modify
Name==User
DefaultValue=anonymous
Successfully modified 1 object
```

7.0 Managing Projects

A project is a name given to a particular undertaking requiring the use of resources or services for a common purpose. Users may be designated as members of a project and may be allowed to share its allocations. The user members may be designated as active or inactive, and as a project admin or not a project admin. Default project properties include the description, the organization it is part of, and whether or not it is active. A project can be created, queried, modified and deleted. A project's user membership can also be adjusted. By default, a standard user may only query projects they belong to.

Project queries allow the specification of filter options which narrow down the projects that will be returned to those having the specified users in them.

7.1 Creating Projects

To create a new project, use the command `gmproject`:

```
gmproject [-A | -I] [-o organization_name] [-d description] [-X, --extension property_name=property_value [,property_name=property_value...]] [-u [^ | !] [+ | -]user_name [, [^ | !] [+ | -]user_name...]] [--createAccount=True|False] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [{"-p"} project_name]
```



Additional detail for this command can be found in the man page by issuing `gmproject --man` at the command line.

When defining users, the optional caret or exclamation symbol indicates whether the user should be created as an admin ([^]) or not (!) for the project. The optional plus or minus sign can precede each member to indicate whether the member should be created in the active (+) or inactive (-) state. By default, a user will be created in the active state but not an admin. Multiple users may be passed to the `-u` option in a comma-delimited list. Alternatively, multiple `-u` options may be specified.



If the Account object's `AutoGen` property is set to true (see [Account Auto-Generation](#)), 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.

Example 1. Creating a project

```
$ gmproject -d "Chemistry  
Department" chemistry  
Successfully created 1 project
```

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

In this example, we make amy the project admin and associate the project with the sciences organization.

```
$ gmproject -d "Chemistry  
Department" -u ^amy,bob,dave  
chemistry -o sciences  
Successfully created 1 project
```

7.2 Querying Projects

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

```
glsproject [-A | -l] [-o organization_name][-X, --extension property_  
name=property_value [,property_name=property_value...] [-u user_name] [--  
full] [--show attribute_name [,attribute_name...]...] [-l, --long] [-w, --  
wide] [--raw] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --ver-  
sion] [[-p] project_pattern]
```



The fields which are displayed by default by this command can be customized by setting the `project.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsproject --man` at the command line.

Example 3. Listing all info about all projects

```
$ glsproject
Name           Active      Users
  Organization
Description
-----
-----
-----
biology        True        amy,
^bob           sciences
Biology Department
chemistry      True        ^amy,
^dave          sciences
Chemistry Department
film           True        amy,
^dave          arts        Film
Department
```

Example 4. Displaying the name and user members of a project in long format

```
$ glsproject --show Name,Users
-l chemistry
Name           Users
-----
chemistry      ^amy
                dave
```

Example 5. Listing all project names

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

Example 6. Listing all project that have dave as a member

```
$ glsproject --show Name -u  
dave --quiet  
chemistry  
film
```


7.3 Modifying Projects

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

```
gchproject [-A | -I] [-o organization] [-d description] [-X, --extension prop-  
erty_name=property_value [,property_name=property_value...]] [--addUser(s) [^  
| !] [+ | -]user_name [, [^ | !] [+ | -]user_name...]] [--addUser(s) [^ | !] [+ | -]  
user_name [, [^ | !] [+ | -]user_name...]] [--delUser(s) user_name [,user_  
name...]] [--modUser(s) [^ | !] [+ | -]user_name [,user_name...]] [--debug] [--  
site site_man] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-p]  
project_name}
```



Additional detail for this command can be found in the man page by issuing `gchproject --man` at the command line.

User members may be added, removed or modified in a project. When adding user members to a project, the optional caret or exclamation symbol indicates whether the user should be created as an admin (^) or not (!) for the project. The optional plus or minus signs can precede each member to indicate whether the member should be created in the active (+) or inactive (-) state. When modifying user members of a project, the caret symbol or exclamation symbol indicates the user should be changed to become an admin (^) or not (!) for the project. The plus or minus signs indicate whether the user should be changed to become active (+) or inactive (-). If an active or admin modifier is not specified, that aspect of the user member will remain unchanged.

Example 7. Deactivating a project

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

Example 8. Adding users as members of a project

```
$ gchproject --add-users  
jsmith,barney chemistry  
Successfully added 2 users
```

Example 9. Deactivating a user in a project

```
$ gchproject --mod-user -dave  
chemistry  
Successfully modified 1 user
```

7.4 Deleting Projects

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

```
grmproject [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-p] project_name}
```



Additional detail for this command can be found in the man page by issuing `grmproject --man` at the command line.

Example 10. Deleting a project

```
$ grmproject chemistry
Successfully deleted 1 project
```

7.5 Project Auto-Generation

It is possible to have projects be created automatically when first encountered in a usage function (charge, reserve or quote). It is also possible for projects to be automatically created when initially added as a member of another object. In order for project auto-generation to occur, the AutoGen property for the Project object must be set to 'True'. This is the default. For project auto-generation to occur when initially encountered in a usage function, the Values property of the project attribute of the UsageRecord object must be set to '@Project'. Additionally, for project auto-generation to occur when a project is added as a member of another object (such as the Organization object) via an association table (e.g. OrganizationProject), the Values property for the project attribute of the Association (e.g. Name) must be set to '@Project', indicating that that value should be constrained to be a valid instance of the Project object. The auto-creation of projects can be completely disabled by setting the AutoGen property for the Project object to 'False'.

Example 11. Enable auto-generation of projects when initially seen in a charge

```
$ goldsh Attribute Modify
Object==UsageRecord
Name==Project Values=@Project
Successfully modified 1
attribute
```

Example 12. Disable all auto-generation of projects

```
$ goldsh Object Modify
Name==Project AutoGen=False
Successfully modified 1 object
```

See [Object Auto-Generation](#) for more information about the auto-generation of objects.

7.6 Default Project

It is possible to set a global default project to which usage would be ascribed in quotes, reservations or charges where no project is specified. This can be accomplished by setting the DefaultValue property for the Project object to the desired project name. It is also possible to set a project default for a specific object, which will result in usage being ascribed to the specified project when the object is attributed to the usage. This is done by creating a default usage override modifier. For example, to specify that debug be the default project for usage associated with the machine sandbox, you might first create an attribute called DefaultProject for the Machine Object with the Values property of @?=Project. Then you would populate the new DefaultProject property for the sandbox machine with the value of debug. See the chapter on [Customizing Objects](#) for more information on default and other usage override modifiers.

Example 13. Assign a global default project

```
$ goldsh Object Modify
Name==Project
DefaultValue=common
Successfully modified 1 object
```

8.0 Managing Machines

A machine is a location where resources or services can be used such as a cluster, cloud or a store. Default machine properties include the description and whether or not it is active. A machine can be created, queried, modified and deleted.

8.1 Creating Machines

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

```
gmkmachine [-A | -l] [--arch architecture] [--opsys operating_system] [-o organization_name] [-d description] [-X, --extension property_name=property_value [,property_name=property_value...]] [--debug] [--site site_man] [-?, -help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-m] machine_name}
```



Additional detail for this command can be found in the man page by issuing **gmkmachine --man** at the command line.

Example 1. Creating a machine

```
$ gmkmachine --arch amd64 --  
opsys linux "Linux Cluster"  
colony  
Successfully created 1 machine
```

8.2 Querying Machines

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

```
glsmachine [-A | -l] [-X, --extension property_name=property_value[,property_name=property_value...]] [--full] [--show attribute_name[,attribute_name...]...] [--raw] [--debug] [--site site_man] [-?, --help] [--man] [--quiet] [--m machine_pattern]
```



The fields which are displayed by default by this command can be customized by setting the `machine.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsmachine --man` at the command line.

Example 2. Listing all inactive machine names and descriptions

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

8.3 Modifying Machines

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

```
gchmachine [-A | -l] [--arch architecture] [--opsys operating_system] [-d description] [-o organization] [-X, --extension property_name=property_value [,property_name=property_value...]] [--site site_man] [--debug] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-m] machine_name}
```



Additional detail for this command can be found in the man page by issuing **gchmachine --man** at the command line.

Example 3. Deactivating a machine

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


8.4 Deleting Machines

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

```
grmmachine [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-m] machine_name}
```



Additional detail for this command can be found in the man page by issuing `grmmachine --man` at the command line.

Example 4. Deleting a machine

```
$ grmmachine colony
Successfully deleted 1 machine
```

8.5 Machine Auto-Generation

It is possible to have machines be created automatically when first encountered in a usage function (charge, reserve or quote). It is also possible to have machines automatically be created when initially added as a member of another object. In order for machine auto-generation to occur, the AutoGen property for the Machine object must be set to 'True'. This is the default. For machine auto-generation to occur when initially encountered in a usage function, the Values property of the machine attribute of the UsageRecord object must be set to '@Machine'. Additionally, for machine auto-generation to occur when a machine is added as a member of another object (such as a hypothetical Site object) via an association table (e.g. SiteMachine), the Values property for the machine attribute of the Association (e.g. Name) must be set to '@Machine', indicating that that value should be constrained to be a valid instance of the Machine object. The auto-creation of machines can be completely disabled by setting the AutoGen property for the Machine object to 'False'.

Example 5. Enable auto-generation of machines when initially seen in a charge

```
$ goldsh Attribute Modify
Object==UsageRecord
Name==Machine Values=@Machine
Successfully modified 1
attribute
```

Example 6. Disable all auto-generation of machines

```
$ goldsh Object Modify
Name==Machine AutoGen=False
Successfully modified 1 object
```

See [Object Auto-Generation](#) for more information about the auto-generation of objects.

8.6 Default Machine

It is possible to set a global default machine to which usage would be ascribed in quotes, reservations or charges where no machine is specified. This can be accomplished by setting the DefaultValue property for the Machine object to the desired machine name. It is also possible to set a machine default for a specific object, which will result in usage being ascribed to the specified machine when the object is attributed to the usage. This is done by creating a default usage override modifier. For example, to specify that whitecloud be the default machine for usage associated with the amy user, you might first create an attribute called DefaultMachine for the User Object with the Values property of @?=Machine. Then you would populate the new DefaultMachine property for the amy user with the value of whitecloud. See the chapter on [Customizing Objects](#) for more information on default and other usage override modifiers.

Example 7. Assign a global default machine

```
$ goldsh Object Modify
Name==Machine
DefaultValue=cloud
Successfully modified 1 object
```

9.0 Managing Accounts

An account is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an account is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each account has a set of constraints designating which entities (such as Users, Projects, Machines, Classes, Organizations, etc.) may access the account or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Account constraints may also be negated with an exclamation point leading the constraint value.

When 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 billable units 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 an allocation 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.

Accounts may have a name which is not necessarily unique for the account. Accounts may also have a priority which will influence the order of account selection when charging. Operations include creating, querying, modifying and deleting accounts as well as making deposits, withdrawals, transfers and balance queries. An account (or all accounts) may also be reset which means that all of the credits and deposited tallies in all active allocations associated with the account are set to zero. By default, a standard user may only query and view the balance for accounts which pertain to them.

Some account operations (Account Query, Account Balance, Account Deposit, Account Withdraw and Account Refund) allow the specification of filter options which narrow down the accounts that will be acted on for that operation. There are two account filter types that can be employed: Exclusive and NonExclusive. If an exclusive filter type is used, the query will return only the accounts for which the specified filters meet all constraints for usage. Another way to think of an exclusive filter is to ask if usage were to be posted given only the specified filter options as ACLs, which accounts would be eligible for charging? For example, Account Query FilterType:=Exclusive Filter:=User=scottmo would not return an account with the sole constraint Machine=blue because Machine=blue was not included in the filters. Not only must the filters be a non-conflicting superset of the account constraints, but all constraint dependencies must also be satisfied (for

example, an appropriate user may need to be specified with the project). If a non-exclusive filter type is used, the query will return all accounts for which the filters do not specifically exclude the constraints. The query assumes that if constraints are not specified within the filters, they can be assumed as a wildcard and will return all accounts that are not specifically excluded by the filter. For example, Account Query FilterType:=NonExclusive Filter:=User=scottmo would return an account whose only constraint was Machine=blue (because it does not conflict) but would not return an account with the constraint User=bob (because it does conflict).

9.1 Creating Accounts

`gmkaaccount` is used to create a new account. You can specify an account name, a description, and any number of account constraints. If a name is not specified and constraints are specified, a name will be automatically generated based on the constraints. A new unique id is automatically generated for the account.

```
gmkaaccount [-n account_name] [-d description] [-X, --extension prop-  
erty_name=property_value [, property_name=property_value...]] [-  
c class_name] [-g group_name] [-m machine_name] [-o organization_  
name] [-p project_name] [-u user_name] [, [-C, --constraint constraint_  
name=[!]constraint_value [, constraint_name=[!] [constraint_  
value...]]] [--parent parent_account_id] [--debug] [--site site_name] [-?, --  
help] [--man] [--quiet] [-v, --verbose] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing `gmkaaccount --man` at the command line.



It is possible to have accounts be created automatically when projects are created by setting the `account.autogen` configuration parameter to true (see [Account Auto-Generation](#)). The auto-generated account will be associated with the new project.

Example 1. Creating an account valid for the chemistry project

```
$ gmkaaccount -p chemistry -n  
"Chemistry"  
Successfully created 1 account  
with id 7 and 1 constraint
```

Example 2. Creating a wide-open account that can be used by anyone for anything

```
$ gmkaaccount -n "Windfall"  
Successfully created 1 account  
with id 8
```

Example 3. Creating an account valid toward all biology project members except for dave and just the machine colony

```
$ gmkaaccount -C  
Project=biology,User=!dave,
```

```
Machine=colony -n "Biology on  
Colony not for Dave"  
Successfully created 1 account  
with id 9 and 3 constraints
```

9.2 Querying Accounts

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

```
glsaccount [-A | -l] [-n account_name] [-X, --extension property_name=property_value [,property_name=property_value...]] [-u user_name] [-g group_name] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value [, filter_name=filter_value...]] [-F, --filter-type Exclusive|NonExclusive] [--full] [--show attribute_name [,attribute_name...]] [-l, --long] [-w, --wide] [--raw] [-h, --hours] [--debug] [--site site_man] [-?, --help] [--man] [--quiet] [-V, --version] [[-a] account_id]
```



The fields which are displayed by default by this command can be customized by setting the `account.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsaccount --man` at the command line.

Example 5. Listing all info about all accounts with multi-valued fields displayed in a multi-line format

```
$ glsaccount -long
Id Name Amount
  Constraints
Description
-----
-----
-----
1 Biology 25000000
  Project=biology
2 chemistry for amy 34802392
  User=amy
  Project=chemistry
3 chemistry not amy 5000000
  User=!amy
  Project=chemistry
4 film on colony 0
  Project=film
  Machine=colony
```

Example 6. Wide listing all info about all accounts useable by amy


```

$ glsaccount -u amy
Id  Name                               Amount
   Constraints
   Description
---  -----
-----
1   biology                             25000000
   Project=biology
2   chemistry for amy                   24802392
   Project=chemistry,User=amy
4   film on colony                       0
   Machine=colony,
   Project=dance

```

9.3 Modifying Accounts

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

```
gchaccount [-n account_name] [--priority account_priority] [-d description]
[-X, --extension property_name=property_value [,property_name=property_value...]]
[--addConstraint(s) constraint_name=[!]constraint_value [,constraint_name=[!]constraint_value...]]
[--delConstraint(s) constraint_name [,constraint_name...]]
[--parent parent_account_id] | --reset [--all]} [-u user_name] [-g group_name] [-p project_name]
[-o organization_name] [-cclass_name] [-m machine_name] [-f, --filter filter_name=filter_value
[,filter_name=filter_value...]] [-F, --filter-type Exclusive|(NonExclusive)] [--debug]
[--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V --version] ([-a] account_id)
```



Additional detail for this command can be found in the man page by issuing `gchaccount --man` at the command line.

Example 6 - Adding a constraint to an account so that it can only be used by the acme organization

```
$ gchaccount --add-constraint
Organization=acme 7
Successfully created 1
constraint
```

Example 7 - Resetting an account

```
$ gchaccount --reset 1
Successfully reset 4512 credits
from 1 allocation
```

9.4 Making Deposits

gdeposit is used to deposit time-bounded resource credits into an account resulting in the creation or increase of an allocation. (See [Managing Allocations](#) for managing allocations). The start time will default to `-infinity` and the end time will default to `infinity` if not specified. Filter options can be specified to help select a unique account for the deposit. If multiple accounts are matched by the filters, the matching accounts will be listed and you will be prompted to respecify the deposit with one of the account ids. If an allocation for the deposit account is found having the start and end times for the deposit, the amount of the allocation will be increased by the deposit amount. Otherwise, a new allocation will be created for the account with the amount of the deposit. If no accounts match your criteria, if the account auto-generation is enabled, an account will be created and the deposit made into it. Otherwise, the deposit will fail (the account will need to be first created using [gmkaccount](#)).

Deposits may be used to extend the debit ceiling by specifying an amount for the deposit (with the `-z` option) or extend the credit floor by specifying a credit limit for the deposit (with the `-L` option) or a combination of both options may be used. Additionally, `Infinity` may be used for either of these option values when Moab Accounting Manager is coupled with a database that supports IEEE Standard 754 for Floating-Point Arithmetic (e.g. PostgreSQL).

```
gdeposit [-L credit_limit] [-s start_time] [-e end_time] [-d description] [-a account_id] [-i allocation_id] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value[,filter_name=filter_value...]] [-F, --filterType Exclusive|(NonExclusive)] [--create-account True|False] [--reset] [-h, --hours] [--debug] [--site site name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-z] amount]
```



Additional detail for this command can be found in the man page by issuing **gdeposit --man** at the command line.

Example 8. Making a deposit into account 1

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

Example 9. Making a deposit "into" a project

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

```
$ gdeposit -z 360000000 -p
chemistry
Successfully deposited
360000000 credits into account
2
```

Example 10. Creating a credit allocation

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

Example 11. Making a reset deposit

Reset the active allocations in the account before making the deposit.

```
$ gdeposit -a 4 -z 36000000 --
reset
Successfully deposited 36000000
credits into account 4
Successfully reset 12767021
credits from 1 allocation
```

Example 12. Creating an infinite allocation

```
$ gdeposit -z Infinity -a 5
Successfully deposited inf
credits into account 5
```



The use of infinite allocations requires the use of a database that supports the IEEE Standard 754 for Floating-Point Arithmetic (e.g. PostgreSQL).

Example 13. Making a series of quarterly allocations

```
$ gdeposit -s 2012-01-01 -e
2012-04-01 -z 25000000 -p
biology
Successfully deposited 25000000
credits into account 6

$ gdeposit -s 2012-04-01 -e
2012-07-01 -z 25000000 -p
biology
```

```
Successfully deposited 25000000  
credits into account 6
```

```
$ gdeposit -s 2012-07-01 -e  
2012-10-01 -z 25000000 -p  
biology  
Successfully deposited 25000000  
credits into account 6
```

```
$ gdeposit -s 2012-10-01 -e  
2013-01-01 -z 25000000 -p  
biology  
Successfully deposited 25000000  
credits into account 6
```

9.5 Querying The Balance

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

```
gbalance [-u user_name] [-g group_name] [-p project_name] [-o organ-  
ization_name] [-c class_name] [-m machine_name] [-f, --filter filter_  
name=filter_value[,filter_name=filter_value...]] [-F, --filterType  
Exclusive|(NonExclusive)] [-l, --long] [-w, --wide] [--raw] [-h, --hours] [--site site_  
name] [--debug] [-? --help] [--man] [--quiet] [-V, --version]
```



The fields which are displayed by default by the **gbalance** command can be customized by setting the `balance.show` configuration parameter in `gold.conf`.



Additional detail for this command can be found in the man page by issuing **gbalance --man** at the command line.

Example 14. Querying amy's balance

```
$ gbalance -u amy
Id Name      Available
Allocated  PercentUsed
---  -
13  biology    2785.87
5000.00    44.28
2   chemistry  1785.87
3000.00    40.47
```

Example 15. Querying the total balance available to bob for the biology project on a colony cluster

```
$ gbalance -u bob -m colony -p  
chemistry --total --available -  
-quiet  
2785.87
```

Example 16. List the available balances that amy can charge against along with the constraints on those balances

```
$ gbalance -u amy --show  
Balance,Constraints
```

Balance	Constraints
-----	-----
25000000	Project=biology
34802392	
Project=chemistry,User=amy	
0	Machine=colony,
Project=film	

9.6 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 accounts that they can charge to:

`mybalance [-h, --hours] [-?, --help] [--man]`



Additional detail for this command can be found in the man page by issuing `mybalance --man` at the command line.

Example 17. List my account balances

```
$ mybalance
Balance      Name
-----
--
25000000    biology
34802302    chemistry for amy
```

Example 18. List my balance in (Processor) hours

```
$ mybalance -h
Balance      Name
-----
--
6944.44     biology
9667.33     chemistry for amy
```


9.7 Making Withdrawals

A withdrawal can be used to debit an account without being associated with the usage charge from some item. To issue a withdrawal, use the command **gwithdraw**:

```
gwithdraw [-a account_id] [-i allocation_id] [-u user-name] [-g group_name] [-p project_name] [-p organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value[,filter_name=filter_value...]] [-F, --filter-type Exclusive|NonExclusive] [[-z] amount] [-d description] [-h, --hours] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-z] amount}
```



Additional detail for this command can be found in the man page by issuing **gwithdraw --man** at the command line.

Example 19. Making a withdrawal

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

Example 20. 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 biology  
Successfully withdrew 12800  
credits from account 1
```

If more than one account exists for the project or filter, you will be asked to be more specific:

```
$ gwithdraw -z 12800 -p  
chemistry  
Multiple accounts were matched  
for the withdrawal.  
Please respecify using one of  
the following account ids:  
2 [chemistry for amy]  
3 [chemistry not amy]
```

9.8 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. Source and destination filters may be used if they result in a single source account and single destination account.

```
gtransfer {--from-account source_account_id | --from-filter source_filter_name=source_filter_value [, source_filter_name=source_filter_value...]} {-i allocation_id} {--to-account destination_account_id | --to-filter destination_filter_name=destination_filter_value [, destination_filter_name=destination_filter_value...]} [-d description] [-h, --hours] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v --verbose] {[-z] amount}
```



Additional detail for this command can be found in the man page by issuing `gtransfer --man` at the command line.

Example 21. Transferring credits between two accounts

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

Example 22. Transferring credits between two single-account projects

```
$ gtransfer --from-filter  
Project=biology --to-filter  
Project=chemistry 10000  
Successfully transferred 10000  
credits from account 1 to  
account 2
```

9.9 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 filter options are 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] [-n account_name] [-u user_name] [-g group_name] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value [, filter_name=filter_value...]] [-F, --filter-type Exclusive|(NonExclusive)] [-s start_time] [-e end_time] [--summarize] [-h, --hours] [--debug] [--site site_man] [-?, --help] [--man] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing **gstatement --man** at the command line.

Example 23. Generating an account statement for all chemistry accounts for the fourth quarter of 2011

```
$ gstatement -p chemistry -s
2011-10-01 -e 2012-01-01

#####-
## End of Report
#####-
#
$ gstatement -p chemistry -s
2011-10-01 -e 2012-01-01 --
summarize

#####-
#####-
#####
#
# Includes account 3 (chemistry
not amy)
# Includes account 2 (chemistry
for amy)
# Generated on Mon Feb 7
```

```
18:44:23 2012.
# Reporting account activity
from 2011-10-01 to 2012-01-01.
#
```

```
#####-
#####-
#####
```

```
Beginning Balance:          0
-----
-----
```

```
Total Credits:
90122212
Total Debits:              -
5308668
-----
-----
```

```
Ending Balance:
84813544
```

```
#####
```

```
Credit Summary
#####-
###
```

```
Object          Action      Amount
-----
-
```

```
Account          Deposit
90100000
UsageRecord      Refund
22212
```

```
#####
```

```
Debit Summary
#####-
###
```

```
Object          Action      Project
  User    Machine      Amount
Count
-----
```

```
--  ----  -----  -----
-----
```

```
UsageRecord      Charge
chemistry  amy      colony      -
```

```
19744 239
```

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



The fields which are used as default discriminators in the detail section of the **gstatement** command (which are by default Project, User and Machine) can be customized by setting the `statement.show` configuration parameter in `gold.conf`.

9.10 Deleting Accounts

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

```
grmaccount [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-a] account_id}
```



Additional detail for this command can be found in the man page by issuing **grmaccount --man** at the command line.

Example 24. Deleting an account

```
$ grmaccount 2  
Successfully deleted 1 account
```

9.11 Account Auto-Generation

It is possible to enable the auto-generation of accounts by setting the `AutoGen` property of the `Account` object to `True`. When creating a new project, if account auto-generation is enabled, an account will automatically be created for the project (unless overridden with the `--create-account` option). The account will be usable only by usage attributed to the new project. Additionally, if account auto-generation is set, a deposit that does not match an existing account will automatically generate an account using the filters as constraint options. Objects associated with the constraint that have `AutoGen` set to `True` will be auto-generated as well (unless overridden with the `--create-account` option).

Example 25. Enable auto-generation of accounts

```
$ goldsh Object Modify
Name==Account AutoGen=True
Successfully modified 1 object
```

9.12 Hierarchical Accounts

An account hierarchy can be established between accounts. When creating an account or by modifying it later, one can specify a parent account id via the `--parent` option to establish the object account as a child of the specified parent account. An account may have multiple children accounts but only a single parent account.

Example 26. Establishing a child relationship with another account

```
$ gchaccount --parent 3 -a 6
Successfully added 1 parent
```

Deposit shares may be established between the parent account and its children that assist to automate a trickle-down effect for funds deposited at higher level accounts (`DepositShare` is an attribute of the `AccountAccount` association object). Deposit shares are integers and are treated as a percentage of each deposit and the sum of the deposit shares for an account's children may not exceed 100. If the deposit shares for the children of an account totals less than 100, the difference is taken to be the share of the deposit that will be allocated to the parent. When a deposit is made into a parent account, for each child account that has a non-zero deposit share a recursive deposit amounting to the designated percentage of the parent deposit is issued to that child. After the share amounts have been deposited to each of the child accounts, the remaining percentage of the deposit is allocated to the parent account. This effect is recursive with each child. If a start time and/or end time are specified in the original deposit, these time frames will be recursively applied to all descendant deposits. You have to use the `goldsh` interactive control program to manage deposit shares. For the `AccountAccount` association object, the `Account` is the parent and the `Id` is the child.

Example 27. Establishing a 10% deposit share between a parent and a child account

```
$ goldsh AccountAccount Modify
Account==3 Id==6
DepositShare=10

Account      Id      DepositShare
Overflow
-----
3            6       10           False
```



```
Successfully modified 1
accountAccount
```

An overflow policy may be established between the parent account and its children to enable a trickle-up effect for charges, reservations and quotes from the lower level accounts (Overflow is an attribute of the AccountAccount association object). The Overflow attribute is a boolean value (True or False). If the overflow value between a child and its parent is set to True, any charges, reservations or quotes issued against the child account that cannot be satisfied by the balance in the child account, will recursively issue the unsatisfied portion of the charge, reservation or quote against the parent account. If the charge, reservation or quote cannot be satisfied by the ancestors, no charges, reservations or quotes will result against any of accounts. The balance in the descendant accounts will be depleted before ancestor accounts. This effect is recursive with each parent. If a parent account is linked with overflow to a child account and a charge, reservation or quote overflows to the parent account, the constraints of the parent account will not be checked against the properties of the item. One must use the goldsh control program to manage the overflow policy. For the AccountAccount association object, the Account is the parent and the Id is the child.

Example 28. Enabling overflow between a parent and a child account

```
$ goldsh AccountAccount Modify
Account==3 Id==6 Overflow=True

Account      Id      DepositShare
Overflow
-----
3            6            10            True

Successfully modified 1
accountAccount
```

9.13 Account Priority

By default, when an item can charge to multiple accounts, accounts with more constraints are chosen over accounts with fewer constraints. For example, if the user amy is charging against the chemistry project for usage of an item and there are two viable accounts, one with a single constraint (e.g. Project=chemistry) and another with two constraints (e.g. Project=chemistry and User=amy), credits will be taken from the more specific account (with 2 constraints) before they are taken from the more general account (with 1 constraint). To override this behavior, it is possible to give a priority to an account. The priority factor of an account has higher precedence than the specificity (constraint count) of the account. Thus, all else being equal, if an account with a lower number of constraints is given a higher priority than an account with a higher number of constraints, the higher priority account will be depleted first. Other factors, such as the end time of the allocation or whether there is an existing reservation for the item against an account, have a higher precedence than the specificity of the account. If you want the allocations in a particular account to be chosen before allocations that expire sooner or that have a reservation, you may need to specify account priorities that are in the millions (see [Allocation Precedence](#) for a discussion of the manner of sorting allocations for charging).

Example 29. Setting an account priority

```
$ goldsh Account Modify Id==3
Priority=1
Successfully modified 1
account
```

10.0 Managing Allocations

An allocation is a time-bounded pool of resource or service credits associated with an account. An account may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-Infinity to Infinity). An active flag is automatically updated to True if the account is within its valid timeframe or False if it is not. An allocation that becomes active because the current time is greater than its start time undergoes an activation which normally registers as a credit to the account. An allocation that becomes inactive because the current time is greater than its end time undergoes a deactivation which normally registers as a debit to the account.

An allocation may have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the account is like a debit account, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the account will be like a credit account, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

An allocation also has a Deposited attribute that is incremented with each crediting deposit. When a deposit is made, if a matching allocation already exists with the appropriate time period, the existing allocation is modified. Otherwise, a new allocation is created. If the deposit results in an increased balance for the account, the Deposited field is incremented by the same amount. Thus, the Deposited field seeks to track the total amount deposited to the allocation over its lifetime. An allocation can be reset, which causes both the Amount and the Deposited fields to be reset to zero.

It is possible for the allocation Amount or CreditLimit to be set to Infinity (via a deposit). If the amount is infinite, debits will not decrease the balance. An infinite deposit will result in an infinite Deposited amount. If the credit limit is infinite, there will be no negative limit for debits. It is not possible to have infinite charges, reservations, quotes, withdrawals, refunds or transfers. However, it is possible to have infinite allocation activations, deactivations and deletions. This capability is only available when using a database that supports IEEE Standard 754 for Floating-Point Arithmetic (e.g. PostgreSQL).

Operations include querying, modifying, resetting and deleting allocations. Allocations can be created by an account deposit, creating an account with allocation auto-generation enabled, refunding a usage record, or a transfer between

accounts. Allocations may also be indirectly modified via charges, withdrawals, transfers, or refunds. By default, a standard user may only query allocations which pertain to them.

Allocation queries allow the specification of filter options which filter the allocations to those with accounts meeting the specified account constraints. There are two account filter types that can be employed: Exclusive and NonExclusive. If an exclusive filter type is used, the query will return only allocations relating to accounts for which the specified filters meet all constraints. For example, Allocation Query FilterType:=Exclusive Filter:=User=scottmo would not return an allocation for an account with the sole constraint Machine=blue. If a non-exclusive filter type is used, the query will return all allocations relating to accounts for which the filters do not specifically exclude the constraints. The query assumes that if constraints are not specified within the filters, they can be assumed as a wildcard and will return all allocations involving accounts that are not specifically excluded by the filter. For example, Allocation Query FilterType:=NonExclusive Filter:=User=scottmo would return an allocation with an account whose only constraint was Machine=blue but would not return an allocation with an account with the constraint User=bob.

10.1 Creating Allocations

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

10.2 Querying Allocations

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

```
glsalloc [-A | -l] [-a account_id] [-X, --extension property_name=property_value [,property_name=property_value...]] [-u user_name] [-g group_name] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value [, filter_name=filter_value...]] [-F, --filter-type Exclusive | (NonExclusive)] [--include-ancestors] [--full] [--show attribute_name [,attribute_name...]] [-l, --long] [-w, --wide] [--raw] [-h, --hours] [--debug] [--site site_man] [--?, --help] [--man] [--quiet] [-V --version] [[-i] allocation_id]
```



The fields which are displayed by default by this command can be customized by setting the `allocation.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsalloc --man` at the command line.

Example 1. Listing allocations for account 1

```
$ glsalloc -a 1
Id  Account  Active
StartTime  EndTime
Amount      CreditLimit
Deposited   Description
-----
-----
-----
-----
-----
2   1        False    2012-
04-01    2012-07-01    25000000
          0 250000

3   1        False    2012-
07-01    2012-10-01    25000000
          0 250000

4   1        False    2012-
10-01    2013-01-01    25000000
          0 250000

1   1        True     2012-
01-01    2012-04-01    24974400
```

0 250000

10.3 Modifying Allocations

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

```
gchalloc [-s start_time] [-e end_time] [-L credit_limit] [-D deposited]
[-d description] [-X, --extension property_name=property_value [, prop-
erty_name=property_value...]] [-h, --hours] [--debug] [--site site_name]
[-?, --help] [--man] [-quiet] [-v, --verbose] [{-i} allocation_id]
```



Additional detail for this command can be found in the man page by issuing `gchalloc --man` at the command line.

Example 2. Changing the end time for an allocation

```
$ gchalloc -e "2013-01-01" 4
Successfully modified 1
allocation
```

Example 3. Changing the credit limit for an allocation

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

Example 4. Resetting an allocation

```
$ gchalloc -e --reset 2
Successfully reset 25000000
credits from 1 allocation
```


10.4 Delete Allocations

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

```
grmalloc [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose]
[-V, --version] [-l | [-i] allocation_id]
```



Additional detail for this command can be found in the man page by issuing **grmalloc --man** at the command line.

Example 5. Deleting an allocation

```
$ grmalloc 4
Successfully deleted 1
allocation
```

Example 6. Purging inactive allocations

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

10.5 Allocation Auto-Generation

It is possible to enable the auto-generation of allocations by setting the `AutoGen` property of the `Allocation` object to `True`. When creating a new account, if allocation auto-generation is enabled, an allocation will automatically be created for the account via a deposit. The deposit will use the default amount and default credit limit (defined in the `DefaultValue` property of the `Allocation Amount` and `Allocation CreditLimit` attributes). The default action for allocation auto-generation is to create an allocation with an infinite credit limit.

Example 7. Enable auto-generation of allocations

```
$ goldsh Object Modify
Name==Allocation AutoGen=True
Successfully modify 1 object
```

10.6 Allocation Precedence

When issuing a charge (or a reservation or quote) for the usage of a resource or service, the feasible allocations are sorted according to a weight given to them for that transaction. The weight for each allocation is calculated as follows: If the instance has a current reservation against one or more allocations, these allocations are given a value of $10000000 + \text{int}((2147483647 - \text{<end_epoch_time>}) / 86400)$. Thus, these reserved allocations will generally have the highest precedence (subject to large account priorities), with those that expire sooner being used first. For the remaining non-nested accounts, allocations will be given a value of $100 * \text{int}((2147483647 - \text{<end_epoch_time>}) / 86400) + 10 * \text{<account_priority>} + \text{<constraint_count>}$. Thus, sooner expiring allocations will be used before later expiring allocations, account priority will be the next highest factor (assuming small priority values of 1-10), followed by the number of constraints on the account (more specific accounts will be used before more general accounts). Of course, since priority is configurable, a sufficiently large priority (in the millions) can be used to override the precedence of earlier expiring allocations or even allocations with reservations. Lastly, nested accounts that become feasible because of overflow to ancestor accounts have a negative weighting and are used last, with the earliest expiring allocations being used before later expiring allocations and closer level ancestors being depleted before ancestor accounts that are at more distant levels. These allocations are given a weight of $\text{<distance * 100000>} - \text{<end_epoch_time>}$. After all feasible allocations are sorted according to the above rules, the charge (or reservation or quote) will be applied against the allocations one by one in sorted order (highest value first) until the request is fulfilled, or until it fails due to insufficient funds. If a transaction is not able to be satisfied in whole, the entire transaction will fail and no partial debits will be applied.

11.0 Managing Reservations

A reservation is a hold placed against an allocation. Before usage of a resource or service begins, a reservation (or hold) is made against one or more allocations within the requesting user's applicable account(s). Subsequent usage requests will also post reservations while the available balance (active allocations minus reservations) allows. When the usage ends, the reservation is removed and the actual charge is made to the allocation(s). This procedure ensures that usage will only be permitted so long as the requestors have sufficient reserves.

Associated with a reservation is the instance name (name of the item being used such as the job id), the usage record (which contains the item details), a start time and end time for the reservation and a description. The reservation will automatically expire and no longer count against the user's balance after the end time passes. Each reservation will be associated with held amounts from one or more allocations. Operations include creating, querying, modifying and deleting reservations. By default, a standard user may only query reservations attributed to them.

Reservation queries allow the specification of filter options which narrow down the reservations that will be returned. There are two reservation filter types that can be employed: `AttributedTo` and `ImpingesUpon`. If `ImpingesUpon` is used, the query will return all reservations associated with accounts satisfying the filters. For example, `Reservation Query FilterType:=ImpingesUpon Filter:=User=scottmo` will return all reservations impinging on Accounts usable by scottmo. If `AttributedTo` is used, the query will return all reservations associated with usage records satisfying the filters. For example, `Reservation Query FilterType:=AttributedTo Filter:=User=scottmo` will return all reservations for resources or services allocated to scottmo.

When a reservation is created via the `UsageRecord Reserve` action (such as via `reserve`), if another reservation exists with the same instance name, the default behavior is to leave the old reservation in place (and create the new one along side it). This behavior assumes that the other reservation is probably a separate reservation created by a resource or service manager that reuses instance ids. However, alternate behaviors may be specified via the mutually exclusive `Modify` or `Replace` options. If the `Replace` option is specified, any pre-existing reservations with matching instance names will first be deleted, thereby ensuring only one reservation per instance name at a time. If the `Modify` option is specified, a pre-existing reservation with matching instance name will be modified to have the new properties (but keeping the same reservation id), and can be used to extend a reservation. This might be used with incremental charging to dynamically stretch reservations along a little at a time as needed. (See [Making Usage Reservations](#) for a description of the action using these options).

Reservation may be granted a grace period (in seconds), which is defined as the difference between the validity period of the reservation (end time minus start time) and the expected duration of the usage. The purpose of a grace period is to account for the fact that we may not know precisely when the usage will begin and the reservation needs to be remain in force during the lifetime of the usage. One can apply a desired grace period for a reservation by setting the end time longer than the specified duration. Alternatively, a grace duration option can be specified with the duration when creating a reservation via greserve as a helper to computing a relatively adjusted end time.

11.1 Creating Reservations

Reservations are normally created by the resource management system with the `greserve` command (See [Making Usage Reservations](#)).

```
gmkres [-J instance_name|job_id] [-s start_time] {-e end_time | -t reservation_duration} [-d description] [-X, --extension property_name=property_value [,property_name=property_value...]] {-A allocation_id<account_id=subreservation_amount[,allocation_id<account_id=subreservation_amount...]} [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing `gmkres --man` at the command line.

Example 1. Creating a manual reservation

```
$ gmkres -J weekend_run -t
84600 -A "5<-2=3600"
Successfully created 1
reservation
```



Use of the `gmkres` command bypasses the normal mechanisms that prevent more reservations from being placed against an allocation than it can support. Use `greserve` instead if you wish to avoid the possibility of oversubscribing the allocations.

11.2 Querying Reservations

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

```
glsres [-A | -l] [-J instance_pattern | job_id_pattern] [-X, --extension property_name=property_value [,property_name=property_value...]] [-u user_name] [-g group_name] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [-f, --filter filter_name=filter_value [,filter_name=filter_value...]] [-F, --filter-type (AttributedTo)|ImpingesUpon] [--full] [--show attribute_name [,attribute_name...]] [-l, --long] [-w, --wide] [--raw] [-h, --hours] [--debug] [--site site_name] [--help] [--man] [-quiet] [-V, --version] [--r] reservation_id
```



The fields which are displayed by default by this command can be customized by setting the `reservation.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsres --man` at the command line.

Example 2. Listing all info about all reservations for amy

```
$ glsres -u amy
Id Instance      Amount
StartTime      EndTime
UsageRecord Accounts
Description
-----
3   PBS.1234.4     57600    2012-
04-06 21:21:48     2012-04-06
22:31:48 7             2
```

Example 3. Listing all info about all reservations that impinge against dave's balance

```
$ glsres -u dave -F
ImpingesUpon
Id Instance      Amount
StartTime      EndTime
UsageRecord Accounts
Description
-----
```

```
-----  
-----  
-----  
4    batch.12          7600  2012-  
04-06 15:30:34    2012-04-06  
15:41:50  244          3
```


11.3 Modifying Reservations

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

```
gchres [-s start_time] [-e end_time] [-t reservation_duration] [-d  
description] [-X, --extension property_name=property_value [,property_  
name=property_value...]] [-debug] [--site site_name] [-?, --help] [--man] [--  
quiet] [-v, --verbose] [-V, --version] {[-r] reservation_id}
```



Additional detail for this command can be found in the man page by issuing `gchres --man` at the command line.

Example 4. Changing the expiration time of a reservation

```
$ gchres -e "2012-06-06  
14:43:02" 1  
Successfully modified 1  
reservation
```

11.4 Deleting Reservations

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

```
grmres [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {-l | -J instance_name | job_id | [-r] reservation_id}
```



Additional detail for this command can be found in the man page by issuing `grmres --man` at the command line.

Example 5. Deleting a reservation by instance (or job id)

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

Example 6. Deleting a reservation by Reservation Id

```
$ grmres 1
Successfully deleted 1
reservation
```

Example 7. Purging stale reservations

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

12.0 Managing Quotes

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

Associated with a quote is the id, the instance name (name of the item being used such as the job id), the amount quoted (assuming full use of the quoted resources or services), the usage record (which contains the usage details), a start and end time for the quote, a duration (how long the item is expected to be used), a boolean indicating whether the quote is pinned or unpinned, and a description. Each guaranteed quote will be associated with one or more saved charge rates. Operations include creating, querying, modifying and deleting quotes. By default, a standard user may only query quotes attributed to them.

Quote queries allow the specification of filter options which narrow down the quotes that will be returned. The query will return all quotes associated with usage records satisfying the filters. For example, Quote Query Filter:=User=scottmo will return all quotes for resources or services allocated to scottmo.

A quote may be pinned (restricted to a particular instance) or unpinned (allowed to be used by any number of different instances). If a quote is pinned and has not been tied to a particular instance when initially created, it will be tied to the first instance that claims it. Once pinned to an instance, it can then be used repeatedly by that same instance until the quote expires, but not by any other instance. If a quote is not pinned, any instances may use the quoted rates while the quote is active.

A quote may be granted a grace period, which is defined as the difference between the validity period of the quote (end time minus start time) and the expected duration of the usage in seconds. The purpose of a grace period is to account for the fact that we may not know precisely when the usage will begin and the quote needs to be valid during the time of completion of the usage in order for the guaranteed charge rates to be applied. One can apply a desired grace period for a quote by setting the end time longer than the specified duration. Alternatively, a grace duration option can be specified with the duration when creating a quote via gquote as a helper to computing a relatively adjusted end time.

A distinction may be made between quotes and quote templates, both of which use the Quote object. A quote will always return a cost estimate and will be associated with a specific usage record. A quote template provides a way to bundle

together a package of special charge rates that can be applied to quotes, reservations and charges. Quote templates use the same Quote object as regular quotes but they are not associated with a usage record and do not generate a quote amount.

In calculating a price, a quote will use (in order of lower to higher precedence) the standard charge rates, the charge rates from a specified quote template, the specified override charge rates, or an externally specified charge amount. In saving guaranteed charge rates, the standard charge rates pertaining to the specified usage record properties will be used unless overridden by a specified quote template or specified charge rates.

There are several key purposes for using quotes and quote templates. First, a quote may be requested to discover the cost of using a resource or service. If this is your sole purpose, then you may want to use the **gquote** command with the `--costOnly` option. Second, a quote can be used to check whether the requestor has sufficient access and funds to use the requested resource. This may be accomplished by invoking the **gquote** command without the `--costOnly` option. Third, a quote or a quote template can be used to lock-in current or specified charge rates for use in future reservations and charges. If the details of the usage are known and you would like to get a quote amount with a quote id that can be referenced to guarantee the quoted charge rates, you may use the **gquote** command with the `--guarantee` option. Override charge rates may be factored in to the cost estimate of the quote by using the **gquote** command with the `--rate` option. If specific override charge rates need to be saved or guaranteed for future use within a quote, reservation or charge without generating a cost estimate, create a pinned quote template by using the **gmkquote** command with the `--pin` and `-R` options. If it is necessary to create a quote template that can be used to override the standard charge rates for multiple instances, use the **gmkquote** command with the `--nopin` and `-R` options.

12.1 Creating Quotes

Quotes are normally generated by the resource management system with the **gquote** command before an instance uses requested resources or services (see [Obtaining Usage Quotes](#)).

12.2 Creating Quote Templates

Quote templates may be created by using the **gmkquote** command. Quote templates provide a way to bundle together a package of special charge rates that can be applied to quotes, reservations and charges.

```
gmkquote [--pin] [-J instance_name|job_id | --nopin] [-s start_time] {-e
end_time | -t quote_duration} [-d description] [-X, --extension prop-
erty_name=property_value [,property_name=property_value...]] {-
R charge_rate_type:charge_rate_name [{charge_rate_instance}]
=charge_rate_amount [,charge_rate_type:charge_rate_name
[ {charge_rate_instance}]=charge_rate_amount...]} [--debug] [--site
site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing **gmkquote --man** at the command line.

Example 1. Creating a pinned quote template

```
$ gmkquote --pin -J vpc.1 -t
86400 -R
VBR:Processors=1.5,
NBM:QualityOfService{Premium}
=1.7
Successfully created 1 quote
template with id 17
```

Example 2. Creating an unpinned quote template

```
$ gmkquote --nopin -t 86400 -R
VBR:Disk=2.5,NBR:License
{Matlab}=4
Successfully created 1 quote
template with id 18
```



Use of the **gmkquote** command will not result in a cost estimate or the creation of a usage record. Use **gquote** instead if you wish to obtain a quote for usage.

12.3 Querying Quotes

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

```
glsquote [-A | -l] [-J instance_name|job_id [-X, --extension property_name=property_value [,property_name=property_value...]] [-u user_name] [-g group_name] [-p project_name] [-m machine_name] [-f, --filter filter_name=filter_value [,filter_name=filter_value...]] [-F, --filter-type (AttributedTo)|ImpingesUpon] [--full] [--show attribute_name [,attribute_name...]] [-l, --long] [-w, --wide] [--raw] [-h, --hours] [--debug] [--site site_name[-?, --help] [--man] [--quiet] [--q] quote_id]
```



The fields which are displayed by default by this command can be customized by setting the `quote.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsquote --man` at the command line.

Example 3. Listing all quotes for user amy on machine colony

```
$ glsquote -u amy -m colony
Id Amount Pinned Instance
UsageRecord StartTime
EndTime Duration
ChargeRates Description
-----
1 57600 True
242 2012-04-06
12:49:53 2012-04-13 13:09:58
3600 VBR:Processors:1
```

12.4 Modifying Quotes

To modify a quote, use the command `gchquote`:

```
gchquote [-s start_time] [-e expiration_time] [-d description] [-X, --  
extension property_name=property_value [,property_name=pro-  
perty_value...]] [-debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v  
| --verbose] [-V, --version] {[-q] quote_id}
```



Additional detail for this command can be found in the man page by issuing `gchquote --man` at the command line.

Example 4. Changing the expiration time of a quote

```
$ gchquote -e "2012-05-01" 1  
Successfully modified 1 quote
```


12.5 Deleting Quotes

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

```
grmquote [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose]
[-V, --version] {-l | [-q] quote_id}
```



Additional detail for this command can be found in the man page by issuing **grmquote --man** at the command line.

Example 5. Deleting a quote

```
$ grmquote 1
Successfully deleted 1 quote
```

Example 6. Purging stale quotes

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

13.0 Managing Usage Records

Moab Accounting Manager can track the usage of resources and services on your system, recording the charge and the details of the usage in a usage record. A usage record is created when a resource or service manager requests a guaranteed quote for usage, places a reservation for usage, or charges for the usage of an item. Usage records can also be created directly via UsageRecord Create (gmkusage). A refund can be invoked to credit a charge amount back to the originating account. Usage records can also be queried, modified or deleted. By default, a standard user may only query usage records attributed to them.

In a typical use case, a quote might be used to discover how much it would cost to use an item (resource or service) and to verify the user had sufficient access to the item and funds to cover the requested usage. Just before the item is about to be used, a reservation (or hold) might be placed against the user's allocated credits for the requested usage. After the usage is complete, a charge for the actual usage can be debited from their account and the reservation removed.

As is the case for other Moab Accounting Manager objects, usage records are highly customizable. One may remove most usage record properties and add new usage record properties. Refer to the section [Customizing the Usage Record Object](#) for examples of customizing usage records.

13.1 Creating a Usage Record

In most cases, usage records will be created by the resource management system via the API or with the `gquote`, the `greserve` or the `gcharge` command.

However, it is also possible to create usage records directly using the `gmksusage` command:

```
gmksusage [-T usage_record_type] [-u user_name] [-g group_name] [-p project_name] [-o organization] [-c class_name] [-Q quality_of_service] [-m machine_name] [-N nodes] [-P processors] [-M memory] [-D disk] [-n network] [-t usage_duration] [-s start_time] [-S service_id] [-e end_time] [-d description] [-X | --extension property=value[,property_name=property_value...]] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [{"-J"} instance_name|job_id]
```



Additional detail for this command can be found in the man page by issuing `gmksusage --man` at the command line.

Example 1. Creating a usage record

```
$ gmksusage -u jsmith -p chem -m
cluster -X Charge=2468 -P 2 -t
1234 -J PBS.1234.0
Successfully created 1 usage
record with id 246
```



The fields which are displayed by default by this command can be customized by setting the `usagerecord.show` configuration parameter in `gold.conf`.



Use of the `gmksusage` command to record usage will not result in the debiting of a user's allocation. Use `gcharge` instead if you wish to charge for the usage.

13.2 Querying Usage Records

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

```
glsusage [-T usage_record_type] [[-J] instance_name_pattern|job_id_pattern] [-u user_name] [-g group_name] [-p project_name] [-o organization_name] [-c class_name] [-m machine_name] [--stage stage] [-s start_time] [-S service_id] [-e end_time] [-X, --extension property_name=property_value [,property_name=property_value...]] [--full] [-show attribute_name [,attribute_name...]] [--raw] [-h, --hours] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --verbose] [[-j] usage_record_id]
```



The fields which are displayed by default by this command can be customized by setting the `usagerecord.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing **glsusage --man** at the command line.

Example 2. Show specific info about usage tallied by amy

```
$ glsusage --show=Type,
Instance,Project,Machine,Charge
-u amy
Type Instance      Project
Machine      Charge
-----
Job    PBS.1234.0  chemistry
colony      22212
```

13.3 Modifying a Usage Record

It is possible to modify a usage record by using the command `gchusage`:

```
gchusage [-T usage_record_type] [-u user_name] [-g group_name] [-p project_name] [-o organization] [-c class_name] [-Q quality_of_service] [-m machine_name] [-N nodes] [-P processors] [-M memory] [-D disk] [-n network] [-t usage_duration] [-s start_time] [-S service_id] [-e end_time] [-d description] [-X, --extension property_name=property_value[,property_name=property_value...]] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-j] usage_record_id | [-J instance_name|job_id]}
```



Additional detail for this command can be found in the man page by issuing `gchusage --man` at the command line.

Example 3. Changing a usage record

```
$ gchusage -Q HalfPrice -X  
Charge=1234 -d "Benchmark" -J  
PBS.1234.0  
Successfully modified 1 usage  
record
```



Changing a recorded charge in this manner will not change the allocated balance (see [Issuing Usage Refunds](#) to refund a charge).

13.4 Deleting a Usage Record

To delete a usage record, use the command `grmusage`:

```
grmjob [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-j] usage_record_id|-J] instance_name|job_id]
```



Additional detail for this command can be found in the man page by issuing `grmusage --man` at the command line.

Example 4. Deleting a usage record

```
$ grmusage -J PBS.1234.0
Successfully deleted 1 usage
record
```

13.5 Obtaining Usage Quotes

Usage quotes can be used to determine how much it will cost to use a resource or service. Provided the cost-only option is not specified, this step will additionally verify that the submitter has sufficient funds and meets all the allocation policy requirements for the usage, and can be used at the submission of the usage request as an early filter to prevent the usage from getting blocked when it tries to obtain a reservation to start 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 quote record and a permanent usage record. A quote id will be returned which can be used with the reservation and charge to claim the quoted charge rates. A cost-only quote can be used to determine how much would be charged for usage without verifying sufficient funds or checking to see if the charge could succeed. A breakdown of the charges in the quote can be returned by specifying the `--itemize` option with the `--verbose` option.

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

```
gquote [-T usage_record_type] [-u user_name] [-g group_name] [-p project_name] [-o organization] [-c class_name] [-Q quality_of_service] [-m machine_name] [-N nodes] [-P processors] [-M memory] [-D disk] [-n network] [-X, --extension property name=property_value[,property_name=property_value...]] [-t quote_duration] [-G grace_duration] [-s quote_start_time] [-S service_id] [-e quote_end_time] [-d quote_description] [-z quote_amount] [--cost-only | --guarantee] [-R charge_rate_type:charge_rate_name[[charge_rate_instance]=charge_rate_amount[,charge_rate_type:charge_rate_name[[charge_rate_instance]=charge_rate_amount...]]] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-j] usage_record_id] [-q quote_template_id] [-J] instance_name|job_id
```



Additional detail for this command can be found in the man page by issuing **gquote --man** at the command line.

Example 5. Requesting a quote

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

Example 6. Requesting a guaranteed quote

13.6 Making Usage Reservations

A usage reservation can be used to place a hold on the user's account before usage starts to ensure that the credits will be there when it completes. The `replace` option may be specified if you want the new reservation to replace existing reservations of the same instance name (associated with the same usage record). The `modify` option may be specified to dynamically extend any existing reservation with the same instance name with the specified characteristics instead of creating a new one. See [Managing Reservations](#) for more information about these options.

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

```
greserve [-T usage_record_type] [-u user_name] [-g group_name] [-p project_name] [-o organization] [-c class_name] [-Q quality_of_service] [-m machine_name] [-N nodes] [-P processors] [-M memory] [-D disk] [-n network] [-X, --extension property name=property_value[,property_name=property_value...]] [-t reservation_duration] [-G grace_duration] [-s reservation_start_time] [-S service_id] [-e reservation_end_time] [-d reservation_description] [-z reservation_amount] [--replace|--modify] [-R charge_rate_type:charge_rate_name [{charge_rate_instance}]=charge_rate_amount[,charge_rate_type:charge_rate_name [{charge_rate_instance}]=charge_rate_amount...]] [--itemize] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [-q quote_id] [[-j] usage_record_id] {-J instance_name|job_id}
```



Additional detail for this command can be found in the man page by issuing `greserve --man` at the command line.

Example 7. Creating a reservation

```
$ greserve -J PBS.1234.0 -p
chemistry -u amy -m colony -P 2
-t 3600
Successfully reserved 7200
credits with reservation id 37
for instance PBS.1234.0 and
created usage record id 87
```

13.7 Charging for Usage

A usage charge debits the appropriate allocations based on the attributes of the usage. The charge is calculated based on factors including the resources and services used, the usage time, and other quality-based factors (see [Managing Charge Rates](#)). By default, any reservations associated with the charge will be removed. The incremental option may be specified if you want associated reservations to be reduced instead of removed. If a usage record already exists for the instance being charged it will be updated with the data properties passed in with the charge request, otherwise a new usage record will be created.

A quote id can be specified to use a previously quoted set of charge rates. This will also ensure the charge will update the usage record instantiated with the quote. A reservation id can be specified to help match up a charge with its reservation (this may assist in deleting the correct reservation if instance ids are not unique). This will also ensure the charge will update the usage record that may have been instantiated by the reservation.

Although, by default, Moab Accounting Manager will calculate the charge for the usage using its default charge rates or using the charge rates saved by a referenced quote or quote template, it is possible to specify override charge rates via the rate option. Alternatively, it is possible to designate an externally calculated charge by specifying the charge amount with the Charge option (-z option to **gcharge**).

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

```
gcharge [-T usage_record_type] [-u user_name] [-g group_name] [-p project_name] [-o organization] [-c class_name] [-Q quality_of_service] [-m machine_name] [-N nodes] [-P processors] [-M memory] [-D disk] [-n network] [-x usage_state] [-X, --extension property_name=property_value[,property_name=property_value...]] [-t charge_duration] [-s charge_start_time] [-S service_id] [-e charge_end_time] [-d charge_description] [-z charge_amount] [--incremental] [-R charge_rate_type:charge_rate_name{charge_rate_instance}=charge_rate_amount[,charge_rate_type:charge_rate_name{charge_rate_instance}=charge_rate_amount...]}] [-h, --hours] [--itemize] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-j] usage_record_id] [-q quote_id] [-r reservation_id] {-J instance_name|job_id}
```



Additional detail for this command can be found in the man page by issuing **gcharge --man** at the command line.

Example 8. Issuing a usage charge

```
$ gcharge -J PBS.1234.0 -p  
chemistry -u amy -m colony -P 2  
-t 1234  
Successfully charged 2468  
credits for instance PBS.1234.0  
1 reservation was removed
```

13.8 Issuing Usage Refunds

A charged amount can be credited back in part or in whole by issuing a usage refund. This action attempts to lookup the referenced usage record 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 instance names (such as job ids) are non-unique), this command will return the list of matched usage records with unique ids so that the correct usage record can be specified for the refund.

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

```
grefund [-z amount] [-a account_id] [-d description] [-A] [-h, --hours] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [-J instance_name|job_id] [-j] usage_record_id
```



Additional detail for this command can be found in the man page by issuing **grefund --man** at the command line.

Example 9. Issuing a usage refund

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

13.9 Customizing the Usage Record Object

The usage record object as natively defined can be customized with the attributes you want to track in your use cases. The chapter on [Customizing Objects](#) goes into some detail on the customization syntax. However, since this may be a common requirement, this section will provide a few examples on modifying, adding and deleting usage record attributes and getting them to be tracked and show up in queries.

Usage record discriminators are those properties which are considered primary differentiators between usage, reservation and quote records. Usage record discriminators are used in the dynamic web portal as filters for the listing, modification and deletion of usage records, reservations and quotes. The default usage record discriminators are Type, User, Group, Project, Organization, Class, QualityOfService and Machine. Any new attributes added to the usage record object will become usage record discriminators. Removing a discriminator attribute from the usage record object will necessarily remove it as a usage record discriminator as well. It will be necessary to log out and back in after adding or removing a discriminator in order for it to be reflected in the web GUI.

Example 10. Setting VM as the default Usage Record Type

As installed, the usage record type defaults to "Job". The default value can be set to NULL if there should be no default value, or to any other default value. This example will demonstrate how to set the default usage record type to "VM".

```
$ goldsh Attribute Modify
Object=UsageRecord Name=Type
DefaultValue=VM
Successfully modified 1
attribute
```

Example 11. Adding a UsageRecord Application Field (and discriminator)

Let's say you would like to track the application run by your usage scenarios. First, you would add Application as an Attribute of the UsageRecord Object.

```
$ goldsh Attribute Create
Object=UsageRecord
Name=Application
DataType=String
Successfully created 1
attribute
```

If you want the new attribute to show up in `glsusage`, you must add it to the `usagerecord.show string` in `gold.conf`.

```
$ vi /opt/mam/etc/gold.conf
usagerecord.show = Id,Instance,
User,Group,Organization,
Project,QualityOfService,
Machine,Stage,Charge,
Processors,Nodes,Application,
Duration,StartTime,EndTime,
Description
```

If you want to filter the usage records by Application, (such as listing all usage records associated with the specified application), use the `-X` (or `--extension`) option in `glsusage`.

```
$ glsusage -X Application=foo -
-show=Type,Instance,Charge,
User,Application
Type Instance Charge User
Application
-----
Job PBS.1234.0 19744 amy foo
```

You could also use Application as the basis of a ChargeRate. See Name-Based Resources or Name-Based Multipliers in the [Managing Charge Rates](#) chapter for details on how to do this.

Although the initial step above allows the application value to be tracked in the usage record, it is also possible to add it as an attribute of the Transaction table so that it will be automatically populated from actions having assignments, conditions, options and data values referring to the Application.

```
$ goldsh Attribute Create
Object=Transaction
Name=Application
DataType=String
Successfully created 1
```

```
attribute
```

Additionally, the `gstatement` client command can show Application as one of its discriminators (which are Project, User and Machine by default) in its debit detail. These statement discriminators are specified by the `--show` argument to `gstatement` and can be configured with the `statement.show` configuration parameter in `gold.conf`.

Example 12. Removing the UsageRecord Class Field

Let's say you were not interested in tracking the class. First, you would delete Class as an Attribute of the UsageRecord Object.

```
$ goldsh Attribute Delete
Object==UsageRecord Name==Class
Successfully deleted 1
attribute
```

Next, we need to make sure `glsusage` doesn't try to list the class.

```
$ vi /opt/mam/etc/gold.conf
usagerecord.show = Id,Instance,
User,Project,Machine,
QualityOfService,Stage,Charge,
Processors,Nodes,Application,
Duration,StartTime,EndTime,
Description
```

If the attribute you want to delete is also an attribute in the Transaction table, you could delete it from there as well.

13.10 Usage Record Property Verification

If a usage record property has an object associated with it, you may want to verify that when that usage record property is specified in a scheduling action (Charge, Reserve, Quote), it verifies that that property is a valid instance of its object type. You can apply a simple verification to a usage record property by setting the property's Values attribute to an '@' sign followed by the name of the object.

Example 13. Ensure that a machine specified in a charge actually exists

```
$ goldsh Attribute Modify
Object==UsageRecord
Name==Machine Values=@Machine
Successfully modified 1
attribute
```

See [Managing Attributes](#) for more information about setting the Values attribute.

13.11 Usage Record Property Defaults

It is possible to set defaults for usage record properties when they are not specified in the usage data for a charge, reservation or quote. There are two cases which must be considered -- when the property has an object associated with it and when the property does not.

If a property does not have an object associated with it, simply set the DefaultValue attribute for the property's UsageRecord Attribute object to the desired value.

Example 14. Setting a system-wide simple default class of batch for usage functions

```
$ goldsh Attribute Modify
Object==UsageRecord Name==Class
DefaultValue=batch
Successfully modified 1
attribute
```

If a property does have an object associated with it, you will need to both set the DefaultValue attribute for the property's UsageRecord Attribute object to the desired value AND set the DefaultValue attribute for the corresponding object to the desired value.

Example 15. Setting a system-wide simple default machine of cloud for usage functions

```
$ goldsh Attribute Modify
Object==UsageRecord
Name==Machine
DefaultValue=cloud
Successfully modified 1
attribute

$ goldsh Attribute Modify
Object==UsageRecord
Name==Machine
DefaultValue=cloud
Successfully modified 1
attribute
```

See [Global Object-based Defaults](#) for more information about setting default values for objects. See [Local Attribute-based Defaults](#) for more information about setting default values for attributes.

13.12 Usage Record Property Auto-Generation

It is possible for usage record properties which have object definitions to automatically create the referenced objects the first time they are encountered in a usage function (charge, reserve or quote). To do this, the referenced object must be set to AutoGen=True and the Values attribute for the UsageRecord attribute corresponding to the object must be set to a string consisting of the '@' sign followed by the object name.

Example 16. Setting the Usage Record Type to auto-generate Items for usage functions

For example, let's assume there were many usage record types that could be charged for (Food, Book, Haircut) and that you had already created an Item object. It would be possible to automatically generate a new Item instance each time a new usage record type was referenced in a charge operation.

```
$ goldsh Object Modify
Name==Item AutoGen=True
Successfully modified 1 object

$ goldsh Attribute Modify
Object==UsageRecord Name==Type
Values=@Item
Successfully modified 1
attribute
```

See [Object Auto-Generation](#) for more information about the auto-generation of objects.

13.13 Usage Record Property Instantiators

It is possible to establish a dynamic correlation between usage record properties in which one usage record property can instantiate another. For example, if a user is specified in a charge but no project is specified then the user's default project should be applied to the account constraints and logged; or if a project is specified in a charge but not its organization then the organization corresponding to that project should be looked up and applied to the account constraints and logged. Three usage record property instantiator types are currently supported and are configured by prefixing the property instance's Values foreign object reference with the appropriate characters: Assign if not defined (@?=), Assign if not different (@!=), Assign always (@:=). We shall look at each of these individually and in different terms.

- **Applying a correlated default (@?=)** -- If property X is specified with the value x in the usage record and property Y is not specified in the usage record and if the object instance referred to by x has a correlated default value of y' for property Y', then y' will be applied as the default value for property Y in the usage record. For example, we could establish the notion of a default project for a user.

Example 17. Establishing a default project for a user

First we add a DefaultProject attribute (the name is arbitrary) to the User object and give it a Values property of @?=Project.

```
$ goldsh Attribute Create
Object=User
Name=DefaultProject
DataType=String
Values="\@?=Project\"
Description="\Default
Project\"
Successfully created 1
attribute
```

Then we can establish the default project for user scottmo to be chemistry.

```
User Modify Name==scottmo
DefaultProject=chemistry
Successfully modified 1 user
```

Subsequently, when a Charge, Reserve or Quote is issued that specifies the User scottmo but does not specify the Project, the Project chemistry Project will be applied to the charge as if originally specified in the usage record charge data.

- **Applying a correlated verification (@!=)** -- If property X is specified with the value x in the usage record and property Y is specified with the value y in the usage record and if the object instance referred to by x has a correlated verification value of y' for the property Y' and if y' does not equal y, then fail with an error message. Additionally, if property X is specified with the value x in the usage record and property Y is not specified in the usage record and if the object instance referred to by x has a correlated verification value of y' for property Y', then y' will be applied as the default value for property Y in the usage record. For example, we could establish a parent-child relationship between organizations and projects in which explicitly specified incongruities result in a failure.

Example 18. Establishing a verification hierarchy with projects and organizations

First we add a VerifyOrganization attribute (the name is arbitrary) to the Project object and give it a Values property of @!=Organization.

```
$ goldsh Attribute Create
Object=Project
Name=VerifyOrganization
DataType=String
Values="\">@!=Organization\"
Description="\">Verify
Organization\"
Successfully created 1
attribute
```

Then we can establish the verify organization for project chemistry to be sciences.

```
$ goldsh Project Modify
Name==chemistry
VerifyOrganization=sciences
Successfully modified 1
project
```

Subsequently, when a Charge, Reserve or Quote is issued that specifies the Project chemistry and specifies the wrong Organization (e.g. arts), the transaction will fail with an error message. Additionally, when a Charge, Reserve or Quote is issued that specifies the Project chemistry but does not specify the Organization, the Organization sciences will be applied to the charge as if originally specified in the usage record charge data.

- **Applying a correlated override (@:=)** -- If property X is specified with the value x in the usage record and if the object instance referred to by x has a correlated override value of y' for property Y', then y' will be applied as the

override value for property Y in the usage record. For example, we could establish a parent-child relationship between organizations and projects in which explicitly specified incongruities are silently overridden with the value from the child.

Example 19. Establishing an override hierarchy with projects and organizations

First we add an OverrideOrganization attribute (the name is arbitrary) to the Project object and give it a Values property of @:=Organization.

```
$ goldsh Attribute Create
Object=Project
Name=OverrideOrganization
DataType=String
Values="\ "@:=Organization\"
Description="\ Override
Organization\"
Successfully created 1
attribute
```

Then we can establish the override organization for project chemistry to be sciences.

```
$ goldsh Project Modify
Name==chemistry
OverrideOrganization=sciences
Successfully modified 1
project
```

Subsequently, when a Charge, Reserve or Quote is issued that specifies the Project chemistry and specifies either the wrong Organization (e.g. arts) or no Organization, the Organization sciences will be silently applied to the charge as if originally specified in the usage record charge data.

14.0 Managing Itemized Charges

The itemized charge table provides an ability to display the components of a composite charge in a line item format. Each charge transaction will write the components of its charge into the charge record so that you can get a line-item breakdown of each charge for usage including the names, values, rates, scaling factors, charge amounts and details listed for each component of the charge. This capability is enabled by setting `charge.itemization = true` in the `goldd.conf` (it is false by default).

Itemized charges may only be queried. They are created automatically in charge transactions and there are no command line clients to change or remove them.

Additionally, an `itemize` option can be specified for quotes, reservations and charges to include an itemized charge breakdown in the response data instead of a single line with the amount.

14.1 Querying Itemized Charges

To display itemized charge information, use the command `glscharge`:

```
glscharge [-j usage_record_id] [-J instance_name] [-n usage_property_name] [-s start_time] [-e end_time] [--full] [-- show attribute_name[,attribute_name...]] [-- raw] [-h, --hours] [-- debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing `glscharge --man` at the command line.

Example 1. Listing all itemized charge information

```
$ glscharge
UsageRecord Instance Name
Value Duration Rate
ScalingFactor Amount
CreationTime Description
-----
-----
-----
-----
-----
24          job.1      Storage
100      86400      1.157e-07 1
          1 2012-04-05
17:49:41
25          job.2      Processors
4        86400      5.787e-07 1
          20 2012-04-05
17:49:42
25          job.2      Memory
4096    86400      1.13e-08 1
          4 2012-04-05
17:49:42
26          job.3      Processors
1        86400      5.787e-05 1
          5 2012-04-05
17:49:43
26          job.3      Memory
1004    86400      1.13e-08 1
          1 2012-04-05
17:49:43
```



14.2 Displaying Itemized Charges for a Transaction

In addition to the itemized charge table, Moab Accounting Manager captures the itemized charges for usage record charges, reservations and guaranteed quotes in the details of the transaction. The itemized charges show the details for the formula used to calculate the charge for the transaction. To display the itemized charges for a scheduling transaction, parse the details from the command `glstxn --full -A Charge|Reserve|Quote`:

Example 14. Extract the itemized charges for a job charge

```
$ glstxn -A Charge -J
PBS.1234.1 -q --show Details |
perl -pe 's/.*(ItemizedCharges
[^,]*).*\/\1/'
ItemizedCharges:= ( ( ( 16
[Processors] * 1 [ChargeRate
{VBR}{Processors}] ) + ( 2000
[Memory] * 0.001 [ChargeRate
{VBR}{Memory}] ) ) * 1234
[Duration] ) = 22212
```

15.0 Managing Charge Rates

Charge Rates establish how much to charge for usage. Charge rates are applied when usage properties matching the charge rate names are found in the usage data. In order for a charge rate of a given name to be applied, a usage record attribute of the same name must exist.

There are four major categories of charge rates: Resource, Usage, Multiplier, and Fee. These are distinguished by the way they are factored into the charge calculation. Resource charge rates are additive charges that are multiplied by the amount of time that they are used in seconds. Usage charge rates are additive charges that are not multiplied by time. Multiplier charge rates apply multipliers to the sum of the Resource and Usage charges. Fee charge rates are added after the multipliers have been applied.

Each of the major charge rate types has two sub-types: value-based and name-based.

- Name-based charge rates are used for usage properties that take strings for values (e.g. QualityOfService=premium or Project=chemistry). The charge rate that is applied will be determined by a lookup of the usage property value to see if there is a matching charge rate value. A default rate may be specified by creating a name-based charge rate with an empty charge rate value. Multiple values may be assigned to the same rate via separate charge rate definitions or by combining the values in a single charge rate value separated by commas.
- Value-based charge rates are used for usage properties that take numbers for values (e.g. Processors=2 or CpuTime=12.67). The charge rate that is applied will be multiplied by the usage property value. The charge rate value is commonly left blank to be taken as the default rate for the full range of usage property values. A particular value may also be specified as the charge rate value which means that that rate will only be used if the usage property value exactly matches the charge rate value. A half-bounded expression may be used by specifying a less than or greater than sign with an optional equal sign, followed by the number. For example, the charge rate value ≤ 4 would match a usage property value of x if $x \leq 4$. A charge rate value may also be specified as a range (of the form $\langle \text{number} \rangle [- \langle \text{number} \rangle]$). For example, the range 1-4 would be match a usage property value of x if $1 \leq x \leq 4$. If you need to be more specific about the boundedness of the ranges, you may replace the dash with a less than sign with an optional equal sign on either side of it to indicate whether the endpoints are included. For example, the range $1 < 4$ would match if $1 < x < 4$, $1 \leq 4$ would match if $1 \leq x < 4$, $1 < \leq 4$ would match if $1 < x \leq 4$ and $1 \leq \leq 4$ would match if $1 \leq x \leq 4$. So you might use ranges like $1 \leq 2$, $2 \leq 4$, $4 \leq 8$, and ≥ 8 . Multiple values or value ranges hav-

ing the same charge rate may be specified in a single expression separated by commas.

Thus there are eight composite types of charge rates referred to by their acronyms: VBR (Value-Based Resource), NBR (Name-Based Resource), VBU (Value-Based Usage), NBU (Name-Based Usage), VBM (Value-Based Multiplier), NBM (Name-Based Multiplier), VBF (Value-Based Fee) and NBF (Name-Based Fee).

- **Value-Based Resource** – Value-Based Resource (or Consumable Resource) Charge Rates define how much it costs per unit of time to use a consumable resource like processors, memory, telescope time, generic resources that are charged per time used, etc. These resource metrics must first be multiplied by the usage duration in seconds before being added to the total charge. Value-Based Resource Charge Rates are of Type "VBR", with the Name being the resource name (such as Processors) and the given Rate (such as 1) being multiplied by the consumed resource value (such as 8).
- **Name-Based Resource** – Name-Based Resource Charge Rates define how much it costs per unit of time to use a named resource like license, etc. The cost for the named resource must first be multiplied by the usage duration in seconds before being added to the total charge. Name-Based Resource Charge Rates are of Type "NBR", with the Name being the resource name (such as License), the Value being the resource value (such as matlab), and having the given Rate (such as 5).
- **Value-Based Usage** – Value-Based Usage Charge Rates define how much to charge for metrics of total resource usage such as cputime, power consumed, generic resources or licenses that are charged flat fees per use, etc. These usage metrics are added to the total charge without being multiplied by the duration. Value-Based Usage Charge Rates are of Type "VBU", with the Name being the resource name (such as Power) and the given Rate (such as .001) being multiplied by the consumed resource value (such as 40000).
- **Name-Based Usage** – Name-Based Usage Charge Rates define how much it costs to use a named attribute having a flat charge such as feature, etc. These usage metrics are added to the total charge without being multiplied by multiplied by the duration. Name-Based Usage Charge Rates are of Type "NBU", with the Name being the resource name (such as Feature), the Instance being the usage value (such as GPU), and having the given flat Rate (such as 200).
- **Value-Based Multiplier** – Value-Based Multiplier Charge Rates are scaled multipliers which apply a multiplicative charge factor based on a numeric scaling factor. These incoming scaling factors are multiplied against the Value-Based Multiplier Rate and then are multiplied against the total of the resource and usage charges. Value Based Multiplier Charge Rates are of Type "VBM", with the Name being the multiplier name (such as Discount) and

the given Rate (such as 1) being multiplied with the scaling factor (such as .5) before being multiplied to the total charge.

- **Name-Based Multiplier** – Name-Based Multiplier Charge Rates are quality based multipliers which apply a multiplicative charge factor based on a quality of the usage such as quality of service, nodetype, class, user, time of day, etc. These charge multipliers are determined by a hash or lookup table based on the value of the usage attribute. These rates are multiplied against the total of the resource and usage charges. Name-Based Multiplier Charge Rates are of Type "NBM", with the Name being the multiplier name (such as QualityOfService), the Value being the quality instance (such as Premium), and having the given multiplier Rate (such as 2).
- **Value-Based Fee** – Value-Based Fee Charge Rates define how much to charge for scaled or enumerated fees such as setup fees, shipping charges, etc. which should be added after the multipliers are applied. These fees are added to the total charge. Value-Based Fee Charge Rates are of Type "VBF", with the Name being the fee name (such as Shipping) and the given Rate (such as 25) being multiplied by the scaling or counted value (such as 4).
- **Name-Based Fee** – Name-Based Fee Charge Rates define how much it costs to use a named attribute having a flat charge such as feature, etc. which should be added after the multipliers are applied. These fees are added to the total charge. Name-Based Fee Charge Rates are of Type "NBF", with the Name being the fee name (such as Zone), the Value being the fee value (such as Asia), and having the given flat Rate (such as 100).

By default, usage charges are calculated according to the following formula: For each value-based resource charge rate matching a usage property in the usage record data, a value-based resource charge is calculated by multiplying the usage property value by the charge rate and by the duration of time it was used. For each name-based resource charge rate matching a usage property name and value in the usage record data, a name-based resource charge is calculated by multiplying the charge rate by the duration of time it was used. For each value-based usage charge type matching a usage property in the usage record data, a value-based usage charge is calculated by multiplying the usage property value by the charge rate. For each name-based usage charge type matching a usage property name and value in the usage record data, a name-based usage charge is given by the charge rate. These value-based and name-based resource charges and the value-based and name-based usage charges are added together. Then, for each value-based multiplier charge rate matching a usage property in the usage record data, a value-based multiplier is calculated by multiplying the usage property value of the charge rate. For each name-based multiplier charge rate matching a usage property name and value in the usage record data, a name-based multiplier is given by the charge rate. The sum of the resource and usage charges is then multiplied by the product of the applicable value-based and name-based multipliers.

Next, for each value-based fee charge type matching a usage property in the usage record data, a value-based fee charge is calculated by multiplying the usage property value by the charge rate. For each name-based fee charge type matching a usage property name and value in the usage record data, a name-based fee charge is given by the charge rate for that fee. Finally, these value-based and name-based fee charges are added to the multiplied usage charge subtotal.

In short, the formula can be represented by $(((((\sum(VBR*value)+\sum(NBR)+\sum(MVBR*value))*duration)+(\sum(VBU*value)+\sum(NBU))) * \Pi(VBM*value)*\Pi(NBM))+(\sum(VBF*value)+\sum(NBF)))$.

15.1 Creating Charge Rates

To create a new charge rate, use the command `gmkrate`:

```
gmkrate -n charge_rate_name [-x charge_rate_value] -T charge_rate_type [-d description] [-- debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[ -z] charge_rate_amount}
```



Additional detail for this command can be found in the man page by issuing `gmkrate --man` at the command line.



If a usage record attribute does not exist for the name of the charge rate you are creating, you must first create the corresponding usage record property. See [Customizing the Usage Record Object](#).

Example 1. Creating a value-based resource charge rate

```
$ gmkrate -T VBR -n Memory -z  
0.001  
Successfully created 1 charge  
rate
```

Example 2. Creating a name-based resource charge rate

```
$ gmkrate -T NBR -n License -x  
Matlab -z 5  
Successfully created 1 charge  
rate
```

Example 3. Creating a value-based usage charge rate

```
$ gmkrate -T VBU -n CpuTime -z  
1  
Successfully created 1 charge  
rate
```

Example 4. Creating a name-based usage charge rate

```
$ gmkrate -T NBU -n Feature -x  
GPU -z 200  
Successfully created 1  
charge rate
```

Example 5. Creating a value-based multiplier charge rate

```
$ gmkrate -T VBM -n Discount -z  
1  
Successfully created 1 charge  
rate
```

Example 6. Creating a couple of name-based multiplier charge rates and a default rate

```
$ gmkrate -T NBM -n  
QualityOfService -x Premium -z  
2  
Successfully created 1 charge  
rate
```

```
$ gmkrate -T NBM -n  
QualityOfService -J  
BottomFeeder -z 0.5  
Successfully created 1 charge  
rate
```

```
$ gmkrate -T NBM -n  
QualityOfService -z 1  
Successfully created 1 charge  
rate
```

Example 7. Creating a value-based fee charge rate

```
$ gmkrate -T VBF -n Shipping -z  
25  
Successfully created 1 charge  
rate
```

Example 8. Creating a name-based fee charge rate

```
$ gmkrate -T NBF -n Zone -x  
Asia -z 200  
Successfully created 1 charge  
rate
```

Example 9. Creating a couple of conditional value-based resource charge rates

```
$ gmkrate -T VBR -n Disk -x
User=dave? -z 0.2
Successfully created 1 charge
rate
```

```
$ gmkrate -T Disk -n User -x
User=mike? -z 0.5
Successfully created 1 charge
rate
```

Example 10. Creating some value-based resource charge rate ranges and a default

```
$ $ gmkrate -T VBR -n
Processors -x 1-4 -z 2
Successfully created 1 charge
rate
```

```
$ gmkrate -T VBR -n Processors
-x 5-8 -z 1.5
Successfully created 1 charge
rate
```

```
$ gmkrate -T VBR -n Processors
-z 1
Successfully created 1 charge
rate
```

Example 11: Creating some value-based usage charge rate ranges for floating point values

```
$ $ gmkrate -T VBU -n Power -x
'<2' -z 0.005
Successfully created 1 charge
rate
```

```
$ $ gmkrate -T VBU -n Power -x
'2=<4' -z 0.004
Successfully created 1 charge
rate
```

```
$ $ gmkrate -T VBU -n Power -x
'>=4' -z 0.003
Successfully created 1 charge
```



```
rate
```

Example 12: Assigning multiple classes to run for free

```
$ $ gmkrate -T NBM -n Class -x  
dev,test -z 0  
Successfully created 1 charge  
rate
```

15.2 Querying Charge Rates

To display charge rate information, use the command **glsrate**:

```
glsrate [-n charge_rate_name] [-x charge_rate_value] [-T charge_rate_type] [--full] [--show attribute_name][,attribute_name...] [--raw] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing **glsrate --man** at the command line.

Example 13. Listing all charge rates

```
$ glsrate
Name          Value
Type   Rate   Description
-----
-----
CpuTime
VBU     1
Discount
VBM     1
Disk          User=dave?
VBR     0.2
Disk          User=mike?
VBR     0.5
Feature       GPU
NBU     200
License       Matlab
NBR     5
Memory
VBR     0.001
Power
VBU     0.001
Processors
VBR     1
Processors    1-4
VBR     2
Processors    5-8
VBR     1.5
QualityOfService NBM
1
QualityOfService BottomFeeder
```

```
NBM      0.5
QualityOfService  Premium
NBM      2
Shipping
VBF      25
Zone           Asia
NBF      200
```

15.3 Modifying Charge Rates

To modify a charge rate, use the command **gchrates**:

```
gchrates [-n choice="plain" charge_rate_name [-x charge_rate_value] [-T charge_rate_type] [-z charge_rate_amount] [-d description] [-- debug] [-- site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing **gchrates --man** at the command line.

Example 14. Changing a charge rate

```
$ gchrates -T VBR -n Memory -z  
0.05  
Successfully modified 1 charge  
rate
```

15.4 Deleting Charge Rates

To delete a charge rate, use the command **grmrate**:

```
grmrate [-n choice="plain"charge_rate_name] [-x charge_rate_instance]  
[-- debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version]
```



Additional detail for this command can be found in the man page by issuing **grmrate --man** at the command line.

Example 15. Deleting a charge rate

```
$ grmrate -T VBR -n Memory  
Successfully deleted 1 charge  
rate
```

16.0 Managing Transactions

Moab Accounting Manager logs all modifying transactions in a detailed transaction journal (queries are not recorded). Previous transactions can be queried but not modified or deleted. By default, a standard user may only query transactions performed by them.

16.1 Querying Transactions

To display transaction information, use the command **glstrans**:

```
glstrans [-O object] [-A action] [-k primary_key_value] [-U actor] [-a account_id] [-i allocation_id] [-u user_name] [-p project_name] [-m machine_name] [-j usage_record_id] [-J instance_name|job_id] [-s start_time] [-e end_time] [-T transaction_id] [-R request_id] [-X, --extension property_name=property_value [,property_name=property_value...]] [-show attribute_name [,attribute_name...]] [--raw] [-h, --hours] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --version]
```



The fields which are displayed by default by this command can be customized by setting the `transaction.show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glstrans --man` at the command line.

Example 1. List all deposits made in 2012

```
$ glstrans -A Deposit -s 2012-01-01 -e 2013-01-01
```

Example 2. List everything done by amy since the beginning of 2012

```
$ glstrans -U amy -s 2012-01-01
```

Example 3. List all transactions related to job moab.1

```
$ glstrans -J moab.1
```

Example 4. List all transactions affecting charge rates

```
$ glstrans -O ChargeRate
```

16.2 Customizing the Transaction Object

The transaction record as natively defined can be customized with the attributes you want to track in your use cases. It is possible to add additional attributes to the Transaction table so that it will be automatically populated from actions having assignments, conditions, options and data values referring to the attribute.

Transaction discriminators are those properties which are considered primary differentiators between transaction records (besides the metadata differentiators of object, action and instance). Transaction discriminators are used in the dynamic web portal as filters for the listing of transaction records. The default transaction discriminators are User, Project and Machine. Any new attributes added to the Transaction object will become transaction discriminators. Removing a discriminator attribute from the transaction object will necessarily remove it as a transaction discriminator as well. It will be necessary to log out and back in after adding or removing a discriminator in order for it to be reflected in the web GUI.

Example 5. Adding an Organization field to the Transaction record (which also makes it a discriminator)

```
$ goldsh Attribute Create
Object=Transaction
Name=Organization
DataType=String
Successfully created 1
attribute
```


17.0 Managing Events

Moab Accounting Manager has an internal event scheduler that can be configured to execute Moab Accounting Manager actions at a designated time in the future or on a periodic basis. Valid actions on an event include Create, Query, Fire, Modify, Refresh and Delete. Event attributes include Id, FireCommand, ArmTime, FireTime, RearmPeriod, EndTime, Notify, RearmOnFailure, FailureCommand, CatchUp and Description.

There are two server configuration parameters which affect event scheduling: event.scheduler which specifies whether the event scheduler is enabled or not (it is disabled by default) and event.pollinterval which is the period in minutes that the event scheduler uses to fire events. The poll interval must divide evenly into the number of minutes in a day (1440).

The command(s) to be fired by an event are expressed in a serialized form of the request identical to the syntax used in the interactive control program (goldsh). There are two commands that can be configured in an event: the FireCommand which is the command to be executed when the event is fired, and the FailureCommand which is the command to be executed if the fired command results in an unsuccessful response status. The FireTime is the target time for the event to be triggered by the event scheduler. The actual fire time may be dependent on the state of the server and will be recorded in the CreationTime property of the corresponding "Event Fire" Transaction. An event may also be fired manually with the Event Fire action.

The RearmPeriod is a time period expression specifying when the event will be rearmed. This period expression is of the form: "<period>[[@<instant>][~|^]!]" where period may be something like 1 day, 2 hours, or 5 minutes. Instant locks the period to a specific instant within the time period such as 1 day @ hour 12 or 1 month @ day 3. The modifiers indicate whether the time period should be relative to now (!), or relative to the start of this (~) designator (month or minute, etc.), or relative to the start of the first (^) designator (month or minute, etc.). For example, assuming the FireTime was 7:15, if you specified "4 hours !" as the rearm period it would be rearmed at 11:15, if you specified "4 hours ~" as the rearm period it would be rearmed at 11:00, and if you specified "4 hours ^" as the rearm period it would be rearmed at 8:00.

The ArmTime is the time the event was last armed or fired. This field is used as a reference time to be able to derive how long the event has been waiting to happen. This field will be initially set to mark the moment the first FireTime is set and updated thereafter to indicate the last time the event was fired. In the case where an event does not have a FireTime set, this field may be set manually and used in a similar manner. If we consider the time between event firings as "laps", this could be thought of as the Lap Start Time. If the RearmOnFailure boolean is set to False, the event will not be rearmed if the command was unsuccessful. If set

to True, the event will be evaluated for rearming even if the command response has a status of Failure. The standard default is False. If the CatchUp boolean is set to True and the server was down during the time this event should have fired, the event scheduler will attempt to make up for the past due events by progressively firing them (rearming based on previous arm time) until catching up to the present. The actions will still show as having occurred in the present rather than in the past. If set to False, and the server is brought back up after an outage, the event scheduler will still fire immediately for a past due event, but it will only fire once and then rearm relative to the current time.

A Notification method can be specified via the Notify parameter and is of the form: [+|=][<delivery_method>:] [<recipient>][,[+|=][<delivery_method>:][<recipient>]]*. If the term is a -, the notification is sent only on failure. If the term is a +, the notification is sent only on success. Otherwise the notification is always sent. There can be multiple notify expressions separated by a comma. All applicable notifications will be sent. See the chapter on [Managing Notifications](#) for more information about delivery method and recipient.


```
10 Payment End Id:=5 2012-04-
10 13:15:39 2012-04-09 13:15:40
      store:amy
False
True 2012-04-09 13:15:40 End
Payment
```

17.2 Deleting Events

To delete an event, use the command **grmevent**:

```
grmevent [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-E] event_id}
```



Additional detail for this command can be found in the man page by issuing **grmevent --man** at the command line.

Example 2. Deleting an event

```
$ grmevent 9
Successfully deleted 1 event
```

18.0 Managing Notifications

When event commands are executed (asynchronously), the success or failure of the operation is communicated back to the initiator via a notification. When an event is created, you may specify the Notify option which will associate a notification method with the event. Currently there is only one DeliveryMethod implemented which is Store. With the Store delivery method, command response information is stored as instances of the Notification object. These messages can later be retrieved by the initiator via a Notification Query. Payments can also route a notification method down to their associated events via a Notify option.

The notification attributes include Id (autogenerated), Type, Event, Status, Code, Message, Key, Recipient, EndTime and CreationTime. Stored notifications can be queried on any of these conditions. The notification type distinguishes what type of command resulted in the notification (Fire or Failure). The notification key is the value of the primary key of the object instance that the command acted on (e.g. the Payment Id). The recipient could be a user name or any tag that identifies the intended reader for the notification. The Notification Query supports a Delete option, which if set to True, will delete the notifications after they have been queried. Additionally, stored notification have an EndTime after which they are automatically deleted by Gold. The Notification actions include Send, Refresh, Create, Query, Delete and Modify.

There are two server configuration parameters which affect notifications: notification.deliverymethod which dictates which deliverymethod is used by default if unspecified and notification.duration which defines how long notifications stick around if the Store delivery method is used.

18.1 Querying Notifications

To display event information, use the command `glsnot`:

```
glsnot [-E event_id] [-T notification_type] [-k primary_key_value] [-k primary_key_value] -u recipient] [-x status] [-s start_time] [-e end_time] [--delete] [--full] [--show attribute_name[,attribute_name...]] [--raw] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-V, --version] [[-N] notification_id]
```



The fields which are displayed by default by this command can be customized by setting the `event-show` configuration parameter in `gold.conf`. Additional detail for this command can be found in the man page by issuing `glsnot --man` at the command line.

Example 1. Listing all failure notifications

```
$ glsnot -x Failure
Id Event Type Status Code
Message

Key
Recipient EndTime
CreationTime
-----4 20 Fire
Failure 782 Payment Begin
failed starting payment: Failed
creating payment starting
reservation: Insufficient
balance to reserve usage
(Instance Moab.1)\nClearing the
```

```
event fire time.\n\nThe controlling event has been deleted. 9 amy 2012-04-23 13:35:01 2012-04-09 13:35:01se True

2012-04-09 13:15:09 Use Payment
4 20 Fire Failure 782
Payment Begin failed starting payment: Failed creating payment starting reservation: Insufficient balance to reserve usage (Instance Moab.1)
\n\nClearing the event fire time.\n\nThe controlling event has been deleted. 9 amy 2012-04-23 13:35:01 2012-04-09 13:35:01
```


18.2 Deleting Notifications

To delete a notification, use the command `grmnot`:

```
grmnot [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose]
[-V, --version] {[-N] notification_id}
```



Additional detail for this command can be found in the man page by issuing `grmnot --man` at the command line.

Example 2. Deleting a notification

```
$ grmnot 4
Successfully deleted 1
notification
```

Example 3. Deleting all successful notifications

To delete many notifications, query them with the `--delete` option:

```
$ glsnot -x Success --delete
Id Event Type Status Code
Message

Key
Recipient EndTime
CreationTime
-----4 20 Fire
Failure 782 Payment Begin
failed starting payment: Failed
creating payment starting
```

```
reservation: Insufficient
balance to reserve usage
(Instance Moab.1)\nClearing the
event fire time.\nThe
controlling event has been
deleted. 9 amy 2012-04-
23 13:35:01 2012-04-09
13:35:01se True
```

```
2012-04-09 13:15:09 Use
Payment
1 11 Fire Success 000
Payment Begin: Successfully
charged 10 credits for instance
Moab.1\nSuccessfully charged 20
credits for instance
Moab.2\nSuccessfully charged 20
credits for instance
Moab.3\nSuccessfully started
payment (6) and created 3
reservations\nClearing the
event fire time.\nThe
controlling event has been
deleted. 6 scottmo 2012-04-
23 13:28:02 2012-04-09 13:28:02
2 14 Fire Success 000
Payment Begin: Successfully
charged 10 credits for instance
Moab.1\nSuccessfully charged 20
credits for instance
Moab.2\nSuccessfully charged 20
credits for instance
Moab.3\nSuccessfully started
payment (7) and created 3
reservations\nClearing the
event fire time.\nThe
controlling event has been
deleted. 7 amy 2012-04-
23 13:31:02 2012-04-09 13:31:02
3 17 Fire Success 000
Payment Begin: Successfully
charged 10 credits for instance
Moab.1\nSuccessfully charged 20
```

```
credits for instance
Moab.2\nSuccessfully charged 20
credits for instance
Moab.3\nSuccessfully started
payment (8) and created 3
reservations\nClearing the
event fire time.\nThe
controlling event has been
deleted. 8 amy 2012-04-
23 13:32:02 2012-04-09 13:32:02
Successfully deleted 3
notifications
```

19.0 Managing Roles

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

The actions for a role consist of a set of tuples of object, action and instance permitted by the role. In other words, each role action defines an object (whether specific or ANY), the action that can be taken on that object (whether specific or ANY) and the instance of the object that action can be taken on (whether specific or ANY).

In the base configuration, there are three default roles: SystemAdmin, Anonymous and OVERRIDE. Other configurations, such as the bank configuration, add additional roles. Roles can be added as desired. The three base roles are required for proper function of Moab Accounting Manager and should not be deleted. By default, the SystemAdmin role can perform any action on any object. This role is usually assigned to the super user. The Anonymous role is intended to define the actions available to your standard unprivileged user. This may include the ability to set your password, query certain public objects and modify objects that belong to you (implemented via the OVERRIDE role). The OVERRIDE role is a special role type that defines those actions that should use special business logic intrinsic to the routine that handles that object and action. For example, in the bank configuration, the OVERRIDE logic for the Project Query routine will only allow the standard user to see information about projects for which he or she is a member. A given user's privileges will be the superset of the actions of all roles that apply to that user.

The instance indicates which specific instances of the object the action can be performed on. There are several special instance types that can be used in certain situations. The ANY instance is supported by all objects and permits the specified action on all instances of the specified object. The SELF instance applies to the user's own instance if the object is User, or to objects that have a User attribute associated with the user. The MEMBERS instance applies to objects for which the user is a direct member. The ADMIN instance applies to objects for which the user is designated as an administrator. Unless otherwise specified, the instance will default to a value of ANY.

19.1 Creating Roles

To create a new role, use the command **gmkrole**. Users and actions may be associated with the role at creation time. When assigning actions to a role, the object, action and instance must be specified in the form shown. Multiple actions or users may be specified for the role.

```
gmkrole [-d description] [-u user_name[,user_name...]] [-A object_name->action_name[[instance_name]] [,object_name->action_name[[instance_name]]...]] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [{"-r"} role_name]
```



Additional detail for this command can be found in the man page by issuing **gmkrole --man** at the command line.

Example 1. Creating a Manager role

```
$ gmkrole -r Manager -d  
"Manages Roles and  
Responsibilities" -v  
Successfully created 1 role
```

19.2 Querying Roles

To display the role information, use the command `glsrole`:

```
glsrole [--full] [--show attribute_name[,attribute_name...]] [-l, --long]
[-w, --wide] [--raw] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-
V, --version] [[-r] role_name]
```



Additional detail for this command can be found in the man page by issuing `glsrole --man` at the command line.

Example 2. Listing all roles along with users and descriptions

```
$ glsrole --show=Name,Users,
Description
Name          Users
Description
-----
Anonymous     ANY      Things
that can be done by anybody

OVERRIDE      ANY      A custom
authorization method will be
invoked

ProjectAdmin  Can
update or view a project they
are admin for

Schedule      root     Scheduler
relevant Transactions

SystemAdmin   scottmo  Can
update or view any object

UserServices  User
Services
```

Example 3. Listing information about the scheduler role

```
$ glsrole -l Scheduler
Name          Users  Actions
Description
```

```
-----  
-----  
-----  
-----  
Scheduler    root  
UsageRecord->Create (ANY)  
Scheduler relevant Transactions  
  
UsageRecord->Quote (ANY)  
  
UsageRecord->Reserve (ANY)  
  
UsageRecord->Charge (ANY)  
  
Reservation->Delete (ANY)
```

19.3 Modifying Roles

To modify a role, use the command **gchrole**:

```
gchrole [-d description] [--AddUser(s) user_name{,user_name...}] [--addAction(s) object_name->action_name [{instance_name}][,object_name->action_name [{instance_name}]...]}] [--delUser(s) user_name[,user_name...]}] [--del-action(s) object_name->action_name [{instance_name}][,object_name->action_name [{instance_name}]...]}] [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-r] role_name]
```



Additional detail for this command can be found in the man page by issuing **gchrole --man** at the command line.

Users may be added to a role or removed from a role. Actions also may be added to a role or removed from a role. When specifying actions, the instance will default to a value of ANY.

Example 4. Adding a user to a role

Let's add dave to our new Manager role:

```
$ gchrole --add-user dave -r
Manager
Successfully added 1 user
```

Example 5. Associating an action with a role

Allow the Manager to change role responsibilities:

```
gchrole --add-action
"RoleAction->ANY" Manager -v
Successfully added 1 action
```


19.4 Deleting Roles

To delete a role, use the command `grmrole`:

```
grmrole [--debug] [-S, --site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] [[-r] role_name]
```



Additional detail for this command can be found in the man page by issuing `grmrole --man` at the command line.

Users may be added to a role or removed from a role. Actions also may be added to a role or removed from a role. When specifying actions, the instance will default to a value of ANY.

Example 6. Deleting the Manager role

Let's add dave to our new Manager role:

```
$ grmrole Manager
Successfully deleted 1 role and
2 associations
```

20.0 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. A `gchpasswd` command line client exists to aid a user or administrator in setting or changing a password. Other operations (deleting or listing password entries) must be performed using the interactive control program (`goldsh`). By default, a standard user may only set or change their own password. A system administrator may set or change any user's password.



Because Moab Accounting Manager caches password information for faster responsiveness, it will be necessary to restart the server after running `gchpasswd` for the GUI to accept that password change.

20.1 Setting Passwords

To set a new password, use the command `gchpasswd`. If the user name is not specified via an option or as the unique argument, then the invoking user will be taken as the user whose password will be set. The invoker will be prompted for the new password.

```
gchpasswd [--debug] [--site site_name] [-?, --help] [--man] [--quiet] [-v, --verbose] [-V, --version] {[-u] user_name}
```



Additional detail for this command can be found in the man page by issuing `gchpasswd --man` at the command line.

Example 1. Setting a password

```
$ gchpasswd amy
Enter your new password:
Successfully created 1 password
```

20.2 Querying Passwords

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

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

Example 2. List the users who have set passwords

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

20.3 Deleting Passwords

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

goldsh Password Delete User==<User Name>]



The goldsh control program allows you to make powerful and sweeping modifications to Moab Accounting Manager objects. Misuse of this command could result in the inadvertent deletion of all passwords.

Example 3. Deleting a password

```
$ goldsh Password Delete
User==amy
User      Password
-----
-----
amy
HZYzwD20o1XIE/gxRYyFKP2sumkCluH-
m
Successfully deleted 1 password
```

21.0 Using the Gold Shell (goldsh)

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



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. Inadvertant mistakes could result in modifications that are very difficult to reverse.

21.1 Usage

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

goldsh [--debug] [--site *site_name*] [-?, --help] [--man] [--raw] [--quiet] [-v, --verbose] [-V, --version] [*<Command>*]



Additional detail for this command can be found in the man page by issuing **goldsh --man** at the command line.

Example 1. Specifying the command as direct arguments

```
$ goldsh System Query

Name
Version  Description
-----  -
-
Moab Accounting Manager
7.0.0 Commercial Release
```

Example 2. Using the interactive prompt

```
$ goldsh

gold> System Query

Name                Version
Description
-----  -
-----
Moab Accounting Manager  7.0.0
Commercial Release
gold> quit
```

Example 3. Reading commands from a file

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

```
$ goldsh <commands.gold
```

```
Name
```

```
Version  Description
```

```
-----  -----
```

```
-  -----
```

```
Moab Accounting Manager
```

```
7.0.0  Commercial Release
```


21.2 Command Syntax

goldsh commands are of the form:

```
<Object> [=<Alias>] [, <Object> [=<Alias>]...] <Action> [ [<Con-  
junction>] [<Open_Parenthesis>...] [<Object>.] <Name> <Operator>  
 [<Subject>.] <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 ')'.
'

21.3 Valid Objects

To list the objects available for use with commands in goldsh commands, use the goldsh command: **Object Query**

Example 4. Listing all objects

```
gold> Object Query Show:="Sort
(Name) "
Name
-----
Account
AccountAccount
Action
Allocation
Attribute
ChargeRate
Constrains
Machine
Object
Organization
Password
Project
ProjectUser
Quote
QuoteChargeRate
Reservation
Role
RoleAction
RoleUser
System
Transaction
UsageRecord
User
```

21.4 Valid Actions for an Object

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

Example 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
```

21.5 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 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="CreditLimit"
  [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="<Integer Number>Integer
Number}</Where>]
  [<Where name="Account"
[op="EQ|NE|GT|GE|LT|LE|-
Match|NotMatch (EQ) "]
[conj="And|Or (And) "]
[group="<Integer Number>Account
Name}</Where>]
  [<Where name="StartTime"
[op="EQ|NE|GT|GE|LT|LE (EQ) "]
[conj="And|Or (And) "]
[group="<Integer Number>YYYY-
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="<Integer Number>YYYY-
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="<Integer Number>Decimal
Number}</Where>]
  [<Where name="CreditLimit"
[op="EQ|NE|GT|GE|LT|LE (EQ) "]
[conj="And|Or (And) "]
[group="<Integer Number>Decimal
Number}</Where>]
  [<Where name="Deposited"
[op="EQ|NE|GT|GE|LT|LE (EQ) "]
[conj="And|Or (And) "]
[group="<Integer Number>Decimal
Number}</Where>]
  [<Where name="Active"
[op="EQ|NE (EQ) "] [conj="And|Or
(And) "] [group="<Integer
Number>True|False</Where>]
  [<Where name="Description"
[op="EQ|NE|GT|GE|LT|LE|-
Match|NotMatch (EQ) "]

```

```

[conj="And|Or (And) "]
[group="<Integer
Number>Description}</Where>]
  [<Option
name="Filter">True|False
(False)</Option>]
    [<Option
name="F-
ilter-
Type">Exclusive|NonExclusive
(NonExclusive)</Option>]
      [<Option
name="-
IncludeAncestors">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>]
              [<Option
name="Offset">Integer Number}
</Option>]
                [<Option
name="ShowHidden">True|False
(False)</Option>]
                  [<Option
name="ShowUsage">True|False
(False)</Option>]
</Request>

```

21.6 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.

21.7 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.

21.7.1 Query Action

The Query action is used to query objects. It accepts selections 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 (defaulting to 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

Additionally, aliases can be applied to selections so that columns can be renamed as desired. Aliases are expressed by adding "=<Alias>" to the target attribute name (and after the trailing parenthesis of the aggregation, if specified).

For example: Allocation Query Show:="GroupBy)Account),Sum(Amount)=Total"

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

!~ Does not match

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 7. Return the number of inactive reservations

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

21.7.2 Create Action

The Create action is used to create a new object. It accepts assignments 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 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
```

21.7.3 Modify Action

The Modify action is used to modify existing objects. It accepts conditions 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
- !~ Does not match

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 9. Change/set scottmo's phone number and email address

```
gold> User Modify Name==scottmo
PhoneNumber="(509) 376-2204"
Email-
Address="scottmo@daptivecomputing.com"
Name          Active      CommonName
              PhoneNumber
              EmailAddress
              DefaultProject
Description
-----
```

```

-----
-----
-----
-----
scottmo      True      Jackson,
Scott M.    (509) 376-2204

scottmo@adaptivecomputing.com

Successfully modified 1 user

```

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

```

gold> Reservation Modify
EndTime+=864000
Project==chemistry
Id Account      Amount
Instance      UsageRecord  User
Project      Machine      EndTime
              Description
-----
-----
-----
1      2          57600
PBS.1234.0  1              amy
chemistry  colony        2012-04-
06 10:47:30
Successfully modified 1
reservation

```

21.7.4 Delete Action

The Delete action is used to delete objects. It accepts conditions 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
- !~ Does not match

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 11. Get rid of the pesky Jacksons

```
gold> User Delete
CommonName~"Jackson*"
Name           Active      CommonName
                PhoneNumber
                EmailAddress
                DefaultProject
Description
-----
scottmo        True       Jackson,
Scott M.       (509) 376-2204
scottmo@adaptivecomputing.gov
Successfully deleted 1 user and
1 association
```

21.7.5 Undelete Action

The Undelete action is used to restore deleted objects. It accepts conditions 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

!~ Does not match

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 12. Resurrect the deleted users that were active

```
gold> User Undelete
Active==True
Name          Active      CommonName
              PhoneNumber
              EmailAddress
              DefaultProject
Description
-----
scottmo      True        Jackson,
Scott M.      (509) 376-2204
scottmo@adaptivecomputing.com
```

```
Successfully undeleted 1 user  
and 1 association
```


21.8 Multi-Object Queries

Goldsh 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 13. Print the sums for active balance and allocated amounts grouped by project

```
gold> Allocation,Constraint
Query
Show:="GroupBy
(Constraint.Value=Project),Sum
(Allocation.Amount=Balance),Sum
(Allocation.Deposited=Allocation)"

Con-
straint.A-
ccount==Allocation.Account
Constraint.Name==Project
Allocation.Active==True

Project      Balance
Allocation
-----
-----
biology      193651124
360000000
chemistry    296167659
360000000
```

Example 14. Show all active projects for amy's privileges

```
gold> RoleUser,RoleAction Query
Show:="RoleAction.Object,
RoleAction.Name=Action"
RoleUser.Role==RoleAction.Role
&& ( RoleUser.Name==amy ||
RoleUser.Name==ANY )
Unique:=True

Object      Action
```

```

-----
Account          Balance

Account          Query
AccountAccount  Query
Action           Query
Allocation       Query
Attribute        Query
ChargeRate       Query
Constraint       Query
Machine          Query
Object           Query
Organization     Query
Password         ANY
Project          Query
ProjectUser      Query
Quote            Query
QuoteChargeRate Query
Reservation      Query
ReservationAllocation Query
Role             Query
RoleAction       Query
RoleUser         Query
System           Query
Transaction      Query
UsageRecord      Query
User             Query

```



Although the forgoing was a good example of a join request, it should be understood that it is not a straightforward way to determine the full extent of a user's privileges. Some of the actions may be tied to specific object instances and many of them are associated with an override method which may not actually permit the user access to any instances of the object. Using `Show:="RoleUser.Role,RoleUser.Name=User,RoleAction.Object,RoleAction.Name=Action,RoleAction.Instance"` may be revealing in this regard. See the chapter on [Managing Roles](#) for more information about managing roles.

22.0 Customizing Objects

Moab Accounting Manager provides the ability to dynamically create new objects or customize or delete existing objects through the interactive control program (goldsh).



The object customizations described in this chapter will be noticeable in subsequent goldsh queries (and in the web GUI after a fresh login). For installations with a database that supports multiple connections (e.g. PostgreSQL) these changes will be visible immediately while others (e.g. SQLite) will require the server to be restarted. Client commands may need to be modified to properly interact with changed objects or attributes.



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. Inadvertent mistakes could result in modifications that are very difficult to reverse.

22.1 Managing Objects

In Moab Accounting Manager, Objects correspond to tables in the repository which have Attributes (such as Name and Color) and Actions (such as Query and Modify). A specific instance of an object is described as an Instance and has Properties (the specific values of the attributes for that object). The instance is uniquely referred to via its primary key(s) (such as its Name or Id).

An object must have a name and may have a description. An object may be set to auto-generate its instances when first seen (see [Object Auto-Generation](#)) and/or a default value may be designated for the object (see [Global Object-based Defaults](#)).

Objects may reference other objects. If a single instance of an object references only a single instance of another object (for example, a usage record may only have one user), then it is sufficient for the first object to have an attribute field for the second object (the UsageRecord object has an attribute called User). However, if there may be a many-to-many relationship between objects (for example, a project may have multiple users and a user may belong to multiple projects), then it is necessary to maintain a separate object as an association table (e.g. ProjectUser). When creating an association object, the object should be given an appropriate name (e.g. ProjectUser), it should be marked as an association (Association=True), and an object needs to be designated for the parent (e.g. Project) and the child (e.g. User). The association object itself may have additional attributes that provide qualitative information about the association (e.g. a particular ProjectUser association may be active or be an administrator).

22.1.1 Creating a Custom Object

To create a new object, use the command **goldsh Object Create**. When an object is created, the 5 default actions are automatically created for the object: Create, Delete, Modify, Query and Undelete. A number of default metadata attributes are created as well: CreationTime, ModificationTime, Deleted, RequestId and TransactionId. These attributes are normally hidden in regular queries.

```
goldsh Object Create Name=<Object Name> [AutoGen=True|(False)] [Default-Value=<Default Value>] [Description=<Description>] [Association=True|False] [Child=<Child Object>] [Parent=<Parent Object>][ShowUsage:=True]
```

Example 1. Creating a Node Object

```
$ goldsh Object Create
Name=Node Description=\"Node
Information\"
Successfully created 1 object
and 5 actions
```

Example 2. Add a node name attribute

```
$ goldsh Attribute Create
Object=Node Name=Name
DataType=String PrimaryKey=True
Successfully created 1
attribute
```

Example 3. Add a processor count attribute

```
$ goldsh Attribute Create
Object=Node Name=Processors
DataType=Integer
Successfully created 1
attribute
```

22.1.2 Querying Objects

To display object information, use the command **goldsh Object Query**.

goldsh Object Query [Name=<Object Name>] [Show:=Name,AutoGen,DefaultValue,Description,Association,Parent,Child] [ShowUsage:=True]

Example 4. List Information for the Node Object

```
$ goldsh Object Query
Name==Node

Name Association Parent
Child DefaultValue AutoGen
Description
-----
- -----
-----
Node False
False Node
Information
```

22.1.3 Modifying an Object

It is possible to modify an object by using the command **goldsh Object Modify**.

goldsh Object Query [Name=<Object Name>] [AutoGen=True|False] [Default-Value=Default Value] [Description=Description] [Association=True|False] [Child=Child Object] [Parent=Parent Object] [ShowUsage:=True]

Example 5. Changing the Node object's description

```
$ goldsh Object Modify
Name==Node Description="\Host
Information\"
Successfully modified 1 object
```

22.1.4 Deleting an Object

To delete an object, use the command **goldsh Object Delete**. When an object is deleted, all associated attributes, actions and other associations are automatically deleted as well.

goldsh Object Delete [Name=<Object Name>] [ShowUsage:=True]

Example 6. Deleting the Node Object

```
$ goldsh Object Delete
Name==Node
Successfully deleted 1 object
```



This is a very dangerous operation and could result in the deletion of all object definitions requiring database repair. The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. Be sure to specify conditions for the object you want to delete.

22.1.5 Object Auto-Generation

It is possible to have object instances be automatically generated the first time they are referenced in designated contexts. For example, you might want a user be auto-generated when newly added to a project. You could have an organization auto-generated when specified as the default for a user. You could have a cost-center be auto-generated when referenced in a usage record. To do this, the referenced object must be set to `AutoGen=True` and the `Values` property for the attribute that you want to trigger the auto-generation must be set to a string consisting of the '@' sign followed by the object name.

Example 7. Auto-generate a machine's organization

For example, let's assume that your machines belong to specific organizations that you may want to run a report against but you don't want to define all of the organizations up front. It would be possible to automatically generate a new organization instance each time an undefined organization is specified for a machine.

```
$ goldsh Object Modify
Name==Organization AutoGen=True
Successfully modified 1 object
```

```
$ goldsh Attribute Modify
Object==Machine
Name==Organization
Values=@Organization
Successfully modified 1
attribute
```

See [Usage Record Property Auto-Generation](#) for a discussion of auto-generating objects referenced in usage records.

22.1.6 Global Object-Based Defaults

It is possible to set a global default for an object that will be applied to all attributes referencing this object. When a new instance of an object is being created which has an attribute referring to another object via its Values property, if that attribute has not been specified and you want it to default to the global default, you will need to set the DefaultValue attribute for the referenced object to the desired value.

Example 8. Setting a system-wide simple default organization called general

```
$ goldsh Object Modify
Name==Organization
DefaultValue=general
Successfully modified 1 object
```

Thereafter each (non-association) object which has an attribute with a Values property set to @Organization will default to general if that attribute is not specified. Perhaps we would want the default value to be taken for the organization when a new project is created.

```
$ goldsh Attribute Modify
Object==Project
Name==Organization
Values=@Organization
Successfully modified 1
attribute
```

See [Local Attribute-based Defaults](#) for more information about setting default values for attributes. See [Usage Record Property Defaults](#) for more information about setting default values for usage record properties.

22.2 Managing Attributes

Objects can have any number of fields called Attributes. When an object is first created, a number of attributes are created for the object by default. These are: CreationTime (time the object was first created), ModificationTime (time the object was last updated), Deleted (whether the object is deleted or not), RequestId (request id that resulted in the last modification of the object), TransactionId (transaction id that resulted in the last modification of the object).

An attribute must have a name and be associated with an object.

An attribute will have a data type which can be one of (AutoGen, TimeStamp, Boolean, Float, Integer, Currency, String) and defaults to String. A data type of AutoGen means the field will be a primary key of type integer which will assume the next auto-incremented value from the `g_key_generator` table. TimeStamps are epoch times stored in integer format. Booleans are strings constrained to the values of True or False (or unset). Float is used to store decimal or floating point values. Currency is like Float but may have special business logic for handling currency values.

An object may have zero or more attributes which are primary keys (PrimaryKey==True), the combination of which are used to uniquely identify an object instance. Moab Accounting Manager will try to ensure that there can only be one object instance with the exact same set of values of its primary keys.

A required attribute (Required==True), must be either specified or be derived via a default value or other dynamic mechanism when the object is created. It can also not be unset.

A fixed attribute (Fixed==True), may not be changed from its initial value.

An attribute may be constrained to certain values via the Values attribute. The values may be constrained to members of a list expressed as a parenthesized comma-delimited list of strings (i.e. Values="(Brazil,China,France,Russia,USA)"). Alternatively, the values may be constrained to be an instance of a particular object type (like a foreign key constraint) by assigning to the Values attribute the name of an object prefixed by the '@' sign (e.g. Values="@Project" -- which would constrain the value of this attribute to be a valid project name). Stronger versions of the @-prefixed object-constrained values may be used in Quote, Reserve and Charge actions to enforce dynamic interactions between usage record properties such as to assign default values if not defined (e.g. Values="@?=Project"), verification values which evoke an error if they differ (e.g. Values="@!=Project"), or designated values which always overwrite the value (e.g. Values="@:=Project"). See [Usage Record Property Instantiators](#) for more information.

A default value may be assigned to an attribute via the DefaultValue attribute. When a new instance of an object is created, if a property is not specified for the attribute, the default value will be used.

The Sequence attribute determines which order an object's attributes will be listed in for queries if no selection list is specified in the query. Attributes with smaller sequence numbers will appear before attributes with larger sequence numbers. The Sequence attribute is also used to enforce a proper attribute display ordering in the web GUI.

The Hidden attribute specifies whether an attribute should be shown in a query by default or not. Hidden attributes can be seen in queries by specifying the ShowHidden option with a value of True.

The Description field is a location to describe the meaning of the attribute and is used in the GUI for field descriptions.

22.2.1 Adding an Attribute to an Object

To create a new attribute for an object, use the command **goldsh Attribute Create**:

```
goldsh Attribute Create Object=<Object Name> Name=<Attribute Name>
[DataType=AutoGen|TimeStamp|Boolean|Float|Integer|Currency|(String)] [Pri-
maryKey=True|(False)] [Required=True|(False)] [Fixed=True|(False)] [Values=<For-
eign Key or List of Values>] [DefaultValue=<Default Value>]
[Sequence=<Integer Number>] [Hidden=<True|(False)>]
[Description=<Description>] [ShowUsage:=True]
```

Example 9. Adding a Country Attribute to User

```
$ goldsh Attribute Create
Object=User Name=Country
Values="\ (Brazil,China,France,
Russia,USA\)" DefaultValue=USA
Successfully created 1
attribute
```

Example 10. Tracking Submission Time in Usage records

```
$ goldsh Attribute Create
Object=UsageRecord
Name=SubmissionTime
DataType=TimeStamp
Successfully created 1
attribute
```

22.2.2 Querying Attributes

To display attribute information, use the command **goldsh Attribute Query**:

```
goldsh Attribute Query Object=<Object Name> Name=<Attribute Name>
[Show:=Object,Name,DataType,PrimaryKey,Required,Fixed,Values,
DefaultValue,Sequence,Hidden,Description] [ShowHidden:=True] [ShowUsage:=True]
```

Example 11. List the attributes of the Node object

```
$ goldsh Attribute Query
Object==Node

Object Name
DataType PrimaryKey Required
Fixed Values DefaultValue
Sequence Hidden Description
-----
-
Node Processors Integer
False False False
20 False

Node Name String
True True True
10 False

Node TransactionId Integer
False False True
990 True
Last Modifying Transaction Id
Node RequestId Integer
False False True
980 True
Last Modifying Request Id
Node Deleted Boolean
False False True
970 True
Is this object deleted?
```

```
Node      ModificationTime
TimeStamp False          False
True      960
      True      Last Updated
```

```
Node      CreationTime
TimeStamp False          False
True      950
      True      First Created
```

22.2.3 Modifying an Attribute

To modify an attribute, use the command **goldsh Attribute Modify**:

goldsh Attribute Modify Object==<Object Name> Name==<Attribute Name>
[Required=True|(False)] [Fixed=True|(False)] [Values=<Foreign Key or List
of Values>] [DefaultValue=<Default Value>] [Sequence=<Integer
Number>] [Hidden=<True|(False)>] [Description=<Description>] [Sho-
wUsage:=True]



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. A mistake made using this command could result in the inadvertent modification of all attributes.

Example 12. Change User Organization values to not be restricted to the set of organization instances

```
$ goldsh Attribute Modify  
Object==User Name==Organization  
Values=NULL  
Successfully modified 1  
attribute
```


22.2.4 Removing an Attribute from an Object

To delete an attribute, use the command **goldsh Attribute Delete**:

goldsh Attribute Delete Object==<Object Name> Name==<Attribute Name>
[ShowUsage:=True]



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. A mistake made using this command could result in the inadvertent deletion of all attributes.



When using Moab Accounting Manager as an Allocation Manager, certain objects and attributes are assumed to exist. For example, a call to UsageRecord Charge would fail if you had deleted the Allocation Amount attribute. The Attribute Undelete command might come in useful in such a case.

Example 13. Removing the Organization attribute from Machine

```
$ goldsh Attribute Delete
Object==Machine
Name==Organization
Successfully deleted 1
attribute
```

Example 14. Perhaps we don't care to track the QualityOfService attribute in a Usage record

```
$ goldsh Attribute Delete
Object==UsageRecord
Name==QualityOfService
Successfully deleted 1
attribute
```

22.2.5 Local Attribute-Based Defaults

It is possible to set a specific default for an object attribute that will be applied when an instance of that object is created but the attribute is not specified. This type of default is intended for attributes which do not refer to another object or which should vary from the object global default. This default value is assigned to an attribute via the DefaultValue attribute. When a new instance of the associated object is created, if a property is not specified for the attribute, the specified default value will be used. A local attribute default will have precedence over a global object default.

goldsh Attribute Delete Object==<Object Name> Name==<Attribute Name> [ShowUsage:=True]

Example 15. Setting a default organization just for the project object

```
$ goldsh Attribute Modify
Object==Project
Name==Organization
DefaultValue=university
Successfully modified 1
attribute
```

Example 16. Setting a default phone for the user object

```
$ goldsh Attribute Modify
Object==User Name==Phone
DefaultValue="\NoPhone\"
Successfully modified 1
attribute
```

See [Global Object-based Defaults](#) for more information about setting default values for objects.

See [Usage Record Property Defaults](#) for more information about setting default values for usage record properties.

22.3 Managing Actions

Moab Accounting Manager defines which actions can be performed by which objects. When an object is first created, five basic actions are created for the object by default. These are: Create, Modify, Query, Delete and Undelete. Specific code must exist in Moab Accounting Manager modules in order for objects to support additional actions.

An action is uniquely specified by its name and the object with which it is associated. An action also has a description and a boolean display attribute which governs whether this action should be displayed in the web GUI or not.

22.3.1 Adding an Action to an Object

To specify that an action is allowed for an object, use the command **goldsh Action Create**:

goldsh Action Create Object=<*Object Name*> Name=<*Action Name*> [Display=True|(False)] [Description=<*Description*>] [ShowUsage:=True]

Example 17. Adding a Modify Action to Transaction

```
$ goldsh Action Create
Object=Transaction Name=Modify
Description=Modify
Successfully created 1 action
```

22.3.2 Querying Actions

To display action information, use the command **goldsh Action Query**:

```
goldsh Action Query [Object==<Object Name>] [Name==<Attribute Name>]  
[Show:=Object,Name,Display,Description] [ShowUsage:=True]
```

Example 18. List the actions of the Node object

```
$ goldsh Action Query  
Object==Node  
Object Name      Display  
Description  
-----  
Node   Create   False   Create  
Node   Delete   False   Delete  
Node   Modify   False   Modify  
Node   Query    False   Query  
Node   Undelete  False  
Undelete
```

22.3.3 Modifying an Action

To modify an action, use the command **goldsh Action Modify**:

goldsh Action Modify [Object==<Object Name>] [Name==<Attribute Name>]
[Display=True|(False)] [Description=<Description>] [ShowUsage:=True]



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. A mistake made using this command could result in the inadvertent modification of all actions.

Example 19. Display all Node actions but Undelete in the web GUI

```
$ goldsh Action Modify
Object==Node Name!=Undelete
Display=True
Successfully modified 4 actions
```

22.3.4 Removing an Action from an Object

To delete an action from an object, use the command **goldsh Action Delete**:

goldsh Action Delete [Object==<Object Name>] [Name==<Attribute Name>]
[ShowUsage:=True]



The goldsh control program allows you to make powerful and sweeping modifications to many objects with a single command. A mistake made using this command could result in the inadvertent modification of all actions.



When using Moab Accounting Manager as an allocation manager, certain actions are assumed to exist. Be careful what you delete!

Example 20. Do not allow projects to be deleted

```
$ goldsh Action Delete  
Object==Project Name==Delete  
Successfully deleted 1 action
```

22.4 Examples Creating Custom Objects

Creating a custom object normally involves defining a new object and adding attributes to the object.

Example 1. Creating a License object to track license usage and charges.

Invoke the Moab Accounting Manager control program in interactive mode.

```
$ goldsh
```

Create the License Object.

```
gold> Object Create
Name=License
Description=License
Successfully created 1 object
and 5 actions
```

Next, define its attributes. Give each record a unique id (so the record can be more easily modified), a license type that can be one of (Matlab,Mathematica,Compiler,AutoCAD,Oracle), the user who is using it, the start and end time, how many instances of the license were used, and how much was charged.

```
gold> Attribute Create
Object=License Name=Id
DataType=AutoGen
PrimaryKey=True
Description="Record Id"
Successfully created 1
attribute
gold> Attribute Create
Object=License Name=Type
DataType=String Required=True
Values="(Matlab,Mathematica,
Compiler,AutoCAD,Oracle)"
Fixed=True Description="License
Type"
Successfully created 1
attribute
gold> Attribute Create
Object=License Name=User
Required=True Values="@User"
Description="User Name"
Successfully created 1
attribute
```



```
gold> Attribute Create
Object=License Name=StartTime
DataType=TimeStamp
Description="Start Time"
Successfully created 1
attribute
gold> Attribute Create
Object=License Name=EndTime
DataType=TimeStamp
Description="End Time"
Successfully created 1
attribute
gold> Attribute Create
Object=License Name=Count
DataType=Integer
Description="Number of Licenses
Used"
Successfully created 1
attribute
gold> Attribute Create
Object=License Name=Charge
DataType=Currency
Description="Amount Charged"
Successfully created 1
attribute
```

Finally, since we would like to manage licenses from the web GUI, set `Display=True`.

```
gold> Action Modify
Object=License Name!=Undelete
Display=True
Successfully modified 4 actions
```

When done, exit the `goldsh` prompt.

```
gold> quit
```

That's about it. Licenses should now be able to be managed via the GUI and `goldsh`. The data source will need to use one of the methods of interacting with Moab Accounting Manager (see [Interaction Methods](#)) in order to push license record usage info to Moab Accounting Manager.

Apart from being used as an Allocation Manager, Moab Accounting Manager can be used as a generalized information service. It can be used to manage just about any object-oriented information over the web. For example, Moab Accounting Manager

could be used to provide meta-schedulers with machine/user mappings, or node/resource information.

Example 2. Using Moab Accounting Manager as a Grid Map File.

Invoke the goldsh control program in interactive mode.

```
$ goldsh
```

Create the GridMap Object.

```
gold> Object Create
Name=GridMap
Description="Online Grid Map
File"
Successfully created 1 object
and 5 actions
```

Next, define its attributes. Each entry will consist of a userid (which will serve as the primary key) and a required public X.509 certificate.

```
gold> Attribute Create
Object=GridMap Name=User
PrimaryKey=True Values=@User
Description="User Name"
Successfully created 1
attribute
gold> Attribute Create
Object=GridMap Name=Certificate
DataType=String Required=True
Description="X.509 Public Key"
Successfully created 1
attribute
```

Exit the goldsh prompt.

```
gold> quit
```

From this point, a peer service will need to use one of the methods of interacting with Moab Accounting Manager (see [Interaction Methods](#)) in order to query the GridMap information.

23.0 Integration with the Resource Management System

23.1 Dynamic versus Delayed Accounting

23.1.1 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 Moab Accounting Manager 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.

23.1.2 Dynamic Accounting

Moab Accounting Manager's design allows it to interact dynamically with your resource management system. Charges for resource utilization can be made immediately when the usage ends or incrementally. Additionally, reservations can be issued when the usage begins to place a hold against the user's account, thereby ensuring that usage will only be permitted to start if it has sufficient reserves to complete. The remainder of this document will describe the interactions for dynamic accounting.

23.2 Interaction Points

23.2.1 Usage Quote @ Usage Creation Time [Optional – Recommended]

When a usage request is submitted to a grid or cloud scheduler or resource broker, it may be useful to determine how much it will cost to use a particular resource or service by requesting a usage quote. If the quote is guaranteed, 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 usage request is exclusively serviced locally, it is useful to obtain a quote at the time of submission to ensure the user has sufficient funds to satisfy the usage request and that it meets the access policies necessary for the charge to succeed. A warning can be issued if funds are low or the usage 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 or usage request might wait in the queue for days to be scheduled only to fail when it tries to obtain a reservation at its start time.

To make a usage quote with Moab Accounting Manager at this phase requires that:

- the grid or cloud scheduler has built-in support for Moab Accounting Manager {Moab}, or
- the resource manager supports a submit filter PBS(submit_filter), {Load-Leveler(SUBMIT_FILTER), LSF(esub)}, or
- a wrapper could be created for the submit command {PBS(qsub), SGE(qsub), LSF(bsub)}.

23.2.2 Usage Reservation @ Usage Start Time [Optional – Highly Recommended]

Just before usage begins (or 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 omitted, it would be possible for users to use more resources or services (i.e. start more jobs) than they have funds to support.

If the reservation succeeds, it will return a message indicating the amount reserved for the usage. In the case where there are insufficient resources to satisfy the usage request 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 usage from occurring.

To make a usage reservation with Moab Accounting Manager at this phase requires that:

- the scheduler or resource manager has built-in support for Moab Accounting Manager {Moab (see the [AMCFG](#) Configuration Parameter in Appendix A in Moab Workload Manager)}, or
- the resource manager is able to run a script at usage start time {LoadLeveler (prolog), PBS(prologue), LSF(pre_exec)}.

23.2.3 Usage Charge @ Usage End Time [Required]

When the usage is ended, 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 usage completes (dynamic accounting). This has the added benefit that usage run times can often be reconstructed from the usage reservation and charge times-tamps in case the resource management usage accounting data becomes corrupt.

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

To make a usage charge with Moab Accounting Manager at this phase requires that:

- the scheduler or resource manager has built-in support for Moab Accounting Manager {Moab (see the [AMCFG](#) Configuration parameter in Appendix F in Moab Workload Manager)}, or
- the resource manager is able to run a script at usage start time {LoadLeveler (epilog), PBS(epilogue), LSF(post_exec)}, or
- the resource management system supports some kind of feedback or notification mechanism occurring at the end of the usage (an email can be parsed by a mail filter).

23.3 Methods of interacting with Moab Accounting Manager

There are essentially six ways of programmatically interacting with Moab Accounting Manager. Consider a simple usage charge in each of the different ways.

23.3.1 Configuring an application that already has hooks for Moab Accounting Manager

The easiest way to integrate with Moab Accounting Manager is to use a resource management system with built-in support for it. For example, the Moab Workload Manager can be configured to directly interact with Moab Accounting Manager to perform the quotes, reservations, and charges by setting the appropriate parameters in their config files.

Example 1. Configuring Moab to use Moab Accounting Manager

Add an appropriate AMCFG line into moab.cfg to tell Moab how to talk to Moab Accounting Manager.

```
$ vi /opt/moab/moab.cfg
AMCFG[gold] TYPE=GOLD
HOST=batchserver PORT=7112
SOCKETPROTOCOL=HTTP
WIREPROTOCOL=XML
CHARGEPOLICY=DEBITALLWC
JOBFAILUREACTION=HOLD,RETRY
TIMEOUT=30
```

Add a CLIENTCFG line into moab-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 /var/moab/etc/moab.cfg
AMCFG[mam] TYPE=NATIVE
CHARGEPOLICY=DEBITALLWC
AMCFG[mam]
RESERVE-
URL=exec://$-
TOOLS DIR/mam/bank.reserve.mam.pl
AMCFG[mam]
RESERVEFAILUREACTION=HOLD
AMCFG[mam]
CHARGE-
```

```
URL=exec://$-
TOOLS DIR/mam/bank.charge.mam.pl
```

23.3.2 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 the bin directory).

Example 2. *To issue a charge at the completion of a job usage, you would use `gcharge`:*

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

23.3.3 Using the interactive control program

The interactive control program, `goldsh`, will issue a charge for a job expressed in xml.

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

```
UsageRecord Charge
Data:="<-
Usag-
eRecord><Instance>PBS.1234.0</Instance><Project>chemistry</P
<User>amy</Use-
r><-
Machine>colony</Machine><Processors>2</Processors><Duration>
</UsageRecord>" Duration:=1234
```

23.3.4 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 functionality via the Perl API.

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

```
use Gold;
my $request = new Gold::Request
(object => "UsageRecord",
action => "Charge");
my $usageRecord = new
Gold::Datum("UsageRecord");
```

```

$usageRecord->setValue
("Instance", "PBS.1234.0");
$usageRecord->setValue
("Project", "chemistry");
$usageRecord->setValue("User",
"amy");
$usageRecord->setValue
("Machine", "colony");
$usageRecord->setValue
("Processors", "2");
$usageRecord->setValue
("Duration", "1234");
$request->setDatum
($usageRecord);
$request->setOption("Duration",
"1234");
my $response = $request-
>getResponse();
print $response->getStatus(),
": ", $response->getMessage(),
"\n";

```

23.3.5 Use the Java API

Although deprecated, the Java API may still be usable to interact with Moab Accounting Manager. The javadoc command can be run on the contrib/java/gold directory to generate documentation for the gold java classes.

Example 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("UsageRecord",
"Charge");
        Datum usageRecord = new
Datum("UsageRecord");

```



```

usageRecord.setValue
("Instance", "PBS.1234.0");
usageRecord.setValue
("Project", "chemistry");
usageRecord.setValue("User",
"amy");
usageRecord.setValue
("Machine", "colony");
usageRecord.setValue
("Processors", "2");
usageRecord.setValue
("Duration", "1234");
request.setDatum
(usageRecord);
request.setOption
("Duration","1234");
Response response =
request.getResponse();
System.out.println
(response.getStatus() + ": " +
response.getMessage() + "\n");
}
}

```

23.3.6 Communicating via the SSSRMAP Protocol

Finally, it is possible to interact with Moab Accounting Manager by directly using the SSSRMAP Wire Protocol and Message Format over the network (see [SSS Resource Management and Accounting Documentation](#)). 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 Moab Workload Manager communicates with Moab Accounting Manager via this method.

Example 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>
    <Request action="Charge"
actor="scottmo">

<Object>UsageRecord</Object>
  <Data>
    <UsageRecord>

<Instance>PBS.1234.0</Instance>

<Project>chemistry</Project>
  <User>amyh</User>

<Machine>colony</Machine>

<Processors>2</Processors>

<Duration>1234</Duration>
  </UsageRecord>
  </Data>
  </Option
name="Duration">1234>/Option>
  </Request>
</Body>
  <Signature>

<Digest-
Value>azu4obZswzBt8-
9OgATukBeLyt6Y=</DigestValue>

<Sig-
nature-
Value>YXE/C08XX3RX4PMU1bWju+5/E5M=</SignatureValue>
  <SecurityToken
type="S-
ymmetric"></SecurityToken>
  </Signature>
</Envelope>
0

```

24.0 Configuration Files

Moab Accounting Manager uses four configuration files: one for the connection information (`site.conf`), one for the server (`goldd.conf`), one for the clients (`gold.conf`) and one for the graphical user interface (`goldg.conf`). For configuration parameters that have hard-coded defaults, the default value is specified within brackets.

24.1 Site Configuration

The site configuration file specifies the connection information for the current site such as the server host name, port, backup server, default security method and the symmetric key. Optionally, it may also have blocks that specify connection information for other sites. This file should be readable only by the accounting admin user.

Example 1. The following is an example site.conf file

```
server.host = red-head1
backup.host = red-head2
server.port = 7071
token.type = Symmetric
token.value =
pBaIapJqbfLd8NiyzTJefFXW

[white]
server.host = white-head1
server.port = 7071
token.value =
Fl7wOkioUpyjJdqJ8ckvWK_ta

[blue]
server.host = blue-head1
server.port = 7071
token.valne =
gVSeQ8Diz5O3pzj01y4inGWq
```

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

- **backup.host** - The hostname of the backup server. Each site can have both a primary server and a hot-standby backup server. They should either point to the same database or separate instances of replicated database. If backup.host is specified, clients will try communicating with the primary server first, and if the connection fails, they will try communicating with the backup server.
- **server.host** - The hostname of the server
- **server.port** [7112] - The port that the server listens on
- **token-type** [Symmetric] - Indicates the default security token type to be used in both authentication and encryption. Valid token types include Password and Symmetric. The default is Symmetric.
- **token.value** - When using the Symmetric token type, token.value is the secret key. It is a base64-encoded symmetric key used between clients and the server for authentication and encryption.

24.2 Server Configuration

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

- `currency.itemization` [`false`] – Enables (`true`) or disables (`false`) the storing of itemized charges to the Charge table for charge transactions.
- `currency.precision` [`0`] – Indicates the number of decimal places in the resource credit currency. For example, if you are will be dealing with an integer billable unit like processor-seconds, 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.
- `database.datasource` [`DBI:Pg:dbname=mam;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)
- `event.scheduler` [`false`] – Specifies whether the event scheduler is enabled (`true`) or not (`false`)
- `event.pollinterval` [`5`] – The period in minutes that the event scheduler uses to check and fire events. The poll interval must divide evenly into the number of minutes in a day (1440).
- `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
- `notification.deliverymethod` [`store`] – Specifies which deliverymethod is used by default if unspecified
- `notification.duration` [`1209600`] – Defines how long in seconds that stored notifications persist before being automatically deleted. The default is two weeks.

- **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 overridden 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.
- **security.authentication** [true] – Indicates whether incoming message authentication is required
- **security.encryption** [false] – Indicates whether incoming message encryption is required
- **super.user** [root] – The primary Moab Accounting Manager 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.

24.3 Client Configuration

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

- **account.show** [Id,Name,Amount,Constraints,Description] – The default fields shown by glsaccount
- **accounting.context** [hpc] – By specifying the accounting context (either hpc or cloud), the behavior of some client commands are adjusted to show the proper fields for that context. The default is hpc.
- **allocation.show** [Id,Account,Active,StartTime,EndTime,Amount,CreditLimit,Deposited] – The default fields shown by gsalloc
- **balance.show** [Id,Name,Amount,Reserved,Balance,CreditLimit,Available] – The default fields shown by gbalance
- **currency.precision** [0] – Indicates the number of decimal places in the credit currency. For example, if you will be dealing with integer billable units like processor-seconds, 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 gold.conf and golddd.conf files.
- **event.show** – [Id,FireCommand,FireTime,ArmTime,RearmPeriod,EndTime,Notify,RearmOnFailure,FailureCommand,CatchUp,CreationTime,Description] -- The default fields shown by glsevent
- **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.show** [Name,Active,Architecture,OperatingSystem,Description] – The default fields shown by glsmachine
- **notification.show** – [Id,Event,Type,Status,Code,Message,Key,Recipient,EndTime,CreationTime] --The default fields shown by glsnot

- **project.show** [Name,Active,Users,Organization,Description] – The default fields shown by `glsproject`
- **quote.show** [Id,Amount,Pinned,Instance,UsageRecord,StartTime,EndTime,Duration,ChargeRates,Description] – The default fields shown by `glsquote`
- **reservation.show** [Id,Instance,Amount,StartTime,EndTime,UsageRecord,Accounts,Description] – The default fields shown by `glsres`
- **response.chunking** [false] – Indicates whether large responses should be chunked (segmented). If set to false, large responses will be truncated
- **response.chunksize** [1000] – 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.
- **security.authentication** [true] – Indicates whether outgoing messages are signed
- **security.encryption** [false] – Indicates whether outgoing messages are encrypted
- **server.promotion** [suidperl] – When using the symmetric key for security authentication or encryption, since the `site.conf` file is readable only by the accounting admin user, a method must be employed to temporarily elevate privileges in order to encrypt the communication with the symmetric key. One of two security promotion methods may be selected: `suidperl` or `gauth`. `Suiperl` allows a Perl script to temporarily elevate privileges to the owner of the script if the `setuid` bit is set on the file. This method is recommended when `suidperl` can be installed on the system. If you prefer not to use `suidperl` or if it is not available for your system (such as with Perl 5.12 and higher), you will need to use the `gauth` security promotion method. `Gauth` is a `setuid` binary that allows the request body to be passed in as standard input and returns the authenticated digest and signature. Currently, only `suidperl` can be used for encryption of client communication. The security promotion method should be configured at install time by specifying the `--with-promotion` configuration parameter and defaults to `suidperl` when it is available.
- **statement.show** [Project,User,Machine] – The default discriminator fields in `gstatement`
- **transaction.show** [Id,Object,Action,Actor,Name,Child,Instance,Count,Amount,Delta,User,Project,Machine,Account,Allocation,UsageRecord,Duration,Description] – The default fields shown by `glstrans`
- **usagerecord.show** [Id,Type,Instance,Charge,Stage,Quote,User,Group,Project,Organization,Class,QualityOfService,Machine,Nodes,Processors,

Memory,Disk,Network,Duration,StartTime,EndTime,Description] – The default fields shown by glsusage

- **user . show** [Name,Active,CommonName,PhoneNumber,EmailAddress,DefaultProject,Description] – The default fields shown by glsuser

24.4 GUI Configuration

The following configuration parameters may be set in the GUI configuration file (`goldg.conf`).

- **`currency.enablehours`** [`false`] – If set to true, the graphical user interface will include a ShowHours radio button (defaulting to True) for certain panels (e.g. Account Balance, Deposit, Query, Statement, Transfer, Withdraw) that will allow the currency inputs or outputs to be divided by 3600.
- **`currency.precision`** [`0`] – Indicates the number of decimal places in the credit currency. For example, if you will be dealing with integer billable units like processor-seconds, 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 `golddd.conf` and `gold.conf` files.
- **`gui.style`** [`viewpoint`] – Modifies the appearance and behavior of the web GUI to be consistent with use within viewpoint or for standalone use. Valid values are `legacy` or `viewpoint`. The default is `viewpoint`.
- **`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`.
- **`response.chunking`** [`false`] – Indicates whether large responses should be chunked (segmented). If set to false, large responses will be truncated.
- **`response.chunksize`** [`1000`] – 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.
- **`security.authentication`** [`true`] – Indicates whether outgoing messages are signed
- **`security.encryption`** [`false`] – Indicates whether outgoing messages are encrypted
- **`server.promotion`** [`suidperl`] – When using the symmetric key for security authentication or encryption, since the `site.conf` file is readable only by the accounting admin user, a method must be employed to temporarily

elevate privileges in order to encrypt the communication with the symmetric key. One of two security promotion methods may be selected: `suidperl` or `gauth`. `Suidperl` allows a Perl script to temporarily elevate privileges to the owner of the script if the `setuid` bit is set on the file. This method is recommended when `suidperl` can be installed on the system. If you prefer not to use `suidperl`, or if it is not available for your system (such as with Perl 5.12 and higher), you will need to use the `gauth` security promotion method. `Gauth` is a `setuid` binary that allows the request body to be passed in as standard input and returns the authenticated digest and signature. Currently, only `suidperl` can be used for encryption of client communication. The security promotion method should be configured at install time by specifying the `--with-promotion` configuration parameter and defaults to `suidperl` when it is available.

- **`statement.discriminators`** – The Account Statement page will group summary entries in the debit detail by these transaction properties.