MICRO FOCUS TROUBLESHOOTING AND DIAGNOSTICS FOR ENTERPRISE AND COBOL PRODUCTS

Mar 1, 2018
Version 2.7

Revision History

How To Use This Document

This document is in 3 parts – the 1st part covering an overview of the diagnostic capabilities, the minimum diagnostic recommendations for a production system, how to capture the required data and how best to report a problem to Micro Focus such that is handled efficiently.

The 2nd part (beginning with the section “Configuration of ES-specific Logs, Dumps and Traces”) contains all the detail required to configure various diagnostic operations and the table of contents allows efficient navigation to the required topic.

Additional supplementary information is contained in appendices which form the last part of the document.

It is recommended that to get the most out of this document the 1st part is initially read and understood, then the table of contents can be used to find specific references in the 2nd part, as and when different diagnostic features need to be configured.

While information in this document is primarily aimed at the Enterprise suite of products, certain aspects will also be relevant to the Cobol product set.

TABLE OF CONTENTS

INTRODUCTION

Contact Micro Focus Professional Services via email at consulting@microfocus.com with any questions, issues or suggestions or inputs. We value your inputs.

This document details diagnostic facilities that can be used to analyze, understand and resolve problems during the project implementation or production running of an application using Micro Focus Enterprise products (Enterprise Developer, Enterprise Test Server and Enterprise Server). In addition certain aspects will also be applicable to the Visual Cobol products.

The Micro Focus products that are covered by this document are listed below

Enterprise Products:

All the Enterprise products contain the same run time execution technology (if they are at the same version of product). They are described briefly below:

- **Enterprise Server (ES)** - the run time execution technology only
- **Enterprise Test Server (ETS)** - exactly the same product as Enterprise Server with the addition of XDB for highly compatible mainframe DB2 testing and Remote Job Step Execution (RJSE) technology to enable the execution of specific JCL and utilities on the mainframe during integration testing. In addition it provides emulation of IBM mainframe assembler language.
- **Enterprise Developer (ED)** - for any organisation developing or modernizing enterprise applications looking to take full advantage of off-mainframe development tools to improve developer efficiency, time to market and reduce their dependency on the IBM mainframe. Applications can be deployed back to the mainframe or developed and tested on alternative server platforms. It contains the same run time technology as Enterprise Server plus the IDEs (Eclipse or Visual Studio), the compiler and generator and the source code debugger technology (COBOL and PL/I compilers).
- **Enterprise Developer Connect** - an Eclipse based development environment that allows any organisation developing applications directly on the IBM Mainframe to take advantage of modern development tools directly integrated into their existing mainframe tools and processes
- **Enterprise Developer for IBM zEnterprise** – for organisations who want to take advantage of modern development tools and want the choice of developing directly on or off the mainframe from the same toolset given them the freedom on where they develop and deploy these applications

Most customers would use the ED product to build and unit test their application and then either deploy to ETS for integration testing before deployment back to the IBM mainframe or deployment to ES for production execution on a distributed platform (Unix or Windows). The ED product only runs on windows although it can be used to remotely debug and publish to Unix platforms.

Cobol products:

- **Visual COBOL for Eclipse** - An Eclipse based COBOL application development toolset supporting Windows, Unix and Linux platforms and JVM deployment.
- **Visual COBOL for Visual Studio** - A Visual Studio based COBOL application development toolset supporting Windows and .NET deployment.
- **Visual COBOL Developer Hub** - A compiler only product supporting Linux and Unix systems and providing backend support for the remote development option available within Eclipse.

- **COBOL Server** - The deployment product for applications developed using Visual COBOL.

This document covers Enterprise Server diagnostics first, then the specific additional features in Enterprise Test Server (XDB, RJSE and Assembler) and then the Enterprise Developer Product – The ES and ETS diagnostic details are all relevant to ED, consequently the ETS section only covers XDB, RJSE and Assembler and the ED section only the compiler and debugger as the rest of the information applicable to these products is under Enterprise Server.

The enterprise products have a varied set of facilities for problem analysis. These diagnostic facilities are a result of the experience of numerous implementations and the realization that there is a varied set of problems that can occur, each requiring its own type of approach. This document will try to suggest which of the many diagnostic facilities to use for these different problems.

At a minimum we suggest that those involved in the implementation of these products should know, for each diagnostic tool or technique:

- What it does
- How to configure it
- What the output is that is produced and where to find it

These three areas are covered for each diagnostic facility.

Once the appropriate facility has been identified, we aim to help any users of the products set up and run the diagnostics and then either analyze the output themselves or send them to Micro Focus for help from our Support teams.

Depending on the type of problem, several of these diagnostic facilities might need to be used simultaneously. For instance, a regular start point for the Enterprise products is the console log. However, this rarely shows the full problem, so very often a trace output and dump are needed to complete the analysis. The console log might identify the failing process and the trace and dump output will show the detail of what the process was doing and how it was interacting with the other processes if that is critical. Alone these may be of little use, together they can pinpoint what led to the failure.

Although it is well understood that during project implementation and production running, speed can be of the essence to complete the project or bring the business system back up, if Micro Focus is to help then it is vital that the diagnostic output is gathered in its entirety and treated as data of very high value to the customer and ourselves in its detail and its completeness.

Please treat this document as a reference and bear in mind that each problem is different and can require its own approach to diagnosis, so pick from the facilities detailed here based on what you believe to be the type of problem.

PART 1 – OVERVIEW AND BEST PRACTICE

1. OVERVIEW OF DIAGNOSTICS IN ENTERPRISE SERVER

Within ES there are logs that are generally always written to and require little or no configuration, while for more detailed tracing, additional parameters and configuration files can be set to capture a much more detailed set of diagnostic information.

1.1 DEFAULT LOGS

These logs require no configuration as they are enabled for at least some logging by default. However, additional configuration is available for some of them (e.g. to control their size and or content).

1.1.1 CONSOLE LOG

The most important log in the system is the console.log. This contains the region activity since startup and hence shows what happened leading up to a failure.

This log should be reviewed 1st to see what errors have occurred, when they occurred and what process was responsible. The Process ID and time from this log will allow entries in other logs to be associated with this particular error or entry in the console.log

1.1.2 MICRO FOCUS COMMUNICATIONS SERVER (MFCS) LOG.HTML

This logs messages from the communications manager process(es) and any associated listeners. It shows specific information related to the addresses and port numbers used.

1.1.3 MICRO FOCUS DIRECTORY SERVER (MFDS) JOURNAL.LOG

This is the log for the Micro Focus Directory Server (MFDS) process. This process provides an information repository and is one of the 1st things to be accessed during region startup. If security is enabled this log can also help show whether and which connections to MFDS have been successful or are rejected etc. Note that the MFDS process is a single process that provides an information repository for the entire Enterprise Server installation. This means therefore that it interacts with all defined regions running on this machine and hence entries in the Journal log could be from more than one region.

1.2 CONFIGURABLE TRACES AND DIAGNOSTICS

These require some additional configuration before they are active.

There is generally a wide range of settings available to allow specific and relevant information to be collected (note that this can result in very large files so disk space should be checked when configuring these traces).

1.2.1 AUXILIARY TRACE

This provides a trace of activity over a time period within the main CAS facilities. Because it runs whilst the system is running it can have an effect on performance of the system, therefore it is very configurable to record the specific types of events that may be needed for different types of problems. Ideally production systems should be able to perform adequately with at least the minimum tracing enabled.

Aux trace is usually configured from the ES Admin Console (i.e. ‘Aux Trace Active’), but can also be configured using the ‘CTRA’ transaction from a 3270 session. Trace information is written to the trace files CASAUXTA and CASAUXTB and the amount of information output is controlled by the setting of specific trace flags (e.g. Application (API), Storage Control (SCP) and Application Container (RTS)).

1.2.2 Casdump

A System dump provides a 'time slice' view of resources within the system at the time that the dump is requested, it is valid only for that specific time. Taking the dump should not have any performance impact on the system. A manual dump can be taken at any time (with no additional configuration required) from the command line or the ESMAC interface, however a region should be configured to generate a system dump whenever a System Abend occurs. This is usually configured in the ES Admin Console (i.e. 'Dump on System Abend').

Additionally "Dump on Transaction Abend" can be set in ESMAC so that any abending transactions will also cause a dump to be generated.

Dump information will be written to the current casdump file (casdumpa.rec, casdumpb.rec or casdumpx.rec) and includes the current contents of process memory, server control blocks and the local and system trace tables.

A dump should probably be taken for most problems when they occur and whilst the system is in the failed state.

1.2.3 CTF Trace

This is the primary logging mechanism for the COBOL system which includes the logging of what happens in most user programs and what is happening within the internal CAS processes themselves (e.g. JCL and the compiler/debugger).

It records activity over a timeline period and is enabled via a separate configuration file. CTF tracing can be restricted to specific components/modules and certain elements within these components. The level of detail included in the trace can be configured for each component if required.

1.2.4 MFCS connector tracing

Each MFCS listener has an associated connector module (specified using the "listener type" setting in the MFDS GUI), and most connector modules have optional tracing that is configured in the Configuration Text area for that listener (generally by using "trace=y" in a [Trace] section).

1.2.5 Memory Strategy

Memory strategy can be enabled where memory corruption is detected or suspected. When it is enabled it allows the COBOL sub-system to verify/validate memory addresses as they are used by a program. This means that when memory operations occur, the system is able to validate that only the assigned memory has been used, if this is not the case then an exception will be generated at the point that the corruption is detected. The Memory Strategy facility provides a fine-grained control over which processes are checked and the extent of the checking that is done, this keeps the overhead of enabling Memory Strategy to a minimum.

1.2.6 The Historical Statistical Facility (HSF)

The Historical Statistical Facility provides the ability to record statistics related to the time spent in a Cobol program and the time that parts of that program spent in CICS API commands/tasks, SQL operations and JCL jobsteps, as well as associated file and TSQ accesses.

1.2.7 Core on Error

This feature causes a program core dump to be generated when that program causes an exception. Once a core dump is generated, it can be loaded into the development environment where the offending line of code should be shown and the call stack and values of variables can be examined. It is most useful in a production environment where it is unlikely that a debugger can be dynamically attached - the core dump can be obtained and then analysed on another system. The .IDY file(s) should be available for the programs that fail as this will allow the core dump to show the most information regarding the state of the program at failure time, without the .IDY file(s) only a limited amount of diagnostics can be performed.

1.2.8 Fileshare Trace

Fileshare is a transactional filehandler for VSAM and other Micro Focus COBOL files and Fileshare trace would only be required if you have Fileshare configured for use. This facility provides specialist tracing of the transactional file handler technology. Generally Fileshare trace should only be used for specific cases during testing as it can have a great impact on performance.

1.2.9 External File Handler Trace

The File Handler module can be configured to generate trace information that can be later replayed and analyzed using the associated utility program. This would show information such as filenames accessed and the operation used (e.g. Open or Read). More detailed information can also be generated for specific file operations (e.g. access mode and file format).

1.2.10 XA Switch Module Tracing

Additional trace can be enabled to provide more detail regarding the XA Resource Manager interface. The console.log will contain basic errors and other messages relating to XA resources, additional AUX and CTF trace flags can be set to output more detailed XA messages to the relevant trace files.

1.3 Security Tracing

Security trace messages can be written to the console.log, the MFDS journal log and the AUX trace. Detailed security tracing is configured in the relevant Configuration Text areas which exist for:

a) MFDS and globally for all regions (‘security’ button in left hand pane) via:

    Default ES Security tab

    Each Security Manager definition.

b) Individual regions via:

    Region’s Security tab (from ‘Server’ -> ‘Properties’ tabs)

The available options for Security Manager tracing depend on the External Security Manager (ESM) module used by that manager.

In addition the security AUX trace flag (saf) can also be enabled for region-specific security tracing.

Note that problems with initial access to MFDS (e.g. to start a region) will be logged in the MFDS journal, however trace must first be setup in the External Security Manager (ESM) to output these messages.

Security tracing can have a significant effect on performance so should not be left on in a production system.

1.3.1 Security Auditing

The security audit facility is sometimes useful for diagnostic purposes. It maintains a record of security decisions (user signon succeeded, resource access request denied, etc.). It requires the Audit Manager to be running (which can be from the command line, or as a service on Windows or a daemon on Unix). It is configured with a configuration file that is specified with the '-c <filename>' option.

1.4 Non-Micro Focus Diagnostics

The utilities available with the Operating System or available from third parties are often very useful in providing additional information that can complete the diagnostic picture or indeed can identify specific problems directly e.g. with the environment setup or network.

On Windows it may be necessary to obtain additional tools to those included in the OS, in particular there is a suite of tools originally supplied by Sysinternals.com and now distributed by Microsoft that include most of the useful tools (e.g. procmon).

On Unix, tools such as strace and truss may be provided as optional installation components or are otherwise available for download and installation.

The following external utilities have proved useful in providing supporting information when a problem occurs or investigating problems that would be exposed directly to the OS:

- Windows
    - Process Monitor (**Procmon** – real-time process activity)
    - **Netstat** (TCP/UDP port connection information)
    - **Process Explorer** (open handles and DLLs), also (from command line):
        - Handle (open handles/files in the OS)
        - listDLLs (DLLs loaded in a process)
    - **Depends** (dependent modules and exported functions)
    - Windows Task Manager
    - Windows Resource Manager
    - Windows Performance Monitor (+SCOM)
    - Windows Event Manager
    - **TCPview** and **Wireshark** (TCP endpoints, addresses and traffic)
    - Debugging Tools for Windows **(ADplus**, **Windbg)**
    - Microsoft Network Monitor (**NetMon**) (can trace VPN networks)
    - **Rawcap** (can trace the loopback interface)
- Unix
    - **strace/truss**
    - **lsof** (list open files)
    - **netstat** (TCP/UDP port connection information)
    - **ipcs** (interprocess communications - for shared memory etc.)
    - **ulimit** (limits on open files)

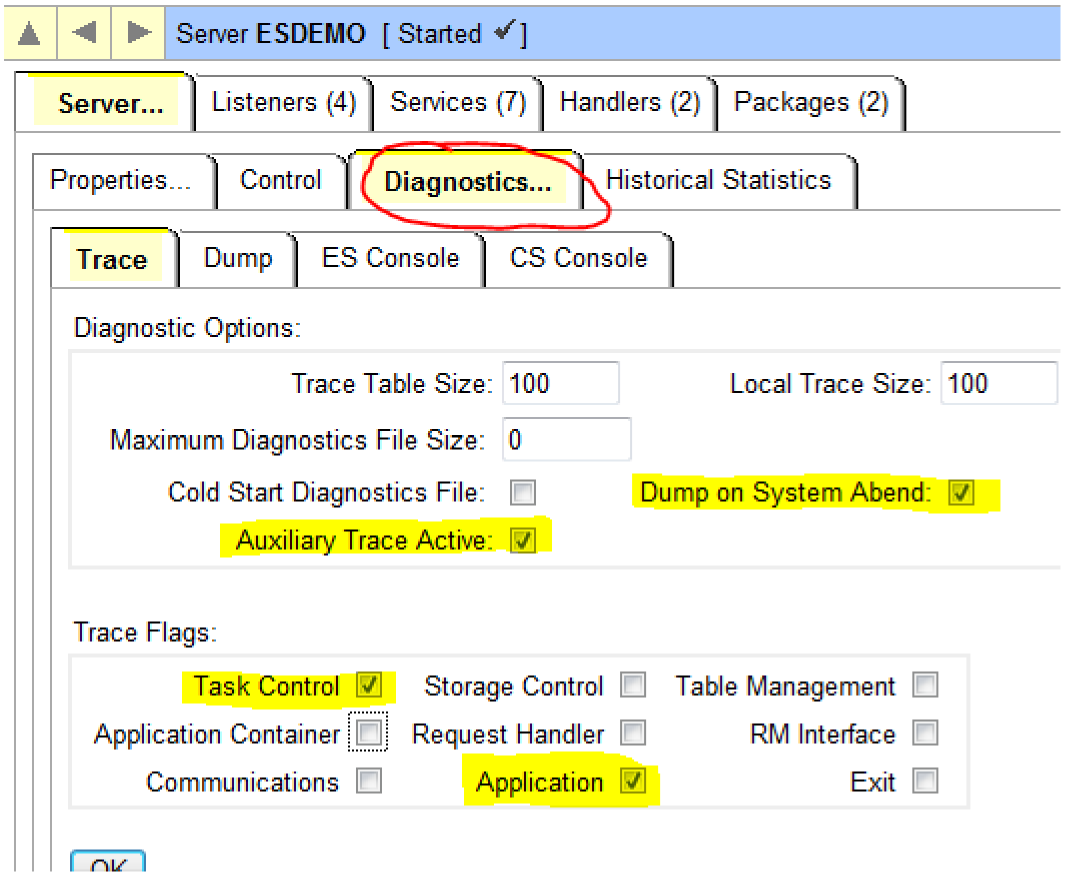
2. Minimum Diagnostic Configuration Recommendations for Production Systems

It is recommended that a production system is run with some diagnostics switched on. This is necessary so that unexpected problems can be investigated when they first occur. Diagnostics consume resources, and there is always contention between problem determination facilities and performance. You need to decide how important it is to be able to diagnose a problem the first time it occurs rather than having to try to recreate it or having to just turn on diagnostics and wait for it to re-occur.

'Dump on System Abend' should always be enabled in an Enterprise Server region, along with some Auxiliary tracing. The following level of diagnostics is the recommended minimum for running in production systems:

From the ES Admin Console -> Diagnostics tab:

- Select "**Dump on System Abend**".
- Select "**Auxiliary Trace Active**".
- Select the **Task Control** (KCP) and **Application** (API) trace flags.



Diagnostic Options:

Trace Table Size: 100
Local Trace Size: 100
Maximum Diagnostics File Size: 0
Cold Start Diagnostics File: ☐
Dump on System Abend: ☑
Auxiliary Trace Active: ☑

Trace Flags:

Task Control	☑	Storage Control	☐	Table Management	☐
Application Container	☐	Request Handler	☐	RM Interface	☐
Communications	☐	Application	☑	Exit	☐

In addition it is strongly recommended that the 'core_on_error' facility is enabled so that application core dumps are created whenever there is a crash – see the section "Core Dumps for run-time errors"

Please also refer to the community article “Minumum Enterprise Server diagnostics recommentations” which additionally details recommendations for JCL tracing.

3. Production System Failure Data Capture

For all troubleshooting exercises it is essential to gather as much information as possible about the state of the enterprise server region when the problem occurred and about the events leading up to the problem. This should include the date/time the problem occurred and observations on what was happening in the system, how long it had been running what the symptoms were and how/where these were observed.

When a problem occurs, it is necessary to capture and provide specific logs, traces and dumps, and ideally the contents of various configuration files, directories (e.g. MFDS), and the output from a number of operating system tools. This should be done as soon as possible after the failure occurs.

3.1 Diagnostic Collection Script

The MFESdiags diagnostic collection script can be used to collect this information automatically.

On Windows the script is MFESdiags.cmd which can be obtained from support. It can be run from any command prompt or from Windows Explorer. The region name is prompted for and the resultant data collection directory should be zipped up and attached to an incident.

The Unix script is called mfesdiags.sh (available from support) which should be transferred to the Unix machine and given the ‘execute’ permission (e.g. chmod +x mfesdiags.sh). The script should be run in the COBOL/Enterprise Server environment. It will prompt for the relevant region name for which to collect diagnostics and will generate “mfesdiags.trz” which should be attached to an incident.

Both the Unix and Windows scripts invoke the relevant ‘mfsupport’ utility which collects information about the OS and machine hardware along with details relating to Micro Focus products installed.

As a minimum (e.g. if MFESdiags is not used to collect the data) the following items should be collected and provided as soon as possible after a failure (which can be simply achieved by zipping or ‘tar’ing the contents of the system/region directory):

- console.log
- log.html or log-*.html (for communications problems)
- Journal log (for MFDS/security problems)
- Any casdump or aux traces
- mfSupportInfo (on Windows)
- mfsupport (on Unix)

However the MFESdiags data collection script will collect all the above and other useful information details as follows:

- All files from the Enterprise Server's System/Region Directory, including :
  - console.log and console.bak The communications process log log.html or log-*.html
  - Any trace diagnostic datasets, casauxta.rec and casauxtb.rec
  - Any system abend dumps (casdumpa.rec, casdumpb.rec or casdumpx.rec)
  - Any HSF output files (*.csv)

- The output from mfSupportInfo (Windows)/mfsupport (Unix) – contains product and system info.
- The Resource Definition File (RDO/RDT), dfhdrdat.
- The Directory Server (MFDS) log file
- The Directory Server configuration directory.
- Additional configuration files (if they exist) including:
  - mf-server.dat
  - mf-client.dat
  - CTF Trace config file and CTF trace files from the configured location
  - COBCONFIG (core_on_error) config file
  - EXTFH file handler config file
- A list of system processes running on the machine.
- A list of network connections and listening sockets using netstat.
- Windows Service information for MFDS (on Windows).
- Windows System and Application Event logs (on Windows).
- A list of shared memory areas and semaphores (on Unix).
- A list of open files (on Unix).

The MFESDIAGS script will collect CTF trace logs if the region is configured to generate CTF trace. See the CTF TRACE section for configuration details.

4. Reporting a Problem

This section describes best practices in providing information when a problem occurs and an incident needs to be raised.

4.1 How to Report a Support Case

If you cannot resolve your issue using Self-Help resources, and you have an active Business Support agreement for your product, you may open a support case with Customer Support. For low and medium severity issues, open the case on the Support Portal. For high severity issues, call us at the telephone numbers referenced in the section below.

Submit or Manage Support Cases on the Support Portal

After logging into the Support Portal, select “Cases” from the top menu top open a new support case, or to manage existing cases. For new product questions or issues, select “Technical Issue”. When reviewing existing support cases, you can use settings and filters to adjust how your lists of support cases are displayed.

Telephone Numbers

Go to the Contact Support page and select your product to see the local telephone numbers for Customer Support.

When creating a case, please:

1. Give the incident an informative short description (see Providing an Informative Short Description)
2. Attach all relevant information that might be useful in supporting issue diagnosis, what this includes will depend on the type of problem that you are encountering but in general should include the following:
   - The MFESdiags output as described above
   - Step–by–step instructions to reproduce the error
   - Any sample programs that can be used to reproduce the problem – ideally provide a zip file containing everything necessary to build any program(s)

- Any directives used if there are problems with compilation
- Note any Micro Focus consulting FDS or other utility that is being used and attach them to the incident if possible

4.1.1 Providing an Informative Short Description

Include explicit and concise information in the Short Description to make it specific rather than general, for example write:

> "Source won't compile, getting error code ###"

or

> "Enterprise Server shows error code ### during restart"

rather than a more general (and less useful) description like:

> "Error with source compile"

or

> "Enterprise Server is crashing"

4.2 Different Types of Incident

We recognise that there can be different types of problem that occur with Micro Focus products, in this section we will address 5 particular types – there are undoubtedly more but hopefully this advice will cover the majority. The 5 types that we concentrate on are:

1) Compiler failures
2) Debugging failures
3) Reproducible run time failures
4) Unpredictable run time failures
5) Performance issues

4.2.1 Compiler related incidents

For this type of issue the minimum that we request you supply is:

a) A cut down of the particular piece of code that is failing, some compiler failures can be related to size of working storage etc. so if it is not possible to cut it down supply the full program, as a tip, if you compile to .lst and supply this to us then this will automatically include all copybooks and show the directives being used.
b) If you do not supply a .lst file then please supply the directives files that you are compiling with
c) Detail the failure symptom or mode

d) Please explain or reference any supporting material if the issue relates to code that you believe should compile successfully, for instance under a particular standard or model.

4.2.2 Debugging Failures

If you encounter a failure during a debug session in the IDE then we would suggest the following as the minimum to complete a high quality incident report:

a) A copy of the source code that shows the failure, cut down as much as possible – again as with compiler issues you may want to supply a listing (.lst) file
b) You may also want to take the run time diagnostics of the failure using the information in the “minimum recommended diagnostic settings” section – and pulled together using MFESdiags
c) Associated debugging files including the IDE CTF trace files and the Java ‘hot spot’ log (for the Eclipse IDE) - see the “Enterprise Developer IDEs” section to configure CTF trace as appropriate.

4.2.3 Reproducible run time failures

If you have a failure at run time that is consistently reproducible then we suggest that when reporting the incident you include the following:

a) The code necessary to reproduce the failure – this should be cut down as much as possible
b) Any data needed to reproduce the failure
c) The details of what the failure actually is, the run time error message, a corruption or the code is not behaving as you expect
d) Set the appropriate first point of failure diagnostics, referring to the “minimum recommended diagnostics” section in this document, adjust them based on your own judgement as to what you believe will help us to understand and diagnose the problem – for instance if it is a filehandler problem refer to that section for possible additional tracing, if a memory corruption seems to be occurring consider the memory strategy settings
e) Collect the output from the run using the MFESDIAGS script
f) Report the problem to Micro Focus, upload the diagnostics captured by the MFESDIAGS script using the naming conventions recommended.
g) We would also be happy to hear from you any hypothesis or theory that you have that may help us address the failure more quickly

4.2.4 Unpredictable run time failures

When tuning production systems or during their implementation failures can be seen that do not occur with any predictable regularity or under definable circumstances. It is these types of failures that require the first point of failure diagnostics to be captured and reported to Micro Focus along with full context and detail of the failure.

We recommend that all systems be run, during implementation or in production, with the minimum diagnostic setting configured as set out in the relevant section. Other diagnostic settings should be configured dependant on the failure scenario, symptoms and area of technology. This document can be consulted for such information as to what is available, for instance for communications tracing, Fileshare/filehandler tracing and other areas.

When the failure re-occurs you should use the MFESDIAGS script to gather the diagnostic files and report the incident to Micro Focus, uploading these diagnostics following the naming convention set out in this section.

The long description should particularly detail as much context and detail as possible including chronology and timeline. For further information please see the "Important Incident Considerations" section.

4.2.5 Performance Issues

A lot of projects identify performance issues at some point and occasionally slow-downs can occur in production systems.

These are actually 2 types of issues and should be approached differently.

For performance testing during implementation we recommend that the following protocol be followed as closely as possible:

1) Define what acceptable performance is, this can be 'the same as previous platform (maybe for example that on the IBM mainframe), as fast as a set of particular users find acceptable or some other criteria, this gives you a target to aim for and to measure how close you are to being complete.
2) You will need to identify a representative benchmark program or execution scenario
3) Create a spreadsheet, in this spreadsheet, for each run of the benchmark you record:
   a. The actual benchmark program that is run for this test – you should run a 'baseline' and this is the start point for tuning hardware, compilations, execution setup etc. and measuring the impact.
   b. The results, maybe elapsed time, I/O time or whichever types of data you want to measure
   c. The variables that are adjusted for each run, maybe the programs are compiled with different directives, maybe the hardware setup is adjusted, maybe the database is moved from a remote machine to a local machine.
   d. Any comments or observations on each run, did it improve performance or not? Did it seem to show up some type of anomaly or restrictions in scaling etc.?

   Please see the performance section in the Appendix where there is further information and some hints and tips on measuring and improving performance.

   For performance slowdowns that occur post-production the obvious question to ask is 'what has changed?' This may be upgraded hardware – we have seen that adding hardware power does not always improve performance, indeed it can slow it down if the architecture or system setup is not configured to take advantage of it, if nothing has seemingly changed then look for whatever else may be running on the system at the time that the slowdown occurs, we frequently have customers find that anti-virus software or mass searches are being done on a machine when performance slows down.

   Another very important aspect of performance to consider is the virtual machine environment if it is part of the setup, Experience has shown that running with a VM can add up to 20% overhead to performance from straight hardware, it is very important to employ a VM consultant to look at the configuration and setup of the VM that is in use to run the Enterprise Server environment.

4.3 Managing the case

Reply to the case within the Web Site with your Web Id, so that any answers are logged right away and the case owner is aware you've replied to the questions sent to you.

Remember to continue to give detailed information and background regarding new situations that are being reported or when responding to requests for further tests/information.

4.4 Important Case Considerations

High quality initial case reporting can considerably shorten the time that it takes to a get to a solution. A high quality case should have:

- An accurate short description of the symptom
- A long description that is full and complete, it should cover the circumstances and context of the failure
- As much context and detail as possible, if you are unsure whether it is relevant, provide it anyway
- Important details such as chronology, symptoms, configuration, workload levels and 3rd party software involved

Always include relevant diagnostics on the initial report.
A good set of diagnostics is:

- Uploaded to the MF fts site using the your login credentials on the case (see "Naming Conventions for Diagnostic Collection")
- Complete – includes traces, dumps, console and other relevant logs
- From the same failure – submitting a console log from Tuesday, a trace from Wednesday and a dump from Thursday does not help as none of the process numbers or details will match.
- Inclusive of specific failure details (probably in the long description) with particular attention to the date/time of the failure, when it started, what symptoms it demonstrated, did it recover or was manual intervention required?

If you believe that a problem reoccurs on a system:

- Always capture a new set of diagnostics – multiple sets may help us see 'patterns'
- Update the existing case number that the problem was reported on.
- Add comments to the case detailing the date/time of the new failure, why you think it is a reoccurrence of the same problem and as much other context on this new failure as possible, including the similarity with previous failures.
- Supply the diagnostics to MF using the naming convention for the diagnostics collections and/or other files as detailed below

4.5 Naming Conventions for Diagnostic Collection

When MFESdiags is used, the data collected will be in a folder or zip file with a name containing the date and time that the collection was done, e.g.

2013-06-05_14-49-36-42-snapshot(.zip)

Please ensure that this convention is carried through so that any additional data provided is identified in the same way.

The case number should also be included in the diagnostics collection filename whenever information is uploaded to the FTS site.

This naming convention is particularly important when multiple failure data collections are taken for the same case and problem - sometimes it can take several failures to help in pattern identification complex problem resolution.

PART 2 – DETAILED CONFIGURATION

5. CONFIGURATION OF ES-SPECIFIC LOGS, DUMPS AND TRACES

The Enterprise Server product allows multiple regions to be configured (or imported), for example there may a region for CICS and a separate one for Batch (i.e. JCL). Diagnostic generation and collection is generally done for a specific individual region. The main exception to this would be MFDS as this acts as a single repository for all regions configured on this Enterprise Server machine.

In the descriptions below, the term ‘Enterprise Server’ is used to mean a single region configured in ES.

5.1 ES SYSTEM/REGION DIRECTORY

This directory is where most of the information that is required to be captured in a production failure exists for a specific region. This directory should be treated with high priority in terms of maintenance and backup.

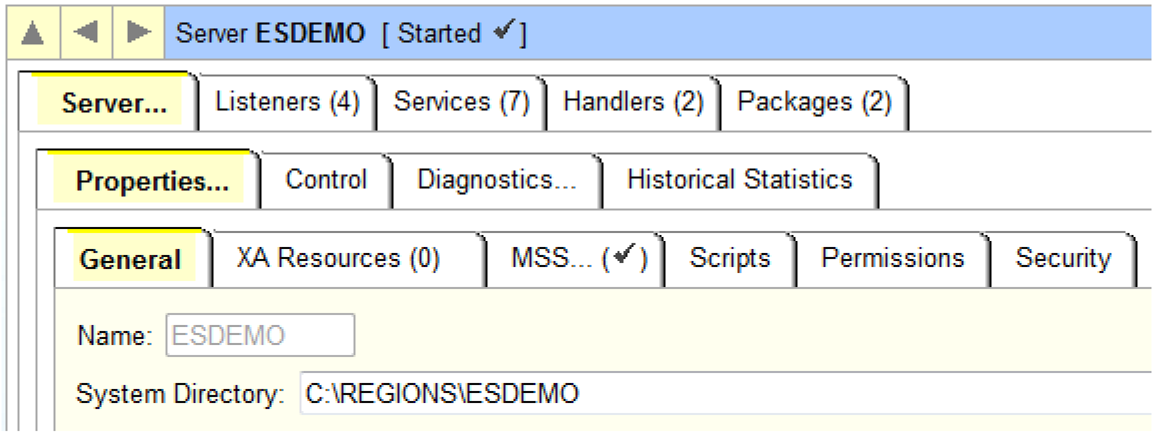
Each Enterprise Server region writes log, dump and trace information to files in its System Directory. The default location for the System Directory in Windows is usually in the ‘Documents’ folder, e.g.:

%USERPROFILE%\Documents\Micro Focus

Under Unix the default is: /var/mfcobol/es

You can change the location of the System Directory by editing the Enterprise Server's definition in Enterprise Server Administration or running the casperm.sh script on Unix

Server -> Properties -> General -> System Directory:



Name: ESDEMO

System Directory: C:\REGIONS\ESDEMO

5.2 ES CONSOLE LOG

An enterprise server logs messages to a file, **console.log**, in its *System Directory*

When an enterprise server starts, it renames any existing console.log file to console.bak, erasing any existing console.bak file. It may be necessary to review console.bak if a region was restarted before logs were collected.

The console log contains an easy-to-read history of Enterprise Server system activity leading up to the problem.

No specific configuration is required to activate console.log as it is always active, however ES Admin Console can be used to configure attributes of the console log such as max size before being backed-up to console.nnn (0=unlimited size, single log) and whether old console.nnn logs are purged on startup.

5.3 MFDS Journal Log

The file "journal.dat" contains information related to the MFDS process. The information in this log will include region startup operations and security information.

A text file version can be created from the current log using the command:

```
mfds –t
```

(The output of this command shows where the text file has been created)

The text file "journal.txt" contains entries in reverse-chronological order (most recent entries are at the top)

It is also possible to view (a text version of) the MFDS journal 'on-the-fly' from the ES Admin console by clicking on the 'journal' button in the left-hand pane

You can change the maximum file size, the number of entries displayed in the GUI and the level of logging in the 'Options -> Journal' page under 'Configure' in ES Admin. When the journal reaches the maximum file size set, the most recent journal entries overwrite the oldest journal entries. For troubleshooting purposes (e.g. when additional tracing is enabled) it is advisable to increase the maximum file size to ensure useful entries are not overwritten (e.g. to 1024 kb)

The file can be deleted from the 'journal' display page in ES Admin Console. This is useful if you are trying to capture a startup problem for example and are no longer interested in what happened before this time.

The three levels of logging available are:

- Errors only
- Errors and warnings
- All information

For troubleshooting purposes, you might want to set the maximum level of logging.

In addition, the 'CTF Trace' option can be checked to allow journal entries to be written via the CTF trace facility. When this is enabled, the same MFDS journal entries are written via the CTF trace facility to a file called "mfds.journal.ctf.txt" this file will have entries in forward chronological order (i.e. with the newest entries at the bottom).

For the trace to be activated, a ctf.cfg file must exist for the MFDS process (i.e. in the environment in which MFDS is started) with at least the 'level' set e.g.:

```
mftrace.level=info
```

The file containing CTF trace entries for MFDS will be located where MFTRACE_LOGS (or mftrace.emitter.textfile#location) points to but by default will be in the directory where the MFDS directory is, e.g.:

- Windows: %ALLUSERSPROFILE%\Micro Focus\Enterprise Developer
- Unix: $COBDIR/etc

5.4 Communications Process Log Files and Listener Tracing

The Communications processes (MFCS) will log messages to the communications process log file, log.html, in the *System Directory*.

If multiple communications processes are running, the date and time stamp at the beginning of each message is followed by an instance identifier in square brackets, for example [2].

By default, there is only one log file which grows continuously until it is deleted by an administrator, however in the newer ED products multiple log files are configured so that as they reach the pre-determined size limit a new log file is started (see MFCS Log File Control regarding configuring rotating log files)

MFCS has a number of options for additional tracing, most of which write messages to the MFCS log.

Trace settings are all case-insensitive, though destination filenames will be case-sensitive on any case-sensitive filesystems. Unless otherwise specified, trace messages go to the MFCS message log (log*.html).

Most connectors will pick up changes to their trace settings while MFCS is running, though some will only apply the new settings to new conversations.

5.4.1 MFCS Trace Settings

Trace settings can be set in the Configuration Information for the main Communications Process (MFCS):

[listeners] section

Logging=<integer>
    For SSL information use logging=3

[**MFCC]** section
  Trace=*n*
    Enable MFCC (Micro Focus Common Client) tracing in MFCS.
    n is the standard MFCC trace-level value.
    0 disables MFCC tracing; higher values produce more output. Currently the highest useful value is 10.

    MFCS uses MFCC during startup to verify that its control channel is working, for some internal processing and for CICS Web Interface (CWI) client operations.

    If MFCC tracing is enabled in the MFCC configuration file (see below), you can set this option to “0” to disable MFCC tracing in MFCS. Alternatively you can set it to a non-zero value to diagnose problems during MFCS startup.

    This can generate a lot of repeated output in the MFDS log file. A value of Trace=1 results in some additional lines at startup eg:

```
[1]  MFCS running in directory D:\cci\ctfTXT, in Enterprise Server mode</li>
[1]  MFCS load location information: main at 00401010, GkProcessConfiguration at 0040165E,
GkOSInitialize at 004016B8, GkUtilityInitialize at 004016D0</li>
[1]  MFCS server &quot;ES1&quot; running as process 356028</li>
```

[**CWI**] section
Tracing=n
Trace CWI operations.
0 disables tracing; higher values trace more information.
Currently 4 is the highest useful value.

[**Async**] section
Trace=n
Trace network async-receive operations.
0 disables tracing; higher values trace more information. Currently 3 is the highest useful value.

[**Tasks**] section
Trace=n
Trace task queuing and dispatching.
0 disables tracing; higher values trace more information.
Currently 4 is the highest useful value and traces Threadpool and TaskMgr (displatch) activity

As well as the above configuration settings (which are accessed via and stored in ES Admin Console/MFDS) the file **mf-server.dat** can also be used to implement MFCS tracing. This would be used when for example there are problems with MFCS trying to retrieve information from the MFDS repository (e.g. if its not able to connect). In this case the following section can be added (see the sample mf-server.dat in the product's bin (Windows) or $COBDIR/etc (UNIX) directory) to allow tracing and diagnosis of the connection problems:

```
[logging]
DSCONTROL=none|standard|all
DIRSVC=none|unusual|processing|not-found|all
```

Where:

DSCONTROL=none|standard|all

Sets the level of logging for control messages. These messages normally come from Directory Server (**MFDS**) though they can come from other sources as well. These messages include server monitor probes, requests for log and dump data, and dynamic configuration changes.

None - Don't log control messages.

standard - Log control messages that update MFCS configuration or status. Don't log messages that just request information, such as the KEEPALIVE probe messages that MFDS uses to check if the enterprise server is still running. This is the default.

All -Log all control messages.

You might want to set this to "all" if you are investigating a problem with the server monitor or other control flows, or set it to "none" if you want to avoid logging messages about configuration changes or other control requests.

Default = standard

DIRSVC=none|unusual|processing|not-found|all

none - Don't log information about **Directory Server** activities.

Unusual - Only log unusual results, such as duplicate object names, which may indicate configuration problems.

processing - Also log some messages about the progress of normal processing. This is the default.

not-found - Also log some messages about searches that return a not-found status. Normally these are not logged because MFCS expects some optional configuration items to be missing if they are not needed. In some cases, though, logging all not-found results may help diagnose configuration issues.

All - Log even more information, including normal MLDAP unbind operations.

Default: processing

Most of the MFCS connector modules (implementing the various different "conversation types" displayed in the MFDS ES Admin console GUI) support some sort of optional trace features. This trace is configured using the ”Configuration Information” text area for a given listener, which is generally accessed from the particular listener’s ’Edit’ button on the ‘Listeners’ tab, as detailed in the following sections.

5.4.2 TN3270 Trace Settings

[**Trace**] section
- Trace=y
  - Enable control trace
- Data=*option*
  - Enable data trace *option* can be:
    - **no**: Disable data trace
    - **yes** or **all**: Trace all data
    - **nvt**: Only trace NVT (non-3270) data
- Length=n
  - If data trace is enabled, trace up to this many bytes (default 40)
- Sem=y
  - Trace semaphore (synchronization) operations
- File=path
  - Trace to this file (as HTML) rather than to the MFCS log
- Capture=path
  - Capture flows to this file, in TN3270Replay format (hex/ASCII/EBCDIC with conversation IDs)

Note that it is possible to enable trace for 3270 with and without the actual data being traced, this is dependent on the ‘Data=’ setting as shown

5.4.3 SOAP (Web Services and J2EE) Trace Settings

This connector is used for SOAP web services, ESMAC, J2EE EJB services, and various internal services used by utilities such as cassub and casfile.
[**Tracing**] section

Trace=y
Enable control tracing.

5.4.4 CTG and ISC Trace Settings

This connector is also used for the P2P protocol (aka Inter-System Communication or ISC, though that term is used for both CTG and ISC). However, P2P protocol trace support is limited; some of the options below are only effective for CTG traffic.

[**Trace**] section
Trace=y
Enable control tracing (trace the operation of the connector)
Data-trace=*option*
Enable tracing at a particular level of detail. The possible values for *option* are:
no: Disable tracing
header: Only trace CTG headers
fmh or **structure**: Also trace CTG structures
all: Trace the entire CTG request/response message
dump: Trace the raw flows as hex/ASCII/EBCDIC rather than parsing and formatting them
Allmax=n
In all data-trace mode, include up to this many bytes of data in the trace output
File=path
Trace to the specified file (as plain text) rather than to the MFCS log

Note that ISC trace can be enabled with different levels of, or no, data being traced, as set by the 'Data=' setting.

5.4.5 IMS Connect (and Remote IMS) Trace Settings

[**Trace**] section

Trace=y
Enable control tracing

Data-trace=*option*
Enable tracing at a particular level of detail. The possible values are:
no : Disable tracing
ll or **length** : Just trace the message length header
all or **data** : Also trace message data
dump: Trace the raw flows as hex/ASCII/EBCDIC rather than parsing and formatting them

Allmax=n
In all data-trace mode, include up to this many bytes of data in the trace output

File=path
Trace to the specified file (as plain text) rather than to the MFCS log

5.4.6 Micro Focus Common Client Tracing

Micro Focus Common Client (MFCC) is the API used for client/server binding for Enterprise Server (where it is used for control-channel probing and CICS Web Interface) and also by a number of other components including ES deployment, CAS utilities (e.g. cassub), COBOL Web Service proxies and the IMTK.

By Default MFCC tracing is enabled in **mf-client.dat** (or the file specified by the **MFC_CONFIG** environment variable if set), which is in the product bin directory (Windows) or $COBDIR/etc (UNIX), however specific applications can override these settings using their own configuration for MFCC trace (see the MFCS tracing above which has an "[MFCC]" section in its configuration area).

When MFCC is used by other programs, its trace level comes from mf-client.dat which is set by adding the following section:

```
[Tracing]
level=n
```

Set *n* to an integer between 0 and 9. Higher numbers produce more output as follows:

0	disables tracing
1	trace high-level MFCC APIs
2	also trace other public MFCC APIs
3	also trace calls to CCI or GK modules
4	also trace (some) private external functions
5	also trace (some) internal functions
6	also include tracepoints within functions
7	also trace returns from CCI/GK modules
8	also trace (some) calls to C library functions
9	also trace returns from C library functions
100	also trace the logging functions themselves

Trace messages are written to the MFCC error log, which by default is the file **mccerror.txt** in the current directory, but this can also be overridden by the application.

5.4.7 CCI Trace Settings

CCI tracing affects anything that uses CCI, which includes MFCS, MFCC, Fileshare, and a number of other components. CCI (Common Communications Interface) is the network-communications component set which continues to be used for network protocols such as TCP, Shared Memory and UNIX Named Pipes for example.

CCI now traces to CTF (in older products it went to a separate binary file). Its traces include information about network I/O.

CCI tracing is primarily used with Fileshare; MFCC and MFCS have their own tracing facilities which usually suffice.

In order to activate CCI trace, it is necessary to set the CCITRACE environment variable and to create/edit cci.ini, as well as activating CTF trace for the CCI component.

```
Set CCITRACE=<log-filename> /f /p /d
```

This will output the trace information to the filename specified (the default is ccitrc1.trc). The specified options will turn tracing on for CCI API, Protocol and Data tracing.

CCI tracing is controlled from the CCI.ini file. This file can be located in %WINDIR% for windows or in $COBDIR/bin for Unix. in which case all processes would be affected by these trace settings.

The CCI.INI file would include:

```
[ccitrace-base]
        force_trace_on=yes
        data_trace=no
        protocol_trace=yes
        internal_net_api=yes
```

A local cci.ini file can also be used (i.e. in the current directory) – this can be used so that only a specific application would be affected by these settings (i.e. when that application is started from this current directory).

The CTF configuration file would include:

```
mftrace.dest=binfile
mftrace.emitter.binfile#Location = path
mftrace.emitter.es#level = 99999
mftrace.level.mf.cci = info
```

(The MFTRACE environment variable would be set to point to this CTF configuration file.)

Note that CCI Tracing can also be used to collect some SSL traces by adding the following to the CCI.ini file:

```
[ccitcp-base]
# Display the negotiated cipher
ssl_display_cipher=yes
# Display details of the peer certificate (if any)
ssl_display_cert=yes
# Display why the certificate content failed its trust test.
ssl_display_cert_fail_report=yes
# Display all details available about the offered certificate.
ssl_display_cert_connection_details=yes
# Switch on or off the SSL options above
ssl_display_options_on=yes
# Where to save the certificate information
ssl_display_destination=/home/diags/tls.txt
```

5.4.8 MFCS Log File Control

Rotating log file can be setup for the MFCS log (log.html) by adding the appropriate entries in mf-server.dat. It uses the ini-file format of section tags in square brackets followed by name=value pairs.

With rotating log files MFCS uses multiple log files, named log-1.html, log-2.html etc.

When a log file reaches a configured size MFCS moves on to the next file (which will be deleted first if it already exists). When it reaches the configured number of log files, it returns to log-1.html, overwriting it.

Note that rotating log files will be the default in ED 2.2 and later releases in the ED product line.

The syntax is as follows:

```
[logging]
FILES=number-of-files
FILESIZE=maxsize
```

To activate rotating log files both files= and filesize= must be set and files= must be set to a number greater than 1. These parameters configure rotating log files, as explained above. If both are set, then the total log file space required by MFCS will not exceed files*filesize (approximately). For example, to limit the log to three files of 1024KB each:

```
[logging]
files=3
filesize=1024
```

The [**logging**] parameters are explained below.

FILES=<number-of-files> - Specifies the number of log files.

FILESIZE=<maxsize> - Specifies the maximum size of a log file in kilobytes (KB)

5.4.9 EZASOKET Tracing

There are various tracing options that can be enabled for a communications process to help trace ezasoket errors and failures. The full list can be found in the documentation here:

https://www.microfocus.com/documentation/enterprise-developer/ed50all/ED-Eclipse/GUID-4917D4CC-96B7-4C8A-828A-156C19E99807.html

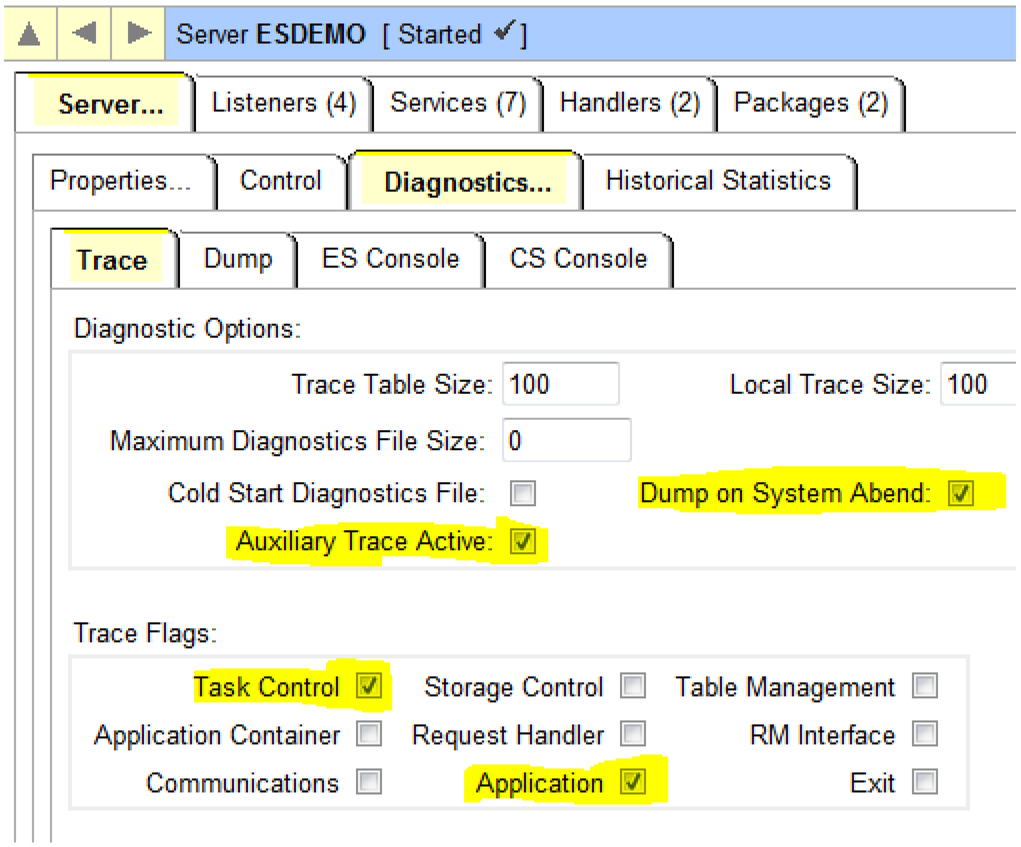
5.5 Configuring a region for Dumps and AUX Trace

This section provides details on how an Enterprise Server region can be configured to generate dump and Auxiliary trace information

You specify settings for traces and internally triggered dumps on the **Edit Server > Diagnostics** page (to take effect at region startup) or in the ESMAC > Control page (for a running system).

As previously mentioned, the following minimum settings for AUX trace and dumps should be configured (from the Diagnostics tab):

- 'Dump on System Abend' – checked
- 'Auxiliary Trace Active' - checked
- The following AUX Trace Flags should be checked:
  - Task Control
  - Application



Diagnostic Options:

Trace Table Size: 100
Local Trace Size: 100
Maximum Diagnostics File Size: 0
Cold Start Diagnostics File: ☐
Dump on System Abend: ☑
Auxiliary Trace Active: ☑

Trace Flags:

Task Control	☑	Storage Control	☐	Table Management	☐
Application Container	☐	Request Handler	☐	RM Interface	☐
Communications	☐	Application	☑	Exit	☐

When 'Dump on System Abend' is enabled, a 'casdump' text file is created in the ES System Directory if/when there is a system abend. This contains information about the entire region including the state/dump of all the processes at the time of the dump. Taking the dump does not incur any performance overhead.

When AUX trace is on, a 'casauxt' text file is created in the ES System Directory containing detailed trace related to the processing in the system. The amount of trace written to this file is controlled by the 'trace flags' which are set in the Enterprise Server region.

Some level of auxiliary tracing should be configured as a minimum in all production systems, however since even the minimum auxiliary trace level can have some performance impact on a running system, each user will need to determine if this is acceptable. Running with the recommended level of trace in a test environment prior to production should allow this aspect to be assessed.

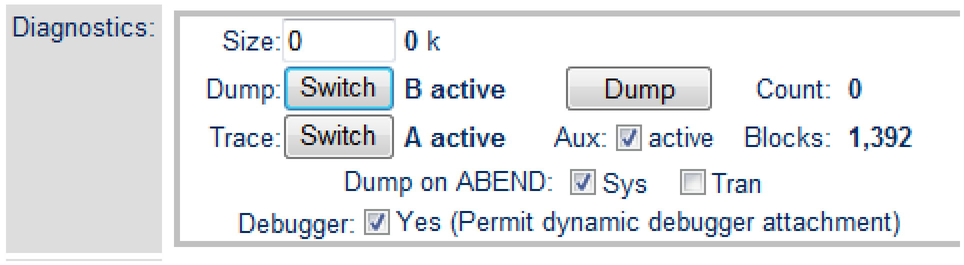
5.5.1 Region Dumps

A region dump captures the state of the Enterprise Server region at the time of the dump. In addition it contains some of the dynamic region configuration parameters so is useful to have for any error situation.

A region system dump is most useful if it is generated at the time of the problem or crash etc. It is recommended that the region is configured to generate such a dump on a 'system' abend.

Additionally the region can be set to generate a dump on an 'application' abend by selecting the relevant flag in ESMAC (see "Enabling Additional Dump Option").

The dump file will be created in the Region's System Directory in a file called casdumpa.rec or casdumpb.rec. By default casdumpa.rec will be used and all dumps will be written (appended) to this file. The alternative file, casdumpb.rec, can be used instead by manually selecting 'switch' for the 'Dump' option ESMAC > control, in which case future dumps will then get written to casdumpb.rec.


Diagnostics:
Size: 0 0 k
Dump: Switch B active Dump Count: 0
Trace: Switch A active Aux: ☑ active Blocks: 1,392
Dump on ABEND: ☑ Sys ☐ Tran
Debugger: ☑ Yes (Permit dynamic debugger attachment)

5.5.2 Generating Dumps Manually

With '**Dump on System Abend**' selected, a (internally triggered) dump will be created when an abend occurs. You can also obtain a (externally triggered) dump that is taken immediately in response to a command. There are several ways of initiating an externally triggered dump:

- Click **Create Dump** on the **Edit Server > Diagnostics > Dump** page
- Click **Dump** on the ESMAC > Control page
- Run the command casdump

5.5.2.1 The Casdump command

This is executed from the product command prompt.

By default casdump attempts to lock shared memory before creating the dump. If Enterprise Server fails in such a way that leaves shared memory locked, the command might hang. To cater for this situation, first run casdump as normal:

casdump /r<Region_Name>

If the above command hangs, you can run it again with the /d option:

casdump /<Region_Name> /d

Using /d means "do not lock shared memory". (Note that this could also cause the command to fail due to shared memory being changed by the running enterprise server while casdump is chasing and dumping storage chains).

Alternatively, use the /b option which dumps all of shared memory as a block at the very beginning before it starts to chase chains and format blocks.

Note that it may be necessary to execute the casdump command in the same user context that started the region in order to generate a dump for that region.

Note that casdumpX.rec is used if casdump is run from the command line with the /d parameter.

5.5.3 Enabling Additional AUX Trace flags

Settings on the Edit Server > Diagnostics page can be changed while the region is stopped and hence persist across region stops and starts (note that any settings changed on the Diagnostics page while the region is started won't take effect until the region is restarted). However settings can also be made to a running system from the ESMAC > Control page and any changes here take effect as soon as they are applied (although they will only persist until the region is stopped).

All the trace flags in the Diagnostics tab are therefore also available from the ESMAC > Control page. The abbreviations used on the ESMAC > Control page are equivalent to the trace flag names on the **Edit Server > Diagnostics** page are as follows (a short description of each trace flag is also provided):

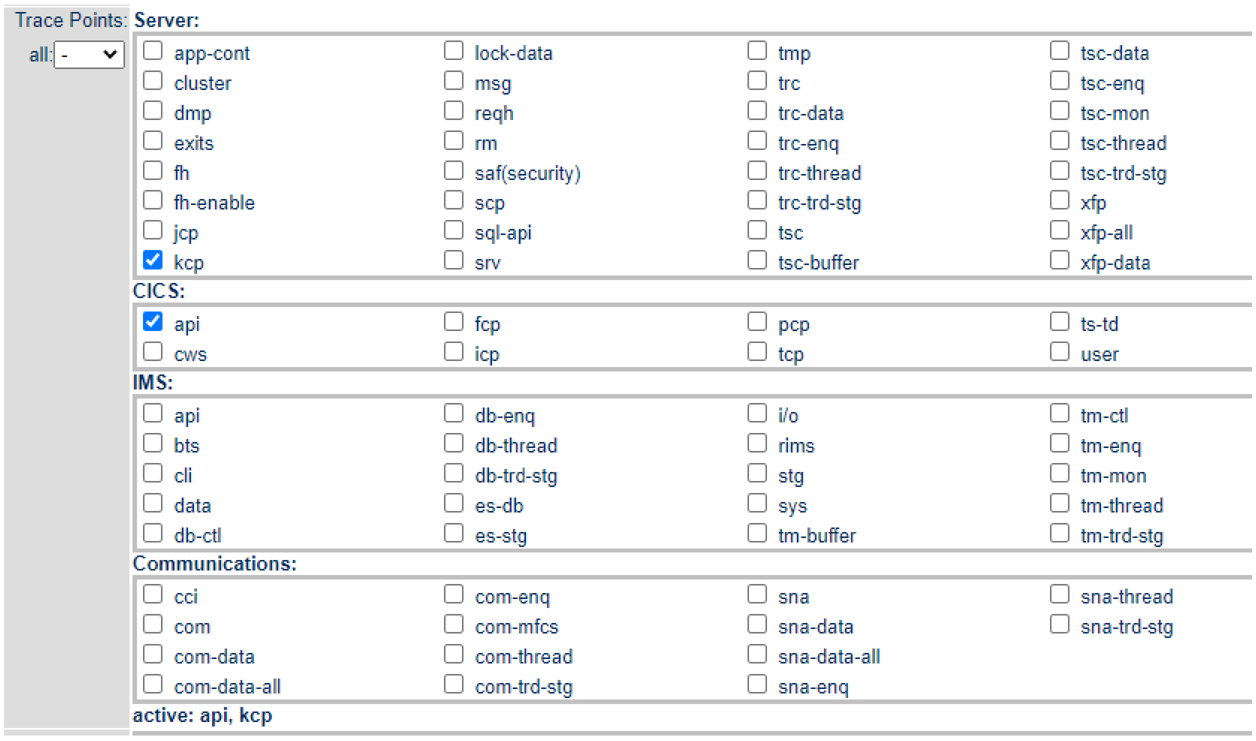
Abbreviation (ESMAC->Control)	**Trace Flag Name (Server -> Diagnostics)**	**Short Description**
KCP	Task Control	Task Scheduling and system management, requests to send or receive (dispatch), Inter-process comms.
API	Application (CICS API calls)	Specifically related to all CICS requests (e.g. EXEC CICS read)
RTS	Run Time System (Application Container)	Cobol Runtime system Trace (output for service-based requests)
SCP	Storage Control	Allocation and release of ES-managed local and shared storage (for memory corruption problems)
TMP	Table Management	Traces lookups (e.g. pct, nqueues). Location of resources such as service and CICS tables.
REQH	Request Handler	Control Request Handler trace for service-based requests e.g. illegal characters in a soap conversation
RM	Resource Manager interface	XA resource manager tracing (e.g. Oracle/UDB/SQL server) all the calls passed to the resource manager
COMMS	Communications	Session information and interactions within the MFCS/casgate interface

EXIT	Exit	trace user and system exit

The AUX trace flags available from the ES Admin Console (Diagnostics tab) are limited to those listed above, however additional flags and settings can be enabled from within ESMAC once the region is running.

5.5.3.1 Dynamically enabling AUX trace flags in a Running Region

Additional AUX trace flags can be dynamically enabled for a running region from the ESMAC > Control page in the 'Trace Points' section:



Trace Points: **Server:**

all: -

app-cont	lock-data	tmp	tsc-data
cluster	msg	trc	tsc-enq
dmp	reqh	trc-data	tsc-mon
exits	rm	trc-enq	tsc-thread
fh	saf(security)	trc-thread	tsc-trd-stg
fh-enable	scp	trc-trd-stg	xfp
jcp	sql-api	tsc	xfp-all
[✓ kcp]	srv	tsc-buffer	xfp-data

CICS:

[✓ api]	fcp	pcp	ts-td
cws	icp	tcp	user

IMS:

api	db-enq	i/o	tm-ctl
bts	db-thread	rims	tm-enq
cli	db-trd-stg	stg	tm-mon
data	es-db	sys	tm-thread
db-ctl	es-stg	tm-buffer	tm-trd-stg

Communications:

cci	com-enq	sna	sna-thread
com	com-mfcs	sna-data	sna-trd-stg
com-data	com-thread	sna-data-all	
com-data-all	com-trd-stg	sna-enq	

active: api, kcp

Any of these flags can be enabled once a region is running. Note that they will only persist until the region is stopped.

Examples of Additional Flags that could be set:

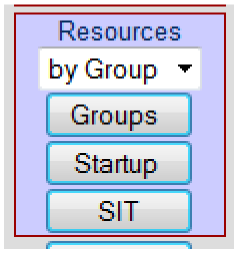
- Server
  - SAF(security) – security trace
  - DMP - dump control
  - MSG – messages
  - SQL-API – SQL calls
- CICS
  - FCP - file control (activity around VSAM file access)
  - ICP - interval control (CICS START requests etc.)
  - JCP - journal control
  - TS-TD - temporary storage/transient data (TS/TD)
  - User
- JCL

- Common
- HSF

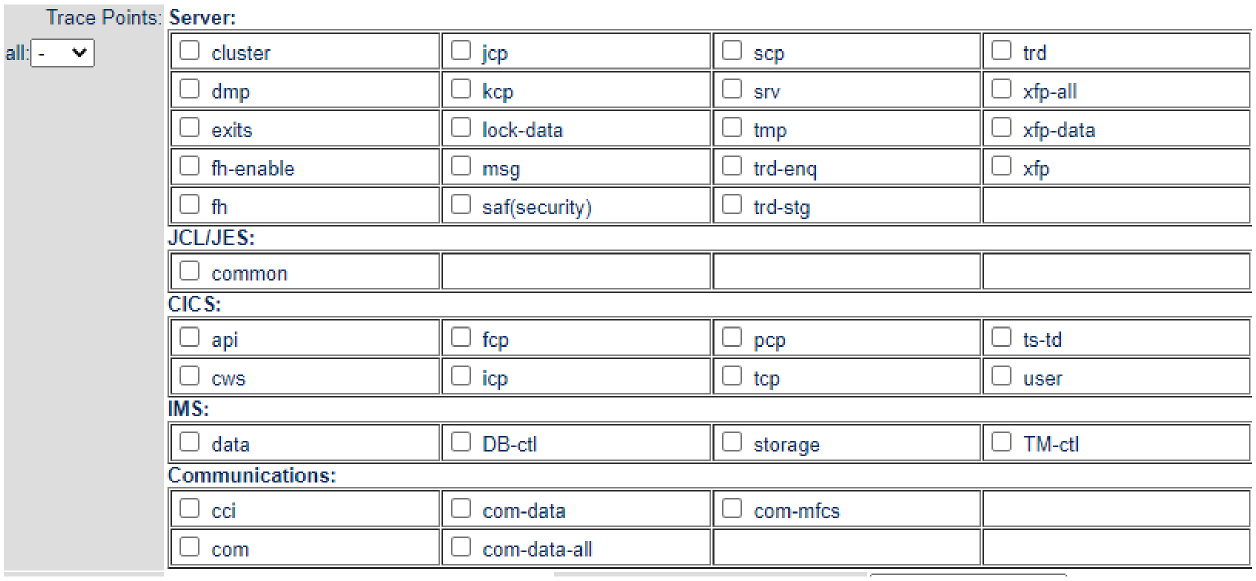
5.5.3.2 Additional Trace flags for Region Startup

Although any additional AUX trace flags selected in ESMAC > Control will not persist when the region is restarted, it is possible to select certain of these additional flags within a specific SIT and this means that these flags can take affect at region startup.

The SIT is chosen from the list presented after clicking on the ‘SIT’ button in the ‘Resources’ section of the left-hand pane (with the drop-down box showing ‘by Group’):



After clicking on the ‘Details’ button for the relevant SIT, the SIT’s ‘Trace Points’ section is available from where specific AUX trace flags can be selected:



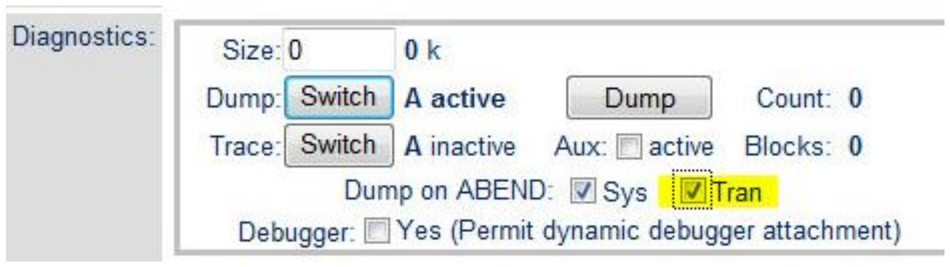
Trace Points: all: -

Server:			
☐ cluster	☐ jcp	☐ scp	☐ trd
☐ dmp	☐ kcp	☐ srv	☐ xfp-all
☐ exits	☐ lock-data	☐ tmp	☐ xfp-data
☐ fh-enable	☐ msg	☐ trd-enq	☐ xfp
☐ fh	☐ saf(security)	☐ trd-stg	
JCL/JES:			
☐ common			
CICS:			
☐ api	☐ fcp	☐ pcp	☐ ts-td
☐ cws	☐ icp	☐ tcp	☐ user
IMS:			
☐ data	☐ DB-ctl	☐ storage	☐ TM-ctl
Communications:			
☐ cci	☐ com-data	☐ com-mfcs	
☐ com	☐ com-data-all		

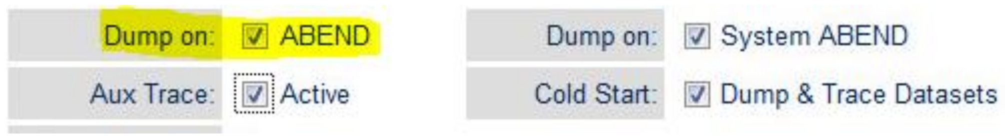
Any flags that are selected in this SIT will then be set to take effect at region startup (i.e. when this SIT is in use). Note that this includes the ‘saf(security)’ trace flag which can be useful to trace security problems during startup.

5.5.4 Enabling Additional Dump Options

As well as the Dump on System Abend that can be set in ES Admin Console it is possible to configure the region to dump on Transaction abends. This is done by putting a check mark in the ‘Tran’ box on the line ‘Dump on ABEND’ in ESMAC as shown below


Diagnostics:
Size: 0 0 k
Dump: Switch A active Dump Count: 0
Trace: Switch A inactive Aux: ☐ active Blocks: 0
Dump on ABEND: ☑ Sys ☑ Tran
Debugger: ☐ Yes (Permit dynamic debugger attachment)

In addition this flag could be set in a specific SIT so that it will be in effect whenever a region (configured to use this SIT) is started as shown below:


Dump on: ☑ ABEND
Aux Trace: ☑ Active
Dump on: ☑ System ABEND
Cold Start: ☑ Dump & Trace Datasets

In 7.0, there is also the possibility to enable Enterprise Server Failure Reporting. This generates a dump and 'readable' failure report designed to help easily identify the error and any pertinent information. More details on this can be found in the documentation (Need link to ES Failure Reporting section. Need to talk to Mark Johnston)

5.5.5 Trace and Dump File Configuration

If required, the size of the Aux trace and dump files can be limited by using the "**Maximum Diagnostics File Size**" setting in the ES Admin 'Diagnostics' tab. When this limit (set in kilobytes) is reached, the active file is switched automatically to the other file.

If there is adequate disk space available then the '**Maximum Diagnostics File Size**' can be left at the default of 0 (i.e. zero, meaning there is no maximum size), however the file can become very big (e.g. if you have a high-activity system) so can be controlled by setting '**Maximum Diagnostics File Size**' to a value in kilobytes. However this dump dataset size should be set such that a minimum of 10 minutes of data can be held within a single trace dataset (so it is dependent on transaction volumes etc.).

In normal operation system processes use a 'trace table' to record trace entries. This trace table is of limited size and when it is 'full' it will simply overwrite the oldest entries in a circular fashion. When Auxiliary tracing is enabled, these trace table entries will be written out to the AUX trace file (casauxta or casauxtb) before they are overwritten.

There are 2 settings for trace tables – one for the general system trace table size (which is shared by the system processes) and another for the 'local' trace table that each SEP has. The number of entries each trace table can hold before it wraps back to the start is controlled by the 2 settings on the Diagnostics tab:

Trace Table Size (number of entries for the system trace table)

Local Trace Table (for each defined SEP – a value of 0 (zero) means that no local trace table is allocated)

The local trace table entries will be flushed to the **aux trace file** (when enabled as above) once the local trace table fills. The default size is 341 entries for both the system trace table and the local SEP trace table.

It may be necessary to increase the size of the trace tables to ensure that trace entries are not lost before they can be written to disk.

If ‘**Cold Start Diagnostics File**’ is checked, the Aux trace and dump files will be deleted at server start (so it is important that these files are captured before a region is restarted if this setting is active).

5.5.6 What is the Performance Impact of Setting Additional AUX Trace Flags?

This is always a relevant question to production systems that require reliable performance and is almost impossible to give an answer to, even an estimate. So much will depend on the configuration of the system, its workload, capacity, the workmix etc and the level of tracing required.

What we suggest is an approach that takes advantage of the dynamic tracing capability within Enterprise Server.

The following approach uses ESMAC to dynamically set trace flags and monitor their effect and is offered as an example, which you might like to adapt to fit your particular circumstance.

The ESMAC ‘Control’ Page is where you can dynamically set or unset the auxillary trace flags whilst the system is running.

The ESMAC ‘Monitor’ buttons (1 and 15) provide a view of the current system performance at short (1 minute) or long (15 minute) intervals. This will therefore give a good indication of whether performance has been affected (i.e. the tasks/sec will reduce).

The procedure is to set one of the trace flags, then check the Monitor short (1 min) interval, and also with users, to see whether there had been any significant performance degradation as follows:

- Set “kcp” on control page and apply
    - Check performance is OK (check the Monitor Short Interval page and with users)
    - Wait and perform further monitoring
- Set “api” on control page and apply
    - Check performance is OK (check the Monitor Short Interval page and with users)
    - Wait and perform further monitoring.
- Set any other specific required/requested flag (e.g. rts) on control page and apply
    - Check performance is OK (check the Monitor Short Interval page and with users)
    - Wait and perform further monitoring
- Repeat for every other required/requested Aux trace flags.

At this stage nothing is written to disk. The trace events can still be seen through the ESMAC SEPs (and Clients) page using the ‘Trace’ buttons.

If performance is acceptable at this stage, enable the ‘Aux’ trace (active) flag on the control page (in the ‘Diagnostics’ section). This will cause the traces to be written to disk to the normal casauxta.rec and casauxtb.rec files. Verify that performance is still acceptable.

Once it has been verified that the system can run OK with these flags, then the ‘api’ and ‘kcp’ traces at least should be set in the Diagnostics tab along with ‘Auxiliary Trace Active’ (as described in the Minimum Diagnostics Configuration section) so that they are in effect after a region restart.

If performance does degrade when enabling a trace flag, then by setting that flag dynamically in in ESMAC while the problem is reproduced would minimize any impact that any reduced performance may have, while still allowing the required traces to be captured.

If at any time you gauge performance is unacceptable you can immediately unset the flag on the control page and click 'apply'.

Note that the size of the AUX trace files can grow quite quickly when the AUX traces are written to disk, see the section Trace and Dump File Configuration for more information (i.e. to ensure that the events that you want are not overwritten).

5.6 HSF Trace

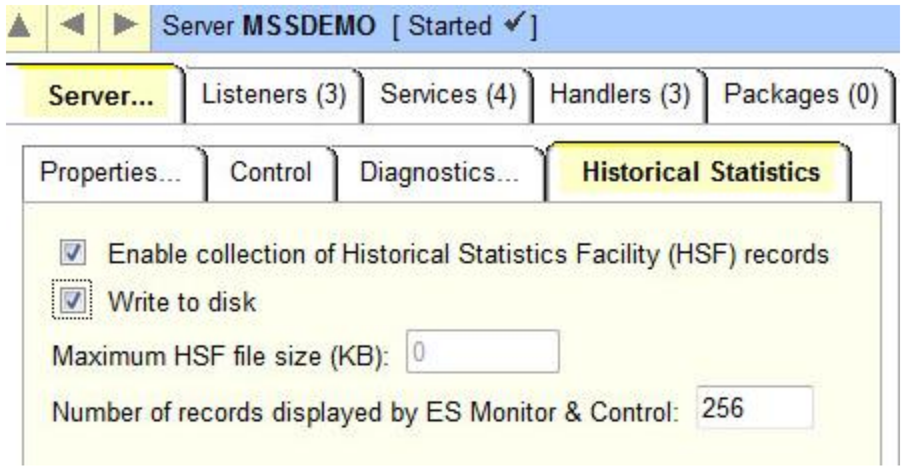
HSF data can be useful in analyzing performance related problems. It can show which transactions are taking most of the time and/or for a particular transaction whether the majority of the time is in the system, API or SQL operations.

HSF creates records for tasks that have been executed and completed and can write this information to .csv files (one record per line) for further external processing, or can be viewed directly in ESMAC (although only the last hours' worth of data in ESMAC).

The recorded information includes:

- Transaction ID/Job Name
- ID of the user that initiated the task
- Response times (Total, API and SQL)
- The 1st 5 files accessed by the CICS transaction
- The 1st 5 TS queues used by the CICS transaction

It can be enabled via the ES Admin Client from the 'Historical Statistics' Tab as shown:

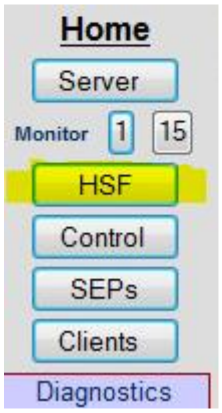


☑ Enable collection of Historical Statistics Facility (HSF) records
☑ Write to disk
Maximum HSF file size (KB): 0
Number of records displayed by ES Monitor & Control: 256

If 'Write to disk' is NOT enabled, the information would only be viewable from ESMAC. However when this option is also enabled .csv files will also be created (in the Region's system directory).

It is also possible to enable HSF in a running region from the ESMAC -> Control page where the same options are presented.

Once it is enabled, an additional 'HSF' tab appears in the left hand pane in ESMAC as shown below:



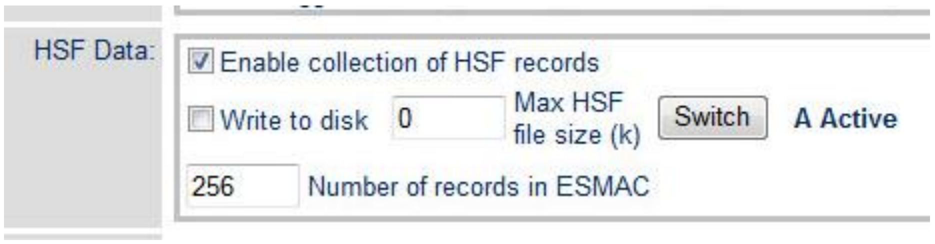
Clicking on the HSF tab opens a window showing statistics from the last hour. In particular the number of transactions and average response time are shown.

Note that HSF Data is flushed to file when this ESMAC 'HSF' button is clicked. Without this action, the CSV file may not contain the latest information.

Clicking on the magnifying glass for a particular entry shows more details for that entry, including the Transaction, time, PID and user and task number

Obtaining the PID in this way can be useful to then explore the relevant SEP trace that ran this transaction etc.

When writing to disk, HSF data is written to cashsf-a.csv or cashsf-b.csv (ESMAC will show which file is 'active') in addition it is possible to 'switch' to the other (A or B) file, otherwise this 'switch' will take place when the 'Filesize' limit is reached (which would be 4GB if no other limit specified):



HSF Data: [✓] Enable collection of HSF records

[ ] Write to disk 0 Max HSF file size (k) Switch A Active

256 Number of records in ESMAC

A backup/archive file is created if the A/B file already exists when a switch occurs (this results in filenames being created with extensions in the range: .001 to .999)

Data in the CSV files can be imported directly into a spreadsheet for further processing.

5.7 ES Feature-Specific Diagnostics

The following describes specific features or aspects of Enterprise Server that may be enabled or used and what diagnostics are available.

5.7.1 Configuration Information for a Region

In order to specify additional configuration information for a region (e.g. environment variables for trace configuration files), they can be entered into the 'Configuration Information' box on the 'General' tab in ES Admin Client.

Note that it is also possible to specify the relevant parameter in the OS environment itself, provided that the region is started from this environment (e.g. from the same command prompt). It is important to note however that if other regions are subsequently started from this same command prompt, they will also inherit these environment variables which may not be desirable (this will also be the case if any environment variables are defined globally in the operating system).

Thus it is generally more convenient and desirable to configure a specific region directly with the required environment variables using the Configuration Information box, in this way no other regions will be affected.

This Configuration Information section must start with the following text/section:

[ES-Environment]

Then parameters can be added to this box (one on each line) to define certain environment variables for this region. This can include some of the trace options as follows:

MFTRACE_CONFIG=

MFTRACE_LOGS=

COBCONFIG= (or COBCONFIG_=)

EXTFH=

IMSDIAG=

It is recommended that at least the MFTRACE_ parameters are set in this region's configuration as this allows for a specific CTF config file to be used for this region (i.e. to investigate a particular problem), while other regions could have different CTF configuration requirements. Furthermore if the MFTRACE_LOGS location is set to the system/region directory then any CTF logs will be collected automatically using the MFESdiags collection script.

5.7.2 Security Tracing

Security doesn't have a dedicated trace file/log but can be configured to output information to other system logs (console.log, MFDS journal and the AUX trace).

Within ESMAC the AUX trace flag 'saf' can be set which will output saf/security trace points to the AUX trace file. This flag needs to be set in a running region. If it is required to have the SAF AUX trace flag on from region startup, it will be necessary to set this in the SIT that is active for this region.

When External Security is configured for a particular External Security Manager (ESM), the 'configuration Information' box for that ESM is used to configure trace information (under the [Trace] section). When this ESM is then used for MFDS, the relevant messages will be output to the MFDS journal log. Similarly if the region is configured to use this ESM (via the Default ES Security setting) then additional trace will be output to the console.log. Note that the Default ES Security setup has its own 'Configuration Information' box which can be used to set parameters to be used by any and all components that are set to use the Default Security manager (which can be any region and/or MFDS). Furthermore each region has its own 'configuration information' box (under the 'Security' tab).

The following is an example of what can be set for security managers using the mldap_esm module:

[Trace]

```
rule=n
groups=n
search=fail
modify=y
Bind=y
```

The available settings are briefly described below

Rule=setting
Enable the logging of the "effective rule" for resource-access decisions.
If this is set to a string beginning with "y" or to "1", the ESM Module will make a log entry for every authorization decision noting which rule was used to make the decision.

Groups=setting
Log various messages regarding the processing of user groups.
If this is set to a string beginning with "y" or to "1", the ESM Module will make a log entry when it determines that a user belongs to a group during Verify, or when it finds a group ACE that applies to a request during Auth.
This is particularly useful when debugging problems with All-Groups mode.

Search=setting
Enable the logging of some LDAP search operations. (Not all searches are currently logged.)
If "fail", a log entry is made for every search that returns an error other than "not found".
If "found", all searches that succeed or return an error other than "not found“ are logged
If "all", "y", or "yes", it will log all search operations, including ones that return "not found"

Modify=setting
Enable the logging of some LDAP modify operations which are normally not logged.
Currently affected operations include setting the last-login-time user attribute and others.

Bind=setting
Enable the logging of LDAP bind operations.
This may be useful if you are investigating an issue with network traffic to the LDAP server.

In addition, very specific ‘filtered’ trace can also be configured using TraceN (where N is a number from 1 to 8). This can be useful on a busy system to only obtain trace information for a specific user and/or operation. EG:

- Trace1=verify:SYSAD:deny
  - traces Verify (signon) requests where the SYSAD user is denied
- Trace2=auth:SYSADM group:TCICSTRN:C*:allow
  - traces Auth (authorization) requests where a member of the SYSADM group requests access to any resource in the TCICSTRN class with a name beginning with C, and the request is allowed

Optional Security tracing features in ESF include:

- In the ESF configuration text area (Security Manager Definition, Directory Server security tab, Default ES

Security, or a region's Security tab):

```
[Trace]

lock level=n
lock trace file=path
lock dump=yn
```

These settings enable tracing thread locks. *n* is a value >= 0; 0 disables tracing and higher values enable more trace entries. Currently the highest value that enables additional messages is 4. If lock level is set, lock trace file must also be set, to the path of the file for trace output. The *yn* value for lock dump can also be set to "y" (or any string beginning with "y") or "1" to enable dumping information about a thread lock when a thread-lock error is detected.

- In a Security Manager that uses the OS ESM module:

```
[Trace]

verify=yn
```

Set *yn* to a string beginning with "y" or to "1" to enable some tracing of Verify (signon) operations that invoke this module.

- In a Security Manager that uses the eTrust ESM module:

```
[General]

trace=n
```

Set *n* to 1, 2, or 3 for progressively more verbose tracing. At level 3, additional information is written to c:\etrustesm.log (Windows) or /var/mfcobol/es/estrustesm.log (UNIX).

5.7.2.1 Security Auditing

The security audit facility is configured with a configuration file (specified with the '-c <filename>' option at startup).

The configuration file contains the following settings:

- mfaudit.dest=AUDITFILE
- mfaudit.emitter.auditfile#location=C:\MFAudit\logs
- mfaudit.emitter.auditfile#collectionsize=3
- mfaudit.emitter.auditfile#file=audit.aud_$(GEN)

(The above creates audit files in the specified directory and ensures that files are number sequentially (GEN).)

The Audit Manager should be started with the '-c' parameter used to specify the configuration file:

```
mfauditmgr [-c cfg-file]
```

For the Audit facility to audit security events, the 'Create audit events' security option should be checked. This is in the Default ES Security tab (in 'Configure Security Options' for MFDS). Security audit events will then be generated when Enterprise Server or Directory Server use this default ESM.

Once audit files have been created/written to (i.e. after some security operations have taken place), the audit files need processing. Use the following command to make the first audit file available for dumping:

```
mfauditadm -p -f audit.aud_1
```

Use the following command to generate an audit report from the dumped file:

```
mfauditadm -r -o audreport.txt -f audit.aud_1
```

The above command creates a file named audreport.txt that contains the audit information.

5.7.3 JCL Tracing (also applies to TSO, MVCATIO, SVC99)

In a batch environment the following information will be required

Console.log

Original JCL (with any includes)

Joblog and Sysouts

SPL*.dat files (These are all the spool files, which are prefixed with SPL*. E.g. SPLDSN.DAT, SPLJOB.DAT)

The console.log is available as described above.

The Joblog (JESYSMSG) is located in the 'datasets' folder and will be of the following format:

Y2020.S1128.S135218.J**nnnnnnn**.D00000.DAT

Where **nnnnnnn** is the job number (from the console.log)

The Joblog can also be viewed from ESMAC as follows:

ESMAC -> Resources -> JES -> spool

Out Hold -> Jobname

JESYSMSG

The SYSOUT file is also found in the datasets folder and has the following name format:

Y2020.S1217.S160553.J**nnnnnnn**.D00006.**SYSOUT**

The file SPL*.dat files will be in the CATALOG folder

Catalog and Datasets location are set in ES Admin Console under the MSS -> JES tab:

System Catalog:

Default Allocated Dataset Location:

In addition to the above, CTF trace can be used to provide more detailed information (see the 'CTF Trace' section)

See the article "Minumum Enterprise Server diagnostics recommentations" for additional help in configuring CTF trace for JCL diagnostics.

5.7.4 IMS Tracing

The minimum recommend IMS diagnostics are to enable the '**API**' flag **under the IMS trace flags** heading (see table below). If performance allows, you should also enable **DB-CTL** and **TM-CTL.**

Specific IMS tracing can be enabled in the Enterprise Server Configuration Information section as follows:

IMSDIAG=222

This will create a .LST file which contains all the IMS calls that have been made and this will also include/enable BTS trace. In order to view the resultant IMS trace it is necessary to run the IMSPRINT command as follows:

MFIMS IMSPRINT BTS-<PID>.LST

Other IMSDIAG values:

IMSDIAG=nnn
1st position – IMS system tracing
2nd position – DB tracing
3rd position – DC/TM tracing

The values in each position can differ, valid values and their meaning are:

0 – no tracing
1 – low level tracing
2 – high level tracing
3 –traces all extfh activity as well (can quickly result in a very large amount of trace output)

For IMS Transactions, you should normally use the IMSDIAG=222 option as noted above.

For BMP use the **BTSCONFG** option within the JCL. This provides better control over the output of the trace (e.g. not including the IOAREA in the output if there is sensitive data). See the 'BTS Trace' documentation for more detail on the various configurations which includes the following sample:

```
//S01        EXEC PGM=DFSRRC00,REGION=4M,
//           PARM='BMP,DEMO001B,DEMO001T,,,,,,,,,CDLI,,N,N'
//REPORT1    DD SYSOUT=*
//IMSERR     DD SYSOUT=*
//PRINTDD    DD SYSOUT=*
//SYSOUT     DD SYSOUT=*
//SYSPRINT   DD SYSOUT=*
//IMSTRACE   DD SYSOUT=*
//BTSCONFG   DD *
SSA
PCB
IOAREA
/*
//
```

In addition, IMS AUX trace flags can be selected from the ESMAC -> Control page as follows:

Trace Point	Description
Api	IMS API calls (minimum recommended)
Bts	Batch Terminal Simulator
Cli	Client processing
Data	Not currently used
db-ctl	DB control processing (recommended)
db-enq	DB lock management
db-thread	DB thread management
db-trd-stg	DB thread storage management
es-db	DB control processing
es-stg	Storage management for MFS processing
i/o	Database I/O
Rims	Remote IMS
Stg	DB storage management
Sys	Not currently used
tm-buffer	TM buffer management
tm-ctl	TM control processing (recommended)
tm-enq	TM lock management
tm-mon	TM buffer monitor
tm-thread	TM thread management
tm-trd-stg	TM thread storage management

It is also possible to set some of these IMS AUX trace flags using the **ES_IMS_TRACE** environment variable as follows:

ES_IMS_TRACE= 11111100000000000000000000000000

The following IMS trace flags can be set in this way:

```
#       rims---------+                +---------db-trd-stg
#        bts--------+|                |+--------db-enq
#        stg-------+||                ||+-------tm-trd-stg
#        i/o------+|||                |||+------tm-mon
#        sys-----+||||                ||||+-----tm-enq
#        api----+|||||                |||||+----tm-buffer
#        cli---+||||||                ||||||+---tm-thread
#     db-ctl--+|||||||                |||||||+--db-thread
#             ||||||||                ||||||||
ES_IMS_TRACE=11111111000000000000000011111110
```

```
Minimum IMS diagnostics (enabling only api and db-ctl) can be setup as follows:
```

`set` **`ES_IMS_TRACE`**`=`**`1`**`0`**`1`**`00000000000000000000000000000`

A dump can also be created by clicking '**/dump TM'** on the ESMAC IMS control page - the results will be shown below this entry field.

5.7.4.1 IMS Message Queue Diagnostics

The IMS-TM message queues summary display in ESMAC is a snapshot of the contents of the message queue file as currently flushed to disk. The display reflects the internal physical structure and organization of the message queue file and is created by sequentially reading the file as a variable-length record file. ESMAC reads the file directly using standard MF COBOL file I/O (i.e. it is not managed by the MF file handler in TM, so TM is not involved in the display).

The message queue file is structurally an MF COBOL variable-length record file format. The content of the file, by record, is as follows:

- Standard MF variable-length record file header
- Record 1: Anchors for message queue objects (held in memory while TM is executing)
- Record 2: Free-space array (held in memory while TM is executing), indicating the maximum number of contiguous free segments in each block
- Records 3 and up: Data blocks, each block divided into 254 248-byte segments, chained together as necessary to hold message data across multiple segments

Message queue file data block records are accessed using available queue buffers. A queue buffer is a 64-KB block of memory that is the interface between the queue file and TM. The queue file is only read in complete 64-KB blocks, i.e. one buffer at a time, but is written according to the size of the data being changed (from a minimum of 8 bytes at a time to a maximum of 64 KB at a time). Disk output is “pushed” at commit/rollback time by telling the operating system to flush its buffers to physical disk. The queue buffers can be thought of conceptually as being similar to virtual memory with 64-KB pages, i.e. TM cannot do anything with an object in a given block (page) unless that block is in one of its buffers. Buffer contents are replaced on a least-recently-used (LRU) basis.

5.7.5 IMTK Service Execution Tracing

5.7.5.1 CICS Service Interface

Tracing for CICS services is configured by setting the Generated trace level, Runtime trace level, and Trace queue name properties for your service interface.

Generated trace and Runtime trace levels are set as follows:

Set the Generated trace level property to specify the level of information you want to generate into your deployed service. This determines the level of trace information available at run time.

These can be set to:

- None - No trace information is generated into the deployed service
- Low - Generate trace code showing the execution of service interface operations, input and output interface fields, and screen records
- High - Generate trace code showing record names and paragraph names as they execute, in addition to the information generated by the Low setting

Set the Runtime trace level property to specify the level of information you want to have written to the trace queue at run time. The options are:

- RUN_LEVEL_1 - No information is written to the trace queue
- RUN_LEVEL_2 - Writes trace records showing the execution of operations and the accessing of screen input and output records
- RUN_LEVEL_3 - Writes trace records showing the names of paragraphs as they execute, input and output interface data, and screen records
- RUN_LEVEL_4 - Writes low-level trace messages from bridge support modules
- RUN_LEVEL_5 - Writes medium-level trace messages from bridge support modules

Notes:

- Because the Generated trace level setting determines the type of information generated into the deployed service, the runtime can only generate messages based on that stored information even if the Runtime trace level is set to write out higher-level messages.
- The reporting for each trace level builds on the lower level or levels, meaning that the higher trace levels also include all trace messages associated with lower levels.
- The file sizes for your deployed service grow in proportion with the level of reporting you specify. Larger files could slightly degrade run-time performance.
- Before redeploying your service for production, set the Generated trace level to None.

5.7.5.2 IMS Service Interface

IMS service execution requires both CTF trace configuration file and the relevant property to be set in the Visual Studio IDE.

IMSolution (Visual Studio IDE trace) requires a CTF configuration file with the following contents:

```
mftrace.level=DEBUG
mftrace.emitter.TEXTFILE#multiproc = true
mftrace.emitter.TEXTFILE#Location=C:\Temp\logs
```

Note: The name of the component when listed in the CTF log output file for IMSolution will be MF.IDE.IMSOLN

The IMS Service interface requires a CTF configuration file containing the following:

```
mftrace.emitter.es#level= 999999
mftrace.dest=textfile
mftrace.emitter.textfile#location=c:\Regions\ESDEMO
mftrace.emitter.textfile#file=ctf.log
mftrace.emitter.textfile#Format = $(TIME) $(THREAD) $(COMPONENT) :$(DATA)
mftrace.level.mer0ctf=info
```

Note: mer0ctf is the IMTK module that implements CTF tracing for IMS Service Interfaces. It can be set to Info, Debug or Error.

This trace is emitted when one (or both) of the properties listed below is set for the service interface in the Visual Studio IDE:

• Generate Trace - writes trace information about the IMS service interface driver program and runtime as the service executes.

This is useful when trying to debug something that has gone wrong with the service interface

• Enable System Trace - traces the executed service interface operation names

When the service fails the final trace entry can help localize the area where the failure occurred. This is useful when failures are first discovered. It writes minimal trace information that allows one to identify the area within the service interface where the failure occurred.

5.7.6 XA Tracing

XA Tracing is required when there are problems with the XA resource interface which includes the initial resource connection attempt.

Any connection failures would be seen in the Console.log as per the following examples:

```
23940 ESDEMO   CASXO0015I a XA interface initialized successfully
```

- this indicates that the switch module loaded correctly

23920 ESDEMO   CASXO0002S Switch Load Module for resource xa failed to load

- you may have specified an incorrect name in the module field

23940 ESDEMO   CASXO0001S XA Resource Manager initialization error. Resource xa open failed, reason -00003 10:29:56

- verify that the open string passed in is correct

In order to trace XA resource problems further, the 'RM' Aux trace flag can be set (available on the ES Admin Client -> Diagnostics tab)

CTF trace can also be used to trace XA resource issues by setting the relevant CTF trace parameter (see the CTF Trace section)

However it may also be necessary to turn to the resource being used itself for additional tracing as shown below.

5.7.6.1 DB2

Error messages can be stored in either the system message log, or the DB2 'first failure' log, db2diag.log.

- On Win32, this file can be located under UDB install dir\DB2.
- On UNIX, the file can be located under UDB instance ID/sqllib/db2dump.

On Linux it will also output XA issues to the Syslog daemon

5.7.6.2 ORACLE

The trace log needs to be setup the open string by using the following additional text:

+LogDir=/usr/local/xalog

(this creates the "/usr/local/xalog" log file)

By default the entry and exit to each procedure in the XA interface will be traced. See Oracle's documentation for more information.

5.7.6.3 SQL SERVER

- Data Access is via OpenESQL (odbc).
- ODBC tracing can be enabled from within the ODBC Administrator Utility
  - Windows (Control Panel -> Administrative Tools), click on the Tracing tab
  - UNIX, set Trace=1 within the ODBC section in the ODBC Configuration file ($ODBCINI).
- Under some circumstances, it may also be worth obtaining OpenESQL Tracing, via the SQL(TRACELEVEL=n) directive. Refer to the Product documentation for more details on this directive.

5.7.7 FILESHARE TRACING AND MONITORING

When the Fileshare Server is active, you can toggle the trace option on and off using the F2 key. If the trace option is on, the Fileshare Server displays the file access requests on the fs console as they happen (note that in this case this information is NOT logged to a file):

F2 = toggle trace on|off

```
ESC = exit
```

When Fileshare is started from the command line the ‘tr’ parameter can be used to enable trace:

```
"start fs /s fsserver /tr f"
```

A Fileshare configuration file can also be used to turn trace on. The default configuration name is fs.cfg – typical contents could be:

```
/s fsserver

/tr f
```

When the trace option is provided on the command line or in the fs.cfg file, trace information will be displayed in the fs console window and will also be written to the file ‘fsscreen.lst’ in the current directory.

The Fileshare Admin utility fsview can be used to toggle the trace options and to monitor other aspects of the Fileshare server and its operations using the command:

```
trace toggle     (Toggle FS trace)
```

Note that FSview console is started simply with the ‘fsview’ command, e.g. on Windows:

```
Start fsview
```

It is then necessary to connect to the Fileshare server, using the name of the server i.e. as provided when starting Fileshare (‘fsserver’ in the above example) as follows:

```
Server set fsserver
```

Then the trace and other commands can be entered, e.g.:

```
server get           (Get server information from the connected server)

files get  (Get list of open files)

users get            (Get list of logged on users)

user get-files       (Get list of open files for specific user)
```

Use the command ‘help’ at the fsview command prompt to list all the available commands.

As well as using trace and fsview to monitor what Fileshare is doing, OS utilities can also be used since the operations involve file access. Using procmon (windows) or ps (Unix) will show what path-string is being used for example (hence any wrong path syntax would be seen)

Procmon Trace example:

```
10:38:23.3920754 cassi.exe 4568 CreateFile

C:\Issues\eza\TESTFCT.DAT PATH NOT FOUND
```

5.7.8 File Handler

XFH trace is enabled in its config file as follows:

```
EXTFH=<path>/extfh.cfg
```

The config file would contain the following entries

```
[XFH-DEFAULT]

TRACE=ON
```

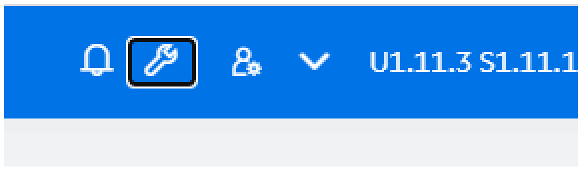
Then it is necessary to use the ‘cobfhrepro’ utility to decode the resultant file as follows:

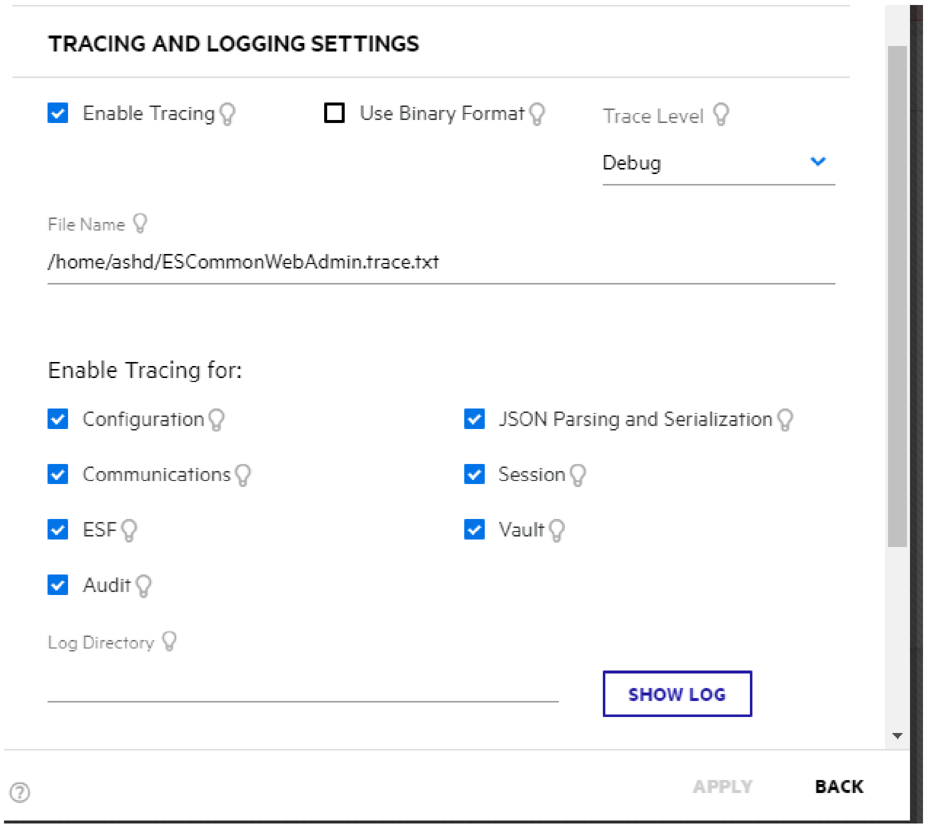
```
cobfhrepro2 /gxfhtrace.xfh /d /j
```

5.7.9 ES Common Web Administration (ESCWA)

In 5.0 onwards there is a new administration interface designed to replace the current MFDS and ESMAC interface, named the ES Common Web Adminitration, or ESCWA for short.

There are some default logs built-in to ESCWA to help diagnose issues within here. The log configuration can be accessed from the spanner in the top right of the screen and then under the Tracing and Logging Settings heading:




TRACING AND LOGGING SETTINGS

[✓] Enable Tracing
[ ] Use Binary Format
Trace Level: Debug

File Name: /home/ashd/ESCommonWebAdmin.trace.txt

Enable Tracing for:

[✓] Configuration
[✓] JSON Parsing and Serialization
[✓] Communications
[✓] Session
[✓] ESF
[✓] Vault
[✓] Audit

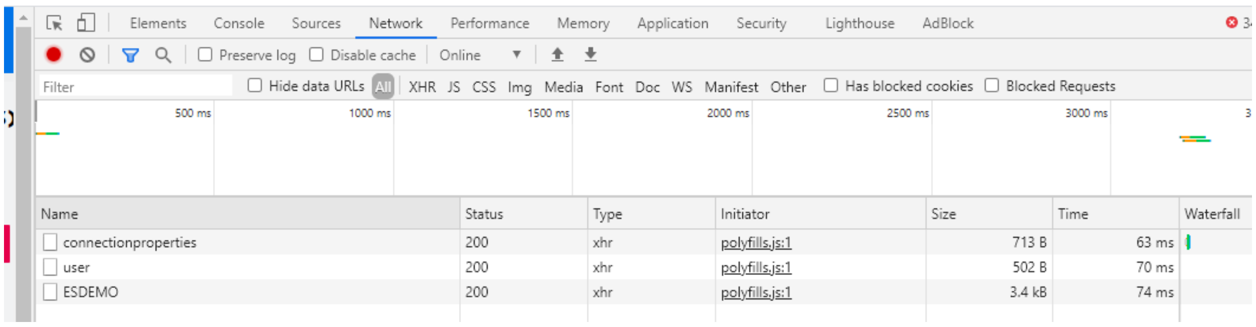
Log Directory:

The ‘Enable Tracing’ option will generate a CTF-style diagnostics file in the location specified in the File Name field, which can then be interpreted using the MF.ES.COMMONWEBADMIN.xml file in the product directory.

The tracing options below this allow you to configure the components that are traced.

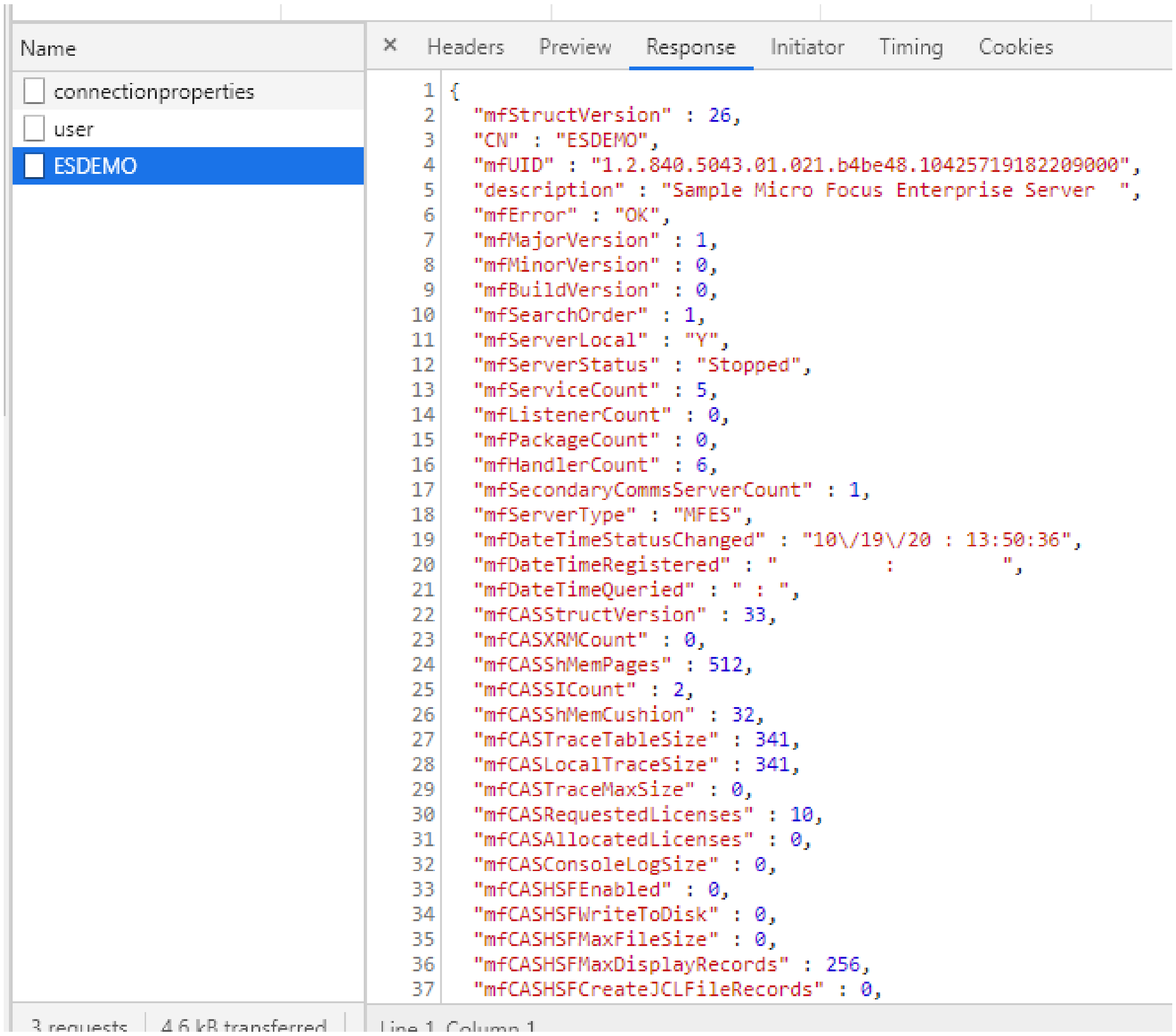
The Log Directory option generates a log similar to the log.html for an ES region and will help diagnose any connection issues or errors within the ESCWA process itself.

A useful trace option for ESCWA issues is to use the Developer Tools feature within the web browser itself to inspect the network requests. Almost all requests within ESCWA are sent/received in JSON format, so errors within ESCWA itself are likely to be caused by malformed JSON. If you have an error within ESCWA that is reproducible, it is advisable to try and capture a network trace and send Micro Focus Support the request that fails. This can be done by opening the Developer Tools within the browser and using the ‘Network’ tab:



Name	Status	Type	Initiator	Size	Time	Waterfall
connectionproperties	200	xhr	polyfills.js:1	713 B	63 ms	
user	200	xhr	polyfills.js:1	502 B	70 ms	
ESDEMO	200	xhr	polyfills.js:1	3.4 kB	74 ms	

Click on the name of the ES region or page that you clicked on that reproduces the error, and choose the 'Response' heading. This shows the JSON response that was sent, and should allow you to identify the particular area of the request that might trigger the error. Please copy this response and send it along with the information you provide in the Support Case.



```
{
  "mfStructVersion" : 26,
  "CN" : "ESDEMO",
  "mfUID" : "1.2.840.5043.01.021.b4be48.10425719182209000",
  "description" : "Sample Micro Focus Enterprise Server  ",
  "mfError" : "OK",
  "mfMajorVersion" : 1,
  "mfMinorVersion" : 0,
  "mfBuildVersion" : 0,
  "mfSearchOrder" : 1,
  "mfServerLocal" : "Y",
  "mfServerStatus" : "Stopped",
  "mfServiceCount" : 5,
  "mfListenerCount" : 0,
  "mfPackageCount" : 0,
  "mfHandlerCount" : 6,
  "mfSecondaryCommsServerCount" : 1,
  "mfServerType" : "MFES",
  "mfDateTimeStatusChanged" : "10\/19\/20 : 13:50:36",
  "mfDateTimeRegistered" : "         :         ",
  "mfDateTimeQueried" : " : ",
  "mfCASStructVersion" : 33,
  "mfCASXRMCount" : 0,
  "mfCASShMemPages" : 512,
  "mfCASSICount" : 2,
  "mfCASShMemCushion" : 32,
  "mfCASTraceTableSize" : 341,
  "mfCASLocalTraceSize" : 341,
  "mfCASTraceMaxSize" : 0,
  "mfCASRequestedLicenses" : 10,
  "mfCASAllocatedLicenses" : 0,
  "mfCASConsoleLogSize" : 0,
  "mfCASHSFEnabled" : 0,
  "mfCASHSFWriteToDisk" : 0,
  "mfCASHSFMaxFileSize" : 0,
  "mfCASHSFMaxDisplayRecords" : 256,
  "mfCASHSFCreateJCLFileRecords" : 0,
```

6. General Micro Focus Diagnostics useful in ES

6.1 CTF Trace

CTF trace allows detailed tracing of specific components in the system. A specific configuration file is required which details exactly what components etc to trace. A trace file will be generated for each component/process traced and the PID can be included in the log filename.

The environment variable MFTRACE_CONFIG is used to point to the CTF config file as follows:

```
MFTRACE_CONFIG=c:\CTFtrace\CTF.cfg
```

This can be set in the OS environment (for when the region is started from the command prompt) or in the region itself (which would override any setting in the OS environment).

The location of the CTF trace logs is determined by the '#location' setting in the CTF.cfg file or, if this isn't set, then by the MFTRACE_LOGS environment variable – which can be set either in the region or in the OS environment (the region setting overrides the OS environment setting).

e.g. for Windows:

```
MFTRACE_LOGS=c:\temp\CTFtrace\Logs
```

Or on Unix:

```
MFTRACE_LOGS=/tmp/ctftrace/logs
```

It is good practice to set the CTF Configuration file in a region's configuration area (i.e. using MFTRACE_CONFIG) rather than in the OS environment so that different CTF config files can be used for different regions. In addition the MFTRACE_LOGS location can also be set in the region's environment configuration area (if '#location' isn't set in the specific CTF.cfg file), again so the CTF trace logs for a particular region will be written to a specific directory for this region

Typical entries in a region's configuration information section could be:

```
[ES-Environment]
MFTRACE_CONFIG=c:\Regions\ESDMO-ctf.cfg
MFTRACE_LOGS=c:\MF\ESDEMO-CTFtrace
```

Or on Unix:

```
[ES-Environment]
MFTRACE_CONFIG=/var/mfcobol/es/ESDEMO-ctf.cfg
MFTRACE_LOGS= /home/mf/ESDEMO-CTFtrace
```

Note that the CONFIG file itself (ESDEMO-ctf.cfg in the above examples) and the MFTRACE _LOGS directory should exist at the relevant locations.

The CTF CONFIG file is a text file containing specific entries on a separate line (lines beginning with a '#' are treated as comments). The configuration parameters for the CONFIG file are described in the online documentation and include the general characteristics (e.g. file size, output format etc) as well as the specific components and elements to trace. However included in the product are sample CTF configuration files e.g. CTFesjcl.cfg, which can be modified to suit different specific situations.

When the location of the CTF trace is to be set in the CTF.cfg file, it can be done for both binary and text files as follows:

```
mftrace.emitter.textfile#location  = C:\MF\ESDEMO-CTFtrace
mftrace.emitter.binfile#location =  C:\MF\ESDEMO-CTFtrace
```

In all cases, the ‘level’ of trace must be specified either globally for all components (using ‘mftrace.level’), or for a specific component. Setting the ‘level’ effectively enables CTF trace.

E.G.: mftrace.level = info
mftrace.**level**.mf.rts = debug

In addition either the specific ‘elements’ of a component to be traced would be set, or the component can have “**all**” its elements set e.g.:

mftrace.**comp**.mf.**rts#all** = true

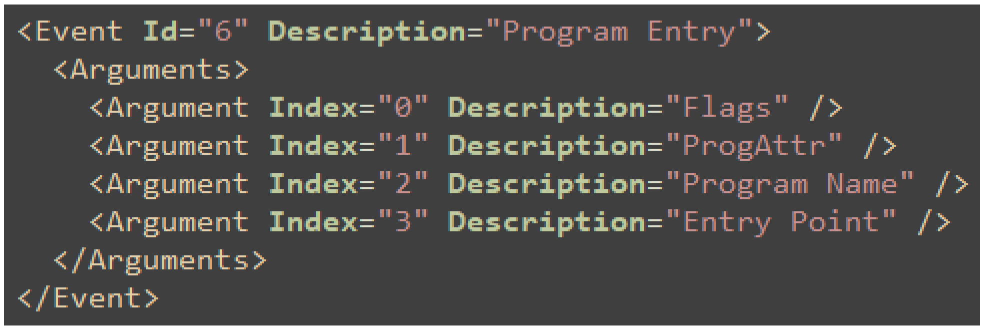
CTF output can be in text (.txt) and/or binary (.ctb) formats and this is specified using the ‘mftrace.dest’ parameter, e.g.:

mftrace.dest=textfile,binfile

Binary format allows post-processing while text format can simply be viewed in a text editor. It is recommended that the CTF configuration file is set to produce binary output (as well as text if required) as the binary file allows for programmatic post-processing.

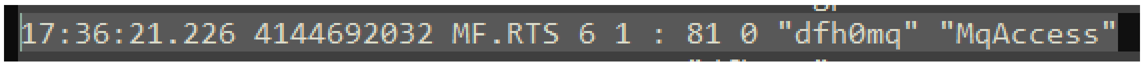
Analysing CTF Trace

CTF in text format is a list of data events that might not make much sense. In the product installation directory, under %COBDIR%\etc\mftrace\annotations, there are XML files that explain what each data item should refer to. Each XML file is named after the component you trace in the CTF configuration file. In general, the format of an annotations file lists the arguments as indexes, and what value that index is corresponds to what that data item refers to. An example of this is below:



```
<Event Id="6" Description="Program Entry">
  <Arguments>
    <Argument Index="0" Description="Flags" />
    <Argument Index="1" Description="ProgAttr" />
    <Argument Index="2" Description="Program Name" />
    <Argument Index="3" Description="Entry Point" />
  </Arguments>
</Event>
```

And the example line in the textfile CTF:



```
17:36:21.226 4144692032 MF.RTS 6 1 : 81 0 "dfh0mq" "MqAccess"
```

You would normally have a timestamp, a thread id and then the component name. After that is the ‘event’, which is 6 here, and the data after the colon is what corresponds to the indexes in the annotations file. So in this example, we know this is a program entry, with flags = 81, program attributes = 0, a program name of dfh0mq and the entry point is MqAccess.

This can sometimes still be difficult to decipher, so instead the preferred way is to generate CTF as binary format. When you format the binary CTF into textfile, or view it in CTF viewer, it uses these annotations file to annotate the entries so you do not need to do the above.

6.1.1 CTF Trace for JCL

As previously mentioned, detailed information relating to JCL operation is available through CTF.

This would include at a basic level:

```
mftrace.level.mf.mvs.mfjcl            = debug
mftrace.comp.mf.mvs.mfjcl#msgs        = true
mftrace.comp.mf.mvs.mfjcl#pgm         = true
```

Then depending on the specific problem being investigated, other CTF flags will be appropriate in addition to the above:

6.1.1.1 Issues with catalog and/or dataset allocation

```
mftrace.level.mf.mvs.mjrm             = debug
mftrace.comp.mf.mvs.mjrm#entry        = true
mftrace.comp.mf.mvs.mjrm#enq          = true
mftrace.level.mf.mvs.mvscatio         = debug
mftrace.comp.mf.mvs.mvscatio#entry    = true
```

6.1.1.2 S806 Abends

```
mftrace.level.mf.rts                  = debug
mftrace.comp.mf.rts#pgmsearch                 = true
mftrace.comp.mf.rts#pgmload                   = true
```

6.1.1.3 Issues with Job Restart

```
mftrace.level.mf.mvs.casspool         = debug
mftrace.comp.mf.mvs.casspool#restart  = true
mftrace.level.mf.mvs.mvsrgdg          = debug
mftrace.comp.mf.mvs.mvsrgdg#entry     = true
```

6.1.1.4 GDG Issues

```
mftrace.level.mf.mvs.mvsrgdg          = debug
mftrace.comp.mf.mvs.mvsrgdg#entry     = true
mftrace.comp.mf.mvs.mvsrgdg#restart   = true
mftrace.level.mf.mvs.mvscatio         = debug
mftrace.comp.mf.mvs.mvscatio#entry    = true
#################################################
```

Note that:

a) General JCL workload management operation (e.g. job submission, dispatch, printing and held SYSOUT) is traced using:

   mf.mvs.casspool

b) JES system catalog information would be trace with:

   mf.mvs.mvscatio

6.1.2 CTF Trace for XA Module

The following parameter should be set in the CTF.cfg file to include ctf trace relevant to the XA module:

```
mftrace.comp.mf.esxa#all=true
```

Note that a sample CTF.cfg file for XA is provided with the product e.g.

```
Enterprise Developer\src\enterpriseserver\xa\ctf.cfg
```

6.1.3 CTF Trace for FileHandler/Fileshare

The following additional CTF trace can be enabled for FileHandler (mffh) tracing:

```
mf.mffh.fs          = error
mf.mffh.xfh         = error
```

6.2 Memory Strategy

Failures can occur in production that are not reproducible, are unpredictable or manifest themselves in different ways. These types of problems can be due to memory corruptions either in shared memory at a systems level or in local memory in specific SEPs. The Memory strategy facility is available to help validate memory usage in local memory in order to try and track down these corruptions. Very often the corruption will occur much earlier than the actual failure occurs, this is because the corruption may not have an effect until that particular area of memory is used at a later time. These corruptions can be caused by application or systems behaviour.

ES Memory Strategy provides a fine-grained control over which processes (CICS SEPs, JES Initiator SEPs, MFCS etc) are checked, whether validation is done for the different processes and how often ES checks for corruption (i.e. just on allocates/frees or after every X tasks as well).

Checking at end of every X task per process provides a method of narrowing down a memory corruption without requiring all processes to perform the validation all the time, keeping the overhead down.

When the check bytes memory management strategy is in place, and the run-time system receives a request to free a memory block, the run-time system checks that the check bytes have not been corrupted, and then releases the memory.

If the run-time system detects a corruption, it issues run-time system error RTS252 indicating the memory block address and type of memory that has been corrupted.

The validation memory management strategy causes memory allocations previously made by the run-time system (using the check bytes memory management strategy) and freed memory to be validated each time the run-time system is requested to perform a memory allocate or free operation. These checks happen on a given memory block for a finite period of time.

If Task Validate Frequency isn't set then memory allocations are (only) checked on any ‘free’ or ‘alloc’ (as above). If this 'frequency' is set to some value, then the memory is (also) checked at the end of every X tasks (where x is the value set for this frequency).

Memory Strategy can be configured from ESMAC -> Control and the following settings can be chosen individually for each of the specific type of processes:

Task Validate Frequency - How often ES will validate memory. 0 is never and 1 always at end of task or step

Retain Free count - How many blocks of storage the RTS will retain on a ‘free’

Validate on Alloc/Free - Asks the runtime to check all memory blocks for corruption every time memory is allocated or freed.

Validate on Container RH - Validate memory prior to and return from request handler functions in the COBOL container

Validate on Container Exec - Validate memory prior to and return from user app execution in the container Web Services and IMS Execute in COBOL Container

Reuse Free - RTS will reuse freed blocks to satisfy new requests

Check Free - Maintain freed memory check array. An array size of 100 blocks is default.

Guard Bytes - Bytes allocated at start and end of blocks. Required to detect corruptions.

This is done from the ‘grid’ in the ‘Memory Strategy’ section ion the ESMAC -> Control page. Settings on one line are for a specific type of process (i.e. SEPs, Batch Initiators, MPRs, MFCS, TMC, DBC and TSC).

Because the memory strategy parameters are set in ESMAC, they will be available in the running region until it is next restarted. If however it is necessary to have a region starting up with memory strategy enabled, this can be effected using the ES_MEM_STRATEGY environment variable, however supportline should be consulted to achieve this.

Note that this memory strategy setting/checking will only check (local) memory allocated by the Cobol runtime, it will not check any shared memory (e.g. allocated with EXEC CICS GETMAIN SHARED)

6.3 Core Dumps for run-time errors (Core on Error)

By default, Enterprise Server does not create a core dump for run-time errors. For example, by default, if the operating system detects a memory access violation caused by your application, no core dump is produced. You must configure the system to produce a core dump for these situations.

Once you have produced a core dump, you can debug the core dump to help determine the cause of the error. Note that this will only show line of failure if the failing module was compiled for debug before the core was generated.

Note that this functionality does not identify memory corruption issues. You can use it to identify application problems only.

To configure ES so that it creates a core dump for run-time errors, you need to create a run-time tunable configuration file, and add the tunable configuration file's details to your Enterprise Server region.

To create a run-time tunable file, create a text file, for example COBCONFIG.CORE.TXT with the following entries for example:

```
SET CORE_ON_ERROR=3
SET CORE_FILENAME="core.%p@%t_%d"
```

Set the relevant config file environment variable (for the platform) to point to this file:

- Unix: COBCONFIG=<path-to-config-file>
- Windows: COBCONFIG_=<path-to-config-file>

Update the Enterprise Server region configuration section with this environment variable setting so the when the region is started core dumps will be generated when run-time errors occur.

Core_on_error settings:

- 0: Never produce a core file on any signal or run-time system error.
- 1: Produce a core file when any system SIGNAL (that would produce a core file) - terminate immediately.
- 2: Produce a core file when any run-time SYSTEM ERROR occurs - terminate immediately.
- 3: Produce a snap-shot core file when any run-time SYSTEM ERROR occurs – process continues as normal (e.g. to display an RTS error message)

Note for LINUX:

CORE_ON_ERROR=3 uses a different mechanism from CORE_ON_ERROR=2 for getting the core and pushing to the location you specify as a filename. On Linux you need to install the "gdb" debugger which in turn provides the "gcore" tool that is required for this. You can (still) use CORE_ON_ERROR=2 and the core.{pid} files will get written to the current directory. To use CORE_ON_ERROR=3 (and the core_filename) then a Linux administrator will need to install the "gdb" package onto the machine. To check whether the relevant package/utilities are installed, enter the following on the command line (to see if it finds the command):

```
gcore
gdb
```

With the idy files that match the cores, you can used "anim mycorefile" and use the debugger to look at why a module crashed.

On Windows when a core dump is generated this needs to be viewed in the debugger. This is achieved by creating an empty project in the development environment, e.g.

Visual Studio:

Click Project > MyProject Properties and navigate to the Debug tab.
Click Start external program and specify the name of the Micro Focus core dump file.
The .idy files should be available for the debugger to show the source.

Eclipse:

Click Run>Debug Configurations…
Create a new launch configuration under the 'COBOL Core Dump' item
Enter the relevant details (coredump locations, IDY files, bitism etc)
Coredump files can be located on remote machines
Click the Debug button and coredump debugging will be started.
The error encountered when the dump was created will be shown and the debugger view enabled.

Note that the compiler directive **IDYSRCFILES** can be used to override the actual paths of main source files contained in the .idy file. This can be used to enable files compiled on one machine to be debugged on another machine and still find all the required src files. The COBIDY, GNT and INT directives would also need to be set.
See the online help for further information on this:
http://documentation.microfocus.com/help/index.jsp?topic=%2Fcom.microfocus.eclipse.help.windows%2Fhtml%2Frhcdircb.htm

If a core file has been generated due to a Micro Focus module error, this can be particularly helpful for Support to identify the problem. Please ensure the core file is sent along with the mfesdiags. If on Unix, it is also helpful to send Support the stack trace from the core file. This can be done by the following, where cassi32 is the name of the executable that the running process crashed with. So for example, if a normal SEP crashed it would be cassi32:
gdb $COBDIR/bin/cassi32 core.filename
This should enter you into the GDB debugger. Then run:

```
(gdb) set logging file gdb-out.txt
(gdb) set logging on
(gdb) thread apply all bt

After this, please send the gdb-out.txt alongside the core file and other
diagnostics.
Note the above is using GDB which is a native debugger to Redhat, so the command may
differ for Windows/Linux variants.
```

6.4 PL/I Trace

In addition to the basic diagnostic collection, specific CTF trace can be enabled for Open PL/i

The following settings form an initial starting point

```
mftrace.level.mf.rts.pli        = debug
mftrace.comp.mf.rts.pli#all     = true
mftrace.level.mf.pli.init       = debug
mftrace.comp.mf.pli.init#all    = true
```

To debug CICS applications the following can be added to the above:

```
mftrace.level.mf.pli.cics       = debug
mftrace.comp.mf.pli.cics#entry  = true
```

In order to 'manage' the amount of trace data collected, the 'level' settings could be reduced from 'debug' to 'info' or even down to 'error' for a further reduction.

7. Enterprise Server Health and Monitoring

When the region is running, the 'Control' tab in ES Admin Client shows how many (CICS) SEPs are running/available for transactions and how many JES initiator SEPs are available for batch jobs. In addition the number of allocated/requested licenses is shown.

7.1 Monitoring the region from ESMAC pages

The ESMAC **Server** Information page provides some statistics of how the system has been performing, including how long the region has been up for and the number of Transactions that have been run together with the maximum number of tasks that can be run at any time.

It additionally shows what the active trace and dump settings currently are, and whether region security is enabled or disabled.

In the **Control** page, the number of available SEPs can be dynamically altered along with other diagnostic parameters etc.

The **SEPs** page shows the running SEPs and their PIDs.

For Standard CICS SEPs it shows the task number and the number of transactions that have been run.

For Batch SEPs the Initiator class and Job details are shown.

Buttons are available to show the active 'Trace' of these SEPs and to 'Stop' a SEP.

The **Clients** page shows any client information including the MFCS communications process and its associated 'Trace' button.

7.2 Checking the Health of the System using MFCS

The MFCS communications process can be used to check the current state of the system which is useful to know when problem related to responsiveness are encountered.

This can be done from a Web Browser as follows:

http://hostname:port/**check**

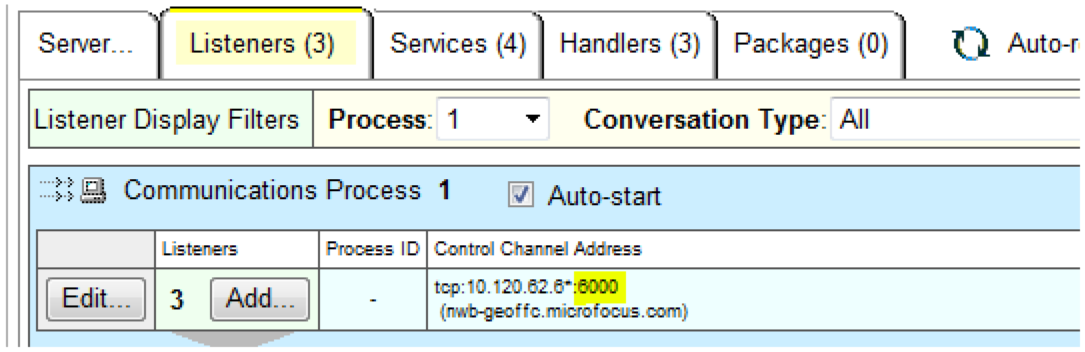
where:

<hostname> is the name of the machine on which Enterprise Server is running

(e.g. "localhost" or "10.123.22.57" or "eth-mach1" etc.)

<port> is the port number that the communications process is running on

The MFCS port number can be found from the ES Admin Console '**Listeners**' Tab under "Control Channel Address" for this Communications Process (it is the number after the colon) as shown below:



	Listeners	Process ID	Control Channel Address
Edit...	3 Add...	-	tcp:10.120.62.6*:6000 (nwb-geoffc.microfocus.com)

If MFCS is working correctly, you should see a response similar to the following:

Server=ESDEMO
ServerType=GKC
Version=1.2.5
Status=Started

In addition the following command will provide statistics on what MFCS is currently doing:

http://hostname:port/**statistics**
(hostname and port as described above)

8. Enterprise Test Server

In addition to the above diagnostics for Enterprise Server, the following are relevant particularly to Enterprise Test Server installations

8.1 XDB Trace

XDB tracing is controlled from XDB.INI

The XDB.ini file for ED will be located in %ALLUSERSPROFILE%\Micro Focus\Enterprise Developer\mfsql\cfg\ on Windows.
The XDB.ini file contains various parameters including the servername (XDBSERVE) and the location of the XDB config directory:

```
[SERVER]
XDBSERVE=<SERVER_NAME>
XSRVCFG=<ED_Install_Path>\mfsql\cfg
```

For XDB Link tracing, add the following lines to the [server] section:

```
[Server]
db2debug=128
```

- generates a file in the mfsql\cfg directory called db2debug.trc

```
TRACEDRDA=1
```

- generates a packet.* file in the mfsql\cfg\trace directory (the trace directory is created if it does not exist once this runs the first time).

X352KeepLog=1

- ensures that if the customer gets a XDB Server violation, the log file is not deleted at XDB Server restart.
- An X352 error is something that should never happen and an X352 should always be reported to Customer Care. An X352 may be intentionally caused by the server (using a divide by zero) to stop the transaction and roll it back when we have detected something has gone terribly wrong. It can also be caused by an inadvertent access to memory that doesn't exist.

Note that by turning on TRACEDRDA=1 or X352KeepLog=1, the log file can grow quite large (e.g. many failures in a short period of time) so the client would have to be responsible for controlling these logs (e.g. by copying the existing X352.LOG to X352.LOG_<Date_Time> every time the machine is booted).

These traces should therefore be turned off once diagnostics have been collected.

The XDB Server must be stopped and restarted, after the xdb.ini information has been added, for these files to be generated.

Client Side Tracing

For client-side tracing, Router and Re-director trace is enabled in the [Trace] section as follows:

```
[Trace]
RouterTrace=c:\tmp\ router.trc
RedirSndRcvTrace=c:\tmp\ redir.trc
```

The Server does not need to be stopped for these client side traces to be generated.

Note: Any SEP that starts when XDB.INI has the above Trace lines in will enable Router and Re-director trace. To avoid doing too much tracing and affecting performance it is recommended that this XDB trace is only configured for the SEP process that will be used to investigate the problem. This can be achieved by modifying the XDB.INI file after the ES has been started, setting the JOB card to use a CLASS that isn't otherwise being used and then starting a SEP manually for that class. In this way only this SEP will pick up the Trace configuration. (It is advisable to comment-out or remove the Trace lines once the SEP is manually started so that no other SEP picks up these Trace configuration settings. Putting any character at the beginning of the line will comment that line out.)

XDBINTRF is the COBOL SQL run time code. It is the layer between COBOL and the XDB client side communications. Router and Redirector traces (as above) would be required to see what is executing in the client's program.

Additional information:

When multiple client side processes need to be traced to determine interactions between them, or when the process to be traced can't be set up ahead of time, it is possible to have each process write to its own trace file. Both RouterTrace and RedirSndRcvTrace may be modified to append the process ID to the trace file using the variables RouterTraceByPID and RedirSndRcvTraceByPID, respectively. For instance, the following section would allow both traces to be turned on and each SEP write to its own trace file:

```
[Trace]
```

```
RouterTrace=c:\tmp\ed2.2\router.trc
RouterTraceByPID=1
RedirSndRcvTrace=c:\tmp\ed2.2\redir.trc
RedirSndRcvTraceByPID=1
```

It should also be noted that when a cursor is opened, both the router and redirector traces will contain the program name which opened the cursor. This can be very useful when trying to assemble a recreate; it is easy to know which programs contain which cursors. Also, the XSTATUS command output will show the options like Isolation Level, Date Format, etc. the client is running with.

For XA tracing the following line can be added to XDB.ini

```
    [Trace]
    XATrace=<filename>
```

Where *filename* is the full path to a file used to collect trace information. Each entry in this file contains a timestamp, the error message generated at that time, and possibly some diagnostic information.
Note that only when an error occurs during XA processing (not an error returned from the XDB server) will anything be written to this file and the error message text can include a ‘reason’ that shows the error is
For instance:

```
XA PID = 7336 : 2011-08-29 15:27:04.247000 :
Error parsing connection string :
LOC = MYLOC, UID=MFIABC, XDBPWD='xxxxxxxx'
Reason = Invalid token XDBPWD at position 26
```

Autobind

See the online help documentation on the ‘Autobind’ SQL compiler directive. This was enhanced (end of 2013) to always produce a log file. It will be located under the project directory along with the DBRM files. The naming convention for the log files is location_name.collection_id.dbrm_name.LOG or if a version is specified, location_name.collection_id.dbrm_name.version.LOG

The log files will contain additional information such as the BIND options provided and the BIND40N commandline generated based on those options.

ECM Trace

See the online docs for compiler directives that enable ECM trace. ECMTRACE.TXT is produced into the project directory. The listing file is in the LISTING directory under the project directory. The trace shows whether ECM generates the correct code.

8.2 RJSE TRACING

The JCL joblog, sysouts and original JCL will be useful for RJSE tracing. In addition, the JCL message that appears in the console.log shows useful information that can be relevant here, e.g.:

JCLCM0199I Program MFJAMS is **non-COBOL ASCII Big-Endian NOAMODE.**

i.e. this shows the "endian-ness", "mode" and the language of the main programs

In addition, the following lists the specific tracing available for Remote Job Step Execution tracing:

- Console.log
  - JCLCM0118E Unexpected error from function RJSECA-FC-Execute RC=00000009 RS=00000222
    - Invalid Password/User (embedded in JCL)
- SET MF_RCCF=[DELETEJOB | KEEPJOB]
    - DELETEJOB – host job logs/sysouts removed when sysouts are downloaded.
    - KEEPJOB - always keeps host job logs/sysouts regardless of sysout downloads.
- JES Spool output
  - Host Step Execution Audit Trail
    - downloads the host job log and spools into in the local spool view under the DD name of %ZJOBLOG
- RJSETRACEFILE=<pathToFile>
  - Logs interactions with Mainframe
- Additional CTF Trace
  - mftrace.comp.mf.mvs.mfjcl#rjse = true
  - mftrace.comp.mf.mvs.mjrm#sysout = true
  - mftrace.comp.mf.mvs.mvscatlu#all = true

8.3 Assembler Tracing and Debugging

An emulation of IBM mainframe assembler language is provided with Enterprise Test Server. This is useful for COBOL applications that rely on some assembler modules for successful testing in Enterprise Test Server. The ability to assemble, link and execute is included. (Note that this is not recommended for the testing of large scale assembler applications.)

Assembler can be traced using CTF trace with the component name 'mf.asm'. For example to trace all Assembler instructions, the following CTF settings can be used:

```
mftrace.level.mf.asm          = debug
mftrace.comp.mf.asm#ins       = true
```

For Assembler programs debugging, it is generally necessary to set the environment variable ES_SERVER_CONSOLE=y before a region is started and for the assembler program to be compiled for debug. Assembler debugging occurs in a command window.

Depending on whether other programs (e.g. COBOL) are to be debugged and whether CICS is involved, slightly different actions may be required as per the following sub-sections.

8.3.1 Debugging a standalone assembler program from the command line

To debug a standalone assembler program from the command line:

- Compile the assembler program for debug
- Use the run command to run the assembler program's .390 file
- The debugger will start in the command window

8.3.2 Debugging an assembler program that is invoked via JCL:

- Compile the assembler program for debug
- In a 32 bit Enterprise Developer command prompt, set the following environment variable
    - es_server_console=y
- In the same 32 bit command prompt, start the server region
- Submit the JCL that invokes the assembler program and the debugger will appear in a command window when the assembler program is executed

8.3.3 Debugging a 3270 CICS or IMS application comprised of entirely assembler programs

To debug a CICS or IMS application that is comprised entirely of assembler programs:

- Compile the assembler programs for debug
- For IMS programs, in a 32 bit Enterprise Developer command prompt, set the following environment variable
    - es_server_console=y
- In the same 32 bit command prompt
    - Start the server region
    - Enter start cassi /r<regionname> /a
    - Respond to the "just in time popup" to select a possible debugger (an IDE instance will open, but the assembler programs won't actually be debugged in an IDE)
- Run the application using a 3270 terminal emulator such as Rumba
- The debugger will appear in a command window when the assembler program(s) are executed

8.3.4 Debugging COBOL and Assembler Programs Together (Non-CICS)

To debug COBOL and Assembler programs together (non-CICS):

- Compile the COBOL and assembler programs for debug
- In a 32 bit Enterprise Developer command prompt set the following environment variable
    - es_server_console=y
- Ensure that the server region is configured to Allow Dynamic Debugging
- In the same 32 bit command prompt, start the server region
- Associate the Visual Studio project with the server region
- Start debugging in the Visual Studio IDE (press F5)
- Run the application in a 3270 terminal emulator such as Rumba
- The COBOL programs will be debugged in the Visual Studio IDE, the assembler programs will be debugged in a separate command window

8.3.5 Debugging COBOL and Assembler Programs Together (CICS)

To debug COBOL and Assembler programs together (CICS):

- Compile the COBOL and assembler programs for debug
- In a 32 bit Enterprise Server command prompt, start the server region
- In Visual Studio, open the project that contains the COBOL code you wish to debug
- In a 32 bit Enterprise Developer command prompt set the following environment variable
    - es_server_console=y
- In the same 32 bit command prompt, enter

    - start cassi /r<regionname> /a

- Select the Visual Studio project that contains the COBOL source to debug from the "just in time popup" that appears
- Run the application in a 3270 terminal emulator such as Rumba
- The COBOL programs will be debugged in the Visual Studio IDE, the assembler programs will be debugged in a separate command window

8.3.6 KNOWN REQUIREMENT FOR RELEASES PRIOR TO ED 2.2:

Failure to set the environment variable *es_server_console=y* prior to executing IMS or JCL assembler programs that are compiled for debug will result in a RTS197 abend when the assembler programs try to execute. Set the *es_server_console=y* environment variable prior to starting the server region.

See Appendix F for more information on using Mainframe Assembler with ES

9. ENTERPRISE DEVELOPER IDEs

The Enterprise Developer product additionally provides for an IDE in which applications can be developed and debugged. The available IDEs are Microsoft Visual Studio or the Open source Eclipse IDE

To debug the Enterprise Developer IDE or the debugger the following options are available.

CTF Trace for the Debugger:

```
mftrace.level.MF.DEBUG.ENGINE.NATIVE = DEBUG
mftrace.comp.MF.DEBUG.ENGINE.NATIVE#ALL = true
mftrace.level.MF.DEBUGCOMMS = DEBUG
mftrace.level.MF.DEBUGCOMMS#ALL = TRUE
```

CTF Trace for the Visual Studio IDE:

```
mftrace.level.mf.ide            = debug
mftrace.comp.mf.ide#vsxprj      = true
```

CTF Trace for the Background Parser in Eclipse IDE:

```
mftrace.level.mf.bgchk          = debug
mftrace.comp.mf.bgchk#all       = true
```

(Often the checker CTF information will also need to be switched on as well)

For Eclipse IDEs, useful information will be found as follows:

- Help/About Eclipse/Installation Details/Configuration
  This provides all the information about the Eclipse environment.

In addition 'hotspot logs' are produced by the Java runtime when the Java runtime crashes – these are located in the directory where Eclipse was started from. The format of the hotspot log filename is as follows:

"HS_err<PID>.log"

Information contained in the hotspot log includes:

Stack trace
Failing module
Register contents (not usually relevant)
Threads
Loaded modules

As well as the hotspot log there is the “eclipse log” this is called “.log” (i.e. no filename) and is located in the eclipse ‘metadata’ directory, e.g.:

C:\Users\<username>\EDforEclipse\.metadata\.log

In order to allow further analysis of such problems, in addition to the normal collection of data, zip up the current project structure and provide this along with full instructions on how to reproduce the problem using this project. Ideally make the reproduction of the problem as simple as possible (by eliminating anything unnecessary that doesn’t affect the reproduction)

10. EDz 2.2 zServer and Workflow Manager

This chapter contains links to the support line URL for the corresponding documentation published with EDz 2.2. The current version is available in PDF only. Please check the online help in later versions of EDz.

For the zServer you can find the *z/Server Messages and Diagnostics* here.

For the Workflow Manager you can find the *Workflow Manager Messages and Troubleshooting* here.

PART 3 – SUPPLEMENTARY INFORMATION

APPENDIX A: SUMMARY OF CONFIGURATION FILES

This section provides additional information that may be useful in order to better understand and use the diagnostics facilities

This is a summary of the configuration files available

File	Description	Configuration Environment Variable/location
ctf.cfg	CTF trace configuration	MFTRACE_CONFIG
fs.cfg	Fileshare configuration	
extfh.cfg	External File Handler	EXTFH
fhredir.cfg	Filehandler redirector	
cobconfig.cfg	Core on error configuration	COBCONFIG (Unix) or COBCONFIG_ (Windows)
mf-server.dat	Comms process log configuration (includes SSL certificate information)	
mf-client.dat	Micro Focus Common Client (MFCC) configuration file	base\bin directory

APPENDIX B: VIEWING THE CONSOLE LOG

When viewing the console log, investigate messages with levels other than information first.

Frequently, the enterprise server will continue to appear normal for some time after a failure so the 10 to 15 minutes before the failure should be investigated in detail (including the information messages).

APPENDIX C: VIEWING REGION DUMP AND AUX TRACE INFORMATION

It is necessary to process the data collected in the *.rec files before they can be viewed.

This is done with the "casdup" utility as follows:

casdup /icasauxtA.rec /fcasauxtA.txt /w

(use "casdup /?" on a command line for additional options)

Note that when sending information to Support the original *.rec files should be provided.

Once the 'casdup' command has been run in the .rec file, the resulting .txt file can be viewed in an editor. It contains the following sections/information:

- Start of storage dump – timestamp
- Server Summary - Trace Flags
- PID of Dump caller process
- Server Control Blocks – Processes and state
- System Trace Table - find entries at relevant time (from console.log)
- Casmgr (MGR)
- Comms server (COM)
- Castsc (TSC)
- SEPs and Batch Initiators
- Local Trace Tables
- Process Memory
- SIT info
- Environment Variables

The dump file can be used to determine detailed activity leading upto the abend/dump.

APPENDIX D: VIEWING CTF TRACE

Binary CTF files (*.ctb) can be viewed using the provided utility 'ctfviewer' – this is available on Windows, and on Linux in the Eclipse IDE. In order to allow the ctfviewer translate/interpret the CTF log entries, the annotations files must be available. These are located in the 'annotations' folder in the product installation directory.

On Windows the following environment must point to the 'annotations' file in the product, e.g.:

```
MFTRACE_ANNOTATIONS=C:\Program Files (x86)\Micro Focus\Enterprise Developer\etc\mftrace\annotations.
```

For the eclipse IDE on Linux, the annotations can be loaded by the CTF item in the Windows/preferences settings pages.

In order to set CTF to generate binary (or binary and text) files, one of the following lines can be included in the CTF.cfg file:

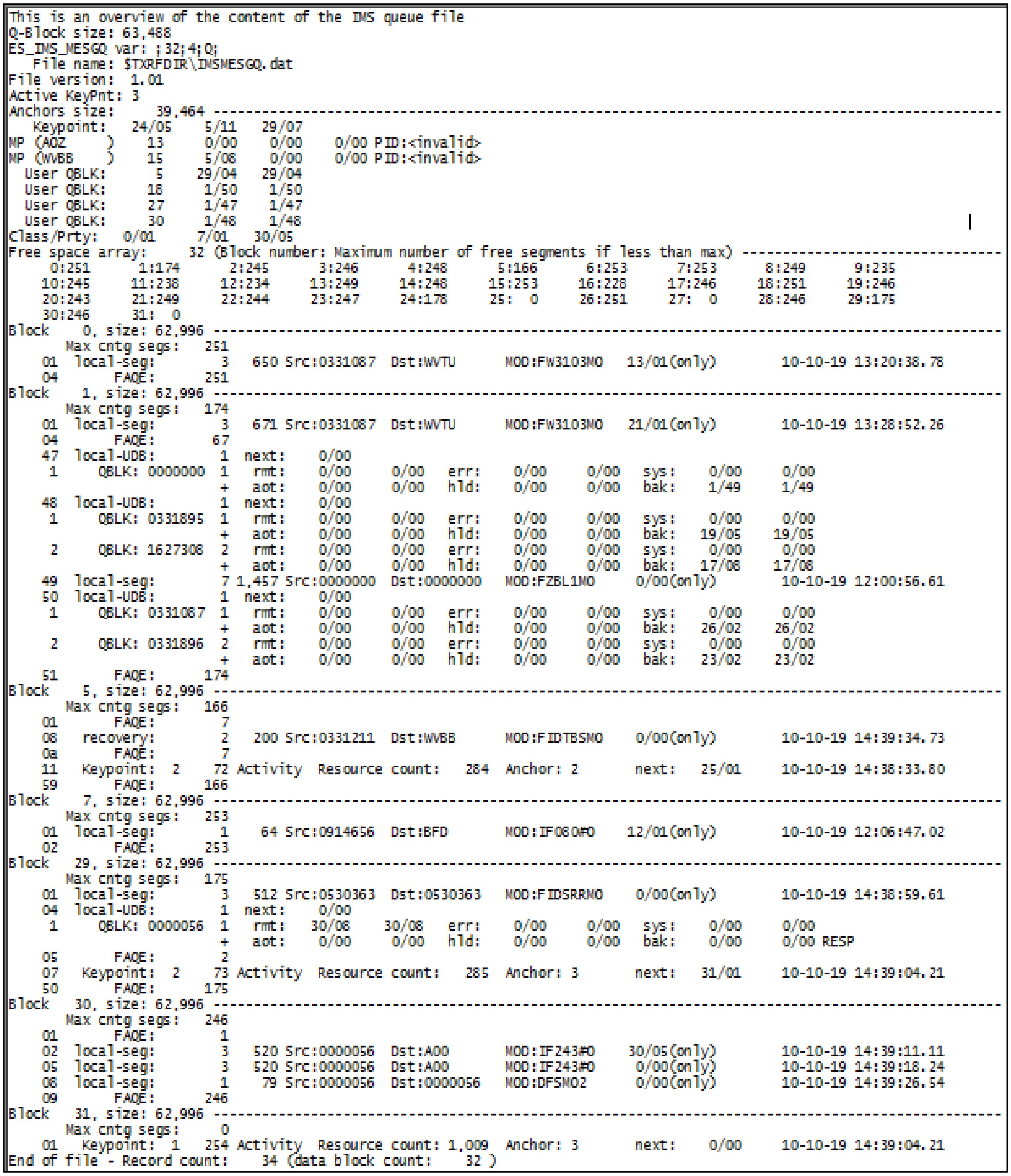
```
mftrace.dest=binfile
mftrace.dest=binfile,textfile
```

The **CTF Viewer**

- Is started from a product command prompt: "ctfviewer.exe" or by opening a binary CTF file (*.ctb)
- Is available on Linux in the Eclipse IDE
- Is a tool for viewing binary files that were created using the CTF binfile emitter
- Displays annotations for trace events to give them meaningful descriptions
- Provides details on a selected/highlighted event
- Can be used to view files from another system including Unix.

Appendix E: Viewing IMS Message Queue Diagnostics

For diagnostic purposes, an Enterprise Server that does not have IMS enabled can be used to view an IMS message queue file by simply copying the file into the server's system directory and going directly to the address of the ESMAC IMS Control page, e.g. http://localhost:9005/esmac/casrdo73 (the queue file is not bit- or OS-sensitive so, for example, you can use ESMAC on a 32-bit ES running on Windows to view the queue file from a 64-bit ES running on UNIX). The message queues summary is displayed as in the following example.



```
This is an overview of the content of the IMS queue file
Q-Block size: 63,488
ES_IMS_MESGQ var: ;32;4;Q;
   File name: $TXRFDIR\IMSMESGQ.dat
File version:  1.01
Active KeyPnt: 3
Anchors size:     39,464 -------------------------------------------------------------------------------------------
   Keypoint:   24/05    5/11   29/07
MP (AOZ     )    13     0/00    0/00    0/00 PID:<invalid>
MP (WVBB    )    15     5/08    0/00    0/00 PID:<invalid>
  User QBLK:      5    29/04   29/04
  User QBLK:     18     1/50    1/50
  User QBLK:     27     1/47    1/47
  User QBLK:     30     1/48    1/48
Class/Prty:   0/01     7/01   30/05
Free space array:     32 (Block number: Maximum number of free segments if less than max) -------------------------------
     0:251      1:174      2:245      3:246      4:248      5:166      6:253      7:253      8:249      9:235
    10:245     11:238     12:234     13:249     14:248     15:253     16:228     17:246     18:251     19:246
    20:243     21:249     22:244     23:247     24:178     25:  0     26:251     27:  0     28:246     29:175
    30:246     31:  0
Block    0, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   251
    01  local-seg:        3   650 Src:0331087  Dst:WVTU      MOD:FW3103MO   13/01(only)       10-10-19 13:20:38.78
    04        FAQE:     251
Block    1, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   174
    01  local-seg:        3   671 Src:0331087  Dst:WVTU      MOD:FW3103MO   21/01(only)       10-10-19 13:28:52.26
    04        FAQE:      67
    47  local-UDB:        1  next:    0/00
     1     QBLK: 0000000  1   rmt:    0/00     0/00   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:    1/49     1/49
    48  local-UDB:        1  next:    0/00
     1     QBLK: 0331895  1   rmt:    0/00     0/00   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:   19/05    19/05
     2     QBLK: 1627308  2   rmt:    0/00     0/00   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:   17/08    17/08
    49  local-seg:        7 1,457 Src:0000000  Dst:0000000   MOD:FZBL1MO     0/00(only)       10-10-19 12:00:56.61
    50  local-UDB:        1  next:    0/00
     1     QBLK: 0331087  1   rmt:    0/00     0/00   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:   26/02    26/02
     2     QBLK: 0331896  2   rmt:    0/00     0/00   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:   23/02    23/02
    51        FAQE:     174
Block    5, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   166
    01        FAQE:       7
    08   recovery:        2   200 Src:0331211  Dst:WVBB      MOD:FIDTBSMO    0/00(only)       10-10-19 14:39:34.73
    0a        FAQE:       7
    11   Keypoint:  2    72 Activity  Resource count:   284  Anchor: 2       next:   25/01     10-10-19 14:38:33.80
    59        FAQE:     166
Block    7, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   253
    01  local-seg:        1    64 Src:0914656  Dst:BFD       MOD:IF080#0    12/01(only)       10-10-19 12:06:47.02
    02        FAQE:     253
Block   29, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   175
    01  local-seg:        3   512 Src:0530363  Dst:0530363   MOD:FIDSRRMO    0/00(only)       10-10-19 14:38:59.61
    04  local-UDB:        1  next:    0/00
     1     QBLK: 0000056  1   rmt:   30/08    30/08   err:    0/00     0/00   sys:    0/00     0/00
                          +   aot:    0/00     0/00   hld:    0/00     0/00   bak:    0/00     0/00 RESP
    05        FAQE:       2
    07   Keypoint:  2    73 Activity  Resource count:   285  Anchor: 3       next:   31/01     10-10-19 14:39:04.21
    50        FAQE:     175
Block   30, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:   246
    01        FAQE:       1
    02  local-seg:        3   520 Src:0000056  Dst:A00       MOD:IF243#0    30/05(only)       10-10-19 14:39:11.11
    05  local-seg:        3   520 Src:0000056  Dst:A00       MOD:IF243#0     0/00(only)       10-10-19 14:39:18.24
    08  local-seg:        1    79 Src:0000056  Dst:0000056   MOD:DFSMO2      0/00(only)       10-10-19 14:39:26.54
    09        FAQE:     246
Block   31, size: 62,996 -------------------------------------------------------------------------------------------
       Max cntg segs:     0
    01   Keypoint:  1   254 Activity  Resource count: 1,009  Anchor: 3       next:    0/00     10-10-19 14:39:04.21
End of file - Record count:    34 (data block count:    32 )
```

The summary display header consists of the following:

- Q-Block size is the file block size (in bytes) of the message queue file.

- ES_IMS_MESGQ is an environment variable that contains:
  - Name of the message queue file (if not the default)
  - Number of queue data blocks (anywhere from 4 to 63,488, default 32)
  - Number of internal queue buffers (anywhere from 2 to 64, default 2)
  - Queue cold-start indicator (N = cold-start nothing, S = cold-start statistics, Y = cold-start resource definitions and statistics, Q = cold-start everything)
- File name is the name of the message queue file.
- File version is the version of the message queue file.
- Active KeyPnt is the number (1, 2, or 3, referring to the Keypoint list in the Anchors section) of the most recent keypoint.

A keypoint is a collection of in-memory message queue destinations (users and transactions) and their operational characteristics (i.e. stopped, etc.) at a certain point in time, recorded in the queue file as a single logical record (which may be segmented if it does not fit in a single data block). The purpose of the keypoint mechanism is to retain operational state across a shutdown/restart, as well as any dynamic transaction changes and transaction statistics for work that occurred up to the keypoint. A keypoint is always generated on shutdown and the frequency of other keypoints is defined by how many commits to the message queue are to occur before the next keypoint (i.e. the keypoint frequency is message-traffic-related). The keypoint frequency may be changed dynamically using ESMAC. At any given time the queue file contains only the three most recent keypoints but currently only the latest one is ever used.

On any start other than a cold start, transaction definitions and state are loaded from the last keypoint (not the resource definitions file), thereby matching any residual traffic that was in the system prior to shutdown (newly defined transactions are loaded at the next start if they had not been dynamically installed prior to shutdown). So upon restart after a normal shutdown, the system is returned to the same operational state it was in when it stopped. On an emergency restart, however, a transaction that was, say, stopped after the last keypoint will be in its earlier non-stopped state when the system returns to service.

The Anchors section of the summary display shows its size (in bytes) and has an entry for every entity to which messages/data are currently attached, each with one or more references to a data block/segment. Block/segment references have the form nnnnn/xx, where nnnnn is the 0-based block number and xx is the 1-based hex segment number (with 0/00 being a null reference). The anchor types are:

- Keypoint, with a reference, for each of the three keypoints, to the first data block/segment in the chain of data for that keypoint.
- MPR (MP or JCL), showing the destination name and PID number, with three references (one to the first segment in the chain of *input* message data for that MPR, one to the first segment in the chain of *output* message data, and one to the *last* in the output chain, respectively).
- User, with two references (one to the first segment in the message data chain and one to the last in the chain, respectively).
- Class/Priority, showing the class/priority followed by two references (one to the first segment in the message data chain, and one to the last in the chain, respectively).

The Free Space Array section of the summary display shows its size (one byte for each data block) and has an entry for any data block that currently contains data, showing the maximum number of contiguous segments in that block that are currently unused.

The summary display has a section for each data block that currently contains data. Each Block section shows its size (in bytes), the maximum number of contiguous unused segments in that block, and an entry for every used segment (as well as an FAQE entry for the first segment in every contiguous sequence of unused segments). Each entry shows the hex segment number and the number of segments that comprise that entry, and has a date/time stamp and a reference to the next segment in the chain. The types of segment data include:

- Keypoint, showing the number of resources (i.e. destinations) it contains.
- Local-segment, showing the message length, source, destination, and MID/MOD name, and whether the message data it contains is part of the first, middle, last or only message segment in the message.
- Local-UDB, showing the user name, with references to the first and last segments in each of the following message data chains:
    - Response Mode (rmt)
    - Error (err)
    - System (sys)
    - All Other Traffic (aot)
    - Held (hld)
    - Backup (bak)

A detail display of the contents of any data block segment can be seen by entering the block/segment reference on the ESMAC IMS Control page.

APPENDIX F: HINTS AND TIPS WHEN USING MAINFRAME OS370 ASSEMBLER

A separate document has been prepared that covers the following aspects when using assembler in a Micro Focus ES environment:

The assembler debugger screen

Expanding the blue source code panel

The Info menu command

The undocumented page down key:

Setting assembler break points.

Break on questionable exit from address space.

The “J”ump instruction.

SOC4 abends at cobol CALL time.

Abends in noAnimate programs.

Abend with no source code.

The document will be included in the main Troubleshooting pack.

APPENDIX G: EXTERNAL TRACING

This section contains information relating to the tracing of 3rd party applications that are used in association with Enterprise Server, to help diagnose problems when this technology is used with ES.
The information is provided as a starting point for when it might be useful to activate external traces, however for full information on these trace settings please refer to vendor's documentation.

We expect to add further information to this section at a later date.

IBM MQ TRACE

This can be enabled from the IBM MQ WebSphere GUI using the menu options – right-click the top node of the tree and select 'trace'
It can also be enabled from a command prompt as follows:

```
strmqtrc -m <Q_Manager_Name> -t all -t detail
```

[now run any tests to generate the traces]

To stop the trace, issue the following command:

```
endmqtrc –a
```

IBM MQ trace will be found under the MQ installation directory e.g.
C:\Program Files (x86)\IBM\WebSphere MQ\trace

Note that in addition there is a 'Qmgrs' directory containing subfolders for each of the individual Q managers, in which there will be an additional 'errors' folder (i.e. for that Q manager).e.g.

C:\Program Files (x86)\IBM\WebSphere MQ\Qmgrs\QM1\errors

ORACALE CLIENT TRACE

Oracle Client trace can be used to show the SQL commands that are generated and can be enabled by editing the file "sqlnet.ora" in the $ORACLE_HOME/network/admin directory of the client by adding the following:

```
diag_adr_enabled = OFF
trace_level_client = SUPPORT
trace_directory_client = <any directory>
trace_file_client = client.trc
trace_unique_client = ON
```

The Enterprise Server region would need to be restarted for these trace changes to take effect (when the XA resource re-establishes connection to Oracle the new trace settings would come into effect)

Oracle documents on this can be found here:

http://docs.oracle.com/cd/B28359_01/network.111/b28317/sqlnet.htm#NETRF948