Deltek

# Deltek Maconomy®

## Logging Options for Maconomy

**November 25, 2022**

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.



This edition published November 2022.

# Contents

# Overview

This document describes the logging features of Maconomy: What is being logged by default, which log files exist and what additional logging can be set up, and how to interpret log output.

Logging on the application server and on the web server will be discussed as well as the logging features of the Maconomy Coupling Service.

The target group of the document is primarily Customer Care and others who deal with Maconomy support. Technical consultants may find it useful as well.

# Application Server

## General Error Log

The log file is named *MaconomyServer.log* on all platforms (note, however, that for version 2.3 LA, the file name is *Maconomy.log*). The location is *<main Maconomy folder\log* on Windows and */usr/maconomy/log* on Unix/Linux.

## Maconomy Server Logging

The standard logging function for the Maconomy server can be enabled in two ways[1]:

- In the file *`CouplingService/configuration/settings/logging.ini`* enable logging by setting the parameter:

  `server.debug.enabled = true`

  This setting is dynamic, and no CouplingServing restart is required. Remember to disable logging again as soon as you are done, by setting the value of the parameter to `false`.

  When logging is enabled using this setting, the log files are written to the folder *`CouplingService/log`*

- Add the parameter "-d" (without quotes) to a command-line execution of the Maconomy server. This will enable logging for the process being started and no others.

  In this case the logging info is written to a file located in *<main Maconomy folder>* *Tmp* on Windows and in */tmp* on Unix/Linux. The log file is named:

  - Odbug<…> for Oracle based applications
  - Mdbug<…> for SQL Server based applications
  - Xdbug<…> for standalone program execution (no matter the database platform)
  - Idbug<…> for Maconomy import/export (no matter the database platform)

The <…> part of the file name includes the process ID (PID) and some random characters.

## Test Masks

***Note that this section applies from Maconomy 2.2.4.***

Enabling logging as above causes some basic info to be logged including database timing. The logged info can be specified further using test masks.

Test masks are specified in a file named *TestMasks* which is to be located in the *MaconomyDir* folder of the targeted application. Please note:

- The file does not exist by default – you must create it yourself. To disable the file later, remove or rename it.

[1] Another way to enable logging is to add a line with the content "-d" (without quotes) to the file *MaconomyServer.<application>.I*. The file is located in *<main Maconomy folder>**IniFiles* on Windows and in */usr/maconomy* on Unix/Linux. However, this is no longer recommended. A restart is required for this setting to take effect, and the log files will not be principal-specific.

- The file name must be *TestMasks*, not *TestMasks.txt*. If you create the file using Notepad, be sure that you end up with a file with the right name, since Notepad may add ".txt" to the name when you save the file.

The TestMasks file consists of one or more lines of the form

```
<log type no.>  <mask no. 1>  <mask no. 2> ….
```

The <*log type no.*> specifies a logging category. The most commonly used categories are 9 for database/SQL logging and 6 for MSL logging (core Maconomy business logic).

The test masks have worked the way they do throughout many Maconomy versions. However, with version 2.2.4 and 2.3 a number of changes are introduced.

There are no changes when you enable logging and don't use test masks, the usual standard log output is written as before, except that time stamps are now added to the log lines. However, if a *TestMasks* file is found, the standard output is no longer written, only the output defined by the test masks will appear in the log file. To add the standard output to the log file when using test masks, add a *TestMasks* line "1 1".

Among the most common used test masks, possibly together with "1 1", are:

- 9 2 3 : Database communication, including detailed cursor definitions, values of bind variables and field values for loaded and saved records.
- 9 4 5 7 8 : Database communication, including e.g. values of bind variables. This produces less output than "9 2 3" and will usually be sufficient
- 6 1 2 : Basic MSL tracing (procedure calls)
- 6 1 2 4 : Advanced MSL tracing on single instructions.

So you can combine multiple masks for a logging category in one line in the *TestMasks* file – e.g. a line "6 1 2" enables masks 1 and 2 for category 6, while masks for different categories must appear in different line, e.g. one line with "6 1 2" and one with "9 2 3".

Many other test masks exist. Please see appendix B.

# Analyzing Odbug Files for Investigating Performance Issues

The timing measures in the Odbug/… files can reveal performance problems related to database operations by showing the performance of particular SQL statements.

A tool for this analysis is available from http://10.4.9.21/cgi-bin/scanOdbug.py, where you can upload an Odbug/… files and see the analysis output.

Alternatively, you can run the tool manually. The tool is a Python 2 script called OdbugFileScanner.py, and you can run like this

```
OdbugFileScanner.py        Odbug6648_a8517e80  >   Odbug6648.txt
```

In this example we analyze the file Odbug6648_a8517e80 and write the result to Odbug6648.txt. In the output the SQL statements found are listed, sorted by time consumption, so that the most time consuming statement appear first. The start of the output may look like this:

```
Ticks/Sec = 1000

SELECT JOBENTRY.* FROM JOBENTRY   WHERE INVOICINGJOBNUMBER= :1 AND (CLOSED=0 OR
BEINGREOPENED=1) AND APPROVEDFORINVOICING=0 AND ENTRYNUMBER<> :2 AND
JOBINVOICESELECTIONLINEINSTA= :3
                          #         Total           Max
Declare                   6             0             0
Open                    136             0             0
FetchNext               136        189879          4814
Insert                    0             0             0
Update                    0             0             0
Delete                    0             0             0
Close                   136           235
Total                             190114
```

This means that 6 times an SQL cursor has been declared for the statement "SELECT JOBENTRY.* FROM JOBENTRY …", 136 times a cursor has been opened for the statement, and 136 times a record has been loaded *(FetchNext)*, and 136 times the cursor has been closed. The FetchNext operations have in total taken 189879 *ticks. Ticks* is a hardware related time measuring unit which may be different for different machines, but the script output tells us what is means here – it says Ticks/Sec = 1000, which means that 189879 ticks is 189.879 seconds, i.e. a little more than 3 minutes. The line "Ticks/Sec = 1000" is actually a line occurring in the original Odbug… file, written by the Maconomy server. Usually Ticks/Sec is 1000 on Windows and 100 on Unix/Linux but always check the value in the Odbug … output.

Interpreting the output above we note that 190 seconds used for only 136 fetches which is to be considered bad performance (a rule of thumb is that if figures in the *Total* column are "much higher" than the corresponding figures in the # column, it's a sign of bad performance). Here is it likely that indexes are not applied in a useful way, so the indexes on the JOBENTRY should be considered, and possibly an extra index might be added.

Another possible interpretation: 189879 ticks for 136 fetches means that average is 1396 ticks, the *Max* value 4814 ticks is clearly higher but not extremely higher. If the Max value is extremely higher than the average, so that most of the time is used by a few operations, it may be an indication that database locks are causing the performance problems.

Debugging Analyzer Problems Using Test MasksTest masks can be used for debugging related to the Maconomy analyzer reports. Here they work in a way that is a little non-standard since they not only produce the usual Odbug…/Mdbug… files but also some files ROE… and XMQL…, located within the same folder – and there may be several of these files for each session – and the info generated through the test masks are in those files. To enable this kind of logging, use a setup specifying test masks "21 1", "22 1" and "24 1" (written in three separate lines).

# Debugging Analyzer Problems Using Test Masks

Test masks can be used for debugging related to the Maconomy analyzer reports. Here they work in a way that is a little non-standard since they not only produce the usual Odbug…/Mdbug… files, but also some files ROE… and XMQL…, located within the same folder There may be several of these files for each session and the info generated through the test masks are in those files. To enable this kind of logging, use a setup specifying test masks "21 1", "22 1" and "24 1" (written in three separate lines).

# MaconomyServer.<…> --users

With Maconomy version 2.3 the monitoring of user sessions, processes etc. changes substantially. The related info is no longer stored in shared memory tables on the Maconomy server machine, but in the database. The purpose of this is to support the *Scalable Server* concept where multiple Maconomy server machines can connect to the same Maconomy database. Furthermore high-level locks are no longer used by Maconomy.

The tools also change: *mmem* no longer exists – instead a new Maconomy Server command option is used for displaying info on users and sessions. The command requires specification of both an application and a shortname. e.g.:

```
MaconomyServer.w_19_0.prod  -Smacoprod  --users
```

The output may look like this:

```
Nov 11 2016 14:40:18
Username        Client   Created    Accessed   Session ID
-------------------------------------------------------------------------------
Jack Jones      RPC User 14:32:27   14:32:38   42a789016f664fddb2a228663d8e4b6a
Jill Jackson    RPC User 14:23:29   14:23:30   85327889a73e4241becf3daa5a7face2
-------------------------------------------------------------------------------
```

Note that RPC User is the client type for the Workspace Client. You do not see the user process related to a Maconomy Server PID, but that is not possible anyway when using Coupling Service and Workspace Client.

There are some additional options for the *--users* parameter modify the output, e.g. *--users:ext* for adding dates and info on timeout status. You may also with *--users:time=relative* get the time displayed relative to "now". See the *"MaconomyServer -h"* help text for all options.

# ShowBlockingSessions.sql

The *ShowBlockingSessions.sql* scripts show the current database lock status. They require sysdba rights, and they show the status for the entire database instance (which may include multiple Maconomy shortnames and non-Maconomy data). There are different scripts for Oracle and SQLServer, for which the output looks quite different but the info is similar.

There are two main cases for using the script.

1. If users are massively being blocked (i.e. their sessions are "hanging"), it is likely to be because of database locks. Running *ShowBlockingSessions.sql* can show which processes are holding and locks, and terminating these processes is often the way to solve the problems.

2. Some performance problems are caused by e.g. batch jobs taking a lot of database locks, thereby blocking each other or blocking human users' activities. One way to explore such cases is to set up *ShowBlockingSessions.sql* to run regularly, usually once every minute. Analyzing the full log output may help with solving the problems.

Some notes on database locks: Database locks are not generally "evil"--on the contrary, they are vital for running database systems by preventing data corruption (e.g. if multiple users update the same records at the same time). Normally, a database lock should only be held for a short time, so if you are dealing with a situation where you suspect database locks to cause problems, only consider those locks that prevail in the *ShowBlockingSessions.sql* output for at least a minute (e.g. by occurring in two consecutive outputs when *ShowBlockingSessions.sql* is run once per minute).

Until now *ShowBlockingSessions.sql* has often been combined with *mmem* in order to determine which Maconomy users are related to the blocking or blocked sessions. However we are about to release new and improved versions of the *ShowBlockingSessions.sql* scripts, which can retrieve info on Maconomy users from the database and include it in the output.

Here is an example of *ShowBlockingSessions.sql* output (a number of columns are omitted below for clarity):

```
LOCKER_PID      WAITER_PID     SQL_TEXT_WAITER
---- ---------- ------------ --------------- ----------------------------------------------------
10636:7052      7312:10368     TX: SELECT SYSTEMNUMBER.* , ROWID FROM SYSTEMNUMBER   WHERE NUMB
11424:11800     7188:8064      TX: SELECT JOBBALANCE.* , ROWID FROM JOBBALANCE   WHERE JOBNUMBE
12348:10180     3784:4884      TX: SELECT FINANCEPERIOD.* , ROWID FROM FINANCEPERIOD  WHERE AC
6052:10768      10188:6968     TX: SELECT JOBHEADER.* , ROWID FROM JOBHEADER   WHERE JOBNUMBER=
6052:10768      6688:776       TX: SELECT JOBINVOICEDISTRIBUTION.* , ROWID FROM JOBINVOICEDISTR
6052:10768      10636:7052     TX: SELECT FINANCEPERIOD.* , ROWID FROM FINANCEPERIOD  WHERE AC
6052:10768      11424:11800    TX: SELECT FINANCEPERIOD.* , ROWID FROM FINANCEPERIOD  WHERE AC
6052:10768      12348:10180    TX: SELECT INTERNALJOBINFORMATION.* , ROWID FROM INTERNALJOBINFO
6052:10768      7428:13288     TX: SELECT JOBHEADER.* , ROWID FROM JOBHEADER   WHERE JOBNUMBER=
6052:10768      11524:8040     TX: SELECT JOBHEADER.* , ROWID FROM JOBHEADER   WHERE JOBNUMBER=
6052:10768      1448:9332      TX: SELECT FINANCEPERIOD.* , ROWID FROM FINANCEPERIOD  WHERE AC
```

This example is from Windows where PIDs are shown with a related thread number as in 6052:10768. Above process 6052:10768 is a LOCKER_PID blocking several other processes shown under WAITER_PID, e.g. 10636:7052 and 11424:11800 which in turn block other processes. You can also see a part of the SQL executed by the waiting (i.e. blocked) processes.

# Portal and Other MScript-based Interfaces

## Logging Setup

The MScript based interfaces/clients include the Portal, the Maconomy Web Services (Maconomy WS), the standalone MScript interface and the MScript side of Maconomy Touch.

The web server log files for the MScript interfaces e.g. for the Portal, reside in the *<web server root>/cgi-bin/Maconomy* folder – the same folder as the web server executable files. Logging is enabled using the parameter file, e.g. MaconomyPortal.<…>.l or MaconomyWS.<…>.l also in the *cgi-bin\Maconomy* folder.

By default, only error messages are written to the log file. To add information about communication, processed data, etc., logging can be enabled this way:

In the file parameter file, e.g. MaconomyPortal.<…>.l, locate the line

```
Log = …
```

By default, the parameter is set to *none,* i.e. *"Log = none",* this means that only error messages are written to the log file. To add information about communication, processed data etc. change the parameter value to switch on different kind of detailed logging, e.g.

```
Log = login
```

for logging login processes or

```
Log = scripts
```

For logging script execution load and time information,.

The logging switches may be combined, e.g.:

```
Log = login;scripts
```

The full set of logging switches are listed in Appendix A.

When replicating a specific problem that is being investigated a useful setup is

```
Log = all
```

which combines most of the available log switches - this produces a lot of log output and should therefore only be used when you are able to replicate a problem within a limited time frame. In cases where logging needs to be switched on for longer time frames, specific switches should be used.

When investigating a performance related issue which also can be replicated within a limited time frame this setup can also be useful:

```
Log = performance;tab
```

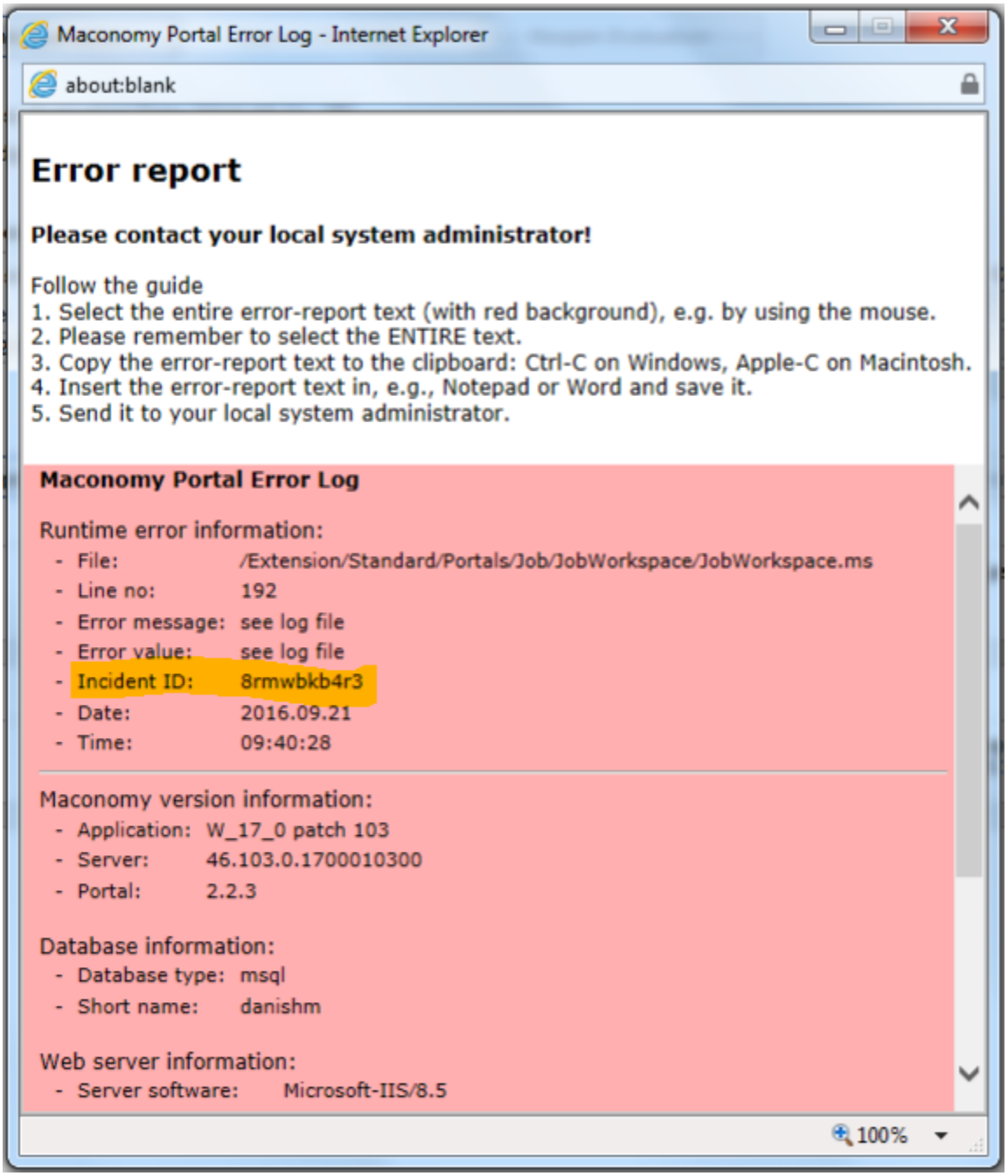
which also produces a lot of output – the *tab* changes the log format to be TAB separated which is convenient for automatic processing of the log.

NB! The "Log = …" line is present in the parameter file by default – always modify that line, do **not** add a new "Log = …" line.

Enabling/disabling logging for Portal etc. does *not* require restart of e.g. Web Daemons.

## Incident IDs

If an error occurs in the Portal, a "pink screen" window like this is often shown:




Note the Incident ID displayed, here 8rmwbkb4r3. The value itself has no meaning, but it points to the output that has been logged. So if you open the Portal log file and search for the displayed incident ID, you can find the place where more detailed info on the incident is logged.

# Using a Parallel Log File and URL

Switching on portal logging will apply not only to a planned replication session but also to other user sessions. On a production system, this may produce a log file where it is hard to filter out the relevant log info.

To avoid this problem, you can set up a parallel portal login:

1. In the cgi-bin/Maconomy folder on the web server, create copies of the Portal executable and the corresponding parameter (.I) file, and add a significant string, e.g. ".debug" to the copy file names. Example: if the Portal executable is named *MaconomyPortal.macoprod.en_US.exe* and the corresponding parameter file is named *MaconomyPortal.macoprod.en_US.I*, name the copy files *MaconomyPortal.macoprod.en_US.debug.exe* and *MaconomyPortal.macoprod.en_US.debug.I*
2. Open the copy parameter file (in the example *MaconomyPortal.macoprod.en_US.debug.I)* and change the *Log* parameter as desired, typically to "Log = all", leave other parameters unchanged and close the file
3. When replicating the problem scenario, log on the portal with a modified version of the URL. E.g. if the URL normally used is http://maconomy.company.com/cgi-bin/Maconomy/MaconomyPortal.macoprod.en_US_MCS.exe/ instead use http://maconomy.company.com/cgi-bin/Maconomy/MaconomyPortal.macoprod.en_US_MCS.debug.exe/

The logging for the session will then be stored in a separate log file – in the example it will be MaconomyPortal.macoprod.en_US.debug.log.

You may combine this setup with setting up a parallel Web Daemon – this is explained in the Web Daemon section of this document. In that case you must modify the DaemonPort setting in the *MaconomyPortal.macoprod.en_US.debug.I* file to match the CGI port of the parallel Web Daemon.

# Web Daemon

You can enable logging for a Web Daemon e.g. to look for errors (e.g. connection problems) or to explore problems with the Web Daemon's server pool being exhausted.

To enable logging for a particular Web Daemon, open the corresponding *WebDaemon.<…>.I* file which located in <*main Maconomy folder*>*IniFiles* (Windows) or */usr/maconomy* (Unix), and insert a line

```
LogFileName = <log file by full path>
```

and restart the Web Daemon.

Besides from error messages, the important lines in the log are those like:

```
2016-10-20(18:13:13.861) Server status:  -bbb/ -.... ..... .
```

Each character in the "-bbb…" part shows the status of the servers in the pool. The important status characters are '.' for *not running*, '-' for *ready to serve* and 'b' for *busy*. With default minimum and maximum values for the server pool, 2 and 16, the status string after starting the Web Daemon and before any user actions will look like

```
2016-10-20(18:11:43.482) Server status:  --... ..... ..... .
```

showing 2 servers ready (corresponding to the minimum) and 14 (= 16 - 2) not yet started.

The critical situation is if all, or almost all, servers frequently are busy during longer intervals, i.e. if the status string looks like

```
2016-10-20(18:17:25.915) Server status:  bbbbb b
```

it is a sign of the server pool running full, and increasing the maximum server pool size, i.e. the *MaxServerProcs* parameter in *WebDaemon.<…>.I,* may solve the problems.

## Setting up a Parallel Web Daemon

Setting up a parallel Web Daemon for problem investigation, including scenario replication, is often useful in order to separate out the log data for a replication session and to avoid using resources for logging other processes. Also, you will not have to restart Web Daemons used by other users.

It is useful to have two Maconomy server processes running for the replication – start the processes like this, each in its own Command Line window:

```
MaconomyServer.w_17_0.tst -Smacoprod -f -d --port 4666 > C:\maconomy\Tmp\Odbug1.txt
2>&1
```

```
MaconomyServer.w_17_0.tst -Smacoprod -f -d --port 4667 > C:\maconomy\Tmp\Odbug2.txt
2>&1
```

Substitute the application name and shortname, and use port numbers that are not currently in use on the machine. The two port numbers must be consecutive, like 4666 and 4667 in this example.

Then, start a WebDaemon process in a third Command Line window like this;

*On Windows:*

```
C:\maconomy\tpu.NTx86.17_0.p103.dir\bin\WebDaemon.exe -debug --
ServerPort=4666 --MultipleServerPorts=1 --CGIPort=4421 --CGICallbackPort=4521 --
MinServerProcs=2 --MaxServerProcs=2 -i WebDaemon.macoprod.4103
```

*On Unix/Linux:*

```
WebDaemon.macoprod.4103 -debug --ServerPort=4666 --
MultipleServerPorts=1 --CGIPort=4421 --CGICallbackPort=4521 --MinServerProcs=2 --
MaxServerProcs=2
```

Here WebDaemon.macoprod.4103 should be substituted by name of the “real” Web Daemon and the port numbers should not be currently in use. On Windows the TPU in the command should be the TPU currently associated with the application.

The ‘--' parameters for the Web Daemon command modify the parameters in the WebDaemon.macoprod.4103.I file, so you get a Web Daemon with a server pool with exactly two processes – which will be processes you started earlier.

The Web Daemon output will be like the log output explained above. Wait for the status line to show ‘-‘ (“ready to serve”) for both servers (“Server status:  --“) before starting the Java Client / Portal.

The parallel Web Daemon in this example runs on CGI port 4421, so when using this Web Daemon you should use a parallel URL for the Java Client or Portal, and in the Jaconomy.<…>.I or MaconomyPortal.<…>.I for the parallel URL the *DaemonPort* setting must be modified to the CGI port of the parallel Web Daemon – in this example 4421.

# Coupling Service

For the Maconomy Coupling Service, some standard logging functionality runs permanently, and different kinds of additional logging can be enabled when needed.

## Standard Coupling Service Logging

The standard Coupling Service log files are located within <*Coupling Service main folder*>*/log* which by default has two or three subfolders *coupling, metrics* and *servicewrapper* (the latter only exists on Windows).

A script is available that can gather these files together with configuration info into a ZIP file. The script is located within <*Coupling Service main folder*>*/exportlog* and named *exportlog.bat* for Windows and *exportlog.sh* for Unix/Linux. The generated ZIP file will be located in <*Coupling Service main folder*>*/exportlog/output*.

### Log Files in *Coupling* Subfolder

These files contain error and warning messages plus notifications from the Coupling Service. There are a number of files for various kinds of information, the most important ones are *maconomy.log* and *maconomy-uncaught.log*. The files are “rotated” daily, meaning that new files are started from scratch every day and files from earlier days will have a date mark included in their names, e.g. *maconomy-2016-10-26.log*, while Today’s log files have no date mark in their names.

### Log Files in *Metrics* Subfolder

These files contain statistic data being gathered constantly, which often is useful when tracing errors and performance problems. The info is generally gathered on an hourly basis and written into CSV files (comma separated files) which can be read with e.g. Microsoft Excel.

The metrics log files are “rotated” so that they are started from scratch when the Coupling Service is restarted. The sets of files from each rotation is stored within a separate subfolder under <*Coupling Service main folder*>*/log/metrics*, e.g.
<*Coupling Service main folder*>*/log/metrics/csv/csv-internal-20161010-160908-3600sec*

### Log Files in *servicewrapper* Subfolder

These only exist on Windows and are related to issues related to the Coupling Service being run as a Windows service. They are usually less important than the coupling and metrics log files.

## Additional Coupling Service Logging

The logging in <*Coupling Service main folder*>*/log/coupling* is defined in the file <*Coupling Service main folder*>*/configuration/settings/logback.xml* and additional logging can be defined by adding to that file. Changes to *logback.xml* will work immediately, i.e. no restart of the Coupling Service is needed.

As an example, adding this to logback.xml will add lo info on launching of workspace containers to the maconomy.log file:

```
<logger name="com.maconomy.api.container.launcher.local.McContainerLauncher" additivity="false">
  <level value="DEBUG" />
  <appender-ref ref="FILE" />
</logger>
```

Use this in conjunction with Maconomy R&D.

# Garbage Collection Logging

Garbage collection may often be a subject of investigation when exploring performance problems possibly related to the Coupling Service, and logging of garbage collection can be enabled this way:

*On Windows:*

Add these lines to <*Coupling Service main folder>/servicewrapper/conf/wrapper.equinox.conf*

```
wrapper.java.additional.<N>=-Xloggc:{some-folder}/gc.log
wrapper.java.additional.<N+1>=-XX:+PrintGCDetails
wrapper.java.additional.<N+2>=-XX:+PrintGCDateStamps
```

where <*N*>… are the next yet unused numbers in the wrapper.java.additional… lines in the file, and *{some-folder}* is your chosen location for the log file. Note that you must use '/' for the log file path, even on Windows – and may of course choose another name than *gc.log* for the log file.

For Java 8, it is possible to add *%t* to the file name to get a date/time stamp added to the log file name, e.g.

```
wrapper.java.additional.<N>=-Xloggc:F:/PUs/Logs/gc-%t.log
```

*On Unix/Linux:*

Add these lines to <*Coupling Service main folder>/CouplingService.ini*

```
-Xloggc:{some-folder}/gc.log
-XX:+PrintGCDetails
-XX:+PrintGCDateStamps
```

For Java 8, it is possible to add *%t* to the file name to get a date/time stamp added to the log file name, e.g.

```
-Xloggc:F:/PUs/Logs/gc-%t.log
```

Note that enabling/disabling logging this way requires restart of the Coupling Service – and be aware that the log files must be collected *before* next Coupling Service restart, otherwise they get overwritten.

You may extend the specification to get log file rotation:

*On Windows:*

Add these additional lines to
<*Coupling Service main folder>/servicewrapper/conf/wrapper.equinox.conf*

```
wrapper.java.additional.<N+3>=-XX NumberOfGCLogFiles=<number of files>
wrapper.java.additional.<N+4>=-XX:GCLogFileSize=<number>M (or K)
```

*On Unix/Linux:*

Add these lines to <*Coupling Service main folder>/CouplingService.ini*

```
-XX NumberOfGCLogFiles=<number of files>
-XX:GCLogFileSize=<number>M (or K)
```

The sizes of the individual files can be specified in megabytes (M) or kilobytes (K).

# Setting up a Parallel Coupling Service

Setting up a parallel Coupling Service for problem investigation, including scenario replication, can sometimes be useful, and when using Maconomy Performance Monitor (see next section), especially on a production system, a parallel Coupling Service may prevent or limit resource problems related to use of MPM. You can set up an additional CouplingService for debugging via MConfig.

# Maconomy Performance Monitor (MPM)

The Maconomy Performance Monitor (MPM) is a debugging tool that is designed to time the different processes on the Maconomy server and Coupling Service.

One instance of the MPM system is initiated inside every MaconomyServer process that is running on the Maconomy server machine. The MPM system monitors and times much of the internal functionality in the Maconomy server, such as actions, SQL expressions, MQL requests, Universe Report requests, and so on.

Using MPM is especially useful for exploring performance problems that are not database related, or problems where we do yet know what they are related to. An advantage of MPM is that it can be enabled dynamically, with no need for restarting Web Daemons or Coupling Services.

The framework contains a set of predefined time measuring objects, called probes, placed in, for example, the code of the Maconomy Server or the Coupling Service. Each of these probes is an object dedicated to a specific monitoring task. An example of a task is measuring the execution time of MSL dialog scripts. Besides measuring time, a set of attributes are assigned to every probe.

The probes can be configured and enabled on a runtime system to monitor performance or system activity. Probes may be enabled only for a selected set of users. The output is logged in XML format.

In recent Maconomy versions, MPM is monitored through the Coupling Service. See the *Maconomy System Administrators Guide* for info on how to use MPM on Maconomy systems with no Coupling Services.

There are probes for the Maconomy Server program and probes for the Coupling Service, and they are controlled differently.

An import rule for using MPM is:

***Never enable all Maconomy Server probes for all users on a running production system. It may cause massive performance problems.***

On a non-production system, e.g. a test system, you may fully enable MPM. On a production system, you may enable MPM, but always do it only for a limited set of Maconomy Server probes and/or a limited set of users – and always monitor the system carefully.

Enabling all probes for the Coupling Service is usually not a problem – but still monitor the system as our experience with this is still limited.

## Setting up MPM through mpm.ini

MPM is enabled/disabled/configured through the <*Coupling Service main folder*>*/configuration/settings/mpm.ini* file.

The *mpm.ini* file contains a listing of all possible settings. You can find descriptions and instructional text on the particular settings in the README.txt file located in the same folder– read this if you are in doubt about the meaning of the settings and the syntax.

By default the MPM log files are stored within <*Coupling Service main folder*>*/log/mpm/xml*. For each enabling or change of configuration a new sub folder is created for log files, e.g. <*Coupling Service main folder*>*/log/mpm/xml/mpm-20161108-110504*.

To enable MPM, locate the setting

```
mpm.enabled = false
```

and change the value to *true* – change back to *false* when you are done.

Up to and including Maconomy v. 2.2.3, enabling MPM this way activates all probes for both Maconomy Server and Coupling Service for all users – i.e. the setup that should be avoided on production systems.

From Maconomy v. 2.2.4 and 2.3, a setting has been introduced:

```
mpm.server.all-probes = false
```

This means that initially all probes for Maconomy Server are not activated, i.e. only the Coupling Service probes are active (which is not expected to cause severe performance problems). Changing the setting value to true will activate all Maconomy Server probes – as before the setting was introduced. Activating selected Maconomy Server probes may be done on the application server, more on that later.

Some filtering options are available in *mpm.ini*. They are all prefixed with `mpm.filters.`

Most settings in the section are by default commented out by a '#' character which must be removed if you want to change the filtering setting. Some of the filtering options are:

- **session-wallclock-time**. By default all sessions shorter than 10 seconds are filtered out, if you e.g. don't want this filtering locate the line

  ```
  session-wallclock-time = >= 10
  ```

  and change 10 to 0. You can use operators such as '<=' or '>'.

- **probe**. If you only want some selected probes to be logged you can e.g. write

  ```
  probe = MSLDialogScript or McContainerLauncher
  ```

  and then only these probes are logged (plus a few more basic ones that are always included). The setting value in this example actually means "probe name contains MSLDialogScript or probe name contains McContainerLauncher". You can also use 'and' or 'not' in the value specification. There can only be one probe line in mpm.ini – the example shows how to specify settings for multiple probes.
  Note that the filtering applies to what will appear in the log. For the Coupling Service probes it also defines which probes are active, while for the Maconomy Server probes the are still active depending upon *the mpm.server.all-probes* setting although not appearing in the log output

- **principal-name**. If you want only to log only sessions for selected users – which may be a very good idea when using MPM – you can use this setting to filter on user names

  ```
  principal-name = John or "Jack D"
  ```

  again this means "… contains John or …" and you can use 'and', 'or' and 'not' and put names within quotes in case they contain space characters.
  This setting prevents sessions for not-specified users from being logged.

# Setting Maconomy Server Probes on the Application Server

For v. 2.2.4 and later where the *mpm.server.all-probes* setting are available, you can activate selected server probes this way.

The *mpm.server.all-probes* setting in *mpm.ini* should still be *false* (the default value). In <*application home*>*/MaconomyDir/Definitions* create a file *MPMDefault.cfg* with contents like this:

```
MPML 1

Client #ALL

Company <shortname>

user #ALL

probe       main log
attributes  all on

probe       MSLDialogScript  log
attributes  Title on
            Start on
            ProcessTime on
            WallclockTime on

probe       ROEPrint  log
attributes  Start on
            ReportId on
            ProcessTime on
            WallclockTime on
```

Substitute <shortname> with the relevant value.

In this example, the probes *MSLDialogScript* (Maconomy core business logic) and *ROEPrint* (printing) are activated. We have experienced that this setup can run without causing performance problems even on production systems. Maconomy Engineering may from time to time ask for activation of other probes.

It is often useful to combine enabling of MPM with setting up a parallel Coupling Service, especially on production systems – see the section on the Coupling Service.

# Analyzing MPM Log Files

The MPM log files are XML files that are often large and not easy to read manually. However, a tool is available for the analysis. The tool is a Python 3 script called mpmAnalyzer.py.

Simply providing an MPM log file to the script like this:

```
mpmAnalyzer.py <MPM log file>
```

will produce output like this:

```
WallClockTime   WallClockTimeTotal      Level   Probe
-------------   ------------------      -----   -----
         3.137 1       Main - McPerformanceMonitorService.XmlEntryVisitor
         3.137 1.1       McCouplingServiceServer - McCouplingServiceServer.executeRequest
         0.002 1.1.1       McServiceProvider.imp1.en_GB_MCS - McServiceProvider.getLease
         0.002 1.1.1.1       McRemoteInterface.imp1.en_GB_MCS - McRemoteInterface.callFunction
 0.000   0.001 1.1.1.1.1               MaconomyServer.Main
         0.023 1.1.2       McRemoteInterface.imp1.en_GB_MCS - McRemoteInterface.callFunction
 0.020   0.020 1.1.2.1       MaconomyServer.Main
         0.001 1.1.3       McRemoteInterface.imp1.en_GB_MCS - McRemoteInterface.callFunction
 0.000   0.000 1.1.3.1       MaconomyServer.Main
         0.003 1.1.4       McContainerLauncher.imp1\ekz074 - McContainerLauncher.open
         0.001 1.1.4.1       McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1               McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1               McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1.1               McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1.1.1               McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1.1.1.1                     McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1.1.1.1.1                       McEvent.imp1\ekz074 - McOpen.runContribution
         0.001 1.1.4.1.1.1.1.1.1.1.1                       McEvent.imp1\ekz074 - McOpen.runContribution
         0.000 1.1.4.1.1.1.1.1.1.1.1.1                       McEvent.imp1\ekz074 - McOpen.runContribution
```

The Level column with contents like *1.1.4.1.1.1.1.1.1* reflects the highly hierarchical tree structure of the XML log file. You should watch the *WallClockTimeTotal* column for high values. The analysis script offers a lot of options for deeper exploration, e.g.:

```
mpmAnalyzer.py <MPM log file>  -m 6
```

for only showing 6 levels of the analysis tree

```
mpmAnalyzer.py <MPM log file>  -k 1.1.4.1.1.1.1.1
```

for only showing the part of the tree below the node that appears as *1.1.4.1.1.1.1.1* in the output

The options may be combined like this:

```
mpmAnalyzer.py <MPM log file>  -m 6  -k 1.1.4.1.1.1.1.1
```

The columns shown in the example may are the default set. However various probes have attributes that may be important, and you can add probe attributes as columns to the output like this:

```
mpmAnalyzer.py <MPM log file>  -s FunctionName  -s SqlQuery
```

This will add two columns to the output, *FunctionName* and *SqlQuery*, and show the attribute values for the probes that have these attributes. You may inspect the XML log files to find out which columns may be interesting, but *FunctionName* and *SqlQuery* may often be good candidates.

# MConfig

The log file for MConfig is named *MaconomyServerInstallation.log* and is located in <*main Maconomy folder*>*MaconomyInstallLogs* (Windows) or */usr/maconomy/MaconomyInstallLogs* (Unix/Linux).

The MConfig log file is important when dealing with MConfig relates issues, but it may also be relevant in other cases, since it contains the entire installation history of a server with Maconomy systems.

Related to MConfig is also the file *registry.server* which contains info on installed applications the PUs being applied, installation location and various setup parameters. On Windows it is a text equivalent of the info stored in the registry, and it is located in the same folder as the MConfig log file, i.e. <*main Maconomy folder*>*MaconomyInstallLogs*. On Unix it is the file MConfig is actively using, and it is located in */usr/maconomy.*

# Oracle

## Alert Log and Trace Files

For Oracle related problems, the alert log is often a good place to look for relevant error messages. The name of the alert log file is *alert_<SID>.log*.To find the location of the alert log:

- For databases (SIDs) created using MConfig the alert log, files are located in *<Oracle base folder>/admin/<SID>/log/diag/rdbms/<SID>/<SID>/trace*.
- If you cannot find it there, look up the value of the Oracle parameter *diagnostic_dest*, the location will be *<diagnostic_dest>/diag/rdbms/<SID>/<SID>/trace*.
- If the Oracle parameter *diagnostic_dest* does not exist, the location is defined by the Oracle parameter *background_dump_dest*

You can look up an Oracle parameter when using SQLPlus by the command

```
show parameters <parameter>
```

e.g.

```
show parameters background_dump_dest
```

The alert log file is one file in which messages accumulate. For certain incidents, trace files may be generated. They are named <SID>…trc, and are usually located in the same folder as the alert log. However, they may also be located in the folder defined by the Oracle parameter *user_dump_dest.*

## Setting up Oracle Tracing

Especially when investigating issues related to the Oracle optimizer (i.e. if you suspect that indexes are not being applied optimally), it may be useful to set up tracing of user sessions where issues are replicated. To set up tracing, create a file *OracleHints* (NOT OracleHints.txt) in *<application home>/MaconomyDir/Definitions* (if it does not already exist) and insert these lines

```
SETUP alter session set events '10053 trace name context forever, level 1';
SETUP alter session set events '10046 trace name context forever, level 12';
```

Web Daemons or Coupling Service (whatever is relevant) must be restarted unless you are using this together with parallel server processes and Web Daemons (if so, you must modify *OracleHints* **before** you start the parallel server processes).

This will generate trace files for client session as explained above. Close down the client, and wait a little while, before you pick up the trace files.

When done, revert the changes to the *OracleHints* setup.

## Explain Plans

Examining explain plans for particular SQL statements can reveal problems with missing indexes and other cases of non-optimal use of indexes. When using SQLPlus this is the way to show explain plans:

To make output more readable:

```
set linesize 150
```

For subsequent SQL statements, run them and display explain plan:

```
set autotrace on
```

For subsequent SQL statements, don't run them, just display explain plan:

```
set autotrace traceonly explain
```

For subsequent SQL statements, run them and don't display explain plan (i.e. default behavior):

```
set autotrace on
```

# Materialized Views

If a Maconomy system has been set up with materialized views for BPM optimization, serious performance problems may arise if the views are somehow not functioning as desired.

Here are some statements to check the status of the materialized view. For the statements shown below, it is assumed that materialized views are applied to two Maconomy tables: JOBENTRY and JOBINVOICELINE. This is how it is today, but it may change in the future, and the statements should be modified accordingly.

To change the view logs run:

```
select count(*) from MLOG$_JOBENTRY;
select count(*) from MLOG$_JOBINVOICELINE;
```

This will show how many changes are currently recorded in the view logs. Under normal circumstances, these should show zero. If they do not, it is because they currently store changes that are not transferred to the materialized view. If it seems like the above tables are not emptied at any time, it can be because the materialized view is not able to empty them as it should. One reason can be because more than the two view logs have been installed but without the corresponding materialized views to empty them.

Over time, SQL statements for updating the materialized view will pile up in the shared pool. This is a known issue in Oracle and requires that you empty the shared pool.

A stored procedure, which is usually set up with MConfig, has been developed for flushing the shared pool once a day. To check that the flushing procedure runs as expected, execute the following command. *Failures* should be zero and *broken* should be 'N'.

```
select job, what, last_date, next_date, failures, broken from dba_jobs where what
like '%maconomy_flush_shared_pool%';
```

To check that the flushing procedure cleaned the shared pool, execute the following command. If successful, it should return no entries.

```
select sql_text from v$sql where sql_text like '/* MV_REFRESH%';
```

# Appendix A: List of Logging Switches for MScript-based Interfaces

Logging switches that can be enabled in the *Log* parameter of e.g. *MaconomyPortal.<…>.l* or *MaconomyWS.<…>.l* in *cgi-bin/Maconomy* on the web server, the switches may be combined by specifying a ';'-separated list, e.g. "Log = login;scripts".

| | |
|---|---|
| **none** | No additional logging at all, only log errors (default) |
| **sql** | Log only maconomy::sql calls. |
| **analyzer** | Log only maconomy::analyze calls. |
| **login** | Log only maconomy::login calls. |
| **functions** | Log all maconomy:: calls. |
| **scripts** | Log script execution load and time information. |
| **serverstat** | Log information about server-side timing. |
| **query** | Log information about received query variables. |
| **postdata** | Log all incoming POST data in a file with the same name as the M-Script executable appended with a .postdata.log. |
| **cleanup** | Log information about session cleanup activities. |
| **localization** | Log information about processing of localization dictionaries. |
| **GUI_windowStatus** | Log information about opening and closing of GUI windows. |
| **GUI_warnings** | Log extra information about the GUI. |
| **client** | Combines script, query and localization. |
| **server** | Combines sql, analyze, login, functions, and serverstat. |
| **GUI** | Combines GUI_windowStatus and GUI_warnings. |
| **all** | Log all of the above. |
| **performance** | Log information about how various parts of M-Script perform. |
| **tab** | Make log entries in a tabular format. |

# Appendix B: List of Available Test Masks for Debugging Maconomy Server Processes

| Category | Mask |
|---|---|
| 1: Legacy – backward compatibility *(from Maconomy v. 2.2.4 / 2.3)* | 1: (Generic) |
| 6: MSL – core Maconomy business logic | 1: Basic MSL |
| | 2: SQL-RTS – detailed basic MSL |
| | 3: Timing |
| | 4: Tracing at instruction level |
| | 6: Input tracing |
| | 7: Stack trace |
| | 8: Trace SQL |
| | 9; Suppress SQL |
| | 20: Dumping |
| | 21: Trace cache calls |
| | 30: Extended tracing |
| | 50: Cursor generation |
| 7: Dialog tracing | 1: (Generic) |
| 9: SQL / Database | 2: SQL tracing |
| | 3: Loaded and modified data records etc. |
| | 4: Query dump |
| | 5: Bind variables *(from v. 2.2.4 / 2.3)* |
| | 6: Returned data |
| | 7: Trace SR SQL Request module *(from v. 2.2.4 / 2.3)* |
| | 8: Trace SS Server SQL module *(from v. 2.2.4 / 2.3)* |

| Category | Mask |
|---|---|
| | 9: MQL Universe dump *(from v. 2.2.4 / 2.3)* |
| 12: Trace TDH component (popup handling) | 1: (Generic) |
| 13: Trace print stacks | |
| 21: MQL (analyzer reports) | 1: Trace compiler |
| | 2: Explain plans |
| | 3: Hints |
| 22: MQL cache | 1: Trace caching |
| | 2: Dump cache |
| 24: MQL output engine | 1: (Generic) |
| 25: Report transcoding | 1: (Generic) |
| 26: Access control management | 1: (Generic) |

# About Deltek

Better software means better projects. Deltek is the leading global provider of enterprise software and information solutions for project-based businesses. More than 23,000 organizations and millions of users in over 80 countries around the world rely on Deltek for superior levels of project intelligence, management and collaboration. Our industry-focused expertise powers project success by helping firms achieve performance that maximizes productivity and revenue. www.deltek.com