Best Practices
Opcenter Execution
Troubleshooting

Customer	I**NTERNAL**
Supplier	JnT Digital s.r.o.
Version	1.0
Created at	17.03.2026
Author	Jan Tichý
Releaser	
File	JnT_BestPractices-Troubleshooting.docx

Revisions

Rev.	Chapter	Description	Date/Department/Signature
1.0	all	initial revision	2026-03-17/JnT-JT/

CONTENTS

LIST OF FIGURE

No table of figures entries found.

TERMS AND DEFINITIONS

Term	Definition
CLF	Configurable Logic Flow. A script-based extension mechanism in Opcenter Execution that allows customization of transaction logic without C++ development. CLFs are attached to CDO events and executions.
CDO	Configurable Data Object. The core data model unit in Opcenter Execution. Each transaction (e.g., MoveIn, Start) is represented as a CDO with fields and events.
WCF	Windows Communication Foundation. The Microsoft framework used by Opcenter for service-oriented communication between the Portal, Application Server, and external integrations.
Transaction Tracing	A built-in Opcenter diagnostic utility (csiTraceLogUtil) that generates per-transaction log files capturing CLF execution, parameter resolution, and optionally XML payloads and SQL statements.
DBUpdate	The Opcenter database schema migration utility. Executed via Management Studio to apply model changes, update WCF service definitions, and synchronize the database schema with the application version.
InputData XML	The serialized XML payload that the Portal (or any WCF client) sends to the Application Server when submitting a transaction. Its structure mirrors the CDO field hierarchy.
Portal Studio	The built-in design and administration interface for the Opcenter Portal layer. Used to configure pages, panels, settings, and tracing options.
CodeBehind	Server-side C# code compiled into Opcenter Portal pages, executed at request time. Used for complex UI logic that cannot be expressed in standard Portal configuration.
DebugView	A Microsoft Sysinternals utility that captures OutputDebugString messages from Windows processes. Used to monitor Application Server and service startup output that is not written to the Event Viewer.
ProcDump	A Microsoft Sysinternals command-line utility for capturing full or partial process memory dumps, triggered on exceptions, hangs, or on-demand.
Minidump / Full Dump	A snapshot of a process's memory state. A minidump captures thread and exception context only; a full dump (-ma) captures all memory regions and is required for deep defect analysis by Siemens R&D.
InSite Server	The Opcenter Application Server component. Hosts the core business logic, CLF execution engine, and WCF service endpoints.
csiTraceLogUtil	The command-line utility used to enable and disable Transaction Tracing on the Opcenter Application Server. Located in C:\Program Files (x86)\Camstar\InSite Server\.
LogXmlDocument	A configuration flag in the WCF Services web.config that enables logging of the full InputData XML payload for every transaction processed by the WCF service layer.
Depth	A numeric indicator in Transaction Trace log files representing the nesting level of the CLF call stack. Depth: 1 is the root level; each nested call or callsuper increments the counter.
PID	Process Identifier. A unique integer assigned by Windows to each running process. Required when targeting a specific process with ProcDump or DebugView.
EULA	End User License Agreement. The -accepteula flag in ProcDump and other Sysinternals tools suppresses the interactive EULA prompt for scripted or unattended execution.

1 INTRODUCTION

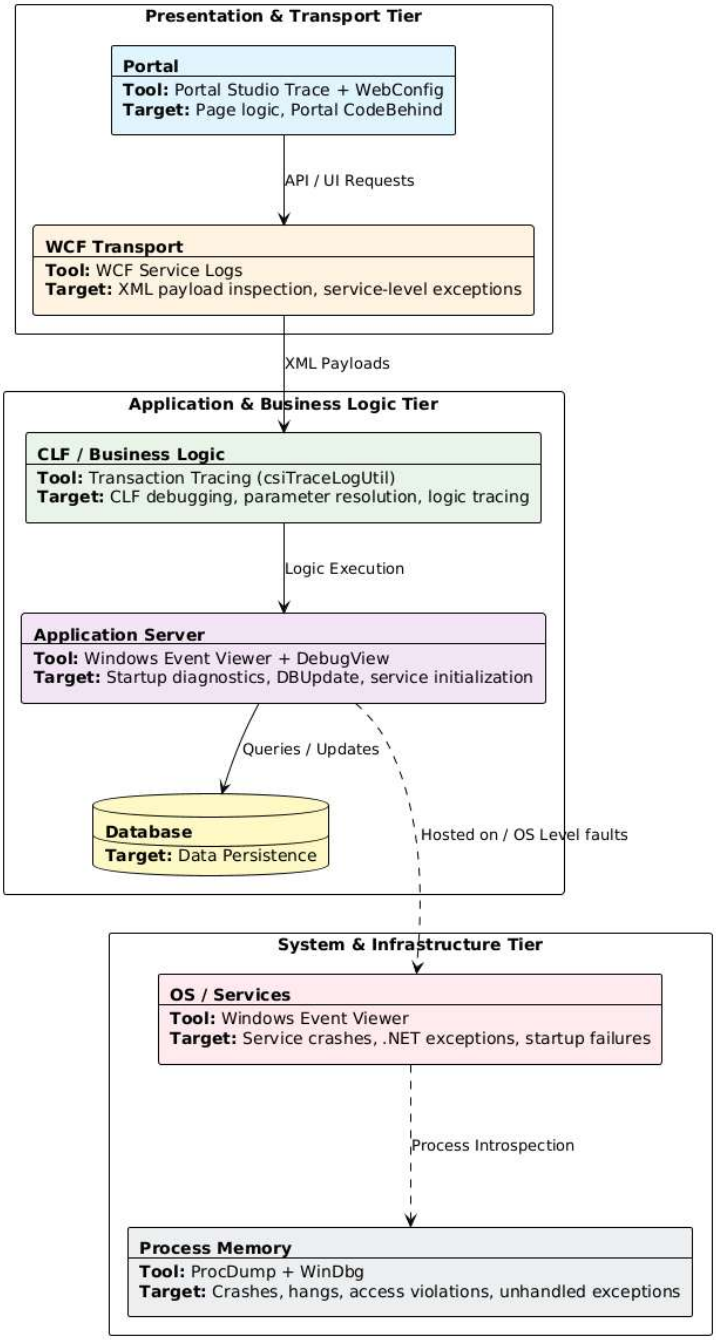
Opcenter is a multi-tier, distributed system: the Portal, Application Server, WCF Services, and database each maintain their own logging subsystem with different levels of granularity.

This guide covers the full diagnostic stack available to Opcenter implementation engineers and system administrators.

1.1 Diagnostic Architecture Overview

The following layers each produce diagnostics that are useful in different failure scenarios:

Layer	Tool / Mechanism	Primary Use Case
OS / Services	Windows Event Viewer	Service crashes, .NET exceptions, startup failures
CLF / Business Logic	Transaction Tracing (csiTraceLogUtil)	CLF debugging, parameter resolution, logic tracing, XML payload
WCF Transport	WCF Service Logs	XML payload inspection, service-level exceptions
Portal	Portal Studio Trace + WebConfig (debug=true)	Page logic, Portal CodeBehind
Application Server	Windows Event Viewer + DebugView (dbgview.exe)	Startup diagnostics, DBUpdate, service initialization
Process Memory	ProcDump + WinDbg	Crashes, hangs, access violations, unhandled exceptions

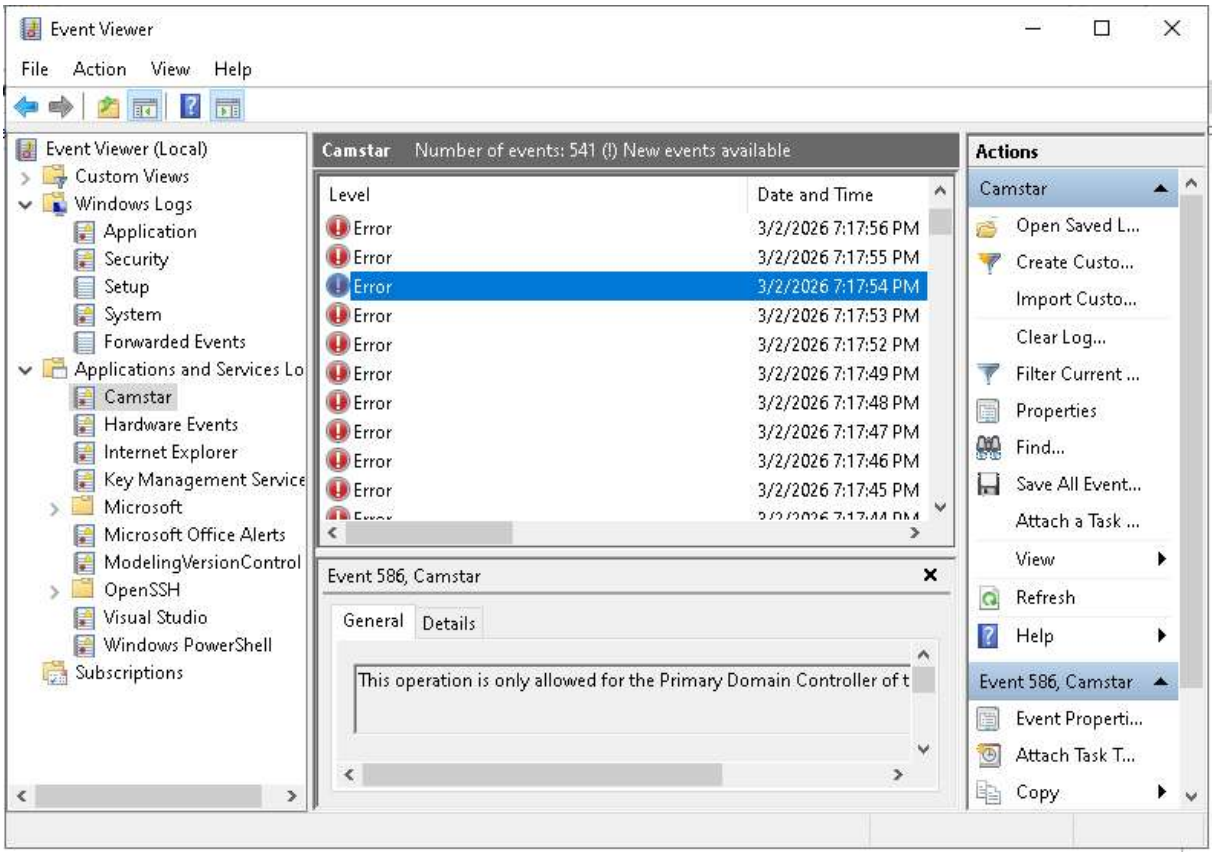

Presentation & Transport Tier
Portal
Tool: Portal Studio Trace + WebConfig
Target: Page logic, Portal CodeBehind
API / UI Requests
WCF Transport
Tool: WCF Service Logs
Target: XML payload inspection, service-level exceptions
XML Payloads
Application & Business Logic Tier
CLF / Business Logic
Tool: Transaction Tracing (csiTraceLogUtil)
Target: CLF debugging, parameter resolution, logic tracing
Logic Execution
Application Server
Tool: Windows Event Viewer + DebugView
Target: Startup diagnostics, DBUpdate, service initialization
Queries / Updates
Database
Target: Data Persistence
Hosted on / OS Level faults
System & Infrastructure Tier
OS / Services
Tool: Windows Event Viewer
Target: Service crashes, .NET exceptions, startup failures
Process Introspection
Process Memory
Tool: ProcDump + WinDbg
Target: Crashes, hangs, access violations, unhandled exceptions

2 WINDOWS EVENT VIEWER

2.1 Diagnostic Architecture Overview

The Event Viewer should be your first stop for any issue involving:

- Opcenter services that fail to start or crash unexpectedly.
- Unhandled .NET exceptions at the application level.
- IIS Application Pool failures.
- Windows service termination events (Camstar App Server, WCF Services).



2.2 Relevant Event Sources

Navigate to: **Event Viewer > Windows Logs > Application OR Camstar**
Filter by these sources:

Source	Description
.NET Runtime	Unhandled CLR exceptions from any Camstar/Opcenter process
Application Error	Process crashes with faulting module info
ASP.NET	IIS / Portal errors
Camstar	Application Server startup, licensing, and runtime errors

2.3 Key Procedure

1. Open **Event Viewer** (eventvwr.msc).
2. Navigate to **Windows Logs > Application OR Camstar**.
3. Click **Filter Current Log** on the right panel.
4. Set the time range to match the incident window.
5. Filter by **Event Level:** Error, Critical.
6. Look for the first error that preceded any cascade of failures — the root cause is almost always higher in the timestamp order than the symptom.

Pro Tip: Export the filtered log to .evtx or .csv before making any system changes. Event logs are volatile and can be overwritten.

3 TRANSACTION TRACING (CSITRACELOGUTIL)

3.1 When to Use

Transaction Tracing is the primary tool for debugging **Configurable Logic Flows (CLFs)**. Use it when:

- A transaction behaves unexpectedly (wrong validation, data not saved, wrong field values).
- A custom CLF is not executing the expected branch.
- You need to verify which executions fired and what values they resolved to.
- You are developing new CLF logic and need to validate parameter resolution.

Warning: Tracing has a **significant performance impact** on the Application Server. Enable it only for the specific user and duration needed. Never leave it enabled in production without active monitoring.

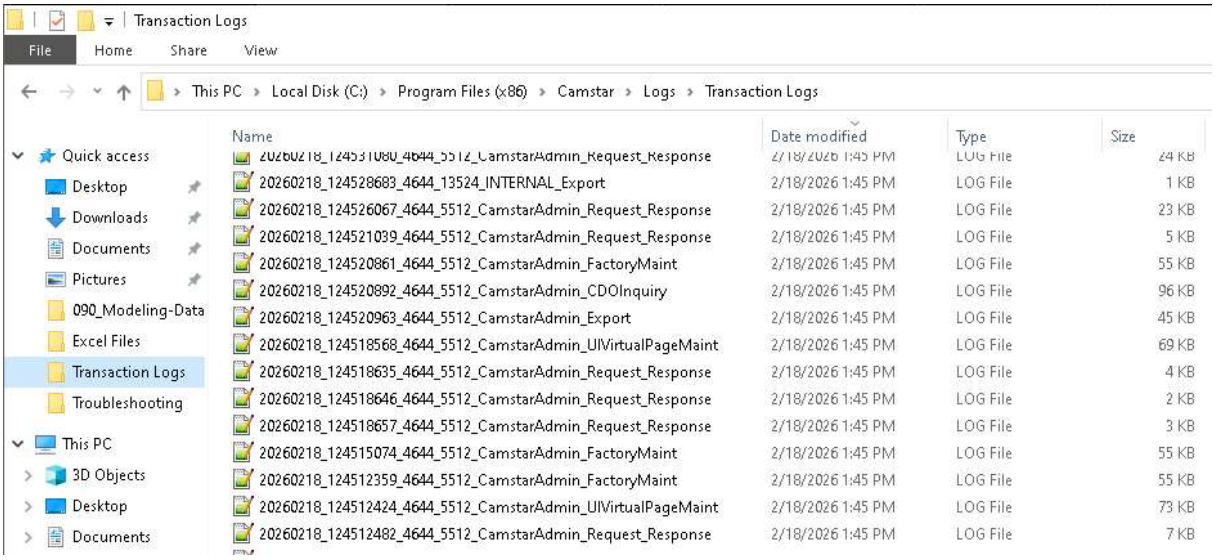
3.2 Utility Location

The csiTraceLogUtil command-line utility is installed with the Application Server:

- *C:\Program Files (x86)\Camstar\InSite Server\csiTraceLogUtil.exe*

Trace log files are stored in:

- *C:\Program Files (x86)\Camstar\Logs\Transaction Logs*



3.3 Trace Levels

Level	What Is Traced
1	CLF names only
2	CLF names + function names
3	CLF names + function names + parameters
4	Level 3 + resolved parameter values
5	Level 4 + XML requests & responses (SerializeXMLResponse)
6	Level 5 + SQL commit statements Hidden feature

Recommendation:

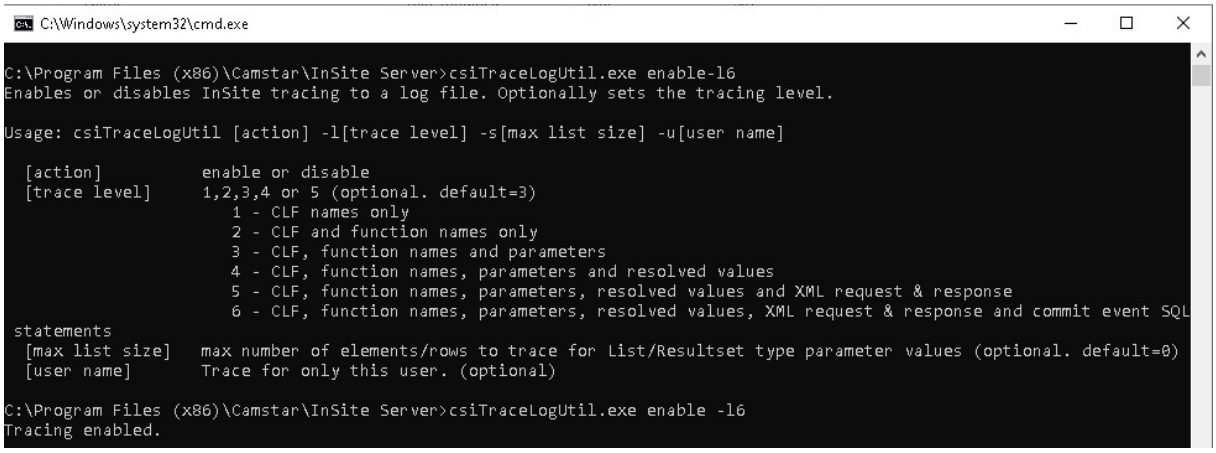
- Use **Level 3** for general CLF debugging.
- Use **Level 4** when you need to confirm what value a parameter actually resolved to at runtime.
- Use **Level 5** only when you need to inspect the full XML payload passing through the server.
- Use **Level 6** sparingly — it generates very large files.

3.4 Enabling Transaction Tracing

1. Open **Command Prompt as Administrator**.
2. Change directory to the InSite Server folder:
   cd "C:\Program Files (x86)\Camstar\InSite Server"
3. Enable tracing with the desired parameters:
   csiTraceLogUtil enable -l[level] -s[max list size] -u[username]

Examples:
csiTraceLogUtil enable -l5

4. A confirmation message trace enabled is displayed.
5. **Reproduce the issue** in the Portal or via the API.



3.5 Disabling Transaction Tracing

Always disable tracing immediately after capturing the relevant log.

1. Open **Command Prompt as Administrator**.
2. Navigate to the InSite Server directory.
3. Execute:
   csiTraceLogUtil disable
4. Confirmation message trace disabled is displayed.

3.6 Trace File Naming Convention

Each transaction generates a separate log file:
yyyymmdd_HHMMSSMMM_<ProcessID>_<ThreadID>_<UserName>_<TransactionName>.log

Example:
20201210_161118849_4540_5328_CamstarAdmin_Start.log

This file documents the **Start** transaction executed on 10 Dec 2020 at 16:11:18.849, by user CamstarAdmin, with process ID 4540 and thread ID 5328.

3.7 Reading a Trace File

Trace files have three columns:

Column	Content
1	Timestamp (yyyyMMddHHmmss.mmm)
2	Thread ID
3	Event detail — CDO name, field name, event name, execution name, and parameter values

Column 3 — Field Event Format:
Depth:<n> **CDO**: <cdo name> **Field**: <field name> **Event**: <event name>
Exec: <execution name>
Value = <parameter value>

Column 3 — CDO Perform Format:
Depth:<n> **CDO**: <cdo name> **Perform**: <method name>
Exec: <execution name>

Depth indicates the nesting level of the call stack. Root-level events are Depth: 1; each nested call increments the depth counter. Track Leaving Depth: N entries to identify where execution returns to the parent.

```
20250126192533.445  19604   Request Document
20250126192533.445  19604   <?xml version="1.0" encoding="utf-16"?><__InSite __version="1.1" __encryption=
"3"><__session><__useSession><user><__name><![CDATA[CamstarAdmin]]></__name></user><sessionId __encrypted=
"no"><![CDATA[d4e98980-d420-46cf-80ef-39e11e5d8c8b]]></sessionId></__useSession></__session><__service
__serviceType="Start"><__txnGUID>cc992476-d0c6-4af3-9e2b-7a8f62b22130</__txnGUID><__utcOffset>+01:00<
/__utcOffset><__inputData><Details><AutoNumber><![CDATA[True]]></AutoNumber><Product><__name><!
[CDATA[JNT_Product_01]]></__name><__useROR>true</__useROR></Product><Qty><![CDATA[10]]></Qty><UOM><__name
><![CDATA[PIECE]]></__name></UOM><Owner><__name><![CDATA[PROD]]></__name></Owner><Level><__name><!
[CDATA[UNIT]]></__name></Level><StartReason><__name><![CDATA[NORMAL]]></__name></StartReason></Details><
CurrentStatusDetails><Workflow><__name><![CDATA[wf_JNT_blank]]></__name><__useROR>true</__useROR></Workflow
></CurrentStatusDetails></__inputData><__execute /><__requestData><CompletionMsg /></__requestData><
/__service></__InSite>
20250126192533.446  19604   starting transaction Start for CamstarAdmin
20250126192533.448  19604   Depth: 1 CDO: CollectDataPoints Field: Constants Event: Initialize
20250126192533.448  19604      Exec: IsVariableDefined
20250126192533.448  19604         Result = CLF::ConstDefined
20250126192533.448  19604         Scope = "Transaction"
20250126192533.448  19604         VariableName = "__Const"
20250126192533.449  19604      Exec: setfieldconditional
20250126192533.449  19604         Condition = CLF::ConstDefined
20250126192533.449  19604         ElseFrom = Constants
20250126192533.449  19604         To = Transaction::__Const
20250126192533.449  19604         Condition, resolved value = ( false )
20250126192533.449  19604         ElseFrom, resolved value = ( 000a8c0000000003, 296868139499.5203 )
20250126192533.449  19604   Leaving Depth: 1 CDO: CollectDataPoints Field: Constants Event: Initialize
20250126192533.449  19604   Depth: 1 CDO CollectDataPoints Event: AfterInitialize
20250126192533.449  19604      Exec: if
20250126192533.449  19604         Condition = User and  User.SessionValues
20250126192533.449  19604      Exec: CreateCDO
20250126192533.449  19604         CDOTypeName = "LocalSettings"
20250126192533.449  19604         NewCDO = User.SessionValues.LocalSettings
20250126192533.449  19604         OverwriteExistingCDO = true
20250126192533.449  19604         UseFieldType = true
20250126192533.449  19604         CDOTypeName, resolved value = ( LocalSettings )
20250126192533.449  19604         UseFieldType, resolved value = ( true )
20250126192533.450  19604         OverwriteExistingCDO, resolved value = ( true )
20250126192533.450  19604         NewCDO, resolved value = (  )
20250126192533.450  19604         NewCDO, resolved value = [Field Reference] )
20250126192533.450  19604   Leaving Depth: 1 CDO CollectDataPoints Event: AfterInitialize
20250126192533.450  19604   Depth: 1 CDO: Start Field: WIPMsgMgr Event: Initialize
20250126192533.450  19604      Exec: FieldCreateCDO
20250126192533.450  19604   Leaving Depth: 1 CDO: Start Field: WIPMsgMgr Event: Initialize
20250126192533.450  19604   Depth: 1 CDO: Start Field: Constants Event: Initialize
20250126192533.450  19604      Exec: IsVariableDefined
20250126192533.450  19604         Result = CLF::ConstDefined
20250126192533.450  19604         Scope = "Transaction"
20250126192533.450  19604         VariableName = "__Const"
20250126192533.450  19604      Exec: setfieldconditional
```

3.8 Parameter Value Formats at Level 4+

Type	Format
List (Type Multiplicity)	ParameterName, resolved value = ([List, size=5] val1, val2, …)
Multi-Value	ParameterName, resolved values [Multiple, size=5, index=4] = (InstanceId)
Resultset	ParameterName, resolved value = ([Resultset, rows=5, columns=2] {row1c1, row1c2}, …)
Query Parameters	ParameterName, resolved value = ([Query Parameters, size=2] Key1=val1, Key2=val2)
Field Reference	ParameterName, resolved value = [Field Reference]
Out/Return	Not traced.

Note: The -s (max list size) parameter controls how many elements from List or Resultset types are written. Set it to 0 to suppress list tracing entirely at Level 4+.

4 WCF SERVICE LOGGING

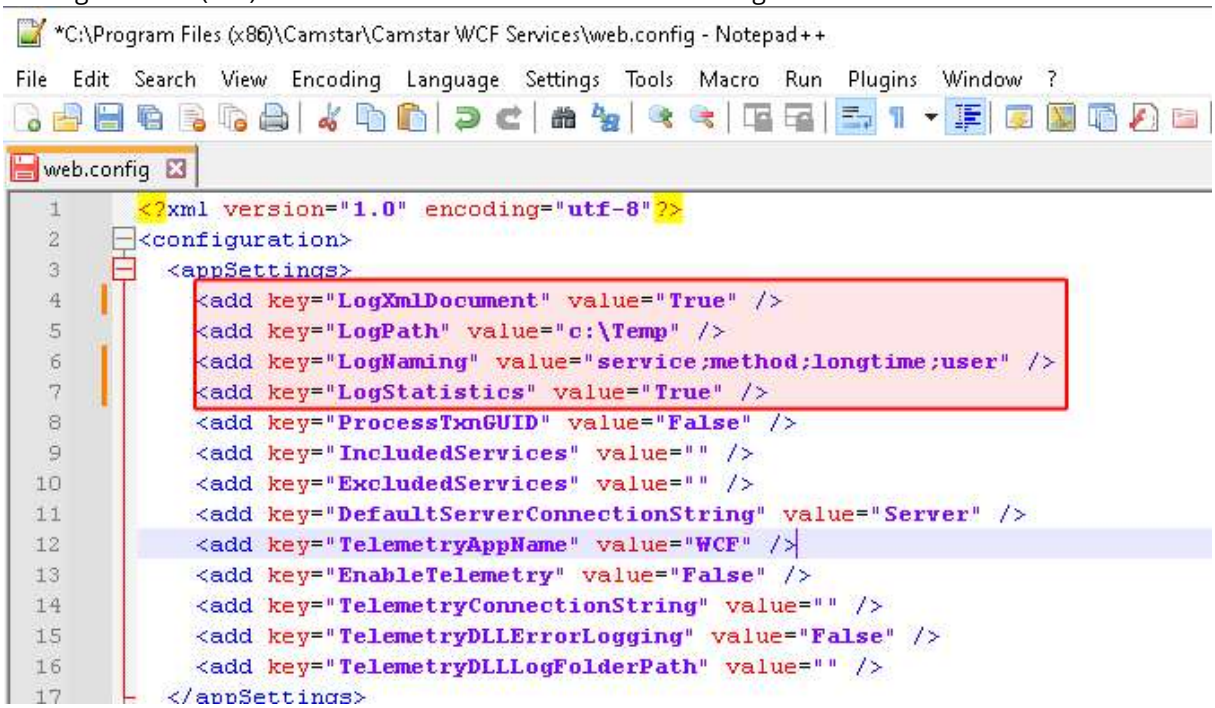
4.1 When to Use

WCF logs capture the **XML payloads** exchanged between the Portal (or any WCF client) and the Opcenter Application Server. Use this tool when:

- A transaction fails and you need to inspect the exact InputData XML being sent.
- You suspect a field value is not being serialized correctly from the Portal form.
- You are integrating a third-party system via WCF and need to verify the request structure.

4.2 Configuration File Location

C:\Program Files (x86)\Camstar\Camstar WCF Services\web.config



4.3 Logging Configuration Keys

Locate the <appSettings> section in web.config and set the following keys:

<!-- Enable XML document logging -->
<add key="LogXmlDocument" value="True" />

<!-- Directory where log files are written -->
<add key="LogPath" value="C:\Program Files (x86)\Camstar\InSite Server\Transaction Logs" />

<!-- Filename components. Available tokens: service, method, time, longtime, user -->
<!-- Use "longtime" for timestamp format: <date>_<time> -->
<!-- Tokens are combined with semicolon (;) delimiter -->
<add key="LogNaming" value="service;method;longtime;user" />

<!-- Enable execution statistics logging to Statistics.csv -->
<add key="LogStatistics" value="True" />

LogNaming token reference:

Token	Output
service	WCF service name
method	API method called
time	Time component only
longtime	Full <date>_<time> timestamp
user	Username making the request

Best Practice: Use service;method;longtime;user for maximum traceability. For lighter logging on busy systems, method;time is a reasonable balance.

4.4 Important Notes

- **Restart Required:** Changes to web.config take effect immediately for IIS-hosted WCF services (no manual restart needed under IIS). For Windows Service-hosted deployments, a service restart is required.
- **Disk Space:** At Level 5 Transaction Tracing combined with LogXmlDocument=True, disk usage can grow very rapidly. Monitor C:\Program Files (x86)\Camstar\InSite Server\Transaction Logs\ and purge after capturing the required logs.
- **Set back to False** after capturing — this is a common oversight that causes disk exhaustion in production environments.

5 PORTAL TRACING

5.1 When to Use

Portal Tracing records a history of Portal page events as they are modified and executed in Portal Studio. Use it when:

- A Portal page is not rendering correctly or throws a client-side error.
- You need to capture server-side request/response data at the Portal layer.
- A user reports a specific page behavior that cannot be reproduced via Transaction Tracing alone.

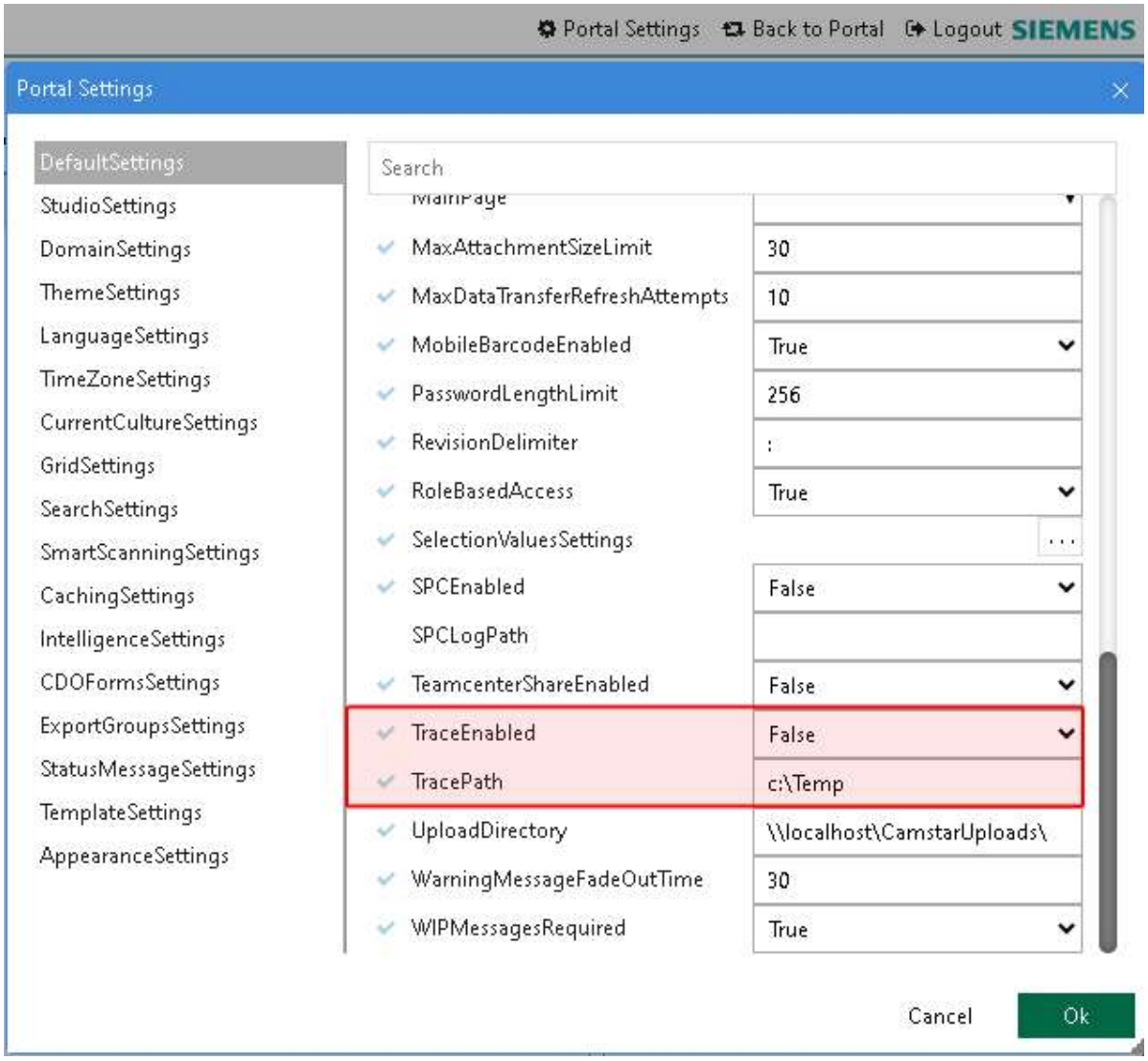
5.2 Portal Settings File Location

Portal Studio settings are persisted in:
C:\Program Files (x86)\Camstar\Camstar Portal\User\Settings.xml

5.3 Enabling Portal Tracing via Portal Studio

1. Open Portal in your browser
2. Click **Studio** in the toolbar to open Portal Studio.
3. Click the **Portal Settings** button (gear icon ⚙).

4. Select **DefaultSetting**.
5. Locate the TraceEnabled setting and set its value to **True**.
6. Note the TracePath value — this is where the trace files will be written. Change it if the default location has insufficient disk space or permissions.
7. **Reproduce the Portal issue.**



5.4 Disabling Portal Tracing

1. Return to **Portal Studio > Portal Settings > DefaultSetting**.
2. Set TraceEnabled to **False**.

5.5 Portal CodeBehind Debug Mode

If you are debugging **CodeBehind** logic (server-side C# code compiled into Portal pages), enable the debug compilation flag in the Portal web.config:
C:\Program Files (x86)\Camstar\Camstar Portal\web.config

Locate the <compilation> element and set debug="true":

```
<compilation
   targetFramework="4.7.2"
   batch="false"
   debug="true"
   optimizeCompilations="true">
```

Warning: Setting debug="true" in production disables ASP.NET page timeout enforcement and increases memory usage. Revert this change after debugging.

6 DEBUGVIEW (SYSINTERNALS)

6.1 When to Use

DebugView is a Microsoft Sysinternals utility that captures **OutputDebugString** messages written by the Opcenter Application Server and services to the Windows debug output channel. It is particularly valuable during:

- **DBUpdate** execution (schema migrations, modeling data updates).
- **Service startup** — when the Application Server or WCF Services fail to initialize.
- Situations where the Event Viewer does not provide enough detail and Transaction Tracing is not yet available (e.g., service is not running).

6.2 Installation

Download from Microsoft Sysinternals:
https://docs.microsoft.com/en-us/sysinternals/downloads/debugview
No installation is required. Execute dbgview.exe directly — it immediately begins capturing debug output.

6.3 Enabling Extended Debug Output

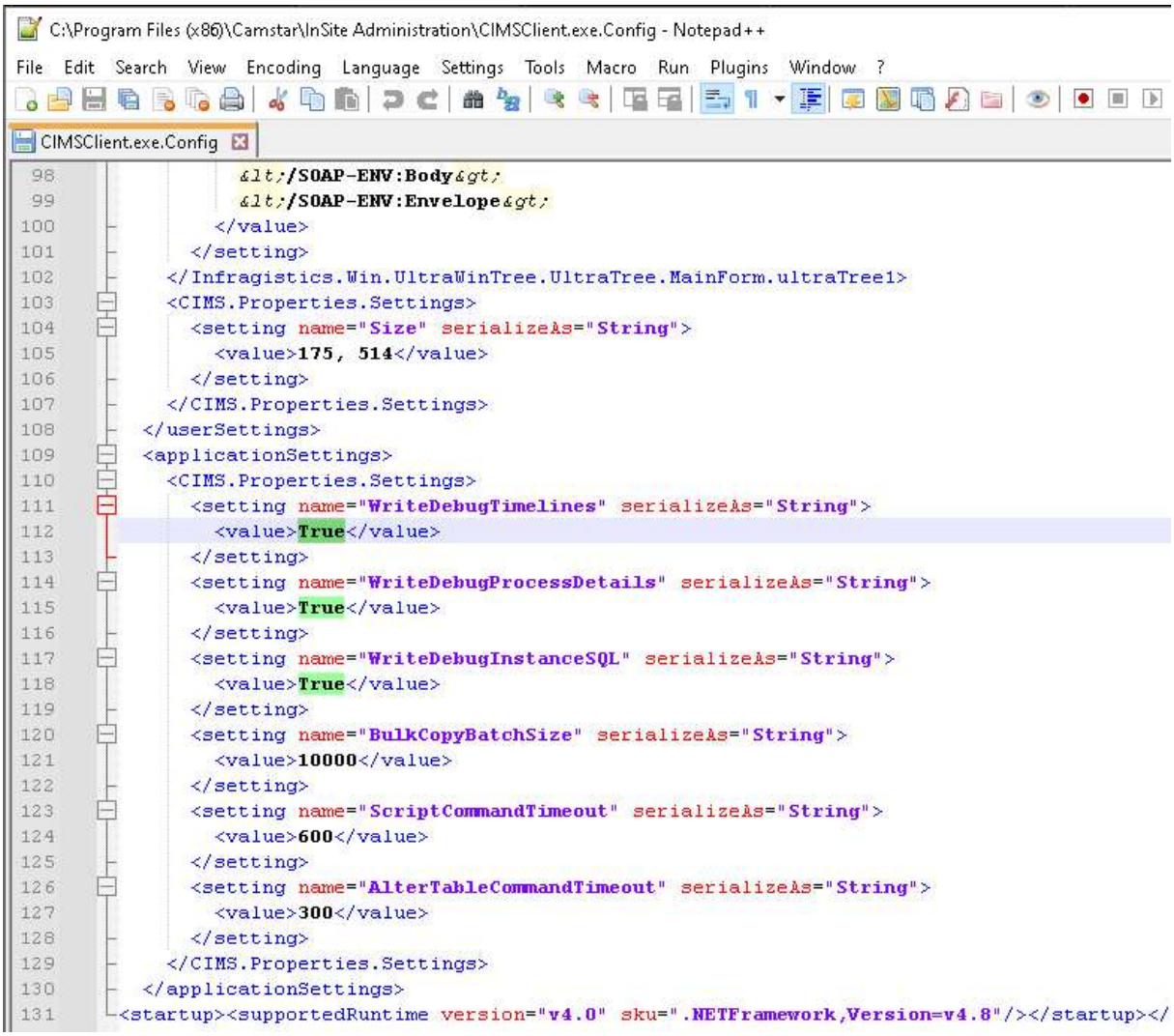
By default, the Application Server writes minimal debug data. To enable extended diagnostics (timeline data, process details, instance-level SQL):

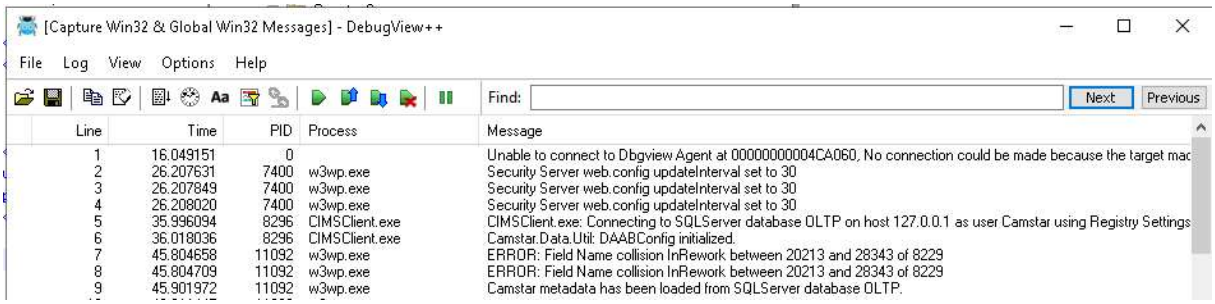
Configuration file:
C:\Program Files (x86)\Camstar\InSite Administration\CIMSClient.exe.config

Locate the <applicationSettings> section and set the following to True:

```
<applicationSettings>
  <CIMS.Properties.Settings>
    <setting name="WriteDebugTimelines" serializeAs="String">
      <value>True</value>
    </setting>
    <setting name="WriteDebugProcessDetails" serializeAs="String">
      <value>True</value>
    </setting>
    <setting name="WriteDebugInstanceSQL" serializeAs="String">
      <value>True</value>
    </setting>
  </CIMS.Properties.Settings>
</applicationSettings>
```

After saving the file, **restart Management Studio (CIMS Client)**.





6.4 DBUpdate Procedure with DebugView

1. Start dbgview.exe — ensure it is running and capturing before you proceed.
2. Open **Opcenter Management Studio**.
3. Launch **DBUpdate** and select **Generate WCF Services** if applicable.
4. Reproduce the problem or monitor the output as DBUpdate executes.
5. In DebugView, use **Edit > Copy All** or **File > Save** to preserve the output before it scrolls away.
6.

Pro Tip: Enable **Capture > Capture Global Win32** in DebugView to capture output from all processes, not just the foreground application.

7 PROCDUMP (PROCESS MEMORY CAPTURE)

7.1 When to Use

ProcDump is a Sysinternals utility for capturing **full process memory dumps** (minidumps or full dumps). Use it when:

- The Application Server or Portal crashes with an access violation or unhandled exception.
- A process **hangs** (stops responding) without crashing — ProcDump can capture the hung state.
- Siemens Support or Siemens R&D requests a memory dump for a defect investigation.
- You need to analyze a native exception that is not surfaced in the Event Viewer.

7.2 Installation

Download from Microsoft Sysinternals:
https://docs.microsoft.com/en-us/sysinternals/downloads/procdump
Copy procdump.exe to the server. No installation required.

7.3 Standard Capture Command

The following command captures a **full memory dump** on any unhandled exception from the target process:
procdump -accepteula -ma -e 1 -f "" <ProcessID> C:\temp

Parameter breakdown:

Parameter	Description
-accepteula	Accepts the Sysinternals EULA silently
-ma	Full dump (all memory regions)
-e 1	Trigger on unhandled exception (first-chance exceptions)
-f ""	Filter on all exception types (empty string = no filter)
<ProcessID>	PID of the target process (find via Task Manager)
C:\temp\	Output directory for the dump file

Example:
procdump -accepteula -ma -e 1 -f "" 23082 C:\temp\

7.4 Capturing a Hung Process

If the process is not crashing but is completely unresponsive:
procdump -accepteula -ma <ProcessID> C:\temp\hung_dump.dmp
This immediately captures the current memory state without waiting for an exception

7.5 Delivering the Dump to Siemens Support

- Full dumps are large (typically 500 MB – 4 GB depending on the Application Server heap size).
- Compress with 7-Zip before uploading to the Siemens Support Portal.

- Always include the corresponding **Event Viewer export** and **Transaction Log** for the same time window.

8 DIAGNOSTIC DECISION TREE

Use this table to quickly identify the right tool for a given symptom:

Symptom	First Tool	Second Tool
Service won't start	Event Viewer	DebugView
DBUpdate fails	DebugView	Event Viewer
CLF not firing / wrong branch	Transaction Tracing (L3)	Transaction Tracing (L4)
Wrong parameter value in CLF	Transaction Tracing (L4)	WCF LogXmlDocument
Portal page error / blank page	Portal Studio Trace	Event Viewer (ASP.NET)
Transaction fails — unknown reason	Transaction Tracing (L3)	WCF LogXmlDocument
Wrong data sent by integration	WCF LogXmlDocument	Transaction Tracing (L5)
Application crash / AV	Event Viewer	ProcDump
Application hang	ProcDump (hung mode)	DebugView
Performance degradation	WCF LogStatistics (Statistics.csv)	Transaction Tracing (L4)
DBUpdate SQL errors	DebugView + WriteDebugInstanceSQL=True	Event Viewer

9 COMMON MISTAKES & BEST PRACTICES

❌ Mistake	✅ Best Practice
Leaving Transaction Tracing enabled in production	Always disable immediately after capturing the required log
Using LogXmlDocument=True and forgetting to revert	Add it to your post-incident checklist; disk exhaustion is a real risk
Using debug="true" in Portal web.config on production	Revert after debugging; document the change in your change log
Not capturing Event Viewer logs before restarting services	Always export the event log before any corrective action
Using debug="true" in Portal web.config on production	Revert after debugging; document the change in your change log
Not capturing Event Viewer logs before restarting services	Always export the event log before any corrective action

JNT Digital s.r.o. | jnt-digital.net | Jan Tichý — Principal Solution Architect This document is provided for educational and consulting reference purposes.