Matisse® Server Administration Guide

May 2014

MATISSE Server Administration Guide

PDF generated 26 April 2014

Contents

Introduction 12

Conventions 12

1 Matisse Server: An Overview 13

1.1 Basic Concepts 13

I/O Parallelism and Copy Semantics 13

Temporal Features 13

Collect Versions 14

Transaction Model and Concurrency Control 14

Disk Fault Tolerance 14

1.2 Database Environment 14

Configuration File 14

Data Files 15

Log File 15

1.3 Managing Your Database 16

1.4 Transferring Databases Between Hosts 16

2 The Matisse Environment 17

`MATISSE_CFG` 17

`MATISSE_HOME` 18

`MATISSE_LOG` 18

`MATISSE_NET_PATH` 19

`MATISSE_PORTMON_ADDR` 20

`MATISSE_PORTMON_NAME` 21

`MATISSE_SMLISTENER_ADDR` 21

3 Matisse Connections 22

3.1 Introduction 22

3.2 Matisse Connections 22

Setting Up a Connection Environment 23

Setting a Transport Priority 24

Port Monitor Daemon Log File 25

Port Monitor Utility mt_pmadm 25

`mt_pmadm` 26

3.3 Connections through Firewalls 29

3.4 Portmon Messages 30

Errors Resulting from the Utility mt_pmadm 30

Error Messages of the Port Monitor Log File 32

4 Server Manager Listener 36

4.1 Introduction 36

4.2 Managing Remote Operations 36

4.3 Controlling Remote Operation Requests 36
4.4 Managing database autorestart 37
4.5 Running mt_smlistener daemon 37
Setting Up a Connection Environment 37
SMListener Daemon Log File 37
Starting a SMListener Daemon 37
Stopping the SMListener daemon 38
4.6 Connections through Firewalls 39

5 Matisse Access Control 40
5.1 Introduction 40
Different Privileges 40
System User 40
Enabling Access Control 40
5.2 Managing Users 41
Operating System Access Control 41
Using Matisse Access Control 41
Add/Drop/ Modify Users 41
Create an Administrator 41
5.3 Database Connection API 42

6 Configuring a Database 43
6.1 Configuration File 43
File Syntax 43
6.2 Configuration Parameters 43
Mandatory Parameters 44
Default Values 44
Automatically Updated Parameters 45
NAME 45
PAGESIZ 45
CACHESIZ 46
SECURITY 46
AUTOEXTEND 47
DATEXTENDSIZ 47
AUTOCOLLECT 47
AUTOCOLLECTFREQ 48
OBJTABLESIZ 48
OBJTABCLRFREQ 49
AUTORESTART 49
DATFULLINIT 49
DATINITSIZ 49
DATINMEMORY 50
MEMORYTRANS 50
MAXSQLDOP 50
MAXSQLTHRDPOOL 51
MAXSRVLOGFILES 51

MAXBKPLOGFILES 51
TCPKEEPALIVE. 52
PORTS 52
PATH 53
6.3 Using Disk Partitions as Datafiles 54
Why Use Partitions? 54
Check for Partitions That Contain the First Sector on UNIX 54
Checking Partitions with File Systems on UNIX 55
Checking for Swap Partitions on UNIX 55
Declaring a Partition in a Configuration File 55

7 Using the Enterprise Manager 57
7.1 Starting the Enterprise Manager 57
7.2 Remote Administration 58
7.3 Creating a Database 58
7.4 Stopping a Database 59
7.5 Monitoring Database Server 59
7.6 Managing Database Server Operation Control 61
7.7 Managing Database Users 61
7.8 Managing Datafiles 62
7.9 Managing Backups 64
7.10 Managing Open Connections 66
7.11 Managing Active Transactions 66
7.12 Monitoring a Database 67
Changing the Refresh Interval. 71
Taking an Activity Snapshot 71
7.13 Restoring a database 72
7.14 Scheduling tasks 73
Executing a User-defined Script 75

8 Collecting the Versions of a Database 77
How the Collect Versions Mechanism Works 77
Automatic Version Collection 77
Kinds of Version Collections 77
Scheduled Collection on MS Windows 78
Version Collection Log File 78

9 Administration Commands 80
Database Shutdown Restart 81
Managing Datafiles 81
Disk Mirroring 81
Using disk partitions 82
Managing Users 82
Managing Connections 82
Managing Transactions 83
Managing Versions 83

Monitoring a Database 83
Extending the Page Server Cache 84
Extending the Object Table Cache 85
Changing the Run Frequency of Operations 85
Managing License Keys 85

10 Database Transactional Replication 86
10.1 Introduction 86
Feature Overview 86
Replication Benefits 86
10.2 Replication Establishing and Disestablishing 87
Before Establishing Replication 87
Establishing Replication 87
Retry or Noretry Mode 87
Disestablishing Replication 87
Swapping Roles Between Master and Replica 88
10.3 Replication Monitoring 88
Replication status 89
10.4 Resynchronization at restart of after replica failure 89
Shutdown Restart 89
Network or Replica Failure 89
Switching to the replica in case of master failure 90

11 Database Snapshot Replication 92
11.1 Introduction 92
Feature Overview 92
Benefits 92
Design Overview 93
11.2 Replication Establishing 93
Before Establishing Replication 93
Establishing Replication 93
Publishing Changes 94
Disestablishing Replication 95
11.3 Replication Monitoring 96
Publisher Sate 96
Subscriber Sate 96
11.4 mt_xsr utility 97
mt_xsr publish 97
mt_xsr subscribe 97
mt_xsr describe 98
mt_xsr unpublish 98
mt_xsr unsubscribe 98

12 Database Backup and Restore 100
12.1 Introduction 100
Full and Incremental Backup 100

Parallel Backup 100
12.2 Running a Full or Incremental Backup 100
Running a Full Backup 100
Running an Incremental Backup 100
Automated Backups 101
Backup Journal Files 101
12.3 Restore 102
12.4 Running a Parallel Backup 102
12.5 Parallel Restore 103

Appendix A Starting Matisse Server as a Windows Service 104
A.1 Introduction 104
A.2 Installation 104
A.3 Specifying Matisse Server and Its Parameters 104
A.4 Starting and Stopping the Matisse Server Service 105
A.5 Uninstall 105

Index 106

Tables

Table 6.1 Default Values of Configuration Parameters 44
Table 7.1 Summary Information about All Connections 68
Table 7.2 Detailed Information about Specific Connections 69
Table 7.3 Information about Datafiles 69
Table 7.4 Configuration Information 70
Table 7.5 Information about Transactions 70
Table 8.1 Collect Levels 77
Table 10.1 Replication info status 89

Figures

Figure 2.1 Matisse Initial Directory Structure 17
Figure 3.1 Matisse Server with portmon 22
Figure 3.2 mt_portmon Daemon under Solaris 23
Figure 7.1 Create Database 58
Figure 7.2 Stop Database 59
Figure 7.3 Database servers state monitoring 60
Figure 7.4 Servers activity and resources monitoring 60
Figure 7.5 Server Operation Control Manager 61
Figure 7.6 Adding a user 62
Figure 7.7 Creating a mirror datafile 63
Figure 7.8 Database Backup Manager 65
Figure 7.9 Killing an active connection 66
Figure 7.10 Aborting a transaction from the Monitor window 67
Figure 7.11 Matisse Monitoring: 68
Figure 7.12 Refresh interval choice 71
Figure 7.13 Freeze button from the monitoring menu-bar 71
Figure 7.14 Unfreeze button from the monitoring menu-bar 71
Figure 7.15 Database restore wizard 72
Figure 7.16 Database Task Scheduler 74

Introduction

Conventions

This document uses the following conventions:

Text	The main text is written in characters like these.
`Code`	All computer variables, code, commands, and interactions are shown in this font. Also, any code and commands that the user must enter are shown in this font on a gray background.
`variable`	In a program example, or in an interaction, a variable (anything that is dependent on the user environment) is written in italics.
References	References to another part of the MATISSE documentation are made as shown here.

1 Matisse Server: An Overview

1.1 Basic Concepts

The Matisse Server operates as a back-end server that manages a repository of persistent objects. Client applications connect to the server through the network or through a local transport.

The two primary tasks of the Matisse Server are to ensure that:

- All objects remain available in a consistent state in the presence of system failures (recovery management)
- When several clients access a shared set of objects simultaneously in read or write mode, each client gets a consistent view of the database

I/O Parallelism and Copy Semantics

The Matisse Server provides high-end parallelism for multimedia streaming and large databases for a large number of users.

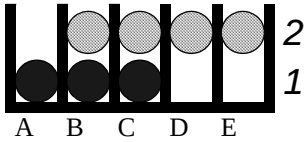
The Matisse Server is implemented on top of kernel threads and scales linearly as new CPUs or new disks are added. Objects are not updated in place: a new version of an object can be written to any available disk with optimal load balancing across disks.

Temporal Features

The Matisse Server intrinsic Versioning is the key underlying technology that differentiates it from other storage management systems. Intrinsic Versioning is the automatic generation and control of object versions.

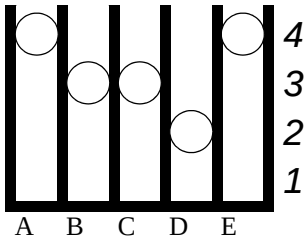
The figure at right shows the creation of three objects by a transaction at time 1.

When the value of an object changes, a new copy of the object is created, rather than the current version of the object being updated in place. The figure at right represents the creation of a fourth and a fifth object, as well as the modification of objects B and C. The database can be queried consistently as of time 1, without affecting the current transaction processing and without locking any data.



Collect Versions

The collect versions mechanism is run automatically to reclaim disk space. It preserves the most recent version and the versions that have been explicitly saved. The figure at right represents the contents of a database after a collect version has been performed.



Transaction Model and Concurrency Control

Concurrency control is enforced by read or write database locks. The locking granularity is at the sub-object level, as the Matisse Server locks separately the relationship part of an object and the attribute part of an object.

In transaction mode, the Matisse Server enforces traditional two phase locking to ensure consistent—serializable—transactions. As mentioned above, transactions are not affected by version access queries, the later can run concurrently without locking.

Disk Fault Tolerance

The Matisse Server provides disk fault tolerance through mirroring.

When there is a disk failure, the database remains online and Matisse Server automatically uses mirrored data as necessary. When a new disk is available, you can use the DBA Tool to reestablish mirroring. It is not necessary to stop Matisse Server to replace the failed disk

1.2 Database Environment

A Matisse database is identified by its name and the name of the host machine where it resides. It is made up of three major components:

- A configuration file
- Files or disk partitions (“datafiles”)
- A log file

Configuration File

The configuration file contains the parameters that define the database—the location and size of its datafiles, the execution parameters of the database, and so on.

You can modify the configuration file with the DBA Tool or the command line administrations commands. The DBA tool is described in *section 7, Using the Enterprise Manager*.

To perform operations such as initialization, you must have sufficient privileges on the database configuration file. Make sure that you have the read (`r`) and execute (`x`) permissions on the directory defined by the environment variable `MATISSE_CFG` before attempting one of these operations. If you do not have sufficient privileges, the DBA Tool will not perform the operation you request.

Data Files

A database may have one or several datafiles. Distributing the database over several disks, with one datafile per disk, provides better security and performance.

NOTE: For maximum safety and best performance, we strongly recommend you not define datafiles on the system disk.

Data files are defined in the configuration file. You define a datafile by specifying either a directory path or an entire unformatted disk partition (raw device) for use by the database. Using a disk partition allows Matisse to access data faster and provides better fault tolerance, as it eliminates the risk of file system corruption.

CAUTION: When it is necessary to add, resize, or delete a datafile, use only the Files menu commands in the Enterprise Manager or the commands discussed in *section 9, Administration Commands*. Manually editing the text of the configuration file could corrupt the database.

Log File

The log file records the main administrative operations that are run on the database. Its purpose is to help you track the activity on a database.

In addition, the log file lists the possible errors that may occur on the database, for instance, “not enough disk space”, when all the datafiles are full.

To perform operations such as initialization, version collection and several others, you should have read (`r`) write (`w`) and execute (`x`) permissions on the directory defined by the environment variable `MATISSE_LOG`. If you do not have sufficient privileges, the DBA Tool does not perform the operation you request.

NOTE: If you are used to relational DBMSes, you may expect a “log file” to contain an entry for every single update, since such entries are required to perform rollbacks and other data-recovery functions after a system failure. This is not the case with Matisse: its data-recovery features rely on its intrinsic versioning architecture, so there is no need for a traditional transaction redo log.

1.3 Managing Your Database

You can manage your database in either of two ways:

- Using the DBA Tool
- Entering shell level commands

This document provides you with all the information required to manage your database either by typing shell commands or by using the DBA Tool.

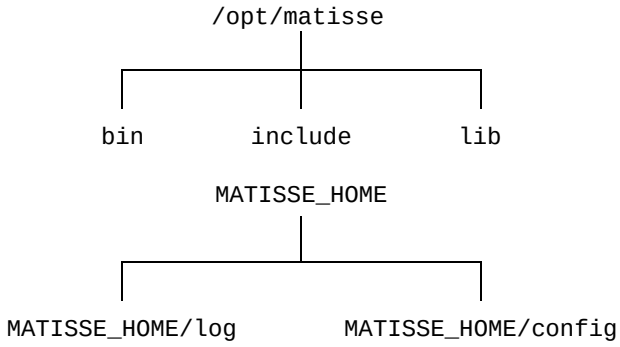
1.4 Transferring Databases Between Hosts

You can transfer databases directly between platforms with identical byte swapping. You can copy a database directly from one platform to the other. If you do this, however, you must copy all the database files. You will be able to use the copied database only if all the files are copied.

2 The Matisse Environment

Assuming that `/opt/matisse` is the directory where Matisse is installed, the product is initially installed as shown in *Figure 2.1*.

Figure 2.1 Matisse Initial Directory Structure



Matisse defines environment variables that:

- Define setup information related to your database
- Help you find where this information is located (that is, in which directory)

It is recommended that you first define `MATISSE_HOME`, in order to make the other variables independent from the location of the Matisse installation. The Matisse environment variables are listed and described in the following sections.

MATISSE_CFG

Purpose

MATISSE_CFG is an environment variable that points to the directory that contains the configuration files.

Immediately after installation, MATISSE_CFG points to the following directory:

```
/opt/matisse/config
```

It is recommended that you set MATISSE_CFG to a directory other than this default. By doing this, you guarantee that the Matisse database configuration files will not be created in the installation directory.

In this way, when an upgrade or a new release is installed, it will not be necessary to copy the configuration file from the default MATISSE_CFG directory to a newly created directory.

Note that both the port monitor and the DBA Tool use MATISSE_CFG, which must be set for these executables to work properly.

NOTE: If the `MATISSE_CFG` environment variable is not set, Matisse uses the config subdirectory of the directory defined by `MATISSE_HOME` as the config directory.

MATISSE_HOME

Purpose

When `MATISSE_CFG` and `MATISSE_LOG` are not specified, Matisse checks that `MATISSE_HOME` is specified. If it is specified, Matisse operates as if `MATISSE_CFG` and `MATISSE_LOG` point respectively to the `config` and `log` sub-directories of `MATISSE_HOME`.

After installation, the variable should point to the installation directory.

It is recommended that you set `MATISSE_HOME` to a directory other than this default. By doing this, you guarantee that the Matisse log and configuration files will not be created in the installation directory.

If you do this, the `MATISSE_HOME` directory is independent from any Matisse version. In this way, when an upgrade or a new release is installed, it is not necessary to copy the files and directories from the previous `MATISSE_HOME` directory to the newly created one.

The `MATISSE_HOME` variable is used by the Port Monitor.

MATISSE_LOG

Purpose

`MATISSE_LOG` is an environment variable that points to the directory that contains the log files of the Matisse databases as well as log files of the port monitor daemons. These log files contain messages concerning either database administration or port monitor operations. They also provide information on errors.

Immediately after installation, `MATISSE_LOG` points to the following directory:

```
/opt/matisse/log
```

It is recommended that you set `MATISSE_LOG` to a directory other than this default. By doing this, you guarantee that the Matisse log files will not be created in the installation directory.

In this way, when an upgrade or a new release is installed, it will not be necessary to copy the files and directories from the default MATISSE_LOG directory to a newly created directory.

Note that both the port monitor and the DBA Tool use MATISSE_LOG. The environment variable MATISSE_LOG must be set for the DBA Tool to work properly.

NOTE: If the MATISSE_LOG environment variable is not set, Matisse uses the log sub-directory of the MATISSE_HOME directory as the log directory.

MATISSE_NET_PATH

Purpose

MATISSE_NET_PATH is an optional environment variable used by the client application that lets you define the order in which Matisse searches for a transport when a client requests connection to the server. It also lets you limit the kind of transport searched to one kind of transport.

By default, the order in which Matisse searches for a transport when a client requests connection (all Unix but Solaris platforms) is the following:

```
local
tcp
```

for Solaris platforms:

```
ticots
tcp
```

or, for MS Windows platforms:

```
tcp
```

If you want to set the order in which Matisse searches for a transport when a client requests connection, you can do so by means of the MATISSE_NET_PATH environment variable.

The variable definition has the following syntax:

```
transport1:transport2
```

The keywords used to specify the different transports are of course tcp and local (or ticots on Solaris hosts). You can specify any order. For example, to specify that tcp transport be searched first, and local next, the MATISSE_NET_PATH environment variable must have the following definition (on a non Solaris host):

```
tcp:local
```

Note that you can also use MATISSE_NET_PATH to limit the kind of transport searched to one kind of transport. To specify that only local transport be used, for example, the MATISSE_NET_PATH environment variable must have the following definition:

```
local
```

MATISSE_PORTMON_ADDR

Purpose

On each machine on which a Matisse server is running there must be a port monitor for each transport used by the database.

When a Matisse server is started on this machine, it has to notify the port monitors of its existence. When a client application needs to connect to a database, it asks to the port monitor on the specified host to initialize the connection. Both server and client need to know at which address the port monitor is listening.

MATISSE_PORTMON_ADDR is an optional environment variable that defines the address the port monitors for each transport are listening to.

The variable definition has the following syntax:

transport-address[:*transport-address*]*

MS Windows

On MS Windows platforms, suitable values for the transport argument is tcp.

To specify, for example, the tcp transports for a server, you can define the MATISSE_PORTMON_ADDR address as follows:

```
tcp-7421
```

UNIX Non Solaris

On UNIX hosts, suitable values for the transport argument are tcp and local.

When a Matisse server is started, it has to notify the port monitors of its existence. To know which port monitor to use, it looks if this variable is defined. If it is the case, the value of MATISSE_PORTMON_ADDR is used to find the port monitors.

Otherwise, the local or NIS file /etc/services is used for tcp.

For the local transport, if MATISSE_PORTMON_ADDR is not defined, the Matisse server will use the default value /tmp/mtportmon_local.socket.

When a client application is running on a non Solaris host, it will use the same operations than described above to find where to address the port monitor.

To specify, for example, the tcp and local transports for a server, you can define the MATISSE_PORTMON_ADDR address as follows:

```
tcp-7421:local-/var/tmp/mt_local.socket
```

NOTE: Solaris platforms do not use the variable `MATISSE_PORTMON_ADDR`.

Solaris

On Solaris hosts, suitable values for the transport argument are `tcp` and `ticots`.

For `tcp`, only the local or NIS `/etc/services` file is used, and for `ticots`, only the file `/etc/net/ticots/services` is used. These files are used by both the Matisse server and client application.

MATISSE_PORTMON_NAME

Purpose

`MATISSE_PORTMON_NAME` is an optional environment variable that defines a name for the port monitor service (port monitor). By default, the name of the port monitor service is `mtportmon`. You can set a different name for the port monitor service by setting this environment variable to the character string you choose.

MATISSE_SMLISTENER_ADDR

Purpose

`MATISSE_SMLISTENER_ADDR` is an optional environment variable that defines the address the Matisse Server Manager Listener (SMListener) is listening to. By default, SMListener address is `7412`. You can set a different address for the smlistener daemon by setting this environment variable to the number you choose

3 Matisse Connections

3.1 Introduction

This section describes how to establish connections between client and server and how to create and administer `mt_portmon` daemons on all supported platforms.

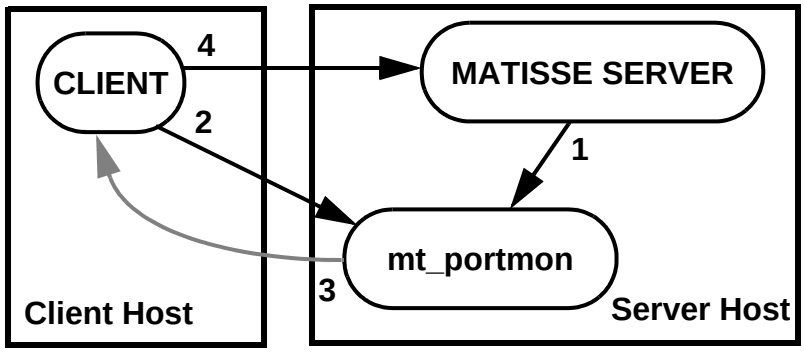
3.2 Matisse Connections

Matisse is a multi-protocol server. For each kind of client-server transport supported, an `mt_portmon` daemon is needed. This daemon must be started before connections between the server and the client can occur.

- There is a different daemon for each kind of transport. The two different kinds of transport currently supported are tcp and local. The tcp transport is TCP/IP on a local area network (`LAN`). The local transport is TCP for connections between a client and server located on the same host.This transport is a virtual circuit-mode transport provider. Under Solaris, the local transport is Ticots-based. Ticots offers connection-oriented service types. It supports the same service types (`T_COTS`) supported by the ISO transport-level model.

Figure 3.1 illustrates the operation of the `mt_portmon` daemon.

Figure 3.1 Matisse Server with portmon

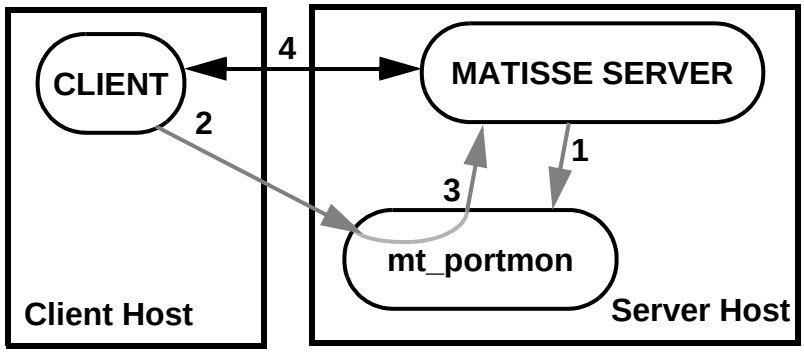


The server connects to an `mt_portmon` daemon and supplies its address. Any client that wants to connect to the server is then free to do so.

A connection between a client and the server is established in the following way. The client requests the address of the server from the `mt_portmon` daemon. The daemon returns the address of the server to the client, as illustrated above by the grey arrow leading to the client. Then the client uses this address to connect to the server.

Under Solaris the connection mechanism is slightly different, the `mt_portmon` daemon connects the client directly to the server, as shown in *Figure 3.2*.

Figure 3.2 mt_portmon Daemon under Solaris



A connection between a client and a server is established in the following way. The client requests the `mt_portmon` daemon for the system and then connects to it. The `mt_portmon` daemon passes the connection descriptor to the MATISSE server. The client and the server are then connected directly. They will have no further communication with the `mt_portmon` daemon.

NOTE: This mechanism simplifies the settings to establishing a connection to a database server protected by a firewall (see *section 3.3, Connections through Firewalls*).

Setting Up a Connection Environment

When you know which kind of transport you use, you can define it. There are two ways you can do this:

- `MATISSE_PORTMON_ADDR` environment variable
- Local or NIS `/etc/services` file (for `tcp` transport only)

Otherwise, defaults are used (`7421` for tcp transport, `/tmp/mtportmon_local.socket` for local transport).

You can use the `MATISSE_PORTMON_ADDR` environment variable to define transport for both tcp and local transport. Note that if your host supports both kinds of transport, you can define both of them by means of this variable.

The variable definition has the following syntax:

```
transport-address[:transport-address]
```

For the argument *transport*, suitable values are tcp and local.

Appropriate values for the argument *address* depend on the transport that is specified. When you specify `tcp`, the address that follows must be the number of a TCP port. When you specify `local`, the address must be the local pathname.

To define both tcp and local transport, you can define the `MATISSE_PORTMON_ADDR` environment variable as follows:

```
tcp-7421:local-/tmp/mtportmon_local.socket
```

This environment variable must also be defined by the client.

For TCP/IP, you can specify the transport in two other ways. The easiest is to add the following line to the file `/etc/services`:

```
mtportmon port/tcp
```

The string `mtportmon` is the default name of the Portmon port monitor.

If you prefer, you can change the default name of the Portmon port monitor to another. To do this, set the `MATISSE_PORTMON_NAME` environment variable to the new name.

The other way is to add similar information to a NIS file. If you choose this third method, see you system manager for further information.

Solaris

Under Solaris, for `ticots` transport, you must define the service associated with the port monitor. To do this, you must enter the name of the service in the file `/etc/net/ticots/services`. To add the service associated with the port monitor, insert the following line in this file:

```
mtportmon mtportmon
```

NOTE: The above example assumes that you have not changed the default name of the Matisse port monitor service.

Setting a Transport Priority

By default, the order in which Matisse searches for a transport when a client requests connection (all Unix except Solaris platforms) is the following:

```
local
tcp
```

Solaris

Under Solaris, the local transport is ticots, so by default the order is the following:

```
ticots
```

```
tcp
```

If you want to set the order in which Matisse searches for a transport when a client requests connection, you can do so by means of the `MATISSE_NET_PATH` environment variable.

The variable definition has the following syntax:

```
transport1:transport2
```

The keywords used to specify the different transports are of course `tcp` and `local` for servers installed on Unix hosts but Solaris ones. You can specify any order. For example, to specify that tcp transport be searched first, and local next, the `MATISSE_NET_PATH` environment variable must have the following definition:

```
tcp:local
```

Solaris

```
tcp:ticots
```

Note that you can also use `MATISSE_NET_PATH` to limit the kind of transport searched to one kind of transport. To specify that only local transport be used, for example, the `MATISSE_NET_PATH` environment variable must have the following definition:

```
local
```

Solaris

```
ticots
```

Port Monitor Daemon Log File

Each time you start a port monitor daemon, a port monitor daemon log file is opened in the `MASTISSE_LOG` directory. The name of the log file is built from the name of the host and the name of the port monitor daemon specified by the *`pmtag`* of the `mt_pmadm` command.

For example, if the name of a port monitor daemon were `mttcp` and the name of its host were `jade`, the name of the port monitor daemon log would be the following:

```
mttcp.jade.log
```

If a problem occurs, the port monitor daemon log file may contain a message. For a description of the error messages, please refer to *section 3.4, Portmon Messages*.

Port Monitor Utility mt_pmadm

The `mt_portmon` daemons (in other words, the port monitor daemons) are managed by means of the `mt_pmadm` utility. The syntax of this command and its uses are described on the following pages.

Note that to use `mt_pmadm`, you must have read and write permissions on the `MATISSE_CFG` and `MATISSE_LOG` directories.

mt_pmadm

Syntax

```
mt_pmadm -p pmtag options
```

UNIX Options

`-p` *`name`*

Specifies a name for the port monitor daemon. This name is used to identify the port monitor daemon. This option must always be specified when using the `mt_pmadm` command.

`-s`

Specifies that a port monitor daemon be started.

`-t` *`transport`*

Specifies the kind of transport that is managed by the port monitor daemon. The two values that are possible are `tcp` and `local`.

`-D`

Specifies if the port monitor daemon is started in enabled mode or disabled mode.

`-k`

Deletes the port monitor daemon of the specified name.

`-d`

Disables a port monitor daemon that has been previously started and enabled.

`-e`

Enables a port monitor daemon that has been previously started and disabled.

`-l`

Lists the status of the port monitor daemon. Indicates if the port monitor daemon is enabled or disabled.

`-L`

Lists the services (or database processes) provided by the port monitor daemon.

`-r`

Removes a service from a port monitor daemon.

`-S`

Specifies the service to be removed from a port monitor daemon.

`-h`

Provides help on the `mt_pmadm` command and its options.

MS Windows Options

`-s`

Specifies that a port monitor daemon be started.

`-d`

Disables a port monitor daemon that has been previously started and enabled.

`-e`

Enables a port monitor daemon that has been previously started and disabled.

`-L`

Lists the services (or database processes) provided by the port monitor daemon.

`-r`

Removes a service from a port monitor daemon.

`-S`

Specifies the service to be removed from a port monitor daemon.

`-h`

Provides help on the `mt_pmadm` command and its options.

Arguments

`pmtag`

The name of the port monitor daemon. The name must be no more than 14 characters in length. The *`pmtag`* does not specify the name of the port monitor. The name of the Portmon port monitor is `mtportmon`. You can change the name of the Portmon port monitor by means of the `MATISSE_PORTMON_NAME` environment variable.

Purpose

This command is used to define, start, enable, disable, and provide the status of a port monitor daemon. Note that if you encounter error messages when using `mt_pmadm`, refer to for an explanation and a possible solution.

Starting a Port Monitor Daemon

To start a port monitor daemon, use `mt_pmadm` with the `-s`, `-p`, and `-t` options. As described under the heading Options, these three options specify respectively:

- That the port monitor daemon should be started
- A name for the port monitor daemon
- The transport managed by the port monitor daemon

To start a port monitor daemon named `mttcp` that handles tcp connections for example, you would type the following:

```
mt_pmadm -s -p mttcp -t tcp
```

After typing this command, the port monitor daemon named `mttcp` is started and enabled.

You can also start a port monitor daemon and leave it disabled. To do this, you need to specify the `-D` option when you start the port monitor daemon, as shown below:

```
mt_pmadm -s -p mttcp -t tcp -D
```

Disabling a Port Monitor Daemon

To disable the port monitor daemon, type the following:

```
mt_pmadm -d -p mttcp
```

After you type this command, the port monitor daemon named `mttcp` is disabled. When the port monitor daemon has been disabled, it cannot register a service (database process).

Enabling a Port Monitor Daemon

To enable a port monitor daemon that has been previously disabled, type the following:

```
mt_pmadm -e -p mttcp
```

Checking a Port Monitor Daemon

To check whether a port monitor daemon is enabled or disabled, use the following:

```
mt_pmadm -l -p mttcp
```

The system then tells you if the port monitor daemon is enabled or disabled.

Listing the Port Monitor Daemon's Services

You can also check which services, or database processes, are available through the port monitor daemon. Note that these services are the different Matisse databases that are registered on the port monitor daemon. To list these services, type the following:

```
mt_pmadm -L -p mttcp
```

The Matisse databases (services) that are registered with the port monitor daemon are then displayed, as shown in the example below:

```
20-Jun 12:40:34 PMADM-I-PMSTATE, Port Monitor is enabled
20-Jun 12:40:34 PMADM-I-TRPTYPE, Transport type: tcp
20-Jun 12:40:34 PMADM-I-SVCLIST, Registered services:
EXAMPLE
MEDIA
```

Removing a Service from a Port Monitor Daemon

If you want to remove one of these services for whatever reason, you can do so with the `-r`, `-p`, and `-S` options. To remove the database `media` from the port monitor daemon `mttcp`, for example, you would type the following:

```
mt_pmadm -r -p mttcp -S media
```

NOTE: There is almost no reason why you would want to remove a service from a port monitor daemon. Do not use this option unless a server that is listed as a service is no longer available.

Removing a Port Monitor Daemon

In addition to removing a service from a port monitor daemon, you can also remove a port monitor daemon by means of the `mt_pmadm` command. To do this, you must specify the `-k` option. The following example shows you how:

```
mt_pmadm -k -p mttcp
```

Getting Help

To get help on the `mt_pmadm` command, use the `-h` option, as shown below:

```
mt_pmadm -h
```

3.3 Connections through Firewalls

In order to establish a connection to a database server protected by a firewall, you must open in the firewall the `TCP` port used by the Port Monitor daemon and the `TCP` ports used by each database server.

To configure your database servers to accept remote `TCP` connections, update your firewall settings as follows:

1. Open the `TCP` port used by the Port Monitor daemon. The default port number is `7421`.
2. Open a `TCP` ports range for the database servers accessed through the firewall.
3. Update the `PORTS` database configuration parameter of each database server with the `TCP` port number range opened in the firewall. Refer to *section 6.2, Configuration Parameters* for more details.

NOTE: The port numbers range should not include the port number used by the Matisse port monitor

NOTE: On Solaris, steps 2 and 3 are not necessary since the Matisse Port Monitor on Solaris forwards the connection on `7421` to the database servers. Therefore opening the `TCP` port used by the Port Monitor daemon is enough.

3.4 Portmon Messages

Errors Resulting from the Utility mt_pmadm

This section lists the different messages related to portmon that you may encounter, either when you use one of the utilities that manage the port monitor or when viewing the port monitor log file.

BADANSWER `Bad answer message from port monitor`

The message received from the port monitor is erroneous.

CMDRCVFAILED `Command message receive failed`

Unable to read from the communication pipe.

Solution Check that the port monitor is running.

CMDSNDFAILED `Command message send failed`

Unable to write to the communication pipe.

Solutions

- Check that the port monitor is running.
- Extend system resources (memory).

CREATEPIPEFAILED `Creation of a communication pipe failed`

Unable to create pipes required for communication between `mt_pmadm` and `mt_portmon`.

Solutions

- Check that `MATISSE_HOME` or `MATISSE_CFG` are defined with valid pathnames.
- Check that the user has sufficient privileges to write to the directory defined by `MATISSE_CFG`.
- Check for system resources (no i-node)

INVTAGSIZE `The specified tag is too long.`

The argument *`pmtag`* is a string longer than the maximum supported length of 14 characters.

Solution Try again with a shorter name.

NOPERM `Unable to kill port monitor, no permission`

It is impossible to send a `SIGTERM` signal to the `mt_portmon` process because it has been started by another user.

Solution Try again while logged in with `root` privileges.

OPENPIPEFAILED `Open communication pipe failed`

Unable to open pipes required for communication between `mt_pmadm` and `mt_portmon`.

Solutions

- Check that `MATISSE_HOME` or `MATISSE_CFG` are defined with valid pathnames.
- Check that the user has access to the pipes (in the `MATISSE_CONFIG` directory).

`PMNORUNNING` `Port Monitor is not running`

The specified port monitor does not exist.

Solutions

- Check the definitions of `MATISSE_HOME` or `MATISSE_CFG`.
- Use the correct port monitor name as previously defined by the *`pmtag`* argument.

`PMNOTFOUND` `Port monitor not found`

The environment variable `PATH` is not correctly set.

Solution Add the path to the Matisse binaries in the `PATH` environment variable.

`PMRUNNING` `Port Monitor already running`

You tried to start a port monitor with the same name as one already running.

Solution Try starting a port monitor with a different name.

`STARTFAILED` `Start of the port monitor failed`

The start of the port monitor failed. Execution of the command was terminated unsuccessfully.

Solutions

- `mt_portmon` must be executable
- Check system resources (processes, etc.)

`SVCLISTFAILED` `Unable to list services`

An attempt to retrieve information about services failed.

Solutions

- Check that some virtual memory is available.
- Check if the port monitor is running.

`SVCNOREGISTER` `Service [service name] is not registered`

The requested service is not registered by the port monitor.

Solutions

- Check values for *`service`* and *`pmtag`*.
- Check the definitions of `MATISSE_HOME` or `MATISSE_CFG`.

`SVCRMFAILED` `Removing service [service name] failed`

The attempt to remove a service resulted in a failure. The port monitor may process subsequent requests incorrectly.

Solutions
- Check if the file system where the MATISSE_CFG directory is located is full.
- For save operation, after error fix, restart the port monitor.

Error Messages of the Port Monitor Log File

This section lists all the error messages that may be logged in the port monitor log file on non Solaris hosts:

`CMDRCFAILED` `Command message receive failed`

An error occurred during message receive on pipe.

Solutions
- *pmadm* has failed. Restart it.
- No more memory available. Increase swap.

`CMDSNDFAILED` `Command message send failed`

An error occurred during message send on pipe.

Solution No more system resources. Increase them.

`CONNBROKEN` `Connection broken for [transport endpoint | STREAM pipe]`

Endpoint communication is broken.

Solution Check your system.

`ENDPOINTFAILED` `Creation of endpoint communication failed`

Unable to create and bind socket.

Solution For `tcp` transport, check if the port number is already used. For `local` transport, check that the path defined is not already used.

`INITFAILED` `Port Monitor initialization failed`

Unable to initialize port monitor.

Solutions
- Check the definitions of `MATISSE_HOME` or `MATISSE_CFG`. An invalid path may be specified.
- Check the privileges on the `MATISSE_CFG` directory. The owner must have write access on this directory.

`INVTRANSPORT` `Invalid transport`

The transport specified is not supported.

Solution Specify a valid transport: `tcp` or `local`.

`OPENLOGFAILED` `Open log file failed`

Unable to open log file for daemon.

Solution Check the paths specified by `MATISSE_HOME` or `MATISSE_LOG`. An invalid path may be specified.

`OPENPIPEFAILED` `Open communication pipe failed`

Unable to open pipes for communication between `mt_pmadm` and `mt_portmon`.

`PMADDRNOTFOUND` `Port Monitor Address not found for specified transport`

Unable to find port monitor address.

Solution Check that in NIS or `/etc/services`, there is a service named `mtportmon` or named the same as the value of the environment variable `MATISSE_PORTMON_NAME`.

`SVCALREGISTER` `Service [service name] is already registered`

A service (database) of the same name is already registered.

Solution Change the name of the database.

`SVCNOREGISTER` `Service [service name] is not registered`

The service is not registered. A client has tried to connect to an unregistered database.

Solutions
- Check if the database is running.
- Check name of the database and the host.

`SVCREGFAILED` `Registering service [service name] failed`

Registering a service failed. An attempt to register a service (a database) has aborted due to a file operation error.

Solution Check space available in the `MATISSE_CFG` directory.

`SVCUNREGFAILED` `Unregistering service [service name] failed`

Unregistering a service failed. An attempt to unregister a service (a database) failed due to a file operation error.

Solution Check space available in the `MATISSE_CFG` directory.

`TRPRCVFAILED` `Transport message receive failed`

An error occurred during message receive on `tcp` or `local` transport.

Solutions
- No more system resources. Check swap.
- Check your network.

`TRPSNDFAILED` `Transport message send failed`

An error occurred during a message send on `tcp` or `local` transport.

Solutions
- No more system resources. Increase them.
- Check your network.

`UNKNOWNMSG` `Unknown message received`

Unknown message received from client or server.

Solution Check your network.

4 Server Manager Listener

4.1 Introduction

Matisse Server Manager Listener (SMListener) manages remote operation requests on a local network. The `mt_smlistener` daemon also controls the denial of operations execution on the machine it is running on. This section describes how to start and administer `mt_smlistener` daemon on all supported platforms.

4.2 Managing Remote Operations

The SMListener daemon is responsible for creating new databases, starting and stopping databases as well as managing backups and restore operations. The SMListener daemon also collects server configuration information as well as activity and resource usage information. It includes CPU activity, memory consumption and disk usage.

4.3 Controlling Remote Operation Requests

The mt_smlistener daemon also controls the denial of operations execution. There are currently four operation types controlled by the SMListener:

- Grant/Revoke Operation controls
- Create/Init/Start/Stop databases
- Add/Update/Remove Datafiles
- Backup/Restore Databases

By default, there is no operation control and anyone who can access locally or remotely a database server can run DBA operations.

To enable DBA operation controls, you need to log on the server, run Matisse Enterprise Manager and select the 'Operation control enabled' check box. This operation will create a Matisse Server Manager administrator who will be able to grant/revoke DBA operation permissions to any Matisse local and remote users. A Matisse user needs to have a login on the local or remote machine.

NOTE: Only the user who has enabled DBA operation controls, can disable it.

4.4 Managing database autorestart

The SMListener utility is also responsible for restarting database servers automatically after a reboot of the machine. It restarts all databases with the `AUTORESTART` database configuration parameter set to 1. When this parameter is set to 0, no action is performed

4.5 Running mt_smlistener daemon

The `mt_smlistener` daemon handles all the remote operation requests sent by local or remote Enterprise Manager tools. The daemon is listening on a `TCP` port. The default port number used by the SMListener is **`7412`**.

Setting Up a Connection Environment

If you cannot use the default port number, you can redefine it. There are two ways you can do this:

- Configuration file `mtsmlistener.cfg` file in the `MATISSE_CFG` directory

You can update the `MATISSE_SMLISTENER_ADDR` parameter in the configuration file before starting the SMListener daemon.

- `MATISSE_SMLISTENER_ADDR` environment variable

You can use the `MATISSE_SMLISTENER_ADDR` environment variable to define port number used by the Enterprise Manager to connect to the SMListener.

SMListener Daemon Log File

Each time you start a SMListener daemon, a log file is opened in the `MASTISSE_LOG` directory. The name of the log file is `mtsmlistener.log`.

If a problem occurs, the SMListener daemon log file may contain a message.

Starting a SMListener Daemon

To start a SMListener daemon, use `mt_smlistener` as follows:

UNIX

1. Logon as root

Before starting the SMListener daemon, make sure that the `MATISSE_CFG` and `MATISSE_LOG` environment variables are defined. In addition, since the mt_smlistener is dynamically linked, it is necessary to update the dynamic library path.

You will need to have the Java Runtime Environment installed on your machine. You may run the `which` command to look for the location of the *`JAVA_DIR`* directory. For instance:

```
% which java
/opt/tools/j2se/bin/java
```

To define the environment variable from the bourne shell, you can set the environment variable with the following command:

Solaris

```
LD_LIBRARY_PATH=INSTALL_DIR/lib
JAVA_DIR/jre/lib/sparc/client\
export LD_LIBRARY_PATH
```

Linux

```
LD_LIBRARY_PATH=INSTALL_DIR/lib
JAVA_DIR/jre/lib/i386/client\
export LD_LIBRARY_PATH
```

1. After defining or updating these environment variables, use the following command to start the SMListener daemon:

```
root% mt_smlistener &
```

Windows

1. Logon with administrator privileges
2. Open a command window

```
Go to ‘bin’ under the Matisse installation directory. For
example,
  > C:
  > cd \Program Files\Matisse\bin
  > mt_smlistener -install
  > net start MATISSE_SML
  The MATISSE SMListener service is starting.
  The MATISSE SMListener service was started successfully.
```

3. Exit the command window

After typing these commands, the SMListener daemon is started and enabled.You can now logoff from this account and logon with your regular account in order to use Matisse.

Stopping the SMListener daemon

To stop the SMListener daemon, proceed as follows:

UNIX

1. Logon as root
2. Retrieve and kill the `mt_smlistener` process

Windows

1. Logon with administrator privileges
2. Open a command window

```
  > net stop MATISSE_SML
  The MATISSE SMListener service was stopped successfully.
  > mt_smlistener -remove
  MATISSE SMListener removed.
```

3. Exit the command window

After typing these commands, the SMListener service is stopped and removed.You can now logoff from this account and logon with your regular account in order to use Matisse.

4.6 Connections through Firewalls

In order to establish a connection to a SMListener daemon protected by a firewall, you must open in the firewall the `TCP` port used by the Server Manager Listener daemon.

To configure your database servers to accept remote `TCP` connections, update your firewall settings as follows:

1. Open the `TCP` port used by the SMListener daemon. The default port number is `7412`.

5 Matisse Access Control

This section describes how to set up access control, create users, and associate privileges to users.

5.1 Introduction

Matisse Access Control provides user/password security for database connections. Upon each connection, the server checks the validity of the user name and password and authorizes the connection if it matches the user description contained in the database system catalogs.

Different Privileges

There are three different levels of privileges which can be associated to a database user:

- The administrator can perform administrative operations, i.e. adding new users, and can read/write data and schema.
- The standard user can read/write data and schema.
- The read-only user can read data, but not modify it.

System User

For a given user the user/password security mechanism can either rely on the operating system access control, or use Matisse access control.

A “system user” is a user of the operating system who is identified by his login name. The system user can access a secured database without specifying a user name/password under the conditions which are described in the next section “Managing Users”.

The notion of system user provides a convenient way to define the default administrator when creating a new database, and to process batch commands without the need to enter a password.

Enabling Access Control

Access control specification is mandatory, so you have to specify a value for the `SECURITY` parameter in the configuration file of your database. In order to enable access control, you will have to set the value of `SECURITY` to 1 and then restart the database.

CAUTION: Once access control has been enabled, on a database it is impossible to disable it for this database.

When access control is not set, the values provided as user name and password at connect time are ignored and all connections are accepted.

5.2 Managing Users

Operating System Access Control

We detail here the notion of “system user” mentioned in the introduction.

The operating system access control is based on the login mechanism. When you connect to the database, Matisse looks for the database user name which corresponds to your login. If this user name exists within this database, and there is no password associated with it, Matisse trusts the system access control, and you can connect to the database.

Only the system user is allowed to connect without password. All other users must provide a non empty password when connecting.

The connection of a system user succeeds if the following conditions are fulfilled:

1. The user name exists in the database.
2. The user name must not have any password associated in the database.
3. The host on which the user is logged must be in relation with the host on which the database is running through a local area network (LAN), and
   a. either the user management is centralized on you LAN (using NIS for instance)
   b. either the user has a system account on the host on which the database server is running

Using Matisse Access Control

If you have to connect from a host that is not on the LAN, or simply if you prefer to use the Matisse access control, you just have to specify a password for the users. In this circumstance, when you connect, you will have to specify both the user name (which may be different from the login) and the password.

Add/Drop/ Modify Users

To manage users, you have to be an administrator. You can go through the Users menu from the Matisse DBA Tool or the shell command `mt_user`.

When adding a user, you can specify one of the privileges:

- ADM: to create an administrator
- RW: to create a standard user who can read/write data objects and schema objects
- RDONLY: to create a read-only user

Create an Administrator

As we just saw, an administrator can create another administrator with the same commands than for creating any other user.

The system user who did start/initialize the database is automatically declared to the database as an administrator with his login name as user name. As a consequence, any user on the system is a potential administrator of the database who has:

- `READ` privileges on the configuration file
- `WRITE` privileges on the `MATISSE_LOG` directory and on the datafiles
- `EXEC` privileges on the Matisse server (`mts` executable)

5.3 Database Connection API

The API for connecting to a database allow you to enter a user/password, and to set several connection options. We provide here an example using the C API:

1. You allocate a `MtConnection` object:

   ```
   MtConnection connection;
   MtAllocateConnection(&connection);
   ```

2. You may set some options, in particular the data access mode. This parameter can have the values: `MT_DATA_DEFINITION` which allows schema modification, `MT_DATA_MODIFICATION` which allows data modification, and `MT_DATA_READONLY` for read only access.

   For instance, to access data in read only mode:

   ```
   MtSetConnectionOption(connection,
                         MT_DATA_ACCESS_MODE,
                         MT_DATA_READONLY);
   ```

   The access mode should always be less or equal the level of permission defined for the user. For instance, a user with read only permission will not be able to connect with the `MT_DATA_MODIFICATION` option.

3. To connect, you use the `MtConnectDatabase` function. If the Matisse access control is not enforced you may specify `NULL` for both user name and password parameters:

   ```
   MtConnectDatabase(connection, host, database,
                     NULL, NULL);
   ```

   Otherwise, you will provide a non-null value for both user name and password:

   ```
   MtConnectDatabase(connection, host, database,
                     myusername, mypassword);
   ```

4. After disconnecting, you can deallocate the connection object:

   ```
   MtDisconnectDatabase(connection);
   MtFreeConnection(&connection);
   ```

6 Configuring a Database

A configuration file is associated to every Matisse database. This file is created by the Enterprise Manager when you create a new database, you may also edit it by hand if needed. Some parameters are used only when initializing or re-initializing a database, some require a database shutdown-restart.

6.1 Configuration File

The configuration file of a database defines the database parameters. You can use this file to:

- Update the values of the parameters
- Define the location and size of the datafiles upon initialization.

The Enterprise Manager process needs to be granted sufficient privileges on this file to be able to read and modify it.

File Syntax

The first line of the configuration file defines the database name. Note that this should be the same as the name of the configuration file without the `.cfg` suffix.

After the name are the parameters and their values. There is one parameter per line:

```
parameter_name:value
```

After the parameter definitions, the configuration file lists the datafile descriptions, with the following syntax:

```
datafile_path size [[, datafile_path size] ...]
```

Or, for mirrored datafiles:

```
datafile_path | mirror_datafile_path size [, ...]
```

6.2 Configuration Parameters

The database configuration file is provided so that you can define an initial setup for your database. It contains a list of database parameters that must be defined before a user can access the database. Other parameters are optional. By specifying values for these parameters you can improve the overall performance of Matisse.

Some of these values can be modified at any time by the current user, while other values require you to restart or reinitialize the database.

Mandatory Parameters

For each database configuration file, you must provide values for:

```
NAME
SECURITY
PATH
```

`NAME` defines the name of the database. `SECURITY` indicates whether access control is enabled or not. `PATH` defines the datafiles that contain the data. Each database must have at least one datafile of 400 data-pages.

Default Values

When you create a database by using the DBA Tool, the parameters are displayed with the default values as listed in *Table 6.1*.

Table 6.1 Default Values of Configuration Parameters

Parameter	Default Value
NAME	blank
PAGESIZ	8
CACHESIZ	64M
SECURITY	0
AUTOEXTEND	1
DATEXTENDSIZ	10
AUTOCOLLECT	1
AUTOCOLLECTFREQ	43
OBJTABLESIZ	0
OBJTABCLRFREQ	34
AUTORESTART	0
DATFULLINIT	0
DATINITSIZ	20
DATINMEMORY	0
MEMORYTRANS	0
MAXSQLDOP	0
MAXSQLTHRDPOOL	0
MAXSRVLOGFILES	7
MAXBKPLOGFILES	7
TCPKEEPALIVE	0
PORTS	0-0
PATH	blank

Automatically Updated Parameters

The parameter PATH is updated automatically when adding, removing or resizing datafiles with the Enterprise Manager or the shell level administration commands. Upon establishing server-side replication, the REPLICA and REPLICATES parameters are automatically added. These replication parameters are not part of the initial configuration for creating your database.

CAUTION: Once the database is initialized, do not update these automatic fields by hand.

NAME

Purpose — This parameter defines the name of the database for which the configuration file was created.

Server Use — The server uses NAME at initialization and restart.

Type — NAME is a character string.

PAGESIZ

Purpose — In a Matisse database, all data is stored in basic units that are called datapages. A datapage is the minimum amount of data that can be read from or written to disk by the Matisse server in any single I/O operation.

The parameter PAGESIZ specifies the size of a Matisse datapage in kilobytes. The parameter range is 8 to 63.

The value of PAGESIZ has some influence on the amount of bytes read or written for each I/O.

When setting PAGESIZ for a database, you must take into account the amount of data that is regularly modified on it.

If the quantity of data modified by a transaction is relatively small—that is to say, if the transactions read, write, or modify a relatively small amount of data—a PAGESIZ somewhat larger than the average amount of data is appropriate. A value that is compatible with the average amount of data to be modified will also help reduce the number of I/O operations required to access the data.

Selecting a datapage size somewhat larger than the object size will also optimize disk space use.

The datapage is also used for internal structures (for example, index, btrees).

Server Use — The server uses the PAGESIZ parameter at initialization.

Type — The PAGESIZ parameter is of type integer.

Default Value — The default value of the PAGESIZ parameter is 8 kB.

CACHESIZ

Purpose — CACHESIZ is the size, expressed in datapages, of the server cache dedicated to the database and located in system memory. The server cache contains the database datapages which have been most recently used.

This parameter has a direct influence on system performance. If the size of CACHESIZ is too close or even larger than the available system memory, the cache can be swapped to disk and thereby reduce system performance.

This parameter can be in one of the following units:

- Kilobytes
- Megabytes
- Gigabytes
- Datapages

The value for this parameter is a number followed with an uppercase or lowercase K, M, or G for kilobytes, megabytes, or gigabytes. If you do not specify one of these units, the DBA will interpret the value as a number of datapages.

The minimum value of CACHESIZ is 1250 datapages. The maximum size is 1.4 gigabytes, except with the 64-bit version of Matisse, which is limited only by available memory.

Server Use — The server uses the CACHESIZ parameter at initialization and restart.

Type — The CACHESIZ parameter is of type integer.

Default Value — The default value of the CACHESIZ parameter is 1250 datapages.

SECURITY

Purpose — Setting a value of 1 for SECURITY will enforce access control security for the database. Once this parameter is set, only the operating system user who started the database can access it. This user must create explicitly new users through the Enterprise Manager or the mt_user command to allow other users to connect to the database.

Server Use — The server uses the SECURITY parameter at initialization and restart. Once this parameter has been set to 1, access control cannot be turned off without reinitializing the database.

Type — The SECURITY parameter is of type integer.

Default Value — The SECURITY parameter has a no default value, it must be explicitly specified.

AUTOEXTEND

Purpose — When this parameter value is set to 1, when a datafile is full the Matisse Server automatically extends it to make room for more objects. When it is set to 0, datafiles must be extended manually using the Enterprise Manager or the mt_file command.

The setting of this parameter has no effect on disk partition (raw device) datafiles.

Type — The AUTOEXTEND parameter is of type integer.

Default Value — The default value of the AUTOEXTEND parameter is 1.

DATEXTENDSIZ

Purpose — DATEXTENDSZ is the minimum datafile extension size used when the datafile is full.

This parameter is expressed in Megabytes.

The minimum value of DATEXTENDSZ is 4 Megabytes. The maximum size is 128 Megabytes.

Server Use — The server uses the DATEXTENDSZ parameter at initialization and restart.

Type — The DATEXTENDSZ parameter is of type integer.

Default Value — The default value of the DATEXTENDSZ parameter is 10.

AUTOCOLLECT

Purpose — When this parameter value is set to 1, the Matisse Server performs the automatic collection of obsolete versions in order to reclaim disk space. When it is set to 0, no automatic collection is performed.

In both cases, you can also run manual collects with the `mt_server collect` command.

When it is set, the automatic collection is triggered automatically in two cases:

- When a datafile becomes full, in order to reclaim disk space.
- When many new versions have been created in the database. The collection of versions occurs regularly for write intensive applications.

This feature prevents the database from growing when large amounts of updates are performed on existing objects, and reduces the need to run manual version collects.

Server Use The server uses the `AUTOCOLLECT` parameter at initialization and restart.

Type The `AUTOCOLLECT` parameter is of type integer.

Default Value The default value of the `AUTOCOLLECT` parameter is 1.

AUTOCOLLECTFREQ

Purpose This parameter defines the run frequency of the automatic version collection operation. This parameter is expressed in seconds. The minimum value is 5 seconds, the maximum size is 360 seconds.

Type The `AUTOCOLLECTFREQ` parameter is of type integer.

Default Value The default value of the `AUTOCOLLECTFREQ` parameter is 43.

OBJTABLESIZ

Purpose This parameter defines the maximum size of the multi-version object table memory cache. The pages in the object table cache are only allocated when required. This parameter is expressed megabytes (M suffix), gigabytes (G suffix). Specifying a value of 0 disables the control of the maximum size.

Type The `OBJTABLESIZ` parameter is of type integer.

Default Value The default value of the `OBJTABLESIZ` parameter is 0.

OBJTABCLRFREQ

Purpose — This parameter defines the run frequency of the object table clearing operation. This parameter is expressed in seconds. The minimum value is 3 seconds, the maximum size is 120 seconds.

Type — The `OBJTABCLRFREQ` parameter is of type integer.

Default Value — The default value of the `OBJTABCLRFREQ` parameter is 34.

AUTORESTART

Purpose — This parameter defines the automatic restart of a database when the machine is rebooted. When this parameter value is set to 1, the Matisse Server Manager Listener (SMListener) restarts the database server automatically after a reboot of the machine. When it is set to 0, no action is performed.

Type — The `AUTORESTART` parameter is of type integer.

Default Value — The default value of the `AUTORESTART` parameter is 0.

DATFULLINIT

Purpose — When this parameter value is set to 1, the Matisse Server performs the full datafile initialization before becoming online. When it is set to 0, the Matisse Server is online as soon as the minimum datafile size (`DATINITSIZ`) is initialized.

Server Use — The server uses the `DATFULLINIT` parameter at initialization.

Type — The `DATFULLINIT` parameter is of type integer.

Default Value — The default value of the `DATFULLINIT` parameter is 0.

DATINITSIZ

Purpose — `DATINITSIZ` is the minimum datafile size to be initialized before the server is online. This parameter has no effect when `DATFULLINIT` is set to 1.

This parameter is expressed in Megabytes.

The minimum value of `DATINITSIZ` is 20 Megabytes. The maximum size is 256 Megabytes.

Server Use The server uses the DATINITSIZ parameter at initialization.

DATINMEMORY

Purpose DATINMEMORY defines the primary location of the datafiles. Specifying a value of 1 enables an in-memory database. The datafiles are in-memory. When it is set to 0, the datafiles are located on disks.

Server Use The server uses the DATINMEMORY parameter at initialization.

Type The DATINMEMORY parameter is of type integer.

Default Value The default value of the DATINMEMORY parameter is 0.

MEMORYTRANS

Purpose When this parameter value is set to 1, the shared memory transport is enabled. See the discussion of MT_MEMORY_TRANSPORT in the *Matisse C API Reference* or one of the other API or binding references for further discussion.

Server Use The server uses the MEMORYTRANS parameter at initialization and restart.

Type The MEMORYTRANS parameter is of type integer.

Default Value The default value of the MEMORYTRANS parameter is 1.

MAXSQLDOP

Purpose MAXSQLDOP defines the maximum degree of parallelism which determines the maximum number of threads that are being used.

Specifying a value of 0 disables parallel processing.

The maximum value of MAXSQLDOP is the number of logical CPUs on the server.

Server Use The server uses the MAXSQLDOP parameter at initialization and restart.

Type The MAXSQLDOP parameter is of type integer.

Default Value The default value of the MAXSQLDOP parameter is 0.

MAXSQLTHRDPOOL

Purpose — MAXSQLTHRDPOOL defines the maximum number of threads in the pool of threads dedicated to parallel processing of SQL queries

Specifying a value of 0 disables parallel processing.

The maximum value of MAXSQLTHRDPOOL is twice the number of logical CPUs on the server.

Server Use — The server uses the MAXSQLTHRDPOOL parameter at initialization and restart.

Type — The MAXSQLTHRDPOOL parameter is of type integer.

Default Value — The default value of the MAXSQLTHRDPOOL parameter is 0.

MAXSRVLOGFILES

Purpose — MAXSRVLOGFILES represents the maximum version number of recycled server log files of the Matisse Server saved.

The minimum value of MAXSRVLOGFILES is 1. The maximum value is 32.

Server Use — The server uses the MAXSRVLOGFILES parameter at initialization and restart.

Type — The MAXSRVLOGFILES parameter is of type integer.

Default Value — The default value of the MAXSRVLOGFILES parameter is 7.

MAXBKPLOGFILES

Purpose — MAXBKPLOGFILES represents the maximum version number of recycled backup log files of the Matisse Server saved.

The minimum value of MAXBKPLOGFILES is 1. The maximum value is 32.

Server Use — The server uses the MAXBKPLOGFILES parameter at initialization and restart.

Type — The MAXBKPLOGFILES parameter is of type integer.

Default Value — The default value of the MAXBKPLOGFILES parameter is 7.

TCPKEEPALIVE

Purpose TCPKEEPALIVE defines the keepalive control of the server TCP/IP connections. When enabled, it verifies on a regular basis that the endpoint at the remote end of the connection is still available. OS-specific value of the keepalive intervals are controllable at the system level. Specifying a value of 1 enables keepalive control. Specifying a value of 0 disables keepalive control.

On Windows, the default settings when a TCP socket is initialized sets the keep-alive time-out to 2 hours and the keep-alive interval to 1 second. The default system-wide value of the keep-alive time-out is controllable through the KeepAliveTime registry setting which takes a value in milliseconds. The default system-wide value of the keep-alive interval is controllable through the KeepAliveInterval registry setting which takes a value in milliseconds. The number of keep-alive probes (data retransmissions) is set to 10 and cannot be changed.

On Linux, the default settings when a TCP socket is initialized sets the keep-alive time-out to 2 hours and the keep-alive interval to 75 second. The number of keep-alive probes (data retransmissions) is set to 9. The default system-wide value of the keep-alive parameters is controllable through the following files:
/proc/sys/net/ipv4/tcp_keepalive_intvl
/proc/sys/net/ipv4/tcp_keepalive_probes
/proc/sys/net/ipv4/tcp_keepalive_time

Server Use The server uses the TCPKEEPALIVE parameter at initialization and restart.

Type The TCPKEEPALIVE parameter is of type integer.

Default Value The default value of the TCPKEEPALIVE parameter is 0.

PORTS

Purpose This parameter is used to specify a port number or a range of port numbers for client-server connections. When this parameter is not specified, Matisse selects the first port number available in the system.

If you need to specify a single port number, simply put the number. If you need to specify a range of port numbers, put the start-number, a hyphen, and the end-number, e.g.,

```
PORTS: 7422-7431
```

The PORTS numbers should not include the port number used by the Matisse port monitor, which is 7421 by default.

PORTS is available on the Windows platforms and on Linux.

Server Use — The server uses the PORTS parameter when establishing a connection to a client.

Type — The PORTS parameter is of type integer.

Default Value — There is no default value for this parameter.

PATH

Purpose — PATH specifies the location of one or more datafiles to be created automatically by Matisse when it initializes the database. A datafile is a quantity of disk space reserved for your database. With Matisse, all data is stored in datafiles.

At least one datafile path must be declared in the database configuration file. If more than one is declared, it is recommended that all the datafiles be declared with the same size. To benefit fully from certain Matisse features, such as load balancing, a database should have datafiles of equal size.

For each datafile, indicate the path where it is located and its size expressed in kilobytes (K), megabytes (M), or gigabytes (G). If you do not specify one of these units, kilobytes is used as the default. A datafile must be greater than 400 datapages (400 x PAGESIZ) and may not exceed 64 million datapages.

There can be more than one datafile per line in the configuration file if they are separated by commas, as in the following example:

```
/path1 10000, /path2 10000
```

Up to 31 datafiles may be specified. Note that since a unit was not specified in the above example, each datafile will be 10,000 kilobytes.

For increased reliability, you may mirror your datafiles. To do so, specify two paths separated by the vertical bar symbol. For example, to mirror two unformatted partitions (raw devices):

```
/dev/rdisk/c1t3d0s1 | /dev/rdisk/c2t3d0s1 2G
```

Up to 31 mirrored pairs may be specified.

CAUTION: In a production environment, we strongly recommend that you specify only one datafile per physical disk.

Server Use — The server uses the PATH parameter at initialization.

Type — The PATH parameter must be defined for one or more datafiles. Each datafile is defined by a pathname and a size.

6.3 Using Disk Partitions as Datafiles

Why Use Partitions?

By using disk partitions, also called raw devices, you eliminate the risk of a file system corruption and can improve the speed of the read and write disk access of a database.

You can initialize a partition that is not used by the operating system and allocate a partition to Matisse either with the DBA Tool or at the command line using `mt_partition init` and `mt_partition alloc`.

Before declaring a partition in the configuration file, the partition must not:

- Contain the first sector of the disk because the first sector contains the disk label and must not be corrupted
- Contain the whole disk
- Have a file system
- Be a swap partition

Choose the partition that you want to use with care. The partition should not already be in use by a different software application or by an operating system utility.

Check for Partitions That Contain the First Sector on UNIX

To find the names of the available disks, you may use the `format` command. You need to be a superuser to use this command.

To see if a partition contains the first sector of a disk, use the `dkinfo` command. This command is an operating system command that displays information on the disk.

For example, if you want to list the partitions on the disk `sd1`, type the following command:

```
dkinfo sd1
```

The type of information returned is as follows:

```
sd1: SCSI CCS controller at addr f0800000, unit # 8
2073 cylinders 21 heads 94 sectors/track
a: 98700 sectors (50 cyls)
  starting cylinder 0
b: 3849300 sectors (1950 cyls)
   starting cylinder 1950
c: 4092102 sectors (2073 cyls)
   starting cylinder 0
d: 144102 sectors (73 cyls)
   starting cylinder 2000
e: No such device or address
```

```
f: No such device or address
g: No such device or address
h: No such device or address
```

For each disk partition, the size and the starting cylinder are displayed. Partition `a` on disk `sd1` for example, has a starting cylinder of 0. This means that partition `a` contains the first sector of the disk `sd1`. This partition does not meet condition a described above. You cannot, therefore, declare the partition `a` on `sd1` for use by Matisse.

Partition d on disk `sd1` on the other hand, has a starting cylinder of 2000. It does not contain the first sector of the disk. This partition and partition b with a starting cylinder of 1950 meet the conditions a and b described above.

Partition `sd1c` contains the entire disk and therefore contains the first sector of the disk. This partition cannot be used by a Matisse database.

Checking Partitions with File Systems on UNIX

You must also verify that a partition does not have a file system. To do this type the following operating system command:

```
mount
```

The host displays information on the partitions that have a file system. For example, on a SunOS host the following partitions may be listed:

```
/dev/sd1a on / type 4.2 (rw)
/dev/sd1b on /usr type 4.2 (rw)
/dev/sd1g on /mima_free type 4.2 (rw)
```

On a Solaris host, the following partitions may be listed:

```
/ on /dev/dsk/c0t3d0s0 read/write/setuid on Thu Apr 7
18:17:49 1998
/usr on /dev/dsk/c0t3d0s6 read/write/setuid on Thu Apr 7
18:17:49 1998
```

The above display indicates that the partitions have a file system and cannot be used by a Matisse database.

Checking for Swap Partitions on UNIX

You must also verify that a partition is not a swap partition. Consult with your system administrator if you do not know if a partition is a swap partition.

Declaring a Partition in a Configuration File

You can define a disk partition that you want to use in the database configuration file directly when you create a database. You can also add a partition at any time to an online database by using the `mt_file` command.

Note that in UNIX, the same physical partition on a disk can have two different names, depending on the mode in which it is accessed. In Solaris, a partition named `/dev/dsk/c0t1d0s2` is referenced as `/dev/rdsk/c0t1d0s2` when accessed in raw device mode.

Matisse accesses partitions in raw mode, but lets you specify a partition with the name used outside raw mode. However, to remind you that Matisse uses the partitions in raw device mode, all the DBA Tool dialogs dealing with partitions specify them in raw mode.

MS Windows

On MS Windows, you can manage disk partition by selecting the Disk Administrator Tool. A partition name E can be referenced as \\.\E: when accessed in raw device mode.

If you define a partition for use before initializing a database, you need to list it in the configuration file as you would any other datafile.

Linux

On Linux, you can setup a disk partition as on the following example, assuming that the partition `/dev/sdb2` is available. Note that on Linux you need to define a symbolic link which ends with the physical name of the partition, as is shown here:

```
[root]# raw /dev/raw/raw1 /dev/sdb2
[root]# ln -s /dev/raw/raw1 /dev/raw/sdb2
[root]# mt_partition init -f /dev/raw/sdb2
```

Then you can declare it in your database configuration file:

```
PATH: /dev/raw/sdb2 2221856K
```

The user starting the Matisse server needs to have the read-write permission on the devices, e.g., `/dev/raw/raw1` and `/dev/sdb2`.

The raw devices need to be listed in the file `/etc/sysconfig/rawdevices` to define a set of raw device mappings automatically created during the system startup, for example:

```
/dev/raw/raw1 /dev/sdb2
```

7 Using the Enterprise Manager

This section shows you how to use the Matisse Enterprise Manager to create and manage your database servers.

The Enterprise Manager regroup in a single tool: the distributed management of database servers, the management of database schemas, the data import and export in table (relational) and XML formats, as well as various security and administration functions. It includes an Object Viewer to browse and edit object hierarchies stored in a database. It also includes a SQL analyzer tool to help optimize complex queries and produce data result-sets in a table format.

7.1 Starting the Enterprise Manager

Before starting the Enterprise Manager, make sure that the `MATISSE_CFG` and `MATISSE_LOG` environment variables are defined. In addition, since the Enterprise Manager is dynamically linked, it is necessary to update the dynamic library path.

You will need to have the Java Runtime Environment installed on your machine. You may run the `which` command to look for the location of the *`JAVA_DIR`* directory. For instance:

```
% which java
/opt/tools/j2se/bin/java
```

To define the environment variable from the bourne shell, you can set the environment variable with the following command:

Solaris

```
LD_LIBRARY_PATH=INSTALL_DIR/lib
JAVA_DIR/jre/lib/sparc/client\
export LD_LIBRARY_PATH
```

Linux

```
LD_LIBRARY_PATH=INSTALL_DIR/lib
JAVA_DIR/jre/lib/i386/client\
export LD_LIBRARY_PATH
```

After defining or updating these environment variables, use the following command to start the Enterprise Manager:

```
mt_emgr
```

7.2 Remote Administration

The Enterprise Manager provides full remote administration features for distributed Matisse database servers on a local network. All administration tasks can be executed remotely via the enterprise manager. This includes database start and shutdown, database backup and restore, server monitoring, database monitoring, database access control and database schema and data manipulation.

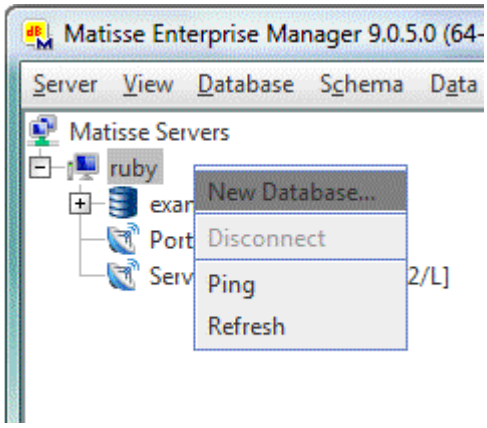
7.3 Creating a Database

The creation of a database involves the following steps:

- Create a configuration file
- Specify the datafile(s) in the configuration file
- Initialize the database

To perform these tasks with the Enterprise Manager, right click on the node that represents your host machine, then select New Database.

Figure 7.1 Create Database



You must then enter a name for the database. Note that the database name should not exceed twelve characters. The other tabs in the Create New Database window allow you to modify the description of the datafiles, or change some configuration options.

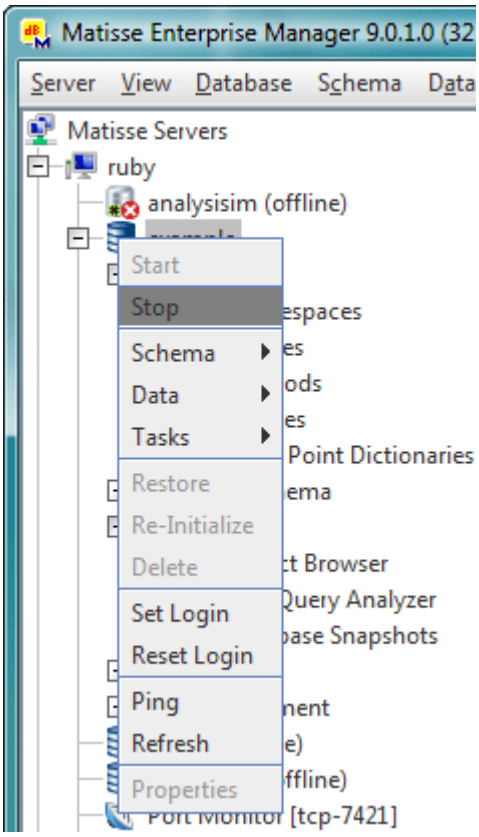
After creating your database, you can start it with a right click on the database node, then select Start. The first start will take more time as it initializes the datafiles.

7.4 Stopping a Database

Before stopping a database, you must verify that there are no users connected on it. After the database has been stopped, it becomes off-line. Users are no longer able to access the data. When the database is off-line, the root or owner account can modify the database configuration file.

To stop a database from the Enterprise Manager, right click on the database node and then select Stop.

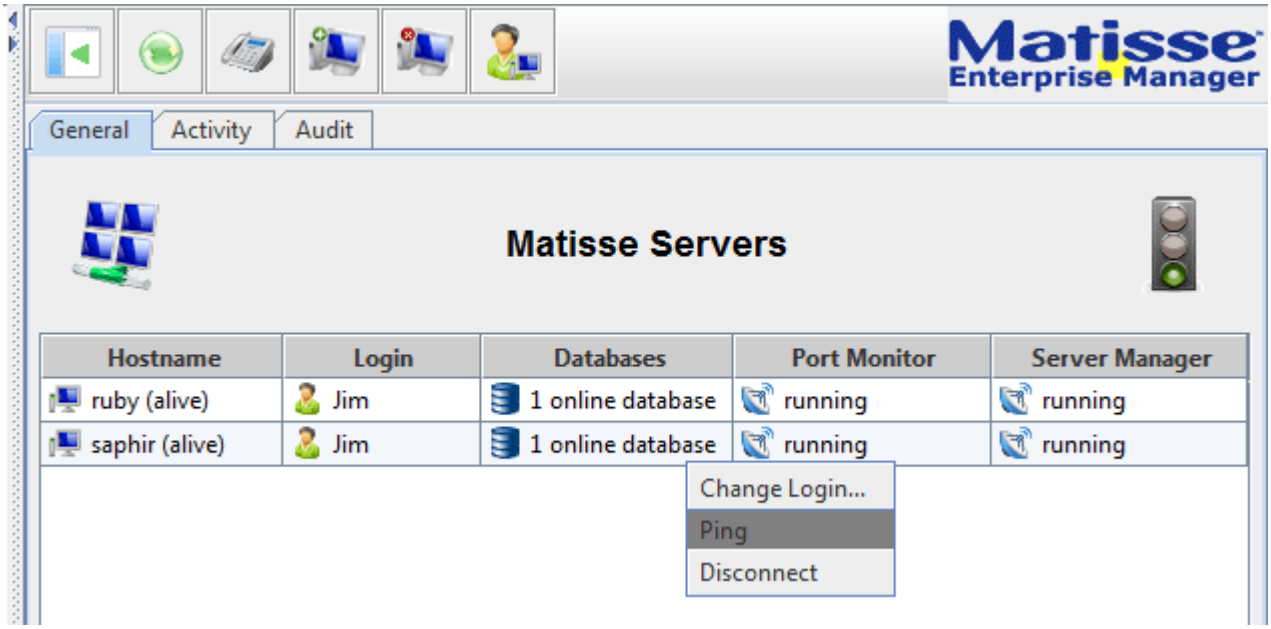
Figure 7.2 Stop Database



7.5 Monitoring Database Server

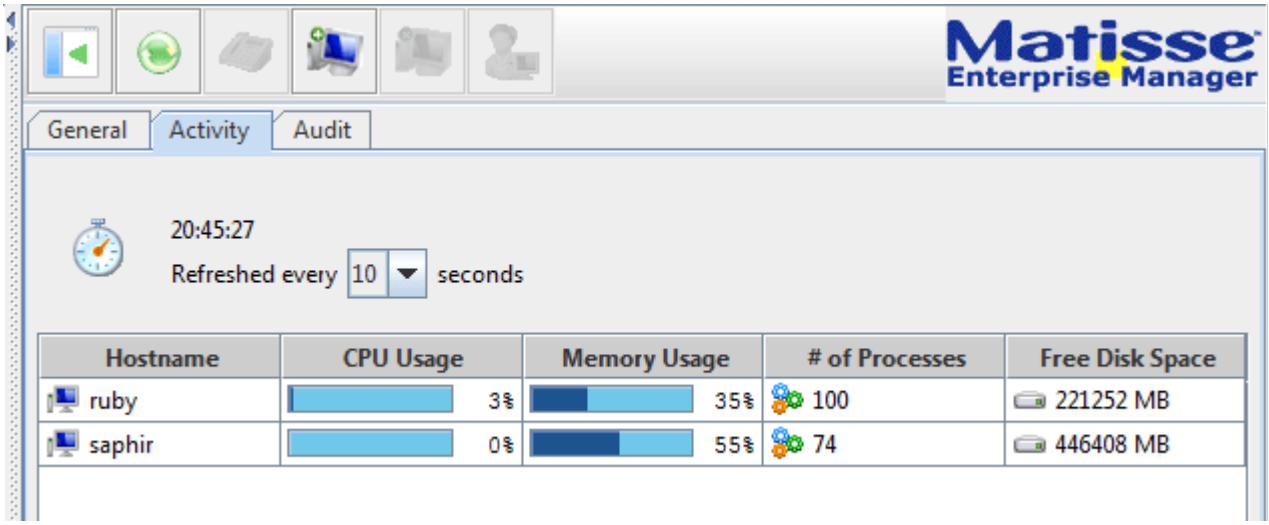
The Enterprise Manager monitors registered database servers reporting in real-time when servers, databases or any other Matisse services are down.

Figure 7.3 Database servers state monitoring



It also includes real-time monitoring of CPU activity, memory consumption and disk usage of database servers.

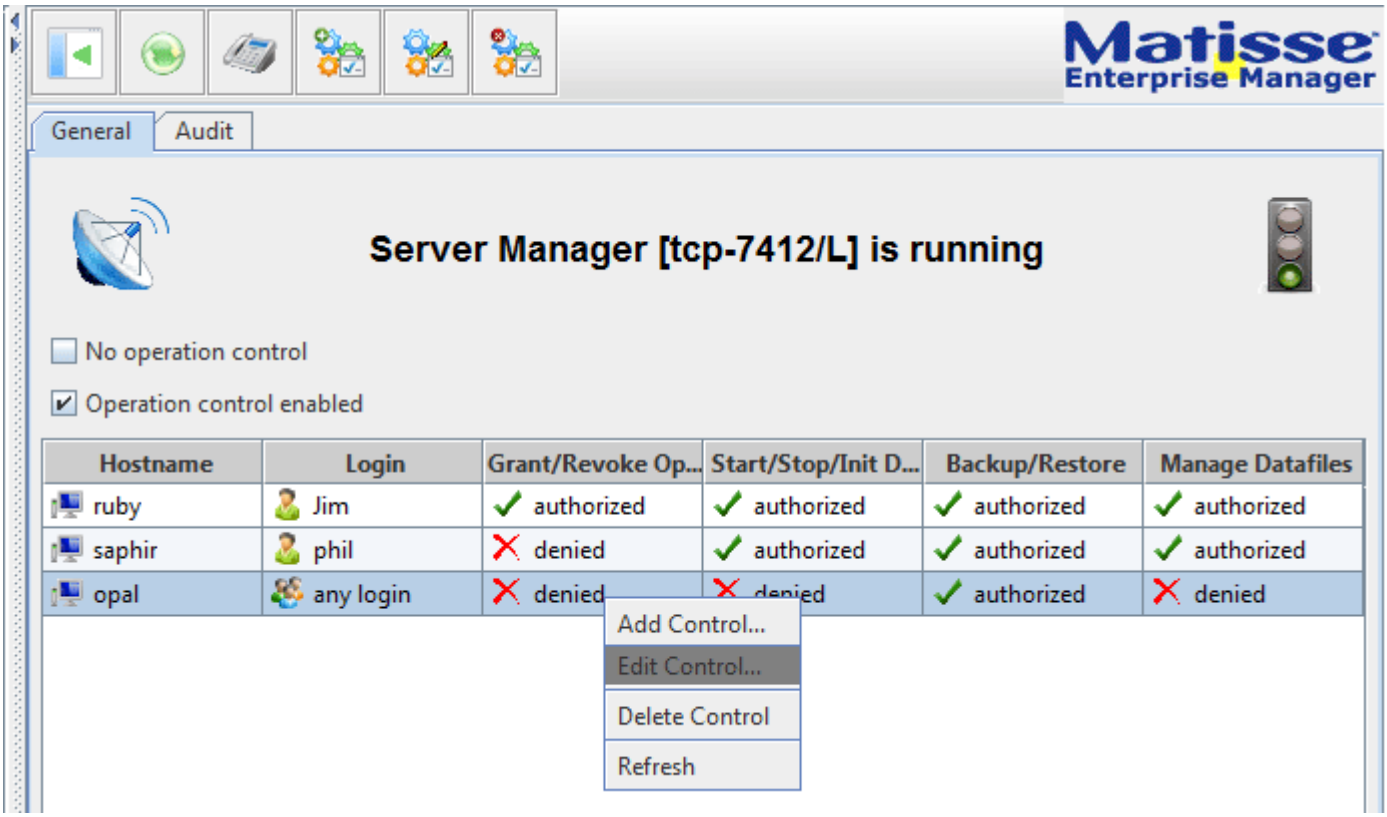
Figure 7.4 Servers activity and resources monitoring



7.6 Managing Database Server Operation Control

The Server Operation Control Manager tool provides security control for executing local or remote administration operations including start/stop databases, backup/restore and datafile management.

Figure 7.5 Server Operation Control Manager



7.7 Managing Database Users

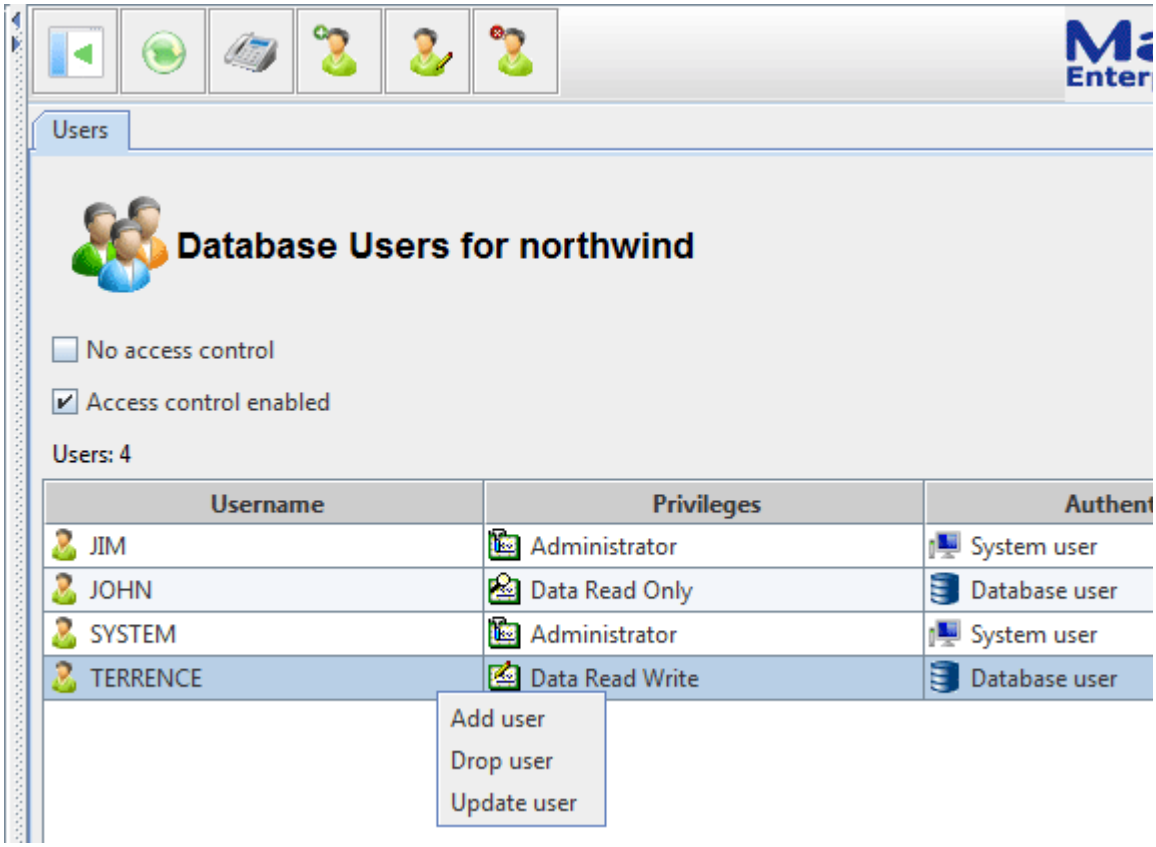
The Matisse access control feature can be enforced in a database by setting to 1 the `SECURITY` parameter in the configuration file. Once access control is enforced all clients must provide a valid user and password to be able to connect to the database server.

You can manage users only when the database is on-line. By clicking on the Users node under Security, you can perform the following operations:

- Add a user
- Drop a user

- Update user password and/or privilege

Figure 7.6 Adding a user



Alternatively, you can perform these tasks using the equivalent shell commands discussed in *section 9, Administration Commands*.

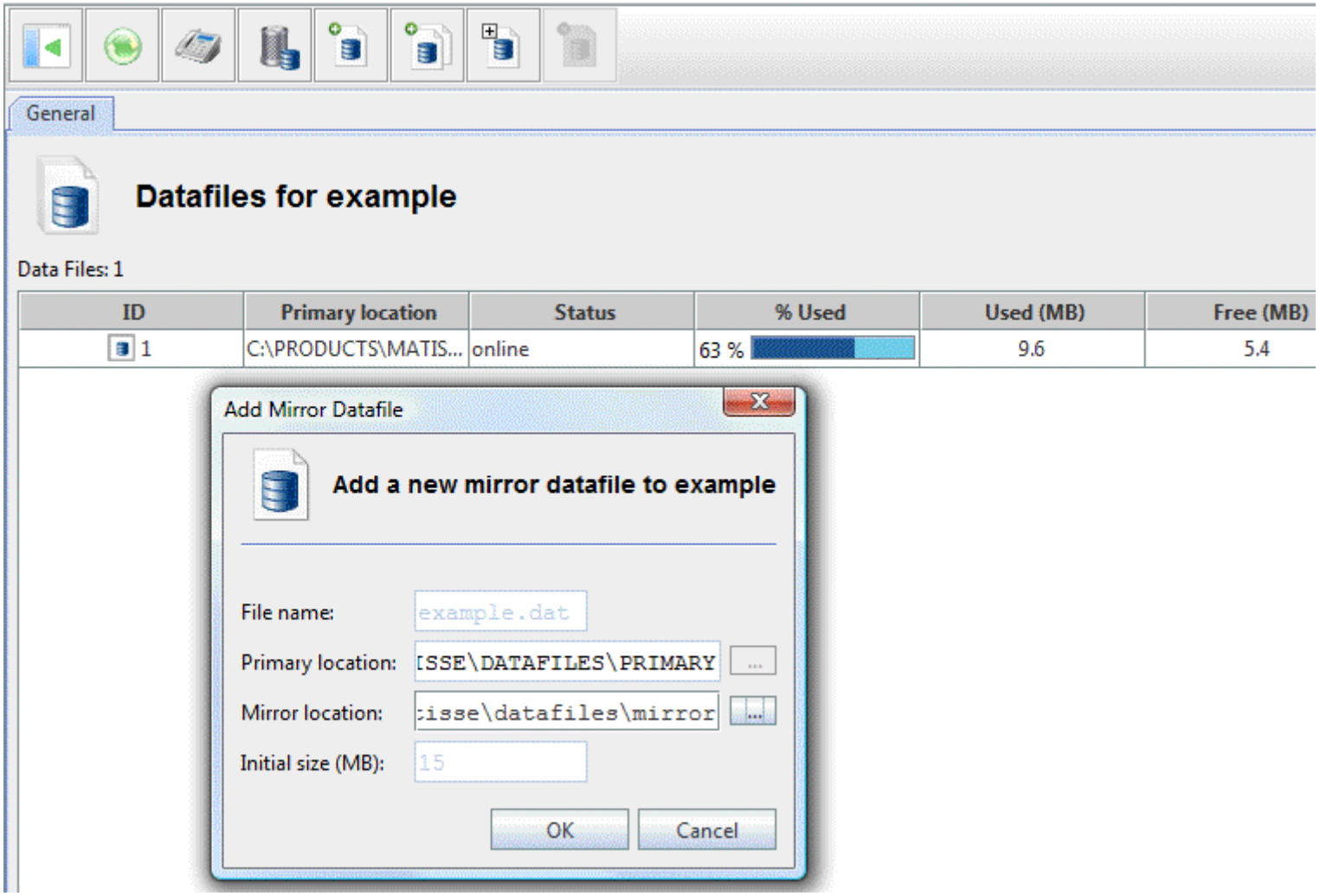
7.8 Managing Datafiles

Once your database has been created, you can manage datafiles only when the database is on-line. Under your database node, open Management, then click on Datafiles node, from the General tab you can:

- Add a primary datafile or a mirror datafile to the database
- Delete a datafile
- Increase the size of a datafile

For instance to create a mirror datafile to an existing primary datafile, you will just re-enter the path for the primary datafile, then the path for the new mirror datafile. The size should be identical to the primary datafile size.

Figure 7.7 Creating a mirror datafile



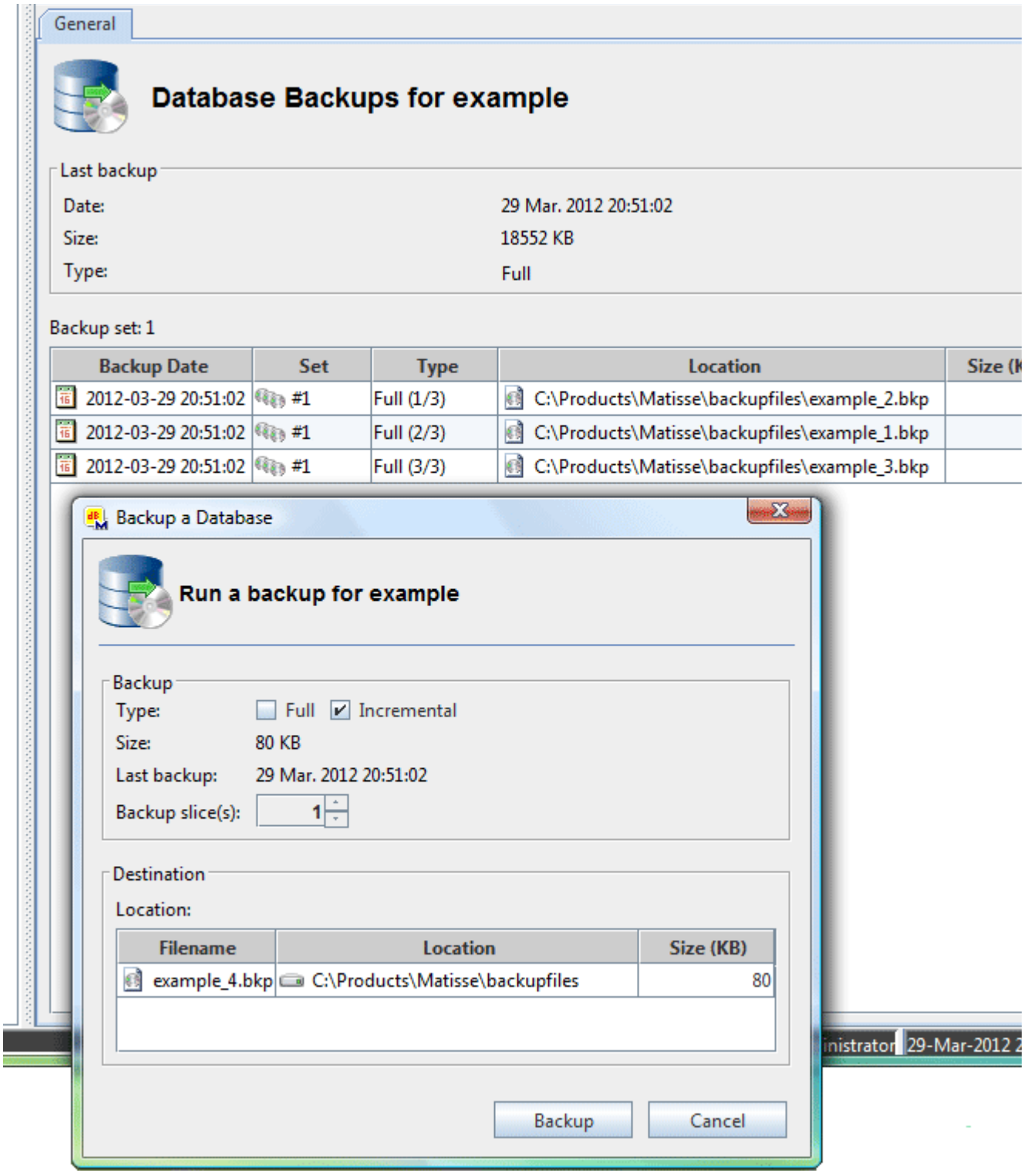
Alternatively, you can perform the same tasks using the equivalent shell commands discussed in *section 9, Administration Commands*.

To take advantage of Matisse automatic load balancing and for optimal performance, all datafiles should be the same size, all disks should be the same type, and should use the same type of controller.

7.9 Managing Backups

Matisse Database Backup tool allows users to perform full and incremental parallel backups of databases while the system in online. There is no need to block updates during a backup, as the Matisse server keeps a snapshot of the database at the time of the beginning of the backup operation.

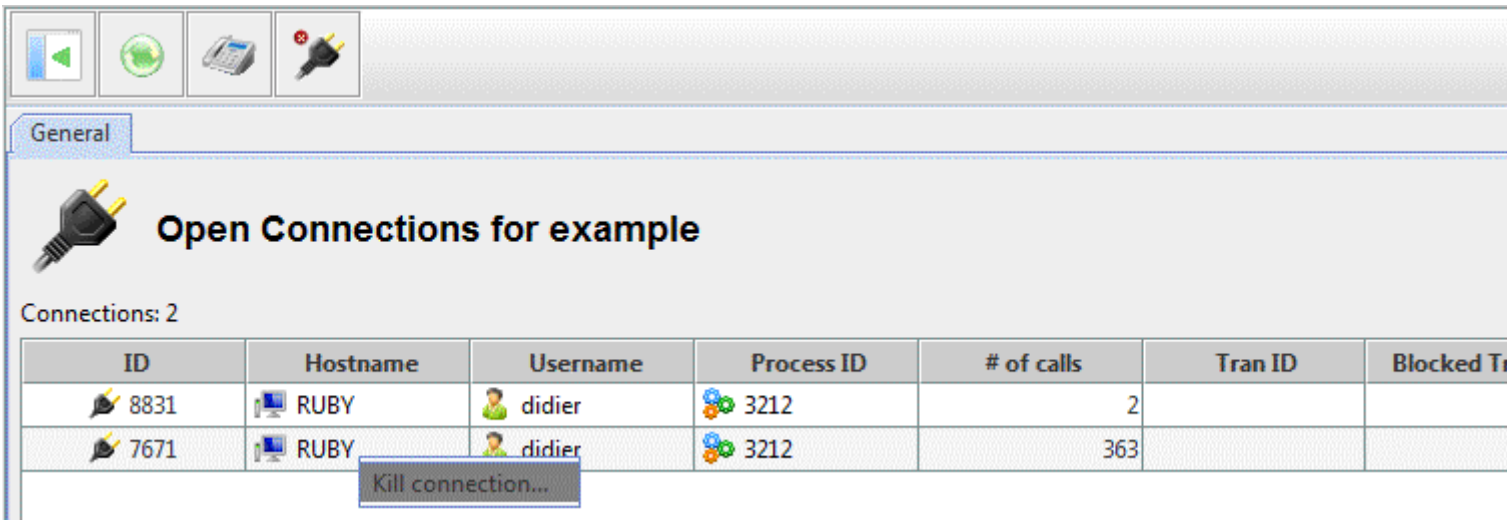
Figure 7.8 Database Backup Manager



7.10 Managing Open Connections

You can see the open connections from the Connections sub-node of an online database. You can also kill a connection. For this, display the connections and right click on an element to display the menu.

Figure 7.9 Killing an active connection

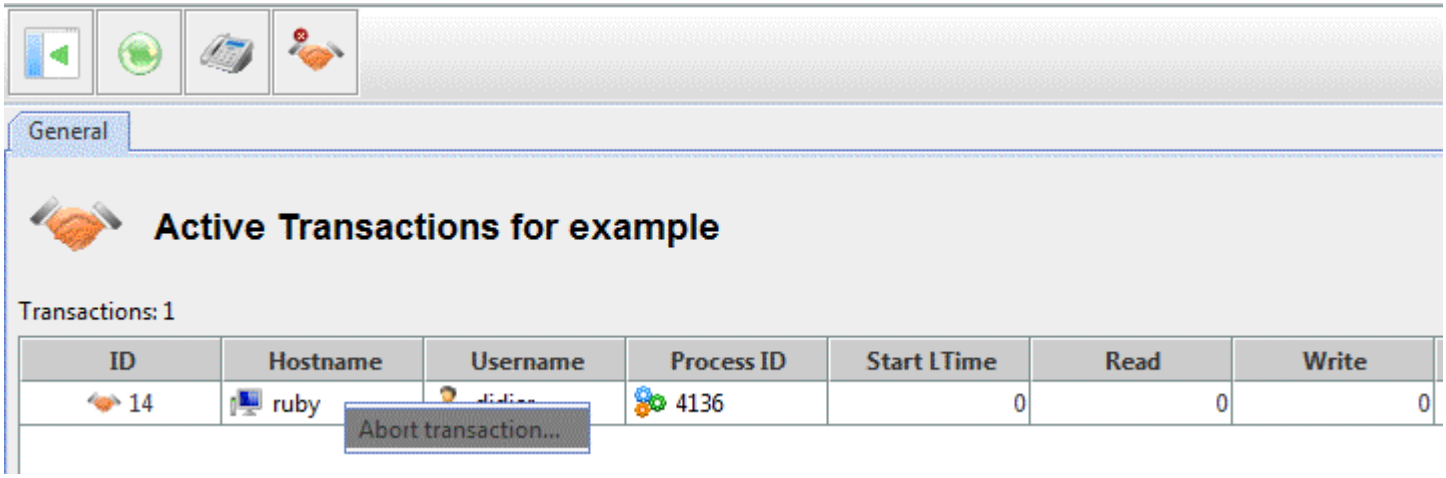


CAUTION: When a connection is killed, the active transaction if any is aborted, all the operations (read, write, create, delete) carried out by the transaction are cancelled.

7.11 Managing Active Transactions

You can see the active transactions from the Transactions sub-node of an online database. You can also abort a transaction from the Transactions window. For this, display the transactions and right click on an element to display the menu.

Figure 7.10 Aborting a transaction from the Monitor window



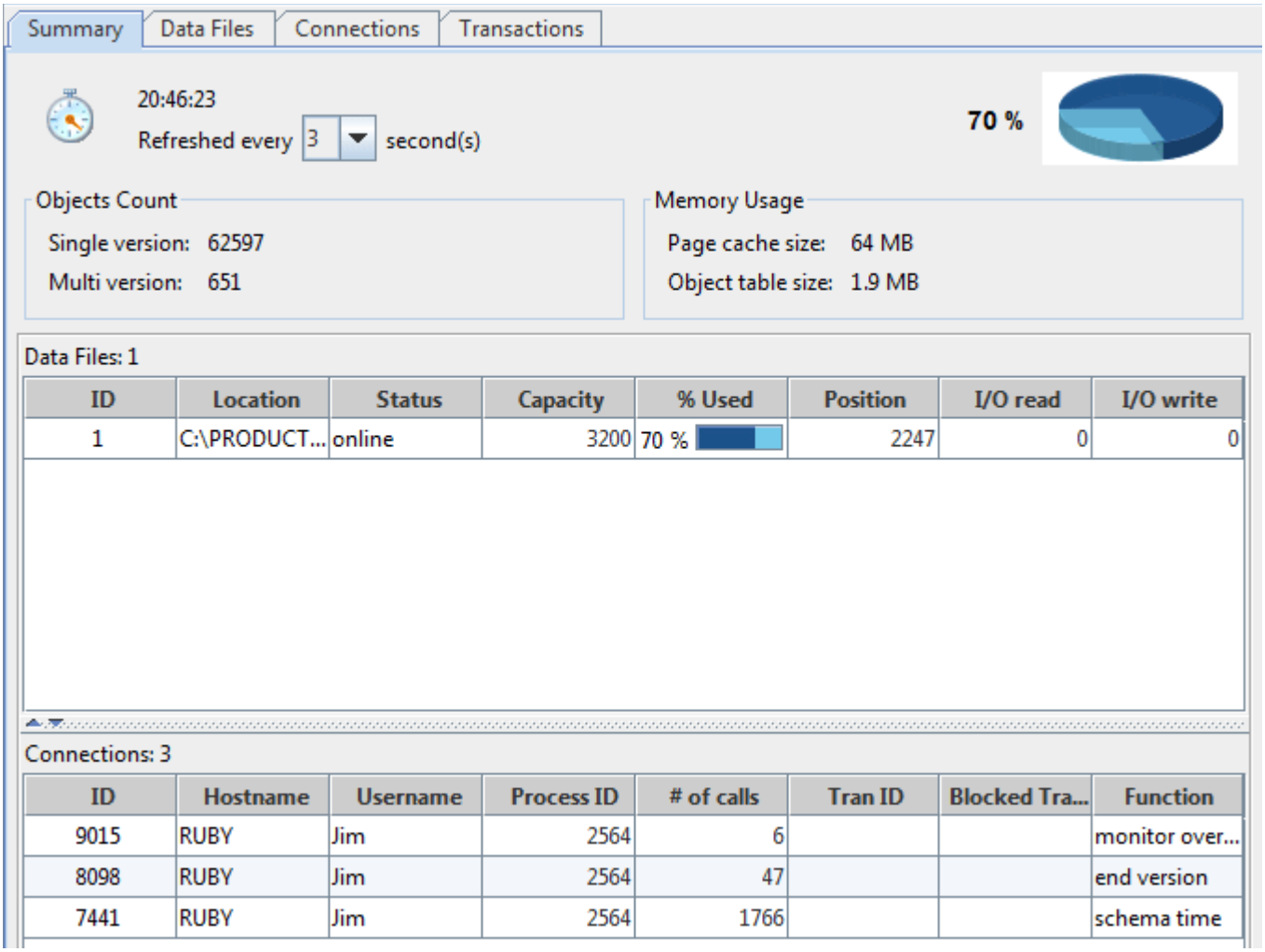
CAUTION: When a transaction is aborted, all the operations (read, write, create, delete) carried out by the transaction are cancelled.

7.12 Monitoring a Database

You can see database snapshots from the Monitor window of the Enterprise Manager. Click on refresh to update the current snapshot.

You can also kill a connection or abort a transaction from the Monitor window. For this, display the connections or transaction and right click on an element to display the menu.

Figure 7.11 Matisse Monitoring:



Matisse Monitor presents detailed information on the currently selected database. It provides information on the connections to the database, transactions performed on the database, and datafile activity.

Summary information about connections, as described in the following table, appears at the top of the Overview and Connection tabs.

Table 7.1 Summary Information about All Connections

Field	Contents
`Committed`	Number of transactions that have committed since the database was started (or restarted)
`Read Locked`	Number of objects currently being locked.
`Aborted`	Number of transactions aborted since the database was started (or restarted)
`Blocked`	Number of transactions currently blocked.
`Open`	Number of open connections
`Deadlocks`	Number of deadlocks since the database was started (or restarted).

Detailed information about connections, as described in the following table, follows the summary on both the Overview and Connection fields.

Table 7.2 Detailed Information about Specific Connections

Field	Contents
`HostName`	Name of the server host
`UserID`	Name of the server host
`ProcessID`	ID of the user connected to the database
`Calls`	ID of the process connected to the database
`TranID`	ID of the transaction that blocks the transaction listed under Blocked. Note that either both the Tran ID and Blocked fields are filled or neither are filled.
`Blocked`	ID of a transaction that is blocked. Note that either both the Tran ID and Blocked fields are filled or neither are filled.
`Function`	Function calling the server when the monitor checks the database connections

The number of possible client-server connections depends directly on the number of file descriptors allowed by the server process. Under UNIX, a file descriptor is opened each time that a datafile or a log file is opened. In addition, file descriptors are opened for other reasons unrelated to datafiles and log files.

UNIX By default, 256 is the maximum number of file descriptors for any process. The total number of possible connections to any database at any time is therefore 256 less the number of connections already opened. If the number of database connections reaches 256, and you anticipate an even greater number of connections, you can increase the number of file descriptors. Use the `limit` C-shell command to set the number of file descriptors for the process. Consult the appropriate operating system manual for further information.

Information about datafiles, as described in the following table, follows the connection information on the Overview tab. It can also viewed separately on the Datafiles tab.

Table 7.3 Information about Datafiles

Field	Contents
`ID`	ID of the datafile. Each datafile has a unique ID assigned by the server.
`Path`	Pathname defined for the datafile
`Status`	Status of the datafile—on-line or off-line
`VC`	Whether a Collect Versions operation is currently taking place. During a Collect Versions, an asterisk (*) appears under this field.
`Size`	Size in datapages of the datafile
`Used`	Number of datapages where data is stored in the datafile

Table 7.3 Information about Datafiles

Field	Contents
`Read`	Number of datapages read between two refresh intervals
`Write`	Number of datapages written between two refresh intervals
`Pos`	Position of the last datapage accessed

Database configuration information, as described in the following table, appears at the bottom of the Overview tab.

Table 7.4 Configuration Information

Field	Contents
`Page Size`	Size, in kilobytes, of the database datapages.
`Total Pages`	Total number of datapages in the database.
`Used Pages`	Number of datapages in the datafiles that are currently used to store data
`Single ver`	Number of objects that have only one version
`Multiple ver`	Number of objects that have multiple versions (may be collectible)
`Cur Version`	Current logical time
`Cache Mem`	Size, in kilobytes, of the memory allocated for the data page cache
`ObjTab Mem`	Size, in kilobytes, of the memory allocated for the object table
`High Water`	Greatest position of datapages used at any time prior to the current logical time. (Similar to the debris left behind when a river overflows its banks and called the *high water mark*.)

Information about transactions, as described in the following table, appears on the Transactions tab.

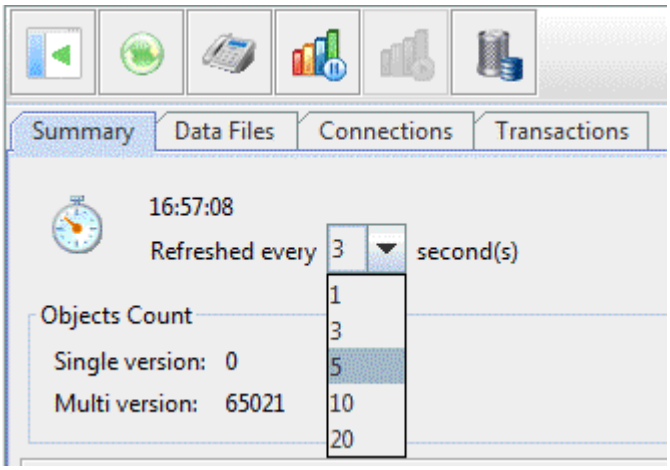
Table 7.5 Information about Transactions

Field	Contents
`TM Disable`	Indicates if the transaction manager is disabled. A “1” means it is disabled.
`Next Time`	Next logical time of the database.
`Commit Tran`	ID of the last transaction that committed
`Active`	Number of transactions that are currently active, that is, open
`Blocked`	Number of transactions that are currently blocked
`Start`	Number of transactions begun since the database was started
`Commit`	Number of transactions committed since the database was started
`Abort`	Number of transactions aborted since the database was started

Changing the Refresh Interval

Click the Refresh Interval pull-down menu to change the number of seconds between updates. The default interval is three seconds.

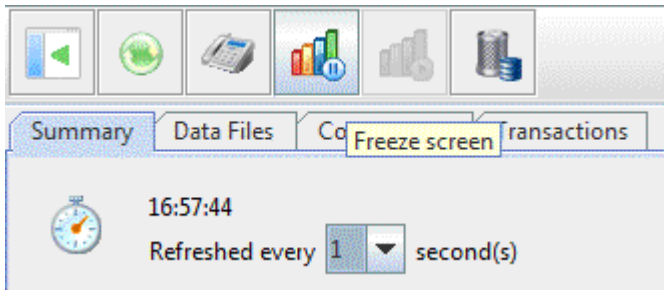
Figure 7.12 Refresh interval choice



Taking an Activity Snapshot

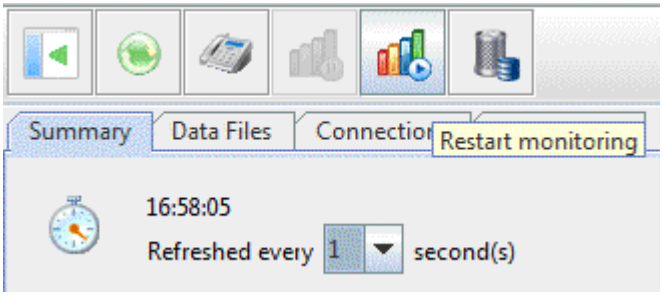
To stop automatic refreshing, click on the Freeze button in the menu-bar.

Figure 7.13 Freeze button from the monitoring menu-bar



You may then resume the automatic refreshing by clicking the Unfreeze button.

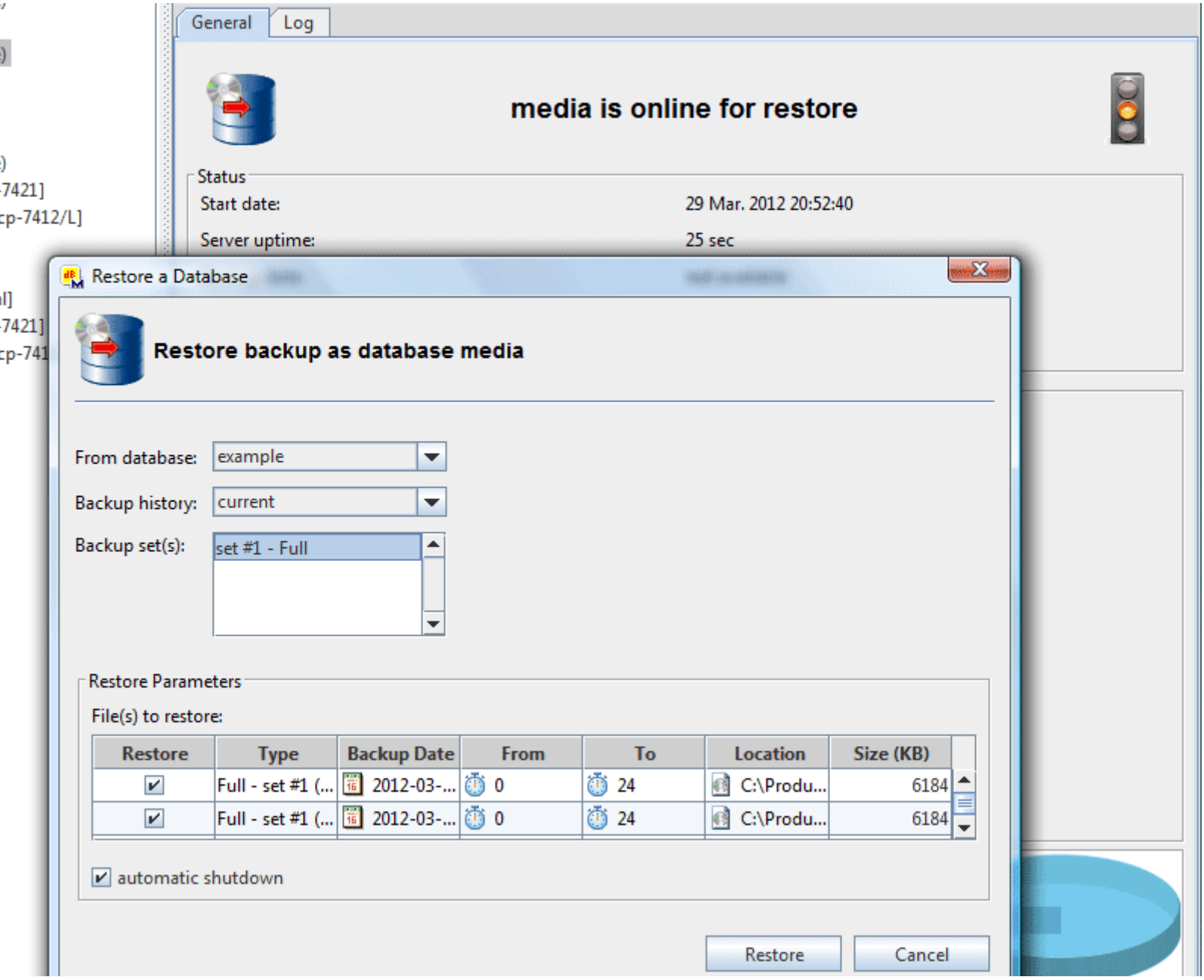
Figure 7.14 Unfreeze button from the monitoring menu-bar



7.13 Restoring a database

Matisse Database Restore tool provides wizards to guide administrators through the restore process. When restoring, you must first preinitialize your database, then you can start the restore process. For restoring from a multi-increment backup, you can restore the full backup files and the incremental backup files in any order, either sequentially or in parallel, and then shutdown and restart the database will complete the process.

Figure 7.15 Database restore wizard

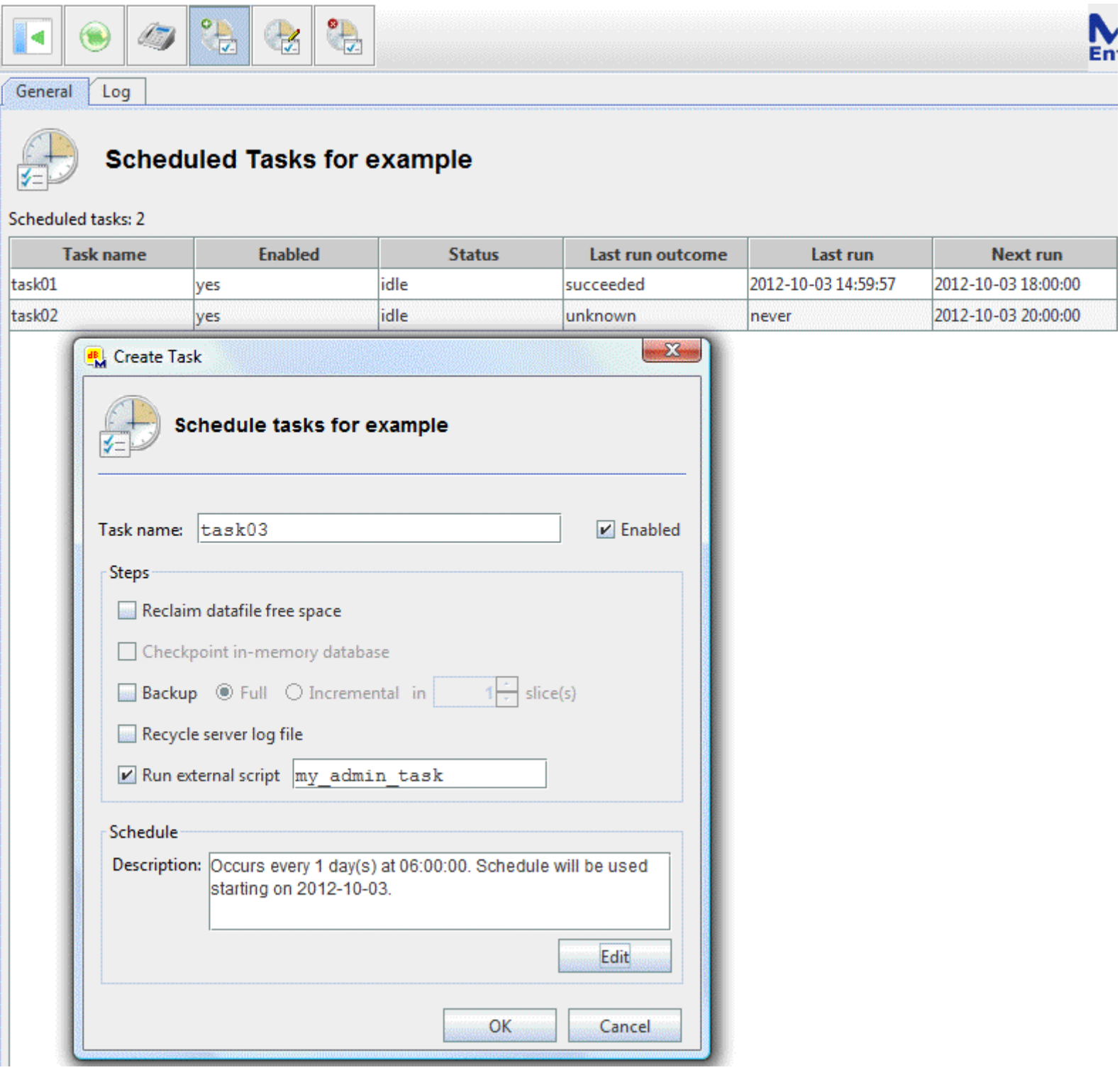


7.14 Scheduling tasks

Matisse Database Task Scheduler provides wizards to guide administrators through the task automation process. The Task Scheduler lets you automate tasks that run on regular or predictable cycles.

By using the task scheduler, you can determine when, and in what order, administrative tasks will occur. You can schedule tasks, such as version collection, backups, log file recycling, user-defined script execution or database checkpoints on in-memory databases. You can also specify the order in which tasks run by creating a multi-steps job. Each task is scheduled to run at desired times and frequency.

Figure 7.16 Database Task Scheduler



Executing a User-defined Script

You can schedule the execution of a user-defined script located in the `scripts/task` directory in `MATISSE_HOME`. The database name is the script only parameter. The script returns 0 to indicate a successful completion while any other number reports an error. The return status as well as the output produced by the execution of the script is logged in the `mtsmlistener.log` file located in `MATISSE_LOG`.

NOTE: The task does not return until the script execution is completed.

The following scripts show how to export the database in an XML file using the `mt_xml` utility.

Unix

```
$ cat mt_runtask.sh
#!/bin/sh
#
# scheduled task script
#
# usage: mt_runtask <database>
#

DBNAME=${1}

mt_xml -d $DBNAME export -f
$MATISSE_HOME/data/XML/$DBNAME_monthly.xml --full
STS=$?

exit $STS
```

Windows

```
$ cat mt_runtask.bat
@echo off
REM
REM scheduled task script
REM
REM usage: mt_runtask <database>
REM

set DBNAME=%1

mt_xml -d %DBNAME% export -f
C:\Products\Matisse\data\XML\%DBNAME%_monthly.xml --full

exit /b %errorlevel%
```

8 Collecting the Versions of a Database

A version collection removes the object versions that are not part of a database version. The disk space used to store these object versions will then be available for other, newer object versions. You can run a version collection on a database only when it is on-line.

How the Collect Versions Mechanism Works

The Matisse versioning architecture uses copy semantics which provide several features such as non-blocking data access to the current version while there are concurrent updating transactions.

The strategy of copy semantics leads to the accumulation of object versions that may no longer be useful. The collect versions mechanism is designed to remove (or “collect”) obsolete object versions.

To prevent the object versions at a given time from being collected, you must declare a version. When you start a collect versions on a database, all the object versions that are not current and are not associated with a version are collected.

Automatic Version Collection

By default the automatic collection is set in your database configuration file. It is triggered upon some database update threshold or when some datafiles become near full. In addition, you may run version collection manually at any time with the `mt_server collect` command, for instance to compact the data before running a backup.

Kinds of Version Collections

As mentioned earlier, the collect versions mechanism removes object versions from the database. Various levels enable you to remove:

- Unreferenced transaction objects (level 2)
- Deleted objects without versions (level 1 and level 2)

Table 8.1 summarizes the principal differences between the three levels of version collections.

Table 8.1 Collect Levels

Level	Purpose
0	Collect obsolete object versions. (Default)
1	Collect obsolete object versions and deleted objects without older versions.
2	Collect obsolete object versions, deleted objects and transaction objects.

Collecting Deleted Objects Without Older Versions

When you delete an object, a deleted object version is written in the database as a marker indicating that the object has been deleted. While no further operation can be performed on this object, previous versions of it may exist in different versions.

A level 1 or higher version collection will collect any deleted object whose previous versions have already been removed.

Note that a level 2 version collection takes longer than a level 0 version collection.

Collecting Unused Transaction Objects

You can use a level 2 version collection to collect unused transaction objects from a database. A transaction object is an object written to the database when a transaction commits.

To determine how frequently you should run a level 2 version collection, you can look at the database log file after running one. Depending on the amount of disk space recovered, it may be best to run a level 2 version collection once a day, once a week, once a month or not at all. It depends on your application.

Data Compaction

After collecting datapages, the collect versions mechanism compacts the valid data from the collected datapages into a minimal number of datapages.

Scheduled Collection on MS Windows

To schedule a collect version on MS Windows, the collect version operation can be scheduled using the `at` command.

For example, to run a collect level 2, each weekday at 11:00 PM, type the following command in a command prompt window:

```
at 23:00 /every:M,T,W,Th,F c:\matisse\mt_server -d
database@host collect -l 2
```

NOTE: Schedule service enables the `at` command. You may check that Schedule service is started using Services tool from the Control Panel.

Version Collection Log File

During a version collection, messages indicating the operations performed by the collect version are written to the log file. To find out what exactly happened during the version collection, you can read the log file. To do this, use the following procedure:

- Start the DBA Tool and then start the version collection.
- Open the Files Menu.
- Select the Log File option. If you start a level 0 version collection on a database that contains two datafiles, for example, the log file of the collect version is likely to contain messages of the following kind:

  ```
  15-MAR 17:48:32 Beginning version collection on file 1
  ```

```
15-MAR 17:48:32 Logical time 3 uncollectible
15-MAR 17:48:33 Completed version collection for file 1
15-MAR 17:48:33 6 pages, 4 versions, 2228 bytes collected
```

The first message indicates that a version collection has begun on file 1. The message, `Logical time 3 uncollectible`, means that all object versions that are current at logical time 3 will be saved from version collection.

9 Administration Commands

Scriptable command-line alternatives and supplements to the DBA Tool are provided by the following commands:

- `mt_backup`
- `mt_connection`
- `mt_file`
- `mt_partition`
- `mt_replicate`
- `mt_server`
- `mt_transaction`
- `mt_user`
- `mt_version`

You may view help for each of these commands by entering it without arguments. For example, entering `mt_file` will display the following:

```
Usage: mt_file -d [user:]dbname[@host[:port]] <command> ...
Possible commands:
  extend            Extend file capacity
  add               Add a new file
  list              List file informations
  remove            Remove a file
```

You may see additional help for the possible commands listed by using the syntax *`command option`* `-h`. For example, the command `mt_file extend -h` will display the following:

```
Usage: mt_file [OPTIONS] extend -f <file> -s <size>[gmk]
[-h]
  -f, --file=...    File location
  -s, --size=...    File Size
  -h, --help      display this help and exit
```

We introduce in this section how to use these commands, except for `mt_backup` and `mt_replicate` which are described in *section 12, Database Backup and Restore* and *section 10, Database Transactional Replication*.

Database Shutdown Restart

Once you have edited a valid configuration file `mydb.cfg` in the `config` directory of your matisse installation, you can initialize your database with the following command:

```
mt_server -d mydb initialize
```

This will initialize and start the database, creating the data files with the path and the size that you have specified in your configuration file.

You can also use the `mt_server` command to stop and restart it:

```
> mt_server -d mydb stop
> mt_server -d mydb start
```

Managing Datafiles

You can use the `mt_file` command to monitor, add, extend, or remove datafiles on an online database. Here are some examples:

```
> mt_file -d mydb@host list
List of files for database mydb at time 156
ID:  FILE:        SIZE K:     USED K:   FREE K:
1:   C:\FILES1:   40960:      5168:     35792:
Total:            40960:      5168:     35792:
```

Add a new datafile:

```
> mt_file -d mydb@host add -f D:\FILES2 -s 40960k
```

Remove a datafile:

```
> mt_file -d mydb@host remove -f C:\FILES1
```

Disk Mirroring

You can also mirror datafiles in order to avoid a single point of failure with the `mt_file add` command. This capability is managed directly within the database server without the need to purchase specialized hardware or volume management software. For instance to add a mirror datafile in the path `D:\FILES2` to an existing database:

```
> mt_file -d mydb@host add -f C:\FILES1 -m D:\FILES2 -s
40960k
```

Once a mirror is established, there is no distinction such as primary datafile and replica datafile within the system. In particular the mirrored datafiles are listed with the same id:

```
> mt_file -d mydb@host list
List of files for database mydb at time 156
ID:  FILE:        SIZE K:     USED K:   FREE K:
1:   C:\FILES1:   40960:      5168:     35792:
1:   D:\FILES2:   40960:      5168:     35792:
```

```
Total:            81920:     10336:     71584:
```

To unmirror a datafile you can execute the `mt_file remove` command, which breaks the mirror and removes the specified datafile:

```
> mt_file -d mydb@host remove -f C:\FILES1
```

Using disk partitions

You may use disk partitions (raw devices) as datafiles and thus skip the file system layer. This brings an additional level of data integrity and performance when deploying databases in a production environment. Before using a partition as a datafile, you need to initialize it with the `mt_partition` command. For example to initialize the drive `G:` for MS Windows:

```
> mt_partition initialize -f \\.\G:
```

You can then use it as a datafile in your database configuration file or with the `mt_file` command:

```
> mt_file -d mydb@host add -f \\.\G: -s 40960k
```

More information is provided in *section 6.3, Using Disk Partitions as Datafiles*.

Managing Users

You can use the `mt_user` command to list, add or remove database users. You must have the administrator privilege to do so. This is part of the database access control mechanism which is fully described in *section 5, Matisse Access Control*.

For instance to add the read-only user `rose` to the database `mydb`:

```
> mt_user -d mydb@host add -u rose -r
> mt_user -d mydb@host list
USER        PRIVILEGE
JOHN        ADM
ROSE        RDONLY
```

Managing Connections

You can view connections, or kill active connections with the `mt_connection` command. For example:

```
> mt_connection -d mydb@host list
1 connections:
ID   NODE    USER  PID  CALLS  TRANID  BLOCKED  FUNCTION
7477 jade    JOHN  308  1                       connection
```

The `kill` option can take any combination of the following parameters:

```
-c, --cid     connection id
-u, --user    user name
-n, --node    client application node (host) name
```

```
-p, --pid    client application process id
-t, --tran   transaction id
```

For instance, to kill the connection(s) from process id 308 on the host `localhost`:

```
> mt_connection -d mydb@host kill -n localhost -p 308
```

You can also get the active connections count. The count includes the connection for the mt_connection command itself, so the minimum value returned by this command is always 1. For example:

```
> mt_connection -d example count
16
```

Managing Transactions

You can view transactions, enable or disable transaction processing, and even abort pending transactions with the `mt_transaction` command. For example:

```
> mt_transaction -d mydb@host list
Current Logical Time: 157
TID     HOST        USER        PID
173     localhost   ROSE        3088
> mt_transaction -d mydb@host abort -t 173
```

Managing Versions

Matisse has the unique ability to declare database versions, or savetimes, for later use, for instance to be able to perform data analysis on a fixed set of data while not hindering concurrent transaction processing. You can view saved versions, declare or undeclare versions with the `mt_version` command. For example:

```
> mt_version -d mydb@host list
Current Logical Time :157
22      FIRST00000015
24      SECOND00000017
> mt_version -d mydb@host undeclare -n first
```

Monitoring a Database

You can monitor the database server activities directly from a command line with the mt_server command with the monitor option. This command provides information about the database state, database objects count, memory usage and disk I/O activities.

```
>mt_server -d mydb monitor
#1
Database State: ONLINE
Object Count:
  Single version: 32456630U
```

```
  Multi version:  65435
Memory Usage:
  Page cache size:   512 MB
  Object table size: 25 MB
Data files: 1
 ID:Location            :   Status:Capacity:    Used:Position: I/ORead:I/OWrite:
  1:C:\WORK\MATISSE\DATA:   online:     640:      11:      11:       0:       0:
```

You can also get additional configuration and status information with the mt_server info command. For example:

```
$ mt_server -d mydb info
Database:
  Name: mydb
  Server version: 8.3.0
  Datafile version: 8.3.0
Status:
  Start date: 05 Jul. 2009 15:33:13
  Server uptime: 52 sec
  Backup date: not available
  Version collect date: not available
Configuration:
   size: 8 Kbytes
   cache size: 32 Mbytes
  Server failover: disabled
  Datafile extension: automatic
  Access control: disabled
  Version collection: automatic
Current State:
  Transaction manager: enabled
  Current logical time: 2
  Highest collected logical time: 1
  Last backup logical time: 0
```

Extending the Page Server Cache

You can extend the size of the server cache on a running database at any time even when clients are connected and transactions are active. For example to extend the server page cache to 1Gbytes:

```
> mt_server -d mydb extendcache -s 1024M
Server cache size extended to 1024M
```

Extending the Object Table Cache

You can extend the maximum size of the in-memory multi-version object table cache on a running database at any time even when clients are connected and transactions are active. For example to extend the object table cache to a maximum of 5Gbytes:

```
> mt_server -d mydb extendcache -o -s 5G
Object table cache size extended to 5G
```

For example to remove the object table cache size constraints:

```
> mt_server -d mydb extendcache -o -s 0
```

Changing the Run Frequency of Operations

You can change on an online database the run frequency of the various automatic operations. For example to change the run frequency of the automatic version collection to 35 seconds:

```
> mt_server -d mydb set runfrequency -a -f 35
```

For example to change the run frequency of the clear object table operation to 15 seconds:

```
> mt_server -d mydb runfrequency -c -f 15
```

Managing License Keys

The mt_server setlicense command allows you to set the customer license key on your server. You need to be logged as root (administrator on Windows) to successfully install the license key. For example:

```
> mt_server setlicense -k B0FF-ED65-11C8-0B2F-A3BD-FE16-E17E-624C
```

The `mt_server checklicense` command allows you to check the customer license key installed on your server.

```
> mt_server checklicense
Your 31 days license expires in 15 days
```

The `--full` option provides a complete description of the installed license.

```
> mt_server checklicense --full
License Description:
Floating License - Developer Edition - x64 - up to 512 concurrent Users - up to 16 logical CPUs
License options: mirroring enabled, raw partition datafiles enabled, multi-datafiles enabled, replication enabled
License expires in 3418 days
```

10 Database Transactional Replication

10.1 Introduction

Feature Overview

The Matisse server provides full distributed synchronous replication between two database servers, one called the 'master' and the other one the 'replica'. The main usage of replication is to provide redundancy across different systems to avoid a single point of failure.

The replication mechanism enforces strong consistency at the transaction level between the databases. A successful commit status is returned to the client application when the client updates have been committed on both the master and the replica.

The updates of both the data instances and the schema are replicated. For instance, the creation of an index on the master will create the same index on the replica.

In case of failure of the master database, the administrator can change the state of the replica database so that it may be used as a standalone database. In case of failure of a replica database, a new replica database can be created and synchronized with the master database.

Replication Benefits

There are several usages for the replica database, that can also be combined together:

- As a *Hot Standby Database*. In case of a master database crash, the applications can be reconnected immediately to the replica database.
- As a *Data Analysis Database*. The replica database is accessible for read only queries thus off loading the master database for this type of processing.
- As a *In-Memory Database*. The replica can be running on main memory storage for fast query access, the data redundancy on the master ensuring recoverability in case of a system crash. The opposite is also possible: the master database may use in-memory datafiles, with the replica enforcing the data recoverability.
- As a *Read-Access Load Distribution System*. The replica database can provide access for read-only queries. Since the master and the replica databases are always in sync, the application can achieve load balancing for data access among the servers, doubling the data serving power of the application.

10.2 Replication Establishing and Disestablishing

Before Establishing Replication

There are two situations to consider:

- If the master database is empty, simply initialize the replica as a new database.
- For a non-empty master database, you need to stop transaction processing on the master, run a full backup of the master, then restore it on the new replica and restart the replica database after restore.

Establishing Replication

The master is notified to start replication to the replica with the `mt_replicate` utility:

```
mt_replicate -d masterdb@host1 establish -r
replicadb@host2
```

This command establishes replication between the two databases and disables transaction processing for all the replica clients other than the master.

Retry or Noretry Mode

The `-m` option can be used to set the `retry` or `noretry` mode. The `noretry` mode indicates that the master will automatically set replication as failed in case the connection is broken with the replica. In this event the master will act as a standalone database.

For instance:

```
mt_replicate -d masterdb@host1 establish -r
replicadb@host2 -m noretry
```

Replication is set by default in retry mode. In this mode the master keeps retrying to connect to the replica until the replica is up and running.

This mode can be switched at any time while replication is active by running the `mt_replicate establish` command.

Disestablishing Replication

Disestablishing is done in a similar way:

```
mt_replicate -d masterdb@host1 disestablish -r
replicadb@host2
```

The replica stays online and becomes a standalone database, and thus it is available for transaction updates. If after disestablishing any update occurs on either side, the replication cannot be re-established besides doing a full restore of the replica from an up-to-date master backup.

You can use the same command without the `-r` option to disestablish replication for a replica database. Then the replica database becomes standalone.

```
mt_replicate -d replicadb@host2 disestablish
```

Swapping Roles Between Master and Replica

To have the master and the replica exchange roles while the system is online, you will first disestablish replication, then re-establish it with the `mt_replicate` command.

To enforce that no update can take place on the databases, you can use the `mt_transaction` command to disable transactions on the initial master until replication has been re-established.

Here is a typical sequence of commands that you may execute:

```
mt_transaction -d masterdb@host1 disable
mt_replicate -d masterdb@host1 disestablish -r
replicadb@host2
mt_replicate -d replicadb@host2 establish   -r
masterdb@host1
```

10.3 Replication Monitoring

The `mt_monitor` tool shows a replica as a standalone database, with a client connection from the master database server. Note that replicas can be accessed by client applications at all times in version mode.

The `mt_replicate info` command provides the following information:

```
mt_replicate -d masterdb@host1 info
  Master database masterdb at time 73600
  REPLICA HOST:  DATABASE:   STATUS:    MODE:  TRAN PENDING
  host2          replicadb   CONNECTED  retry  1


mt_replicate -d replicadb@host2 info
  Replica database replicadb at time 73600
  MASTER HOST:   DATABASE:    STATUS:
  host1          masterdb     CONNECTED
```

Replication status

The replication status indicates the current state of the master-replica synchronization, as detailed on the following table.

Table 10.1 Replication info status

Status	Comments
`CONNECTED`	Normal status.
`CONNECTING`	Master currently connecting to the replica. If the replica is offline or restarting the master keeps retrying and shows the state `CONNECTING`.
`FAILED`	The status `FAILED` appears generally when the replica is disynchronized. For instance if replication is turned off, then new updates have been performed on either side, and then replication is turned on again. In case of `FAILED` status, the administrator must terminate replication explicitly with the mt_replicate command.

10.4 Resynchronization at restart of after replica failure

Shutdown Restart

For shutdown we recommend to first shutdown the master so that it can disconnect from the replica, and then shutdown the replica:

```
mt_server -d masterdb stop
mt_server -d replicadb stop
```

After a graceful shutdown and restart or a crash restart, the master connects automatically to the replica and resynchronizes to the latest committed transaction. You may first restart the replica, so that the master database can see it upon its own restart. It is mandatory to restart in this order if you used the `noretry` option to establish replication:

```
mt_server -d replicadb start
mt_server -d masterdb start
```

Network or Replica Failure

In case of network failure or replica failure, by default the master keeps retrying to connect. The updating transactions are blocked on master until the replica becomes online or the administrator turns off replication. The retry mode can be turned off setting the `noretry` option with the `mt_replicate` command.

Switching to the replica in case of master failure

Upon failure of the master, or by decision of the administrator, the replica can become a standalone database by disestablishing replication on it with the `mt_replicate` command.

```
mt_replicate -d replicadb@host2 disestablish
```

The user application initially connecting to the master must re-establish connections with the replica. This is not currently automated by the Matisse client library.

11 Database Snapshot Replication

11.1 Introduction

Feature Overview

Matisse XML-based Snapshot Replication is a full distributed asynchronous replication. The Publisher-Subscribers model applies to describe how incremental changes are propagated from the Publisher (master database) to Subscribers (replica databases) as they occur. Snapshot replication typically starts with a full data snapshot of the Publisher database. As soon as the initial snapshot is taken, subsequent data changes made at the Publisher are delivered on demand to the Subscribers. All data snapshots (full and increment) are published into XML documents.

The Subscriber initiates the replication by requesting a full data snapshot of the Publisher database. The XML documents produced are equivalent to a full XML export of the database. Subsequent requests from the Subscriber produce data snapshot increment reflecting the net data change since the previous request.

The Subscriber database is synchronized with the Publisher when all the data snapshots are loaded. The data snapshots must be loaded in the order they have been produced.

The production of data snapshots into XML document files give database administrators a great latitude to design the most appropriate replication workflow. The XML format is simple, extensible and universal and XML documents compress very well which is ideal for network transfers. The Enterprise Manager Task Scheduler is well suited to automate the replication workflow.

Production environments that require a minimum downtime can benefit from Snapshot replication to streamline major software and hardware upgrades.

Benefits

There are several usages for the replica database, that can also be combined together:

- As a *Hot Standby Database*. In case of a master database crash, the applications can be reconnected immediately to the replica database.
- As a *Data Analysis Database*. The replica database is accessible for read only queries thus off loading the master database for this type of processing.

- As a *In-Memory Database*. The replica can be running on main memory storage for fast query access, the data redundancy on the master ensuring recoverability in case of a system crash. The opposite is also possible: the master database may use in-memory datafiles, with the replica enforcing the data recoverability.
- As a *Read-Access Load Distribution System*. The replica database can provide access for read-only queries. Since the master and the replica databases are always in sync, the application can achieve load balancing for data access among the servers, doubling the data serving power of the application.

Design Overview

Matisse XML-based Snapshot Replication publishes logical data snapshot of the master database. The objects are identical, but the oid of each object from the master is different in the replica database.

The Publisher database relies on Matisse version tags to keep track of the publishing state. The first data snapshot contains all the data created up to the publishing time. Subsequent publishing produce data snapshot increment reflecting the net data change since the previous one. This data snapshot includes references to deleted, updated and inserted data.

The Subscriber database alters the Publisher database schema to preserve a reference to the OIDs of the Master database as well as a information about the last snapshot increment loaded. When the replication is de-established, the schema changes are removed and the database schema is identical to the original Publisher database schema.

11.2 Replication Establishing

Before Establishing Replication

There are no constraints for the Publisher database before establishing the replication. The Subscriber database must be empty of the Publisher database. In case the snapshot replication is limited to a specific namespace, the Subscriber database can manage application data into different namespaces.

Establishing Replication

The Publisher database must export the database schema and the replication is established with the publishing of a full data snapshot.

```
$ mt_sdl -d master@localhost export --odl -f
masterDbSchema.odl

$ mt_xsr -d master@localhost --verbose=2 publish -f
masterDb_01f.xml -n xsrExample --full
  [INFO] task #1 writing masterDb_01f_xsr_ia001.xml
  [INFO] task #1 writing masterDb_01f_xsr_ir002.xml
  [STAT] Number of top-level objects published: 8
```

```
  [STAT] Number of object insert published: 8
  [STAT] Number of object update published: 0
  [STAT] Number of object delete published: 0
  [OPTN] Number of prefetch objects: 128
  [OPTN] XML data with OID xml attribute: YES
  [OPTN] Media data into external files: NO
  [OPTN] Namespace: xsrExample
  [OPTN] XML data file I/O mode: stream
  [TIME] Start schema info building:       15:48:29.703
Elapsed 00:00:00.000
  [TIME] End schema info building  :       15:48:29.704
Elapsed 00:00:00.000
  [TIME] Start extracting:                15:48:29.686
Elapsed 00:00:00.000
  [TIME] End extracting  :                15:48:29.708
Elapsed 00:00:00.022
```

On the Subscriber side, the first step requires to import the database schema of the publisher database and to establish the replication by loading the full data snapshot.

```
$ mt_sdl -d replica@localhost import --odl -f
masterDbSchema.odl
```

```
$ mt_xsr -d replica@localhost --verbose=2  subscribe -f
masterDb_01f.xml -n xsrExample
  [INFO] task #1 loading masterDb_01f_xsr_ia001.xml
  [INFO] task #1 loading masterDb_01f_xsr_ir002.xml
  [STAT] Number of top-level xml objects read: 8
  [STAT] Number of objects created: 8
  [STAT] Size of oid mapping table: 0.01 MB
  [OPTN] Namespace origin: xsrExample
  [OPTN] Namespace destination: xsrExample
  [OPTN] Number of xml objects parsed at once: 256
  [OPTN] Number of objects per transaction: 20480
  [TIME] Start loading:                   16:06:05.519
Elapsed 00:00:00.000
  [TIME] End loading  :                   16:06:05.526
Elapsed
```

Publishing Changes

The Publisher database publishes on demand the net data change since the previous publication.

```
$ mt_xsr -d master@localhost --verbose=2 publish -f
masterDb_01i1.xml -n xsrExample --increment
  [INFO] task #1 writing masterDb_01i1_xsr_ia001.xml
  [INFO] task #1 writing masterDb_01i1_xsr_ir002.xml
  [INFO] task #1 writing masterDb_01i1_xsr_ua003.xml
```

```
  [INFO] task #1 writing masterDb_01i1_xsr_ur004.xml
  [STAT] Number of top-level objects published: 6
  [STAT] Number of object insert published: 2
  [STAT] Number of object update published: 4
  [STAT] Number of object delete published: 0
  [OPTN] Number of prefetch objects: 128
  [OPTN] XML data with OID xml attribute: YES
  [OPTN] Media data into external files: NO
  [OPTN] Namespace: xsrExample
  [OPTN] XML data file I/O mode: stream
  [TIME] Start schema info building:       15:55:10.271
Elapsed 00:00:00.000
  [TIME] End schema info building  :       15:55:10.280
Elapsed 00:00:00.009
  [TIME] Start extracting:                15:55:10.258
Elapsed 00:00:00.000
  [TIME] End extracting  :                15:55:10.292
Elapsed 00:00:00.033
```

When a new data snapshot increment is available, it can be loaded into the Subscriber database.

```
$ mt_xsr -d replica@localhost --verbose=2  subscribe -f
masterDb_01i1.xml -n xsrExample


  [INFO] task #1 loading masterDb_01i1_xsr_ia001.xml
  [INFO] task #1 loading masterDb_01i1_xsr_ua003.xml
  [INFO] task #1 loading masterDb_01i1_xsr_ir002.xml
  [INFO] task #1 loading masterDb_01i1_xsr_ur004.xml
  [STAT] Number of top-level xml objects read: 8
  [STAT] Number of objects created: 4
  [STAT] Size of oid mapping table: 0.01 MB
  [OPTN] Namespace origin: xsrExample
  [OPTN] Namespace destination: xsrExample
  [OPTN] Number of xml objects parsed at once: 256
  [OPTN] Number of objects per transaction: 20480
  [TIME] Start loading:                   16:06:36.903
Elapsed 00:00:00.000
  [TIME] End loading  :                   16:06:36.915
Elapsed 00:00:00.012
```

Disestablishing Replication

The Publisher can disestablish the replication with the `unpublish` command.

```
$ mt_xsr -d master@localhost unpublish -n xsrExample
```

The Subscriber can disestablish the replication with the `unsubscribe` command.

```
$ mt_xsr -d replica@localhost unsubscribe  -n xsrExample


$ mt_xsr -d replica@localhost describe --subscriber

No XML-based Snapshot Replication subscriber on database
replica at time 14
```

11.3 Replication Monitoring

Publisher Sate

The Publisher database maintains the current state of the replication.

```
$ mt_xsr -d master@localhost describe --publisher
XML-based Snapshot Replication publisher on database master
at time 6
  Publisher #1
    Publisher name:      master@localhost
    Snapshot type:       full (#1)
    Version name:        MTXSR00001086_00000001_00000005
    Version time:        6
    Publisher namespace:  xsrExample
```

The data snapshot files also contains the publishing information.

```
$ mt_xsr -d master@localhost describe -f masterDb_01f.xml
XML-based Snapshot Replication Document:
  Filename:        masterDb_01f.xml
  Publisher:        master@localhost
  Generation date:   2013-05-02 15:48:29
  Snapshot type:     full (#1)
  Version name:      MTXSR00001086_00000001_00000005
  Version time:      6
  Namespace name:    xsrExample
  Insert count:      8
  Update count:      0
  Delete count:      0
```

Subscriber Sate

The Subscriber database maintains the current state of the replication.

```
$ mt_xsr -d replica@localhost describe --subscriber
XML-based Snapshot Replication subscriber on database
replica at time 12
  Subscriber #1
    Publisher name:      master@localhost
    Snapshot type:       increment (#2)
    Version name:        MTXSR00001086_00000002_00000009
```

```
Version time:        10
Publisher namespace:  xsrExample
Subscriber namespace: xsrExample
```

11.4 mt_xsr utility

mt_xsr publish

The `mt_xsr` utility with the `publish` command allows you to publish into XML documents the database incremental changes.

```
$ mt_xsr publish -h
MATISSE XML-based Snapshot Replication Manager x64 Version 9.1.0.0 (64-bit
Edition) - Apr 29 2013.
(c) Copyright 2013 Matisse Software Inc. All rights reserved.

Usage:
  mt_xsr [OPTIONS] publish -f <xmlfile> [-s <size>[M|G]] [-p <n>] [-x <n>] [-n
<nsname>] [-d|-m] -a|-i [-h]
   -f, --file=...    Specify the XML-based Snapshot Replication document file.
                     The XML data is published into a collection of XML
                     segment files named <xmlfile>_xsr_do<docid>.xml,
                    <xmlfile>_xds_ia<docid>.xml, <xmlfile>_xsr_ir<docid>.xml,
                     <xmlfile>_xds_ua<docid>.xml and
                     <xmlfile>_xsr_ur<docid>.xml.
   -s, --size=...     Specify the XML segment file max size.
   -p, --parallel=... Publish data with <n> tasks running in parallel.
   -x, --prefetch=... Specify the number of objects to be prefetched when
                     exporting data. The default value is 128. The values
                     range between 1 and 128.
   -d, --iobuffer     Write XML data to the file in buffered I/O mode.
   -m, --iostream     Write XML data to the file in stream I/O mode.
   -a, --full         Publish the entire database.
   -i, --increment    Publish the database increment since the last
                     publication.
   -n, --ns=...       Specify the namespace from which the objects are
                     exported.
   -h, --help         Display this help and exit.
```

mt_xsr subscribe

The `mt_xsr` utility with the `subscribe` command allows you to establish replication with a master database and to synchronize with the master by loading the database incremental changes from XML documents.

```
$ mt_xsr subscribe -h
MATISSE XML-based Snapshot Replication Manager x64 Version 9.1.0.0 (64-bit
Edition) - Apr 29 2013.
(c) Copyright 2013 Matisse Software Inc. All rights reserved.

Usage:
  mt_xsr [OPTIONS] subscribe -f <xmlfile> [-n <nsname>] [-p <n>] [-x <n>] [-c
<n>] [-h]
   -f, --file=...    Specify the XML-based Snapshot Replication document file
                     to be loaded into the database.
```

```
    -n, --ns=...       Specify the subscriber namespace into which the objects
                       are imported. When the --ns option is ommitted, each
                       object is imported in a namespace matching the schema
                       class namespace.
    -p, --parallel=... Import data with multiple tasks running in parallel. The
                       number of tasks is limited by the number of XML segment
                       files.
    -x, --parse=...    Specify the number of xml objects to be parsed in one
                       sequence. The default value is 256 (1024 in parallel
                       mode). The values range between 1 and 2048.
    -h, --help         Display this help and exit.
```

mt_xsr describe

The `mt_xsr` utility with the `describe` command allows you to view publishers and subscribers settings information.

```
$ mt_xsr describe -h
MATISSE XML-based Snapshot Replication Manager x64 Version 9.1.0.0 (64-bit
Edition) - Apr 29 2013.
(c) Copyright 2013 Matisse Software Inc. All rights reserved.

Usage:
  mt_xsr [OPTIONS] describe [-a|-p|-s] [-f <xml_file>] [-h]
    -a, --all       Provide publishers and subscribers settings information
                    from the database.
    -p, --publisher  Provide publishers settings information from the database.
    -s, --subscriber Provide subscribers settings information from the
database.
    -f, --file=...   Specify the XML-based Snapshot Replication document file
                    to be checked.
    -h, --help      Display this help and exit.
```

mt_xsr unpublish

The `mt_xsr` utility with the `unpublish` command allows you to de-establish the replication of a the master database with a replica database.

```
$ mt_xsr unpublish -h
MATISSE XML-based Snapshot Replication Manager x64 Version 9.1.0.0 (64-bit
Edition) - Apr 29 2013.
(c) Copyright 2013 Matisse Software Inc. All rights reserved.

Usage:
  mt_xsr [OPTIONS] unpublish -a | -n <nsname> [-h]
    -a, --all     Remove all publishers settings from the database.
    -n, --ns=...  Specify the namespace in the database from which the publisher
                  settings are removed.
    -h, --help    Display this help and exit.
```

mt_xsr unsubscribe

The `mt_xsr` utility with the `unsubscribe` command allows you to de-establish the replication of a replica database with a master database.

```
$ mt_xsr unsubscribe -h
MATISSE XML-based Snapshot Replication Manager x64 Version 9.1.0.0 (64-bit
Edition) - Apr 29 2013.
(c) Copyright 2013 Matisse Software Inc. All rights reserved.
```

```
Usage:
  mt_xsr [OPTIONS] unsubscribe -a | -n <nsname> [-h]
    -a, --all     Remove all subscribers settings from the database.
    -n, --ns=...  Specify the subscriber namespace in the database from which
```

12 Database Backup and Restore

12.1 Introduction

You can perform full and incremental parallel backups of databases while the system in online with the `mt_backup` utility. There is no need to block updates during a backup, as the Matisse server keeps a snapshot of the database at the time of the beginning of the backup operation.

The `mt_backup` utility performs a binary backup of the data pages that contain valid data at the time of backup. It cannot be used to upgrade a database to a major revision of Matisse, or to migrate a database from different system architectures, as for instance between SPARC and Intel platforms, for these purposes you may use the `mt_xml` utility instead.

Full and Incremental Backup

A full backup copies all the database content to the backup file or tape. An incremental backup copies only the updates that have occurred since the last full or incremental backup.

Parallel Backup

You can backup a database to several files or tapes in parallel, and restore the generated files in parallel.

12.2 Running a Full or Incremental Backup

Running a Full Backup

You must first ensure that the database is online, and that there are no ongoing administrative operations like adding, extending or removing a data file for the database. We provide here a simple example, running a full backup for the database `mydb` on one backup file:

```
% mt_backup -d mydb start -f
164 Kbytes to Backup
% mt_backup -d mydb write -f C:\mydb.bkp -s 164k
done
% mt_backup -d mydb end
```

Note that for backup `start`, we specify the `-f` option for full backup.

Running an Incremental Backup

You can then run incremental backups for the same database, by using the `-i` option:

```
% mt_backup -d mydb start -i
```

```
64 Kbytes to Backup
% mt_backup -d mydb write -f C:\mydb1.bkp -s 64k
done
% mt_backup -d mydb end
```

Automated Backups

The backup commands can be easily integrated into scripts, for instance a full backup can be automated with the following `.bat` file on MS Windows:

```
rem @echo off
set DB=%2@%1
set BKP=%3
if exist %BKP% del %BKP%
for /f %%f in ('call mt_backup -d %DB% start -f') do set
bkpsz=%%f
echo backup size:   %bkpsz%
for /f %%f in ('call mt_backup -s -d %DB% write -f %BKP% -s
%bkpsz%k') do set bkpres=%%f
echo backup result: %bkpres%
call mt_backup -d %DB% end
```

You may then execute it as follows, assuming you call it `backup.bat`:

```
C:\> backup host mydb C:\mydb.bkp
```

The same set of commands written in a Unix style `.sh` script:

```
DB=${2}@${1}
BKP=${3}
/bin/rm -f ${BKP}
res=`mt_backup -d ${DB} start -f`
echo $res
for bkpsz in $res;  do
 mt_backup -d $DB write -f ${BKP} -s ${bkpsz}k
 break;
done
mt_backup -d ${DB} end
```

Backup Journal Files

For each database that is backed up, a backup journal file is created in the `MATISSE_LOG` directory. It contains information on the backup files. For instance the journal file for `mydb` will be named `mydb.bjl` and may contain the following:

```
2002-12-27:19:13:00.00000000000 0000000004 C:\mydb1.bkp
```

```
2002-12-27:19:15:00.00000000005 0000000007 C:\mydb2.bkp
```

The first field is the date when the backup files where produced, the next two fields are the logical times (or versions) that are saved in each backup file. In this example the file `mydb1.bkp` contains the versions of the data objects from 0 to 4, the file `mydb2.bkp` contains the versions of the data objects from 5 to 7.

Thus at the time of backup, the current version was 7. It may be checked when the database is online with the `mt_version` command:

```
% mt_version -d mydb list
Current Logical Time :7
```

12.3 Restore

When restoring, you must first preinitialize your database, then you can restore your data with the `restore` command as on the following example:

```
% mt_server -d mydb preinitialize
% mt_backup -d mydb restore -f C:\mydb.bkp -s
Restore completed from device 'C:\mydb.bkp'
```

The option `-s` can be used when restoring the last backup file that has been generated by backup. It indicates to shutdown the server upon completion. The server must then be restarted to complete the operation:

```
% mt_server -d mydb start
```

For restoring from an incremental backup, you can restore the full backup files and the incremental backup files in any order, either sequentially or in parallel, and then shutdown the database.

For instance if you generated the full backup file `mydbf.bkp` and then the incremental backup file `mydbi.bkp`, you can restore your database in the following way:

```
% mt_server -d mydb preinitialize
% mt_backup -d mydb restore -f C:\mydbf.bkp
% mt_backup -d mydb restore -f C:\mydbi.bkp -s
% mt_server -d mydb start
```

12.4 Running a Parallel Backup

For a parallel backup you will run the backup `write` command in asynchronous mode once for each backup file, and then wait for completion of all commands with the backup `end` command, as shown on this example:

```
% mt_backup -d mydb start -f
2000 Kbytes to Backup
% mt_backup -a -d mydb write -f C:\mydb1.bkp -s 1000k
% mt_backup -a -d mydb write -f C:\mydb2.bkp -s 1000k
% mt_backup -d mydb end
```

The option `-a` for the backup `write` commands specifies the asynchronous mode. As for most database administration commands, the backup `end` command is by default synchronous and waits for all the backup `write` commands to complete.

If you want to cancel an ongoing backup, you may use the backup `end` command with the force option `-f`:

```
% mt_backup -d mydb end -f
```

12.5 Parallel Restore

Once you have generated several backup files with either a sequential or a parallel backup, you can restore them in parallel by running the `restore` command asynchronously, once for each backup file. For example to restore from the files `mydb1.bkp` and `mydb2.bkp`:

```
% mt_server -d mydb preinitialize
% mt_backup -a -d mydb restore -f C:\mydb1.bkp
% mt_backup -a -d mydb restore -f C:\mydb2.bkp
% mt_server -d mydb stop
% mt_server -d mydb start
```

Here the backup files are restored in asynchronous mode, while the server `stop` command is synchronous by default and waits until restore and shutdown completion.

Appendix A Starting Matisse Server as a Windows Service

A.1 Introduction

This appendix describes how to start a Matisse server as a Windows service, allowing the Matisse Server to come up and service requests even when no user is logged on. It requires the Windows Resource kit `srvany.exe` and `instsrv.exe` utilities.

A.2 Installation

Install `srvany.exe` utility as a Windows service:

```
instsrv matisse_<dbname> <path>\srvany.exe
```

Configure via the Services applet (Startup dialog) of the Control Panel as automatic or manual, as appropriate. Then, if needed, change the Account Name and Password that this newly installed service will use for its Security Context (do not choose LocalSystem account, since it does not have network access).

A.3 Specifying Matisse Server and Its Parameters

Run the Registry Editor (`regedit.exe`).

Create a `Parameters` key under:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\matisse_<dbname>
```

Under this key, create the `Parameters` key by selecting “Add key” from the Edit menu.

Select the `Parameters` key created in the previous step. Under the `Parameters`, create an `Application` value of type `REG_SZ` and specify the full path of Matisse server executable `mts.exe`.

For example:

```
Application REG_SZ C:\matisse\bin\mts.exe
```

Under the above key, create an `AppParameters` value of type `REG_SZ` and specify the Matisse database to start.

For example:

```
AppParameters REG_SZ dbname
```

A.4 Starting and Stopping the Matisse Server Service

Start: If the service is configured as Automatic, the user does not need to start it explicitly: it is started automatically every time when the system is rebooted.

For Manual services, the user may start the service via the Services applet of the Control Panel or via the `net start matisse_<dbname>` command.

Stop: When you stop the service, it will shutdown the Matisse server service. The way to stop the service is to use the Services applet of the Control Panel or the `net stop matisse_<dbexample>` command.

A.5 Uninstall

If you want to prevent a Matisse server service from running until further notice, you should configure it via the Services applet (Startup dialog) of the Control Panel as Disabled.

If you want to remove permanently a Matisse server service: If the service is running, stop it and run:

```
instsrv matisse_<dbname> remove
```

Index

A

abort a transaction 66, 67
aborting a transaction 83
adding a new datafile 62, 81
adding a new user 61, 82
administrator 40, 41
asynchronous backup 103
`at` command 78
`AUTOCOLLECT` configuration parameter 47
`AUTOCOLLECTFREQ` configuration parameter 48
`AUTOEXTEND` configuration parameter 47
automated backup 101
automatic version collection 77
`AUTORESTART` configuration parameter 49, 50

B

backup journal files 101

C

`CACHESIZ` configuration parameter 46
checking a license key 85
concurrency control 14
configuration file 14, 17, 43, 58
counting active connections 83
creating a mirror datafile 62

D

database log file 78
database preinitialize 102
datafiles 15, 43, 81
`DATEXTENDSIZ` configuration parameter 47
`DATFULLINIT` configuration parameter 49
`DATINITSZ` configuration parameter 49
`DATINMEMORY` configuration parameter 50
DBA Tool 18, 19, 57
declaring a version 83
deleting a datafile 62
disabling transaction processing 83, 87
disestablishing replication 87, 95
disk mirroring 81
disk partitions 47, 54, 55, 82
`dkinfo` 54
dropping a user 61

E

enabling transaction processing 83
environment variables 17
establishing replication 87, 93
extendcache 84, 85

F

fault tolerance 14
`format` 54
full backup 100

I

incremental backup 100
info on a database server 84
intrinsic versioning 13

K

kill a connection 66, 67

L

`LD_LIBRARY_PATH` environment variable 38, 57
License Keys 85
load balancing 13
locking granularity 14
log file 15, 18
logical time 70, 79

M

managing Users 61

master database 87
master-replica synchronization 89
MATISSE_CFG environment variable 17, 18
MATISSE_HOME environment variable 18, 19
MATISSE_LOG environment variable 18, 25, 37
MATISSE_NET_PATH environment variable 19, 25
MATISSE_PORTMON_ADDR environment variable 20, 23
MATISSE_PORTMON_NAME environment variable 21
MATISSE_SMLISTENER_ADDR environment variable 37
MAXBKPLOGFILES configuration parameter 51
MAXSQLDOP configuration parameter 50
MAXSQLTHRDPOOL configuration parameter 51
MAXSRVLOGFILES configuration parameter 51
MEMORYTRANS configuration parameter 50
mirroring datafiles 14, 43, 81
monitoring a database 83
mount 55
mt_backup 100
mt_connection 82
MT_DATA_ACCESS_MODE 42
MT_DATA_DEFINITION 42
MT_DATA_MODIFICATION 42
MT_DATA_READONLY 42
mt_emgr 57
mt_file 55, 81
mt_monitor 88
mt_partition 54, 56, 82
mt_pmadm 25, 26
mt_portmon 22
mt_replicate 87
mt_server 81
mt_smlistener 36
mt_smlistener 36
mt_transaction 83, 88
mt_user 82
mt_version 83
MtAllocateConnection 42
MtConnectDatabase 42
MtConnection 42
MtFreeConnection 42

N

NAME configuration parameter 45
non-blocking data access 77

O

Object Table Cache 85
OBJTABCLRFREQ configuration parameter 49
OBJTABLESIZ configuration parameter 48
operating system access control 41

P

Page Server Cache 84
PAGESIZ configuration parameter 45
parallel backup 100, 102
parallel restore 103
PATH configuration parameter 53
port monitor 18, 19, 20, 21, 22, 25
PORTS configuration parameter 52

R

raw devices 47, 54, 56
read-only user 40
refresh interval 71
removing a datafile 81
removing a user 82
replica database 87
replication noretry mode 87, 89
replication resynchronization 89
replication retry mode 87
replication status 89
restore 102
runfrequency 85

S

schedule service 78
SECURITY configuration parameter 40, 46
setting a license key 85
SMlistener 49
standalone database 90
stopping a database 59

swap partitions 55
swapping roles between master and replica 88
system user 40, 41

T

tcp transport 22
`TCPKEEPALIVE` configuration parameter 52
transport 19, 20, 22
two phase locking 14

U

undeclaring a version 83
Unix file descriptors 69
unmirroring datafiles 82

V

version collection 77
versioning architecture 77