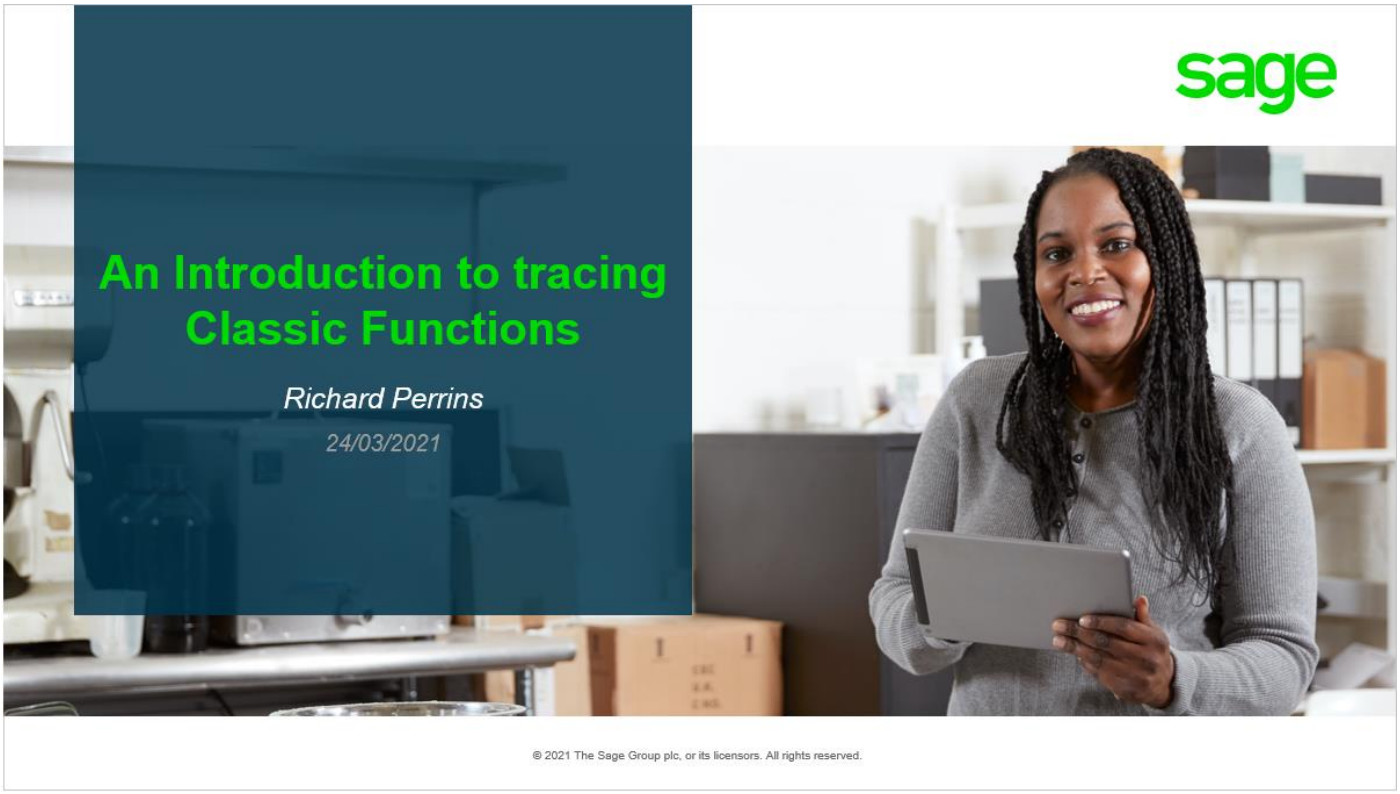
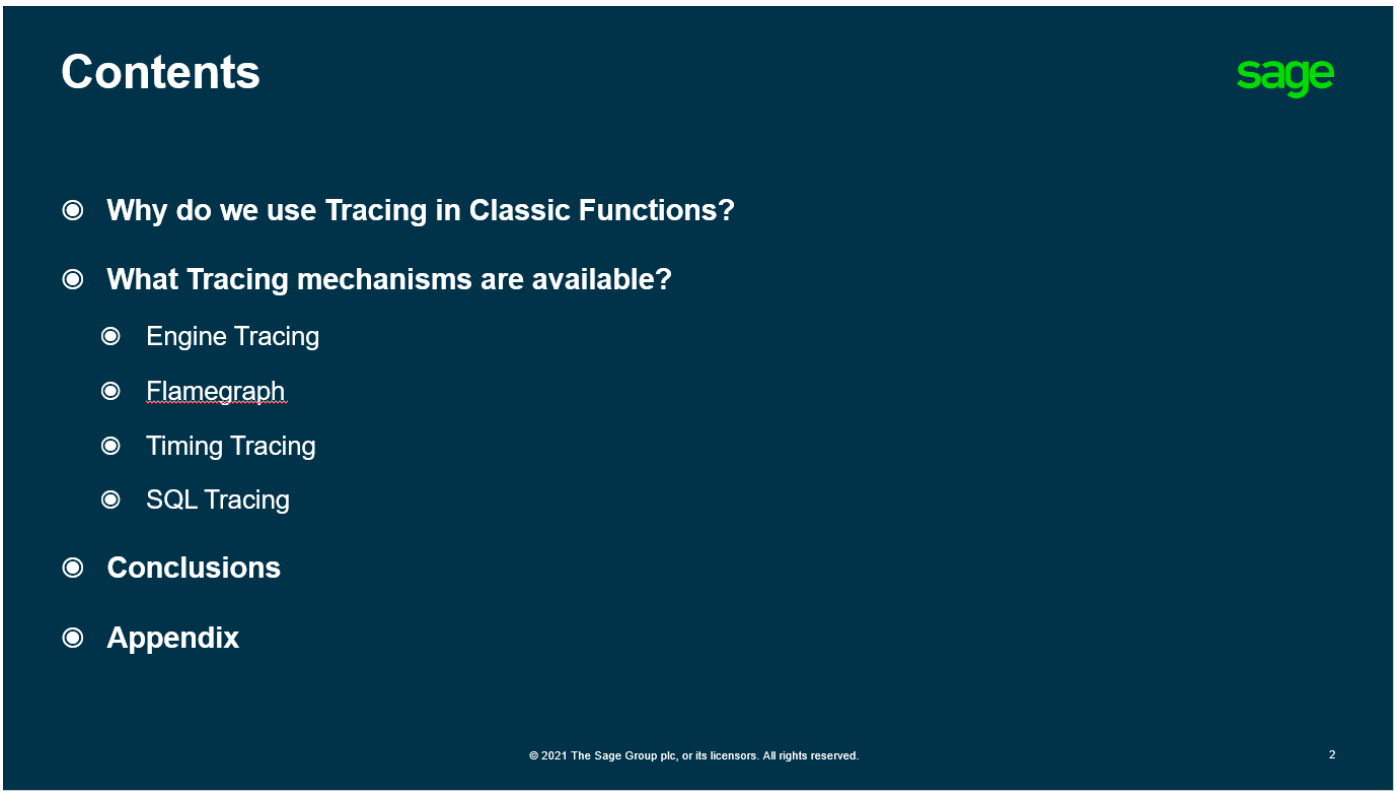
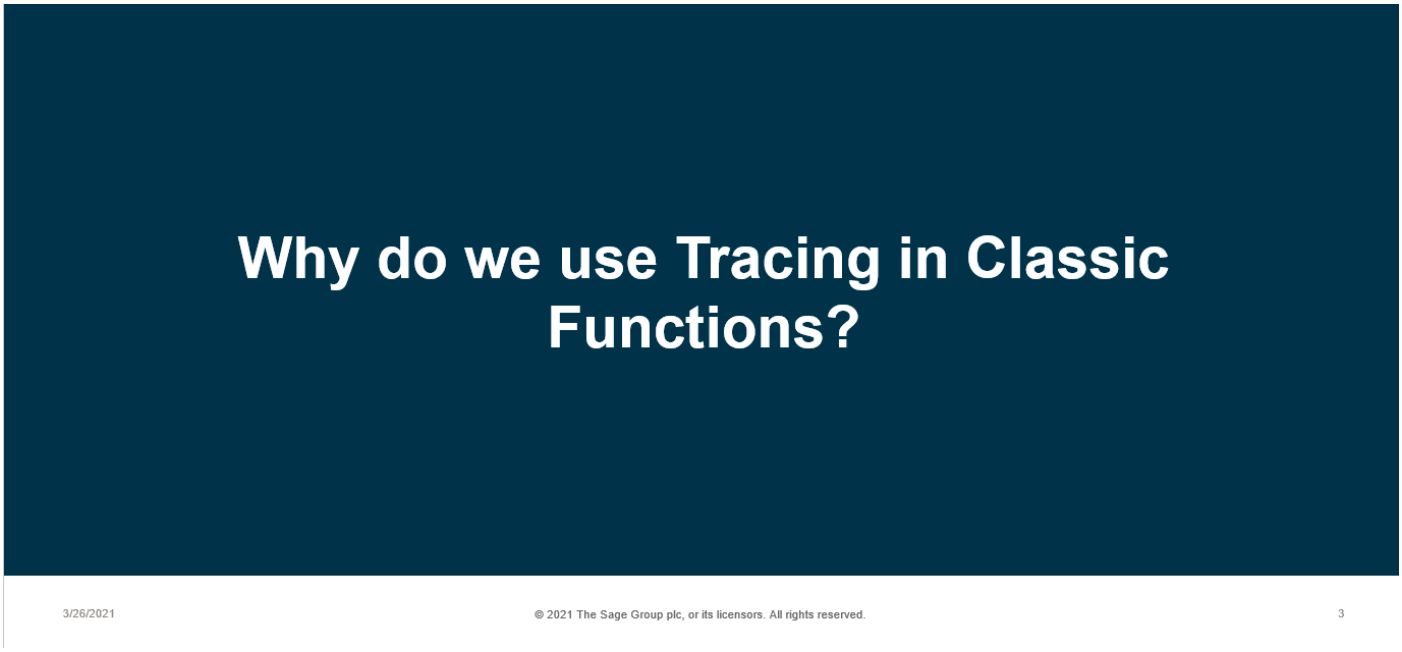


An Introduction to tracing Classic Functions



Contents

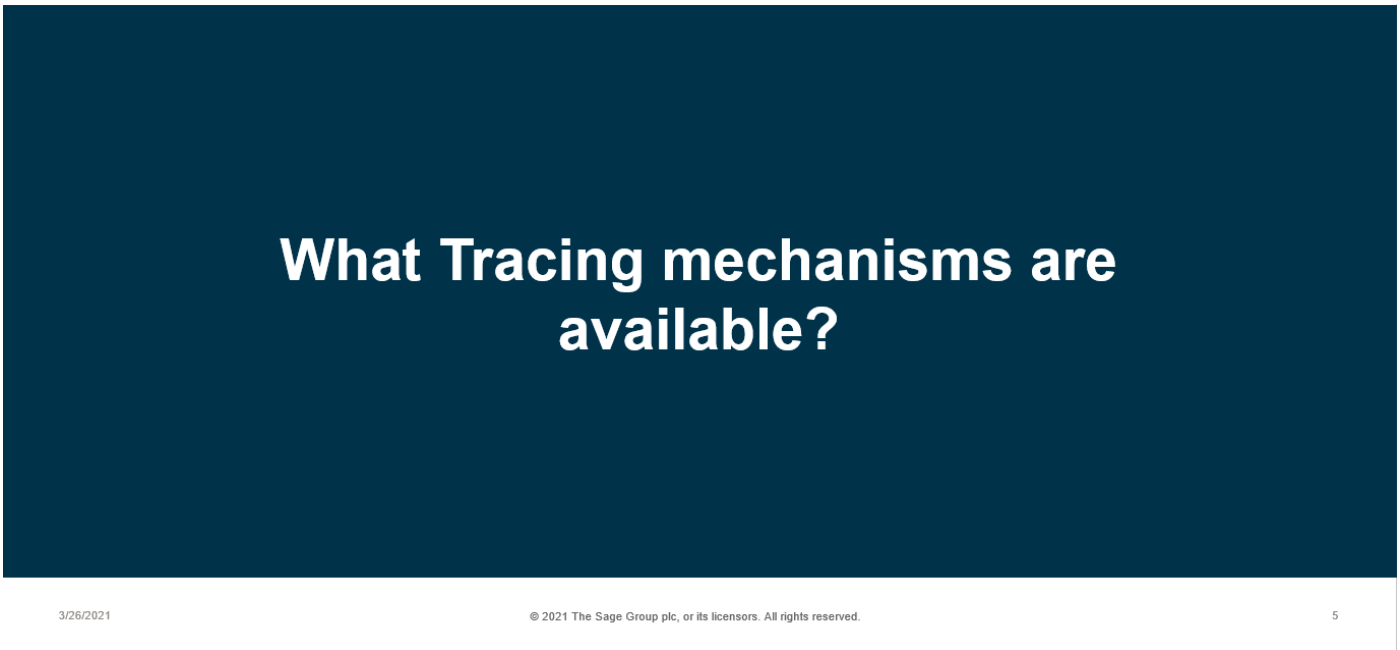
- **Appendix 2: Further Reading**



Why do we use Tracing in X3 Classic Functions?

Whilst investigating some issues in X3, it may be advantageous to have insight into what code is being executed during the running of a Classic Function – some typical instances are:

1. Un-expected behaviour of a Function
2. Errors in a Function
3. Slow performance of a Function



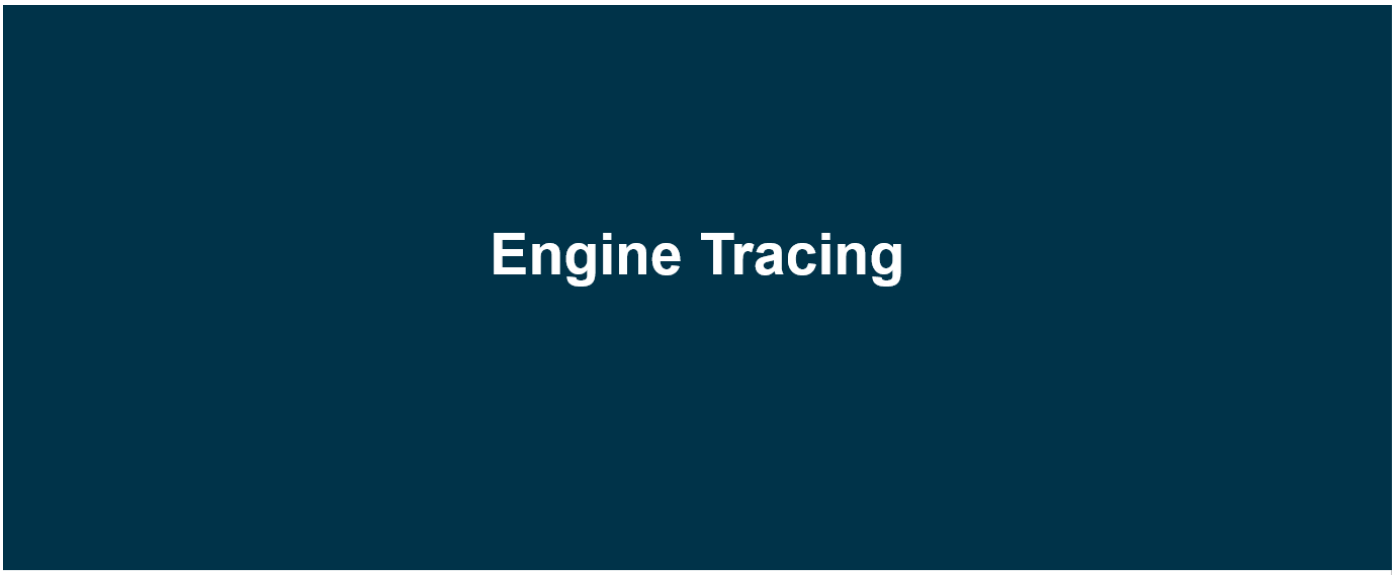
What Tracing Mechanisms are available?

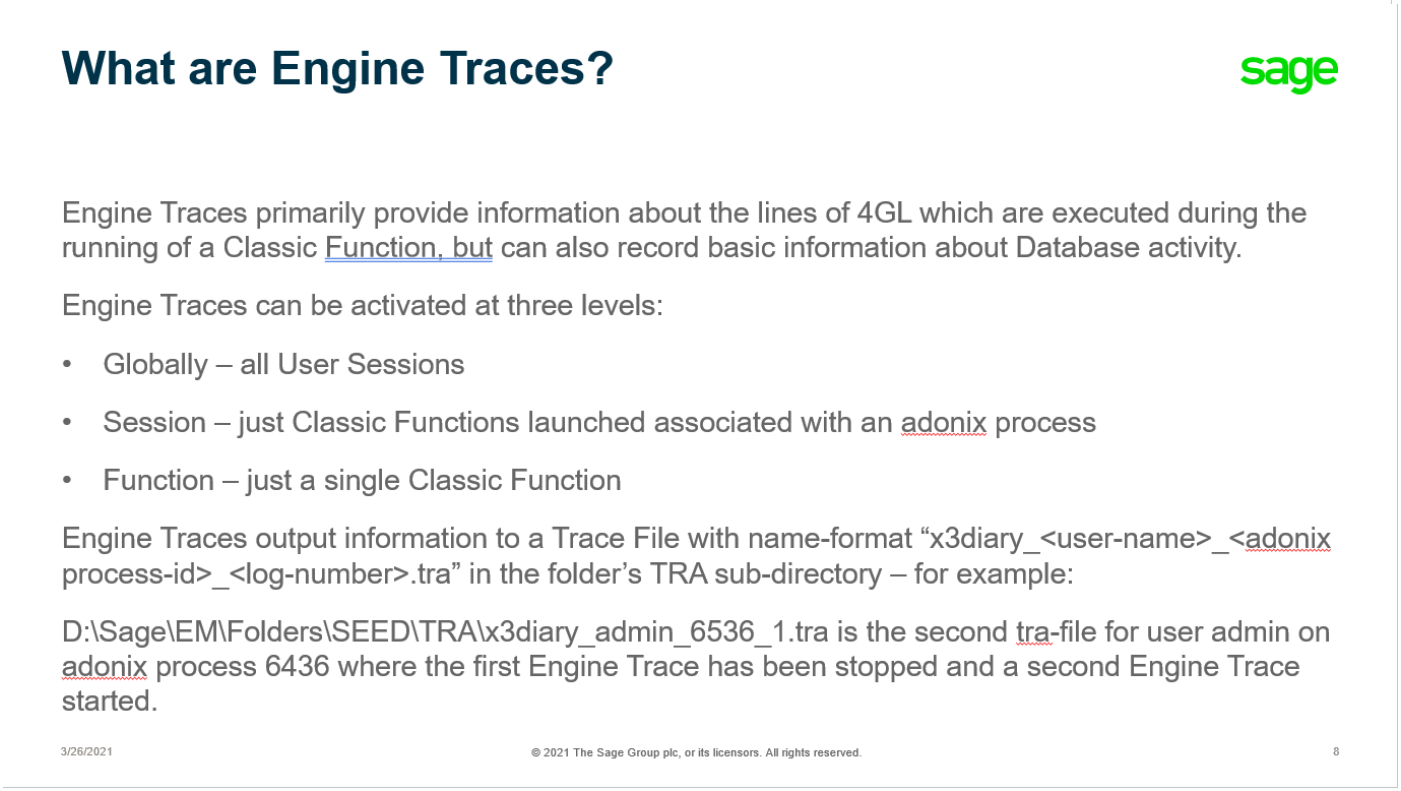
Sage X3 provides tracing mechanisms which can record the flow of execution - this can assist investigations into types 1 and 2, and also function timing and SQL traces which can assist with type 3 issues.

There are four types of tracing:

1. Functional “Engine Traces”
   This records control-flow within a Function such as what lines of code are executed, Subprog calls and while-loop iterations, as well as basic Database operations (Reads/Writes/Updates).

2. Flamegraph
   This shows the order in which blocks of code are called, and how the calls are nested.
   Simple Database interaction is also shown, in the form of light-blue blocks in the graph.
3. Timing Trace
   Information about the amount of time spent executing blocks of code (you get a summary and detailed output).
4. SQL Tracing
   This provides a record of SQL Statements executed during a process – no contextual information is recorded, purely SQL.





Engine Traces

Tracing can be activated at the global, session or function level.

These Engine Traces output to a file with name-format "x3diary_<user-name>_<adonix process-id>_<log-number>.tra" in the folder's TRA sub-directory – for example :

1) …\Folders\SEED\TRA\x3diary_admin_6536_0.tra is the first tra-file for user admin on adonix process 6436
2) …\Folders\SEED\TRA\x3diary_admin_6536_1.tra is the second tra-file for user admin on adonix process 6436 where the first Engine Trace has been stopped and a second Engine Trace started.

When starting an Engine Trace, you can specify a Flag value which denotes what type of event you wish to trace. The following table shows what sorts of events you can ask for:

Value	Element logged
1	Only the execution of the Gosub, Call, Callmet, and Fmet engine instructions.
2	Execution of all engine instructions.
4	Execution of the Read or For statements when they are used to access the database.
8	Technical data feed exchanged between the SAFE X3 engine and the database driver (sadora or sadoss query).
16	Technical JSON data feeds between the client and the web server.
32	Technical binary data feeds between the web server and the Sage X3 server for Classic pages.
64	Technical exchanges between the LDAP server and the Sage X3 server.
128	Only the execution of the Opldap instruction (Technical).
256	Technical exchanges linked to the runtime starting phase. (Technical)

Please note that these values can be added together in order to trace more than one type of event.

For example, flag-value 7 will record for events 1, 2, 3 and 4:

- Gosub and Call executions, and the corresponding returns
- Flow-control commands such as If-statements, Case, Next
- Database statements and for-loops related to them
- Filter and link commands
- Miscellaneous commands such as assignments (Assign) and initialisations (Raz), Datetime

Specifying flag-value 4 will only record events of type 4 – i.e. SQL Statements.

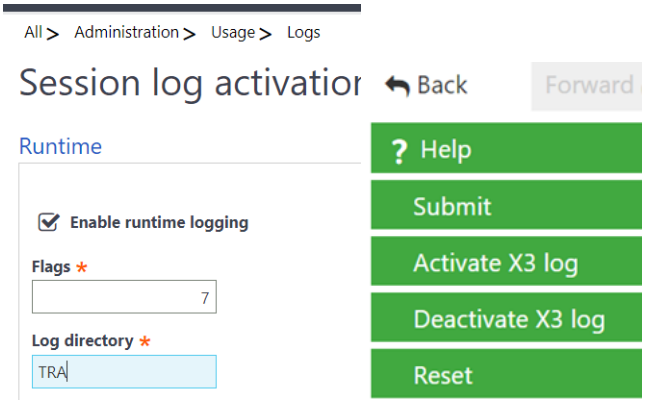
Specifying flag-value 5 will record events of type 1, 3 and 5.

Specifying flag-value 6 will record events of type 2, 3 and 4.

When looking at an Engine Trace file, you will see a “Channel” at the start of each line – this shows which sort of event it relates to.

Global Traces

Engine Traces can be enabled for all sessions via the **Administration > Usage > Logs > Engine Trace** function



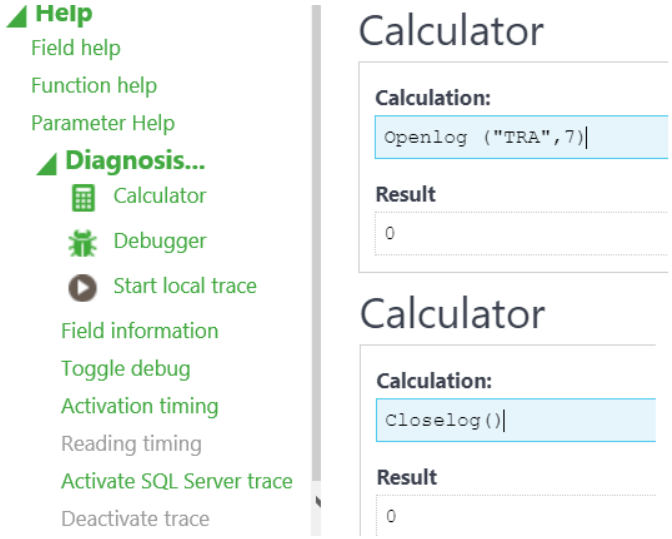
Provide a Flag-value and Volume for the tra-file.

This will enable Tracing for all sessions launched after it is submitted – when a Function is launched, it envokes an adonix process and that is the numeric part of the Trace file's name. This means, each browser session has its own Trace File.

Session Level Tracing

This can be done using Openlog/Closelog mechanism where tracing is initiated by the User from within a Classic Function.

Openlog and Closelog are executed in the **Help > Diagnosis > Calculator** which is available in all Classic Functions.



OpenLog () expects an X3 Volume and a Flag value.

This mechanism allows a User to specify the same Flag Values as are available in the Global Tracing, but it is the responsibility of the User to start and stop the Tracing, and it is only for the current Session.

Once the Tracing has been started, it will continue and will record the events for all the Classic Functions used on a Tab with the same adonix process until it is closed, or until a Closelog() is issued.

Note that the Tracing only applies to the Browser Tab in which it is initiated – if a User starts additional secondary Tabs, then the Trace will not record any activity in any Classic Functions used in the Secondary Tabs.

Any activity on other Browsers, on the same or different Client Machines, will not be recorded.

This level of tracing could be used when you're trying to trace a Function *before* the screens are loaded – you're not able to enable Tracing until the screens are loaded.

The procedure would be

1. Load a Classic Function
2. Enable Tracing
3. Exit the Classic Function
4. Load the Classic Function under investigation
5. Run through the task
6. Disable the Tracing – you may need to load another Classic Function to do this

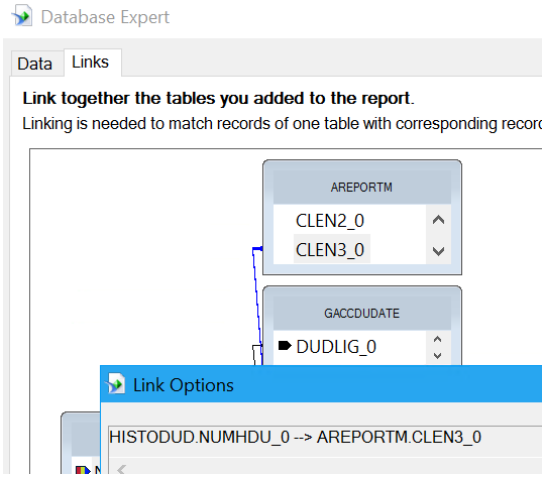
Function Level Tracing

This is the same as Session-level Tracing except that the Openlog and Closelog calls are done solely within a Classic Function – i.e. the Closelog is done before the User exits the Function and no other Functions are launched during the period of Tracing.

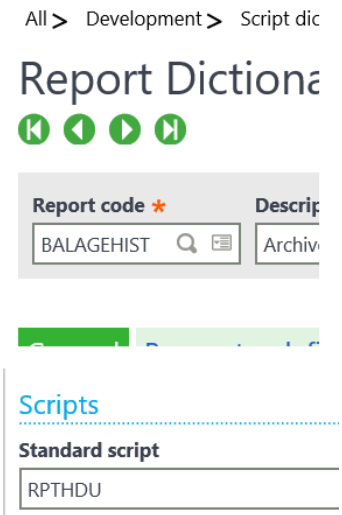
Example of Engine Tracing

BALAGEHIST Report is not producing the expected output.

On examining Crystal Report definition, one can see it relies on AREPORTM.

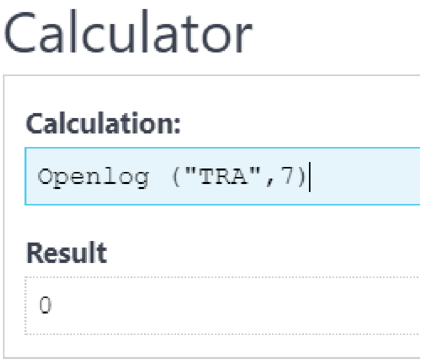


This table must be populated by the Script associated with the Report (as defined in Development > Script Dictionary > Reports) – in this case RPTHDU.

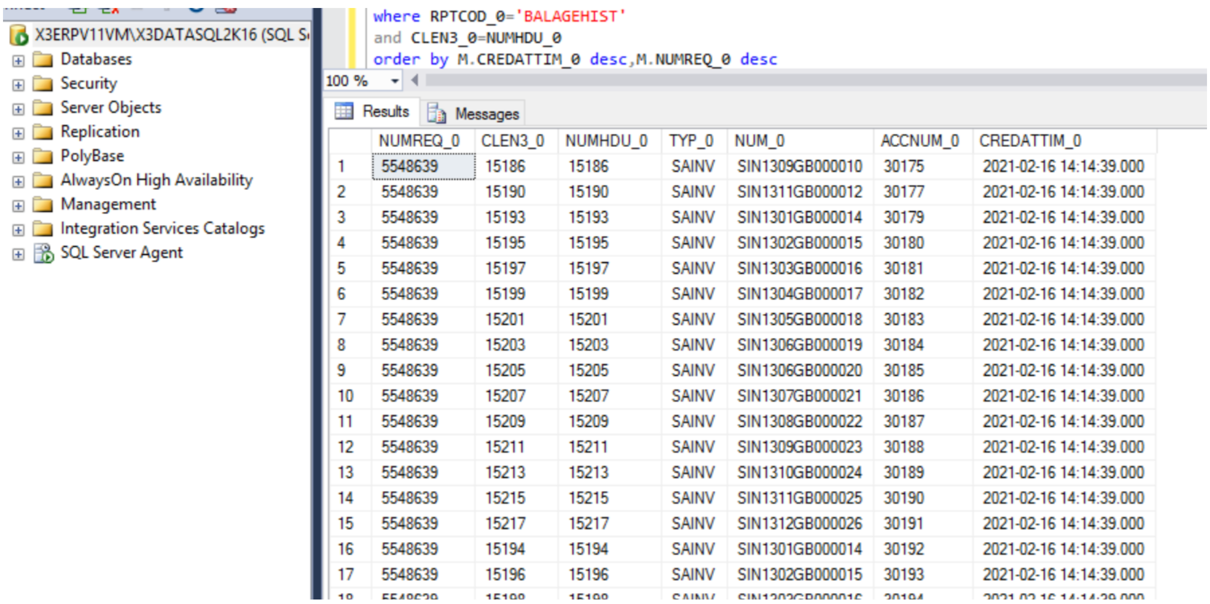


RPTHDU.src has several conditions and routes to write to AREPORTM, so use Openlog to record what route the program takes.

Using Flag 7, this shows basic SQL activity which can then be corelated with SQL Profiler and the Source-code.



Check AREPORTM table to see what data RPTHDU has selected:



	NUMREQ_0	CLEN3_0	NUMHDU_0	TYP_0	NUM_0	ACCNUM_0	CREDATTIM_0
1	5548639	15186	15186	SAINV	SIN1309GB000010	30175	2021-02-16 14:14:39.000
2	5548639	15190	15190	SAINV	SIN1311GB000012	30177	2021-02-16 14:14:39.000
3	5548639	15193	15193	SAINV	SIN1301GB000014	30179	2021-02-16 14:14:39.000
4	5548639	15195	15195	SAINV	SIN1302GB000015	30180	2021-02-16 14:14:39.000
5	5548639	15197	15197	SAINV	SIN1303GB000016	30181	2021-02-16 14:14:39.000
6	5548639	15199	15199	SAINV	SIN1304GB000017	30182	2021-02-16 14:14:39.000
7	5548639	15201	15201	SAINV	SIN1305GB000018	30183	2021-02-16 14:14:39.000
8	5548639	15203	15203	SAINV	SIN1306GB000019	30184	2021-02-16 14:14:39.000
9	5548639	15205	15205	SAINV	SIN1306GB000020	30185	2021-02-16 14:14:39.000
10	5548639	15207	15207	SAINV	SIN1307GB000021	30186	2021-02-16 14:14:39.000
11	5548639	15209	15209	SAINV	SIN1308GB000022	30187	2021-02-16 14:14:39.000
12	5548639	15211	15211	SAINV	SIN1309GB000023	30188	2021-02-16 14:14:39.000
13	5548639	15213	15213	SAINV	SIN1310GB000024	30189	2021-02-16 14:14:39.000
14	5548639	15215	15215	SAINV	SIN1311GB000025	30190	2021-02-16 14:14:39.000
15	5548639	15217	15217	SAINV	SIN1312GB000026	30191	2021-02-16 14:14:39.000
16	5548639	15194	15194	SAINV	SIN1301GB000014	30192	2021-02-16 14:14:39.000
17	5548639	15196	15196	SAINV	SIN1302GB000015	30193	2021-02-16 14:14:39.000

Since data needs to be inserted into AREPORTM, look for Trbegin in the Trace-file (TrBegin stands for Transaction Begin).

Check the x3diary tra-file to see what code was run:

```
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(172)      | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(172)      | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(172)      | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(172)      | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(171)      | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(172)      | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(174)      | | | | | | | | Link
<channel 2>@X3.TRT/RPTHDU$adx(180)      | | | | | | | | Columns
<channel 2>@X3.TRT/RPTHDU$adx(182)      | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(183)      | | | | | | | | Assign
<channel 3>@X3.TRT/RPTHDU$adx(184)      | | | | | | | | Call DEBTRANS From  GLOCK, tick:560661
<channel 2>@X3.TRT/GLOCK$adx(82)      | | | | | | | | | Assign
<channel 2>@X3.TRT/GLOCK$adx(83)      | | | | | | | | | Assign
<channel 2>@X3.TRT/GLOCK$adx(84)      | | | | | | | | | Assign
<channel 2>@X3.TRT/GLOCK$adx(85)      | | | | | | | | | Assign
<channel 2>@X3.TRT/GLOCK$adx(86)      | | | | | | | | | End
<channel 1>@X3.TRT/RPTHDU$adx(184)      | | | | | | | | End Call, tick:560661
<channel 2>@X3.TRT/RPTHDU$adx(185)      | | | | | | | | Trbegin
```

```
<channel 2>@X3.TRT/RPTHDU$adx(189)                | | | | | | | | For
<channel 4>Execution SQL on For clause in @X3.TRT/RPTHDU$adx at line 189, tick : 560663
<channel 4>Select HDU_.ROWID, HDU_.NUMHDU_0, HDU_.DATEVT_0, HDU_.TYP_0, HDU_.NUM_0, HDU_.FCY_0, HDU_.SAC_0,
HDU_.BPR_0, HDU_.ACCNUM_0, HDU_.DUDLIG_0, HDU_.BPRPAY_0, HDU_.UPDTICK_0, HAE_.CAT_0, HAE_.FLGREP_0, FGR_.CPY_0,
ACC_.CONSUL_0, AFF_.PRFCOD_0 From SEED.HISTODUD HDU_ JOIN SEED.GACCENTRY HAE_ ON ((HAE_.TYP_0 = HDU_.TYP_0) AND
(HAE_.NUM_0 = HDU_.NUM_0)) JOIN SEED.FACGROUP FGR_ ON ((FGR_.CPY_0 =  ?) AND (FGR_.FCY_0 = HDU_.FCY_0)) LEFT OUTER
JOIN SEED.GRPSAC GSC_ ON ((GSC_.COA_0 =  ?) AND (GSC_.GRU_0 =  ?) AND (GSC_.SAC_0 = HDU_.SAC_0)) JOIN SEED.BPARTNER BPR_
ON ((BPR_.BPRNUM_0 = HDU_.BPR_0)) LEFT OUTER  JOIN SEED.ACCES ACC_ ON ((ACC_.USR_0 =  ?) AND (ACC_.CODACC_0 =
BPR_.ACS_0)) LEFT OUTER  JOIN SEED.AFCTFCY AFF_ ON ((AFF_.FCY_0 = HDU_.FCY_0) AND (AFF_.PRFCOD_0 =  ?) AND (AFF_.FNC_0 =
?)) Where FGR_.CPY_0 = ? And HDU_.DATEVT_0 <= ? And HAE_.FLGREP_0 <> ? And ACC_.CONSUL_0 = ? And AFF_.PRFCOD_0 <> ? And
((HAE_.CAT_0 =  ?) Or (HAE_.CAT_0 =  ?)) Order by HDU_.ACCNUM_0,HDU_.DUDLIG_0,HDU_.NUMHDU_0
<channel 4>Query end, tick : 560889
<channel 2>@X3.TRT/RPTHDU$adx(190)                | | | | | | | | Raz
<channel 2>@X3.TRT/RPTHDU$adx(191)                | | | | | | | | Filter
<channel 2>@X3.TRT/RPTHDU$adx(192)                | | | | | | | | For
<channel 4>Execution SQL on For clause in @X3.TRT/RPTHDU$adx at line 192, tick : 560889
<channel 4>Select HD2_.ROWID, HD2_.* From SEED.HISTODUD HD2_ Where HD2_.ACCNUM_0 = ? And HD2_.DUDLIG_0 = ? And
((HD2_.DATEVT_0 !=  ?) Or (HD2_.FLGCLE_0 =  ?)) Order by HD2_.ACCNUM_0,HD2_.DUDLIG_0,HD2_.NUMHDU_0
<channel 4>Query end, tick : 560905
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(244)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(244)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(244)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 3>@X3.TRT/RPTHDU$adx(247)                | | | | | | | | Gosub TR1  , tick:560905
<channel 2>@X3.TRT/RPTHDU$adx(273)                | | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(280)                | | | | | | | | | Return
<channel 1>@X3.TRT/RPTHDU$adx(247)                | | | | | | | | End Gosub, tick:560905
<channel 2>@X3.TRT/RPTHDU$adx(248)                | | | | | | | | Filter
<channel 2>@X3.TRT/RPTHDU$adx(249)                | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(190)                | | | | | | | | Raz
<channel 2>@X3.TRT/RPTHDU$adx(191)                | | | | | | | | Filter
<channel 2>@X3.TRT/RPTHDU$adx(192)                | | | | | | | | For
<channel 4>Execution SQL on For clause in @X3.TRT/RPTHDU$adx at line 192, tick : 560905
<channel 4>Select HD2_.ROWID, HD2_.* From SEED.HISTODUD HD2_ Where HD2_.ACCNUM_0 = ? And HD2_.DUDLIG_0 = ? And
((HD2_.DATEVT_0 !=  ?) Or (HD2_.FLGCLE_0 =  ?)) Order by HD2_.ACCNUM_0,HD2_.DUDLIG_0,HD2_.NUMHDU_0
<channel 4>Query end, tick : 560905
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(244)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(197)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(200)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(201)                | | | | | | | | Raz
<channel 2>@X3.TRT/RPTHDU$adx(202)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(203)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(204)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(205)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(206)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(207)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(208)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(209)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(210)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(211)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 2>@X3.TRT/RPTHDU$adx(195)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(215)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(233)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(234)                | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(235)                | | | | | | | | Assign
<channel 2>@X3.TRT/RPTHDU$adx(246)                | | | | | | | | Next
<channel 3>@X3.TRT/RPTHDU$adx(247)                | | | | | | | | Gosub TR1  , tick:560907
<channel 2>@X3.TRT/RPTHDU$adx(273)                | | | | | | | | | If
<channel 2>@X3.TRT/RPTHDU$adx(274)                | | | | | | | | | Write
<channel 2>@X3.TRT/ATRIFIL$adx(16)               | | | | | | | | | | Char
```

Judging by the output, a Transaction is started at code line 185 and the actual action is taken in subprog TR1 – potentially including or excluding data as appropriate.

So, the Trace shows that some records are rejected :

```
<channel 2>@X3.TRT/RPTHDU$adx(280)                | | | | | | | | | Return
```

While others are accepted and written to AREPORTM :

<channel 2>@X3.TRT/RPTHDU$adx(274) | | | | | | | | Write

If you have access to the code, this can be examined to see what conditions cause a record to be rejected from the Report – if you don't have access to the code, then it could be sent to Sage's X3 Customer Services to provide more information.

```
273   $TR1
274   If OK=1
275    Write [ARM]
276   If fstat
277    GOK = 0 : Call FSTA("ARM") From GLOCK
278   Endif
279    If GOK<1 : Goto AB_TR1 : Endif
280   Endif
281   Return
```

Please remember to do the Closelog() to turn the logging off after the task has been completed.

What are Flamegraph Traces?

Flamegraph Traces are very similar to Engine Traces but provide a visual representation of the Subprog and Label calls executed during the running of a Classic Function.

Flamegraph Traces output information in two forms:

- A Trace File with name-format “x3diary_<user-name>_<adonix process-id>_<log-number>.tra” in the folder’s TRA sub-directory – for example D:\Sage\EM\Folders\SEED\TRA\x3diary_admin_6536_0.tra.
- A browser-based Flamegraph representation of the Trace File accessed via the Browser.

Flamegraph

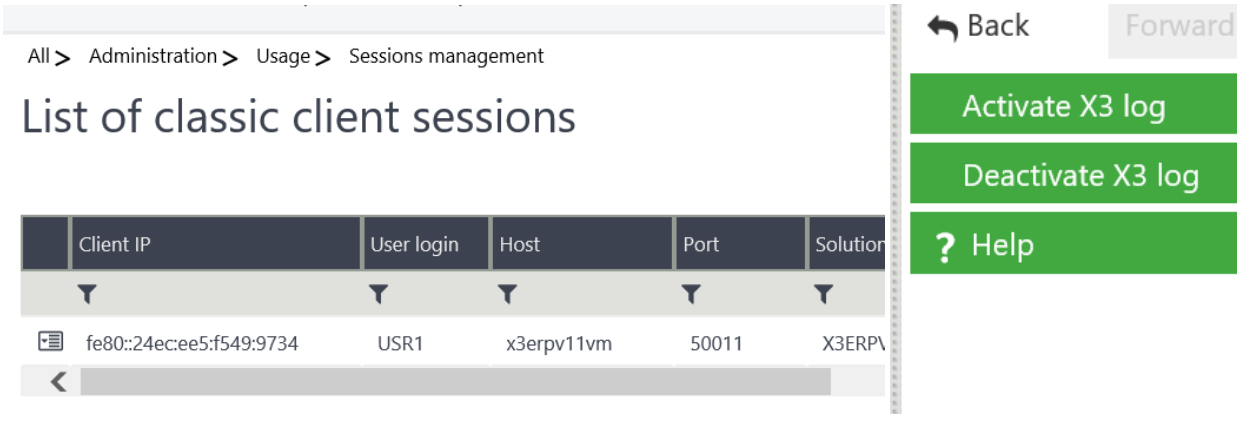
Flamegraph is a mechanism for exploring the time/process consumption of a classic Function, showing the order in which the Sub Programs are called, and also showing any nested Calls.

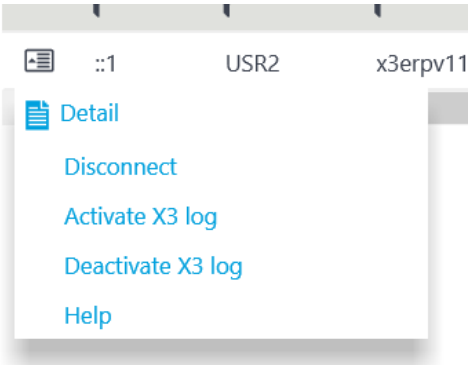
Flamegraphs are activated within the **Administration > Usage > Session Management > Classic Client sessions** option.

How to turn on Flamegraph Tracing:

1. Administration-->Usage-->Classic Client Sessions
2. Click Activate X3 Log Button
3. Open a secondary Session in another Tab and launch the Classic Function under investigation
4. Carry out the task in question – do not exit the Function
5. Press F5 to refresh the Classic Client Sessions session-list
6. Open the session you see here (wait until it fully loads as it takes extra load time to compute the Flamegraph we want to get)
7. Once the page is fully loaded, Click on the CPU Graph
8. Save this image to your disk (right click, save as..)

Flamegraph also generates an x3diary tra-file in the folder’s TRA sub-directory, including the SQL queries that are used

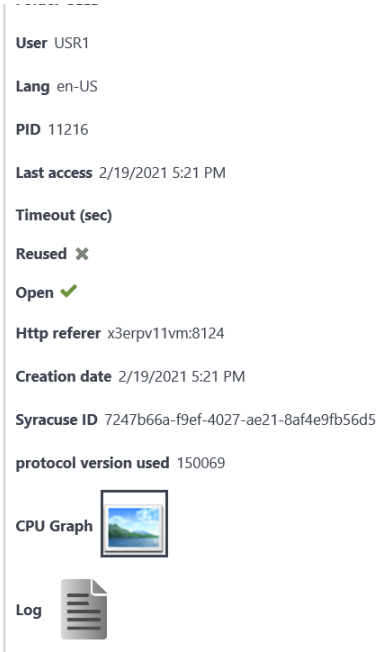




Please note that the tracing is only active for Classic Functions launched on the Browser which Activated the logging – if three Classic Functions are opened on secondary Tabs of the Browser, then a Flamegraph-trace will be generated for each one of those, but any Classic Functions on other Browser will not have Flamegraph-traces generated.

Flamegraph Traces will be generated for all Classic Functions on the Browser session until logging is Deactivated.

The Flamegraph is only available while the Classic Function is open – although the tra-file continues to exist after it is closed - any attempt to access/refresh the Flamegraph will fail after the function is exited. Therefore, it is essential to save a copy of the Flamegraph before closing the Classic Function in question – this can be done using the Browser's builtin "Save as…" feature which is made available on right-clicking within the body of the Flamegraph.



This Trace generates an x3diary tra-file for the information analysed in the Flameraph – it can be found in the Folder's TRA sub-directory – in this example, SEED\TRA\x3diary_usr1_8772.tra.

The consumption can be seen in graphical form by clicking the CPU Graph icon.

The Graph also shows any Database I/O in the context of the Subprog.

Since Flamegraph reads the x3diary tra-file during the generation of the Graph, you can see the Flamegraph change during the lifecycle of the function being examined – the graph can be updated by refreshing the browser-tab with the graph in it.

Thus, several snapshots of a graph can be taken and even saved in the svg format for inspection at a later date.

For example: Sales > Orders > Orders

User USR1

Lang en-US

PID 11216

Last access 2/19/2021 5:21 PM

Timeout (sec)

Reused ✖

Open ✔

Http referer x3erpv11vm:8124

Creation date 2/19/2021 5:21 PM

Syracuse ID 7247b66a-f9ef-4027-ae21-8af4e9fb56d5

protocol version used 150069

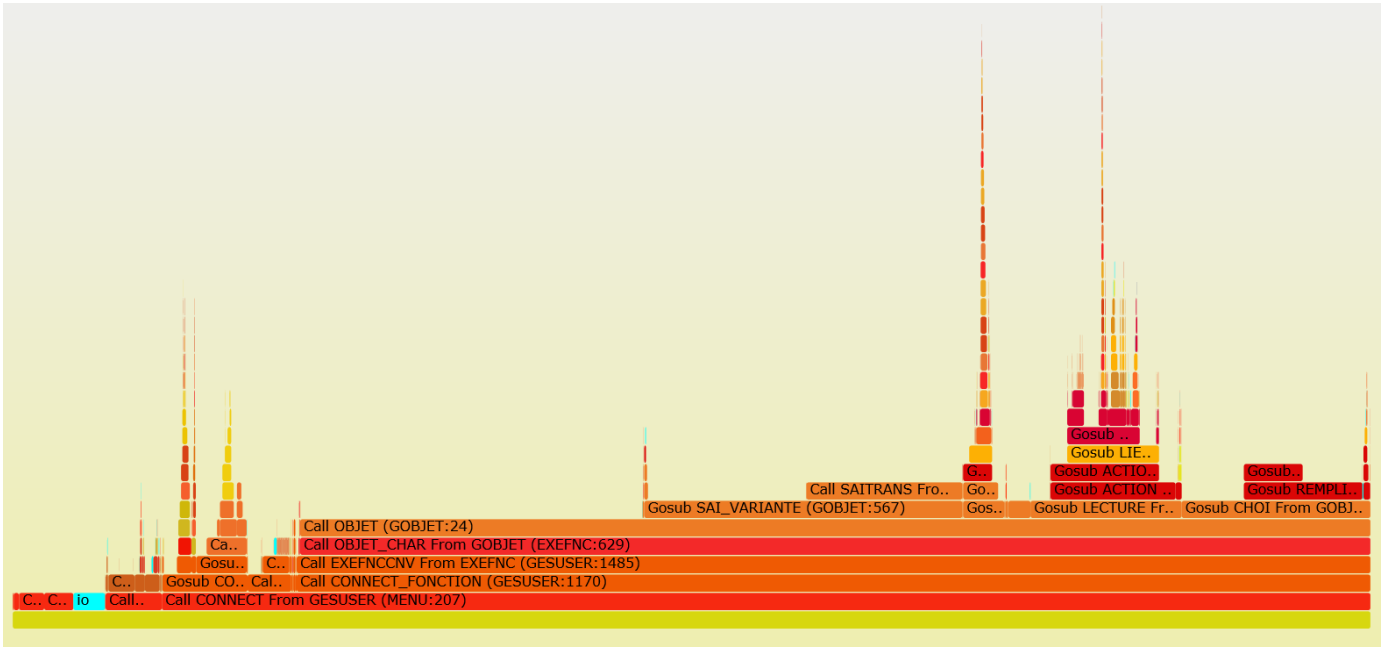
CPU Graph 

Log 

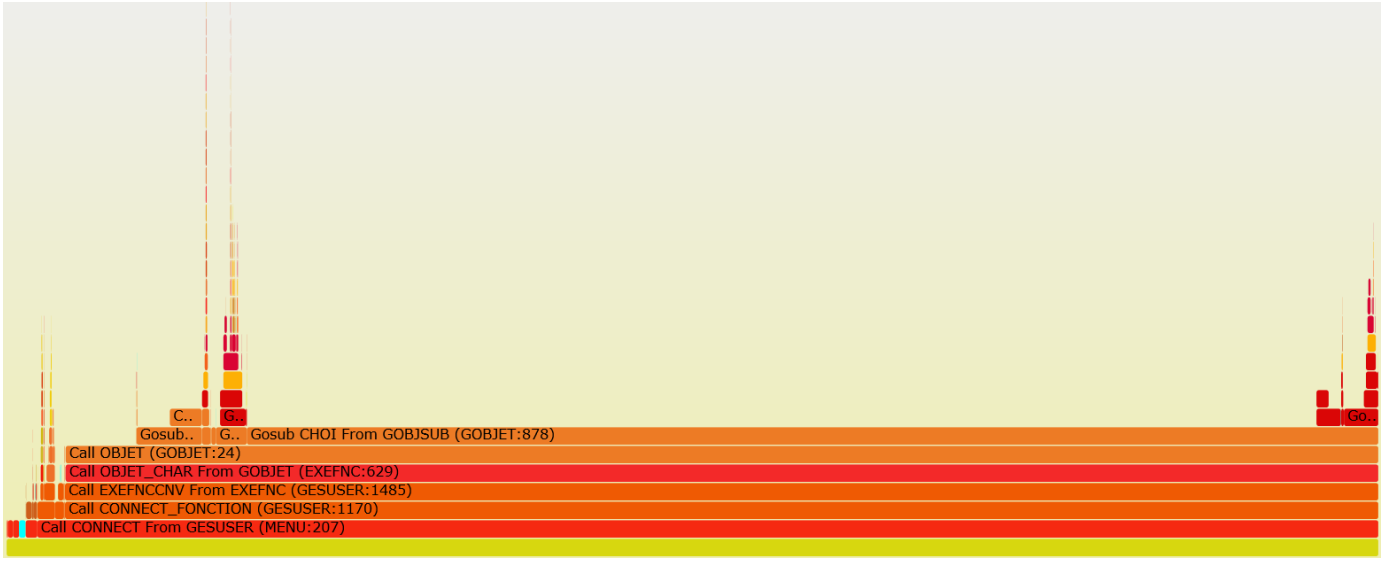
Associated Process : 11216.

Trace file name SEED\TRA\X3diary_admin_11216_0.tra

Snapshot 1:



Snapshot 2



Details of the time spent in each $-label and Subprog can be found by hovering over the appropriate bar, and this will correspond to the section of tra-file recording progress through the code.

As mentioned, the Flamegraph is based on the x3diary tra-file. This means, you can pin-point a section of the Flamegraph and then examine the corresponding section of the tra-file to see more detail.

For example, the section Call CONNECT From GESUSER (MENU:207) corresponds to the following lines from the x3diary tra-file

```
6500 <channel 4>Execution SQL on Read clause in @X3.TRT/APRODUCT$adx at line 78, tick : 776
6501 <channel 4>Select ADW2_.ROWID, ADW2_.* From X3.ADOVAL ADW2_ Where ( ADW2_.CMP_0 = ? And ADW2_.FCY_0 = ? And ADW2_.PARAM_0 = ? ) Or
6502 <channel 4>Query end, tick : 777
6503 <channel 1>@X3.TRT/APRODUCT$adx(132)                    |  |  | End Func, tick:777
6504 <channel 3>@X3.TRT/APRODUCT$adx(133)                    |  |  | Gosub VERSABR From  AGESW33, tick:777
6505 <channel 1>@X3.TRT/APRODUCT$adx(133)                    |  |  | End Gosub, tick:777
6506 <channel 1>@X3.TRT/GLOBVAR$adx(1069)                    |  | End Func, tick:777
6507 <channel 1>@X3.TRT/GLOBVAR$adx(175)                     | End Gosub, tick:777
6508 <channel 1>@X3.TRT/MENU$adx(190)                        End Call, tick:777
6509 <channel 3>@X3.TRT/MENU$adx(207)                        Call CONNECT From  GESUSER, tick:777
6510 <channel 3>@X3.TRT/GESUSER$adx(82)                       | Func FILL_ADS From  DOSSUB, tick:781
6511 <channel 3>@X3.TRT/DOSSUB$adx(534)                       |  | Func PARENTHISTFOLD  , tick:781
6512 <channel 4>Execution SQL on Read clause in @X3.TRT/DOSSUB$adx at line 516, tick : 782
6513 <channel 4>Select ADS__.ROWID, ADS__.* From X3.ADOSSIER ADS__ Where ADS__.DOSHIS_0 = ? Order by ADS__.DOSSIER_0 Option (FAST 1)
6514 <channel 4>Query end, tick : 783
6515 <channel 1>@X3.TRT/DOSSUB$adx(534)                       |  | End Func, tick:783
6516 <channel 4>Execution SQL on Read clause in @X3.TRT/DOSSUB$adx at line 548, tick : 783
6517 <channel 4>Select ADS_.ROWID, ADS_.* From X3.ADOSSIER ADS_ Where ( ADS_.DOSSIER_0 = ? ) Order by ADS_.DOSSIER_0 Option (FAST 1)
6518 <channel 4>Query end, tick : 783
6519 <channel 1>@X3.TRT/GESUSER$adx(82)                        | End Func, tick:784
6520 <channel 3>@X3.TRT/GESUSER$adx(92)                        | Func SEARCH_LANX3 From  ADS_TOOL, tick:784
6521 <channel 4>Execution SQL on For clause in @X3.TRT/ADS_TOOL$adx at line 116, tick : 786
6522 <channel 4>Select TLA_.ROWID, TLA_.* From SEED.TABLAN TLA_ Where (LOWER( TLA_.LANISO_0 ) =  ?) Order by TLA_.LAN_0 Option (FAST 1)
```

The following is a manually generated and formatted representation of calls and sub-calls in a section of the X3diary_admin_11216_0.tra file

Line 6509 : <channel 3>@X3.TRT/MENU$adx(207) Call CONNECT From GESUSER, tick:777
…….
Line 6600 : <channel 3>@X3.TRT/GESUSER$adx(892) | Gosub CONNECT_SYRACUSE , tick:797
…….
Line 6646 : <channel 3>@X3.TRT/GESUSER$adx(2783) | | Call CONTEXT_NEW From ASYRMAIN, tick:815
…….
Line 7227 : <channel 1>@X3.TRT/GESUSER$adx(2783) | | End Call, tick:850
…….
Line 15411 : <channel 1>@X3.TRT/GESUSER$adx(892) | End Gosub
…….
Line 38017 : <channel 1>@X3.TRT/MENU$adx(207) End Call, tick:1306163

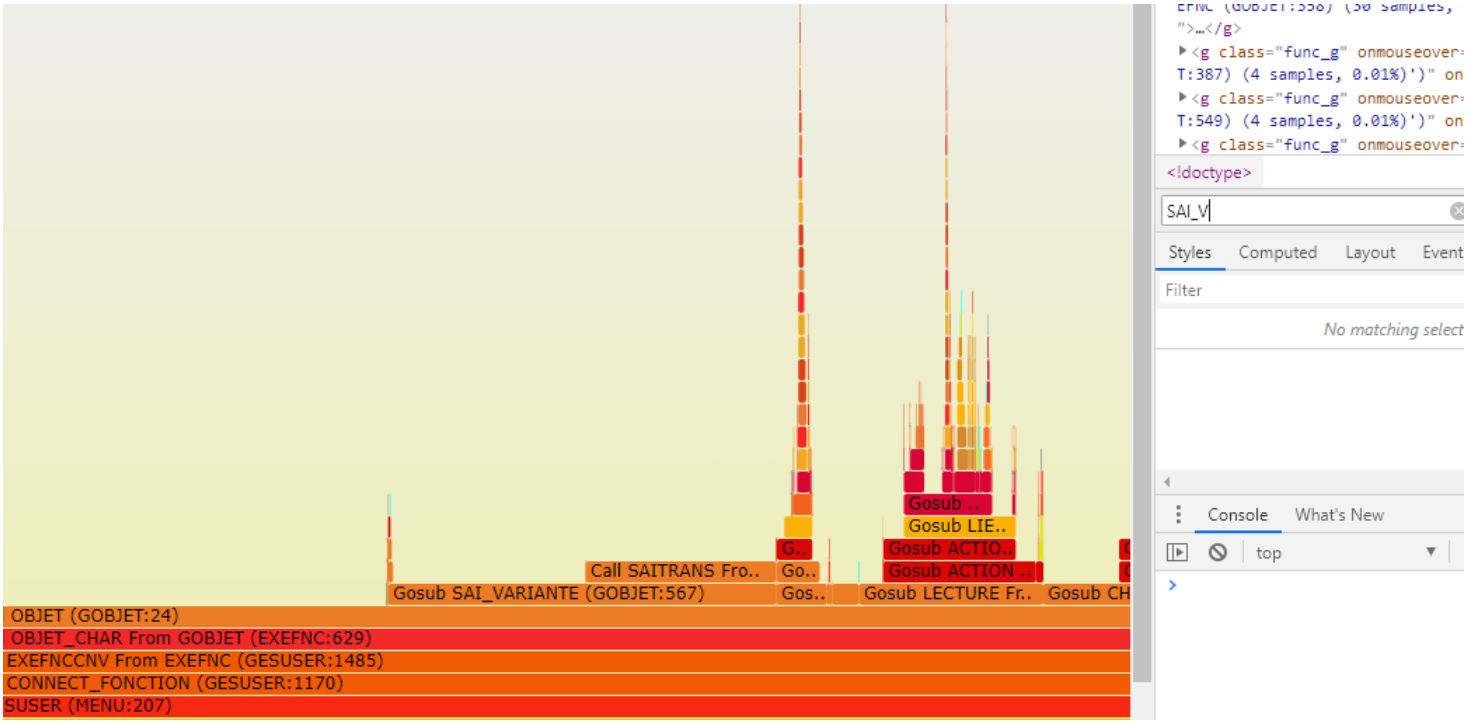
You can see from this, that the pipe-symbol indicates the call-level for each Call/Gosub so these line can be interpreted in the following way

Line	Origin	Level 1	Level 2	Level 3
6509	<channel 3>@X3.TRT/MENU$adx(207)	Call CONNECT From GESUSER		
6600	<channel 3>@X3.TRT/GESUSER$adx(892)		Gosub CONNECT_SYRACUSE	
6646	<channel 3>@X3.TRT/GESUSER$adx(2783)			Call CONTEXT_NEW From ASYRMAIN
7227	<channel 3>@X3.TRT/GESUSER$adx(2783)			End Call
15411	<channel 3>@X3.TRT/GESUSER$adx(892)		End Gosub	
38017	<channel 1>@X3.TRT/MENU$adx(207)	EndCall		

Knowing this, you could import the x3diary file into Excel with pipe-delimiter and you'd get a spreadsheet with lines like

2595	<channel 1>@X3.TRT/ADOVAL$adx(1263)			End Gosub, tick:329
2596	<channel 1>@SEED.TRT/WWAGLOBVAR$adx(598)		End Call, tick:329	
2597	<channel 3>@SEED.TRT/WWAGLOBVAR$adx(601)		Call PARAM From ADOVAL, tick:329	
2598	<channel 3>@X3.TRT/ADOVAL$adx(1263)			Gosub PARAM_ , tick:329
2599	<channel 4>Execution SQL on Read clause in @X3.TRT/ADOVAL$adx at line 1271, tick : 329			
2600	<channel 4>Select ADP_.ROWID, ADP_.* From SEED.ADOPAR ADP_ Where (ADP_.PARAM_0 = ?) Order by ADP_.PARAM_0 Option (FAST 1)			
2601	<channel 4>Query end, tick : 329			
2602	<channel 4>Execution SQL on Read clause in @X3.TRT/ADOVAL$adx at line 1287, tick : 330			
2603	<channel 4>Select ADW_.ROWID, ADW_.* From SEED.ADOVAL ADW_ Where (ADW_.CMP_0 = ? And ADW_.FCY_0 = ? And ADW_.PARAM_0 = ?) Order by ADW_.CMP_0,ADW_.FCY_0,ADW_.PARAM_0 Option (FAST 1)			
2604	<channel 4>Query end, tick : 330			
2605	<channel 1>@X3.TRT/ADOVAL$adx(1263)			End Gosub, tick:330
2606	<channel 1>@SEED.TRT/WWAGLOBVAR$adx(601)		End Call, tick:330	
2607	<channel 3>@SEED.TRT/WWAGLOBVAR$adx(604)		Call PARAM From ADOVAL, tick:330	
2608	<channel 3>@X3.TRT/ADOVAL$adx(1263)			Gosub PARAM_ , tick:330

As the Flamegraph is an SVG file, you can examine the page in a Browser and search for elements to see where they appear in the flow by using the Browser's built-in F12 Developer Tools.



When you search for a string in the body of the Graph, it will highlight it in the actual Graph (it pops-up a g.func_g over it).

You can also do a View Source to see all the information about the calls in written form.

```
<text text-anchor="" x="10" y="614" font-size="12" font-family="Verdana" fill="rgb(0,0,0)" id="details" > </text>
<g class="func_g" onmouseover="s('all (33274 samples, 100%)')" onmouseout="c()">
<title>all (33274 samples, 100%)</title><rect x="10.0" y="577" width="1180.0" height="15.0" fill="rgb(215,215,15)" rx="2" ry="2" />
<text text-anchor="" x="13" y="587.5" font-size="12" font-family="Verdana" fill="rgb(0,0,0)"  ></text>
</g>
<g class="func_g" onmouseover="s('Func EXISTE   (MENU:9) (18 samples, 0.05%)')" onmouseout="c()">
<title>Func EXISTE   (MENU:9) (18 samples, 0.05%)</title><rect x="10.0" y="561" width="0.6" height="15.0" fill="rgb(246,41,17)" rx="2" ry="2" />
</g>
<g class="func_g" onmouseover="s('Call DEFGLOBVAR1 From  GLOBVAR (MENU:67) (6 samples, 0.02%)')" onmouseout="c()">
<title>Call DEFGLOBVAR1 From  GLOBVAR (MENU:67) (6 samples, 0.02%)</title><rect x="10.6" y="561" width="0.3" height="15.0" fill="rgb(246,41,17)"
</g>
<g class="func_g" onmouseover="s('Call INIGLOBVAR1 From  GLOBVAR (MENU:68) (126 samples, 0.38%)')" onmouseout="c()">
<title>Call INIGLOBVAR1 From  GLOBVAR (MENU:68) (126 samples, 0.38%)</title><rect x="10.9" y="561" width="4.4" height="15.0" fill="rgb(246,41,1
</g>
<g class="func_g" onmouseover="s('Call INSTALL From  INSTALL (MENU:97) (4 samples, 0.01%)')" onmouseout="c()">
<title>Call INSTALL From  INSTALL (MENU:97) (4 samples, 0.01%)</title><rect x="15.3" y="561" width="0.2" height="15.0" fill="rgb(246,41,17)" rx
</g>
<g class="func_g" onmouseover="s('Call INIGLOBVAR From  GLOBVAR (MENU:190) (615 samples, 1.85%)')" onmouseout="c()">
<title>Call INIGLOBVAR From  GLOBVAR (MENU:190) (615 samples, 1.85%)</title><rect x="15.5" y="561" width="21.8" height="15.0" fill="rgb(246,41,
<text text-anchor="" x="18.461321151649937" y="571.5" font-size="12" font-family="Verdana" fill="rgb(0,0,0)"  >C..</text>
</g>
<g class="func_g" onmouseover="s('Call CONNECT From  GESUSER (MENU:207) (718 samples, 2.16%)')" onmouseout="c()">
<title>Call CONNECT From  GESUSER (MENU:207) (718 samples, 2.16%)</title><rect x="37.3" y="561" width="25.4" height="15.0" fill="rgb(246,41,17)"
<text text-anchor="" x="40.27114263388833" y="571.5" font-size="12" font-family="Verdana" fill="rgb(0,0,0)"  >C..</text>
</g>
<g class="func_g" onmouseover="s('io (777 samples, 2.34%)')" onmouseout="c()">
<title>io (777 samples, 2.34%)</title><rect x="62.7" y="561" width="27.6" height="15.0" fill="rgb(0,255,255)" rx="2" ry="2" />
<text text-anchor="" x="65.73366592534713" y="571.5" font-size="12" font-family="Verdana" fill="rgb(0,0,0)"  >io</text>
</g>
```

Timing Tracing

What are Timing Traces?

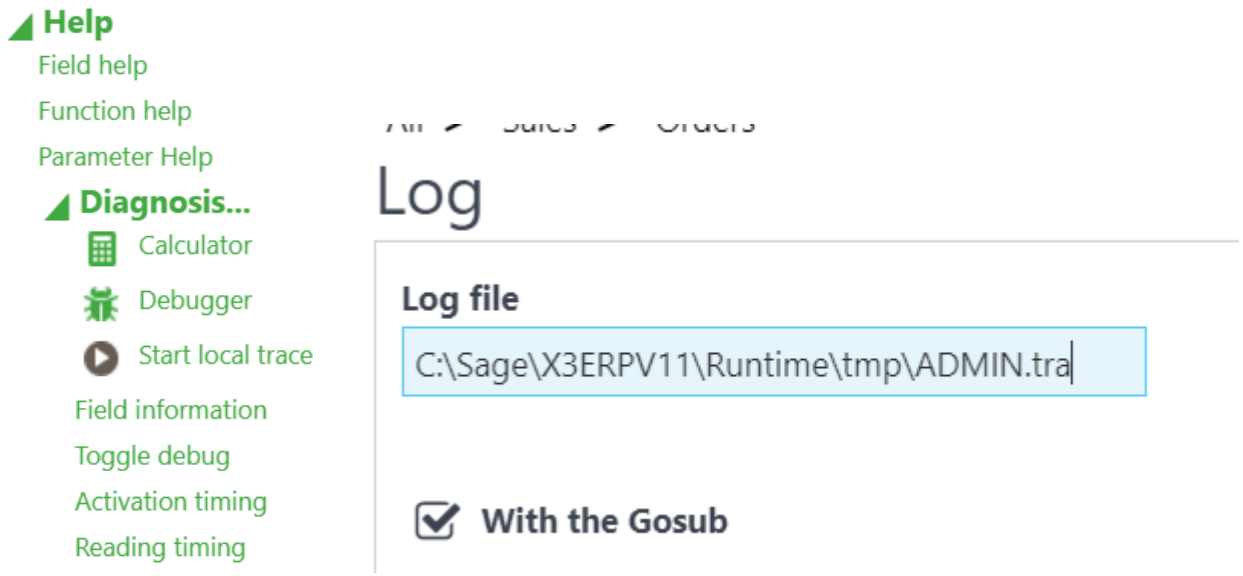
Timing Traces provide information about the amount of time spent in each block of 4GL Code which is executed during the running of a Classic Function.

Timing Traces produce two outputs:

- A Fxxxxx tra-file displayed in the Browser, summarising time spent in each block of code
- An x3diary tra-file detailing each execution of blocks of code

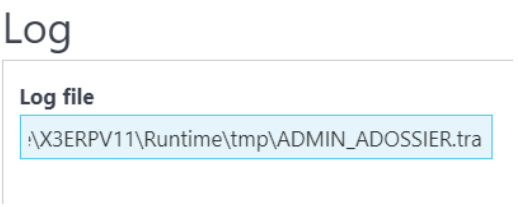
Timing Trace

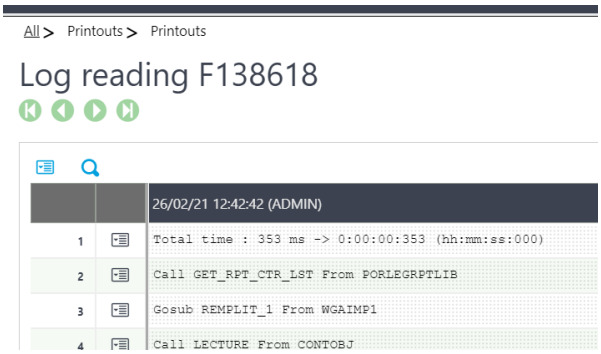
X3 can record the duration of calls in a Classic Functions by selecting **Help > Diagnostics > Activation timing** on the right-hand side of the Function



Once the steps under investigation have been carried-out, then select **Reading timing** and that will display a "normal" tra-file with F-prefix which in the folder's TRA sub-directory with a summary of the timings – the details are in the specified file-name, defaulting to <user-name>.tra in runtime\tmp folder – for example, …\runtime\tmp\ADMIN.tra – not the folder's TRA sub-directory.

This feature could even be used to trace the running of a Report – just make sure the Log-file name is unique/meaningful so it won't overwrite or be overwritten by other traces.





Sage Support can carry-out additional analysis on the two tra-files generated by this Timing Trace feature – send them in and ask for them to be run through X3Analyzer.

SQL Tracing

What is a SQL Trace?

SQL Traces provide information about Database Events during the running of a Classic Function.

It is similar to SQL Profiler.

SQL Traces can be activated at three levels:

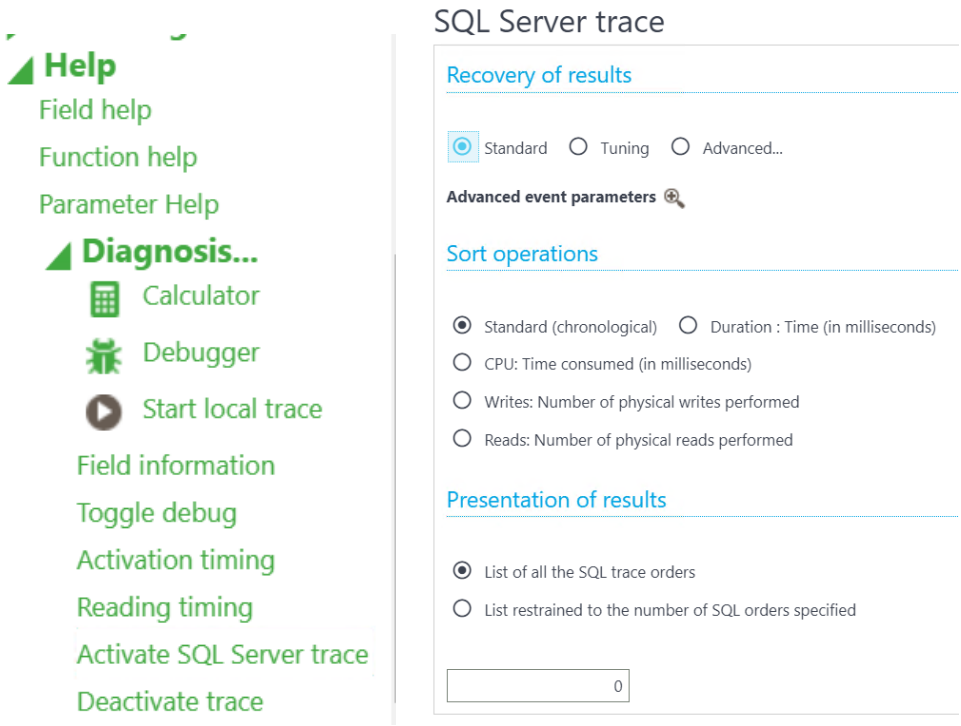
- Standard – basic Execution Plan statements
- Tuning – includes Cursor Prepare and Cursor Execute statements
- Advanced – allows you to specify SQL Events to be recorded

SQL Traces output in two forms:

- An x3diary tra-file detailing all the SQL Events during the running of a Classic Function
- A trc-file with the same information as the x3diary – it can be loaded into SQL Profiler.

SQL trace

As with Timing Traces, these can be activated/deactivated via **Help > Diagnosis > Activate SQL Server trace**



The SQL Trace is output in the form of a SQL Profiler trc file of the format SQLTRC67-<timestamp>.trc in the …\Folders\Database\trace directory – for example, d:\Sage\X3\Database\trace\SQLTRC67-2021-02-26T16-35-21-667.trc.

After deactivating the trace, a tra-file is generated in the folder's TRA sub-directory based on the information in the trc-file.

The .trc file can be loaded into SQL Profiler where it's easier to examine – you can search for particular tables and it can also be saved to a Database Table and searched and filtered in SQL as well.

"Standard" Recovery of results

By default, **"Standard"** Recovery of results is used - the information recorded is basically the Execution Plans for selects, inserts and updates performed during the trace-time – this will be familiar to you if you have used SQL Profiler during investigations in the past. Unfortunately, the Execution Plans don't include the data – they are parameterised as seen in the following example:

```
Execution Tree
--------------
Table Insert(OBJECT:([x3erpv11].[X3].[AFCTCUR]), OBJECT:([x3erpv11].[X3].[AFCTCUR].[AFCTCUR_ROWID]),
OBJECT:([x3erpv11].[X3].[AFCTCUR].[AFCTCUR_AFU0]), SET:([x3erpv11].[X3].[AFCTCUR].[UPDTICK_0] = RaiseIfNullInsert([@P1]),[x3erpv11].[X3].[AFCTCUR].[F
CT_0] = RaiseIfNullInsert([Expr1004]),[x3erpv11].[X3].[AFCTCUR].[UID_0] = RaiseIfNullInsert([Expr1005]),[x3erpv11].[X3].[AFCTCUR].[USR_0] =
RaiseIfNullInsert([Expr1006]),[x3erpv11].[X3].[AFCTCUR].[LOGIN_0] = RaiseIfNullInsert([Expr1007]),[x3erpv11].[
X3].[AFCTCUR].[MODULE_0] = RaiseIfNullInsert([@P6]),[x3erpv11].[X3].[AFCTCUR].[CREDAT_0] =
RaiseIfNullInsert([Expr1008]),[x3erpv11].[X3].[AFCTCUR].[CREUSR_0] = RaiseIfNullInsert([Expr1009]),[x3erpv11].[X3].[AFCTCUR].[UPDUSR_0] =
RaiseIfNullInsert([Ex
pr1010]),[x3erpv11].[X3].[AFCTCUR].[CREDATTIM_0] = RaiseIfNullInsert([Expr1011]),[x3erpv11].[X3].[AFCTCUR].[UPDDATTIM_0] =
RaiseIfNullInsert([Expr1012]),[x3erpv11].[X3].[AFCTCUR].[AUUID_0] = RaiseIfNullInsert([@P12]),[x3erpv11].[X3].[AFCTCUR].[ROWID]
 = [Expr1003]))
  !--Compute Scalar(DEFINE:([Expr1004]=CONVERT_IMPLICIT(nvarchar(80),[@P2],0), [Expr1005]=CONVERT_IMPLICIT(nvarchar(12),[@P3],0),
[Expr1006]=CONVERT_IMPLICIT(nvarchar(5),[@P4],0), [Expr1007]=CONVERT_IMPLICIT(nvarchar(20),[@P5],0), [Expr1008]=CONVERT_
IMPLICIT(datetime,[@P7],0), [Expr1009]=CONVERT_IMPLICIT(nvarchar(5),[@P8],0), [Expr1010]=CONVERT_IMPLICIT(nvarchar(5),[@P9],0),
[Expr1011]=CONVERT_IMPLICIT(datetime,[@P10],0), [Expr1012]=CONVERT_IMPLICIT(datetime,[@P11],0)))
      !--Compute Scalar(DEFINE:([Expr1003]=getidentity((98151445),(5),NULL)))
           !--Constant Scan
```

"Tuning" Recovery of results

If you select **"Tuning"** for Recovery of results, you can get a bit more information – you get to see cursor-prepare statements such as

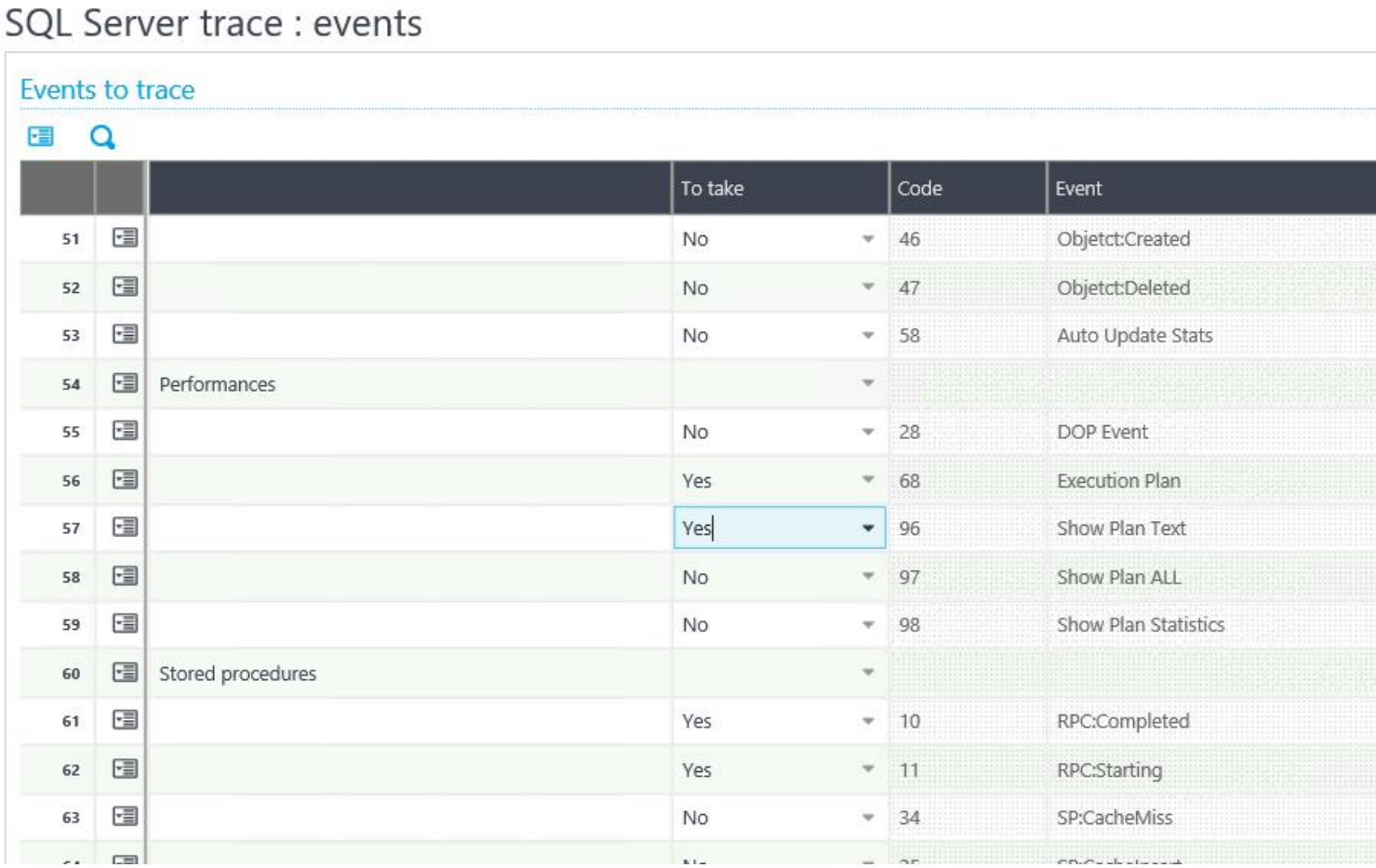
```
declare @p1 int
set @p1=1073742628
declare @p2 int
set @p2=180161391
declare @p5 int
set @p5=16
declare @p6 int
set @p6=1
declare @p7 int
set @p7=0
exec sp_cursorprepexec @p1 output,@p2 output,N'@P1 nvarchar(16),@P2 nvarchar(21),@P3 smallint',N'Select TPT_.ROWID, TPT_.*
From SEED.TABPAYTERM TPT_
Where ( TPT_.PTE_0 = @P1 And TPT_.LEG_0 = @P2 And TPT_.PTELIN_0 = @P3 )
Order by TPT_.PTE_0,TPT_.LEG_0,TPT_.PTELIN_0
Option (FAST 1)',@p5 output,@p6 output,@p7 output,N'CH30NET',N'BRI',1
select @p1, @p2, @p5, @p6, @p7
```

And then you can search for subsequent occurrences of that cursor:

```
exec sp_cursorexecute 1073742628,@p2 output,@p3 output,@p4 output,@p5 output,N'CH30NET',N' ',1
select @p2, @p3, @p4, @p5
```

"Advanced" Recovery of results

Finally, by selecting Recovery of results "**Advanced**", you can stipulate event-types to trace – these are selected from the "Advanced event parameters" section. Unfortunately, the event-types which are pre-ticked are hard-coded to Standard and/or Tuning runs – in order to record any events during an Advanced run, they must be marked as "Yes" in the "To take" column of the Advanced event parameters screen after clicking on the Advanced button in the Recovery of results section.



		To take	Code	Event
51		No	46	Objetct:Created
52		No	47	Objetct:Deleted
53		No	58	Auto Update Stats
54	Performances			
55		No	28	DOP Event
56		Yes	68	Execution Plan
57		Yes	96	Show Plan Text
58		No	97	Show Plan ALL
59		No	98	Show Plan Statistics
60	Stored procedures			
61		Yes	10	RPC:Completed
62		Yes	11	RPC:Starting
63		No	34	SP:CacheMiss

For example F138641.tra/SQLTRC59-2021-03-01T17-25-07-497.trc – these show entries for multiple events such as RPC:Starting and RPC_Completed events which include the parameter-values to the SQL statement, and Execution Plan events which show the SQL and associated Execution Plans

[EventClass]=11 RPC:Starting [StartTime]=2021-03-01 17:34:55 exec sp_cursorexecute 1073742362,@p2 output,@p3 output,@p4 output,@p5 output,N'ADMIN',N'AUTOSEL' select @p2, @p3, @p4, @p5
[EventClass]=68 Execution Plan Nested Loops(Inner Join, OUTER REFERENCES:([Bmk1000])) !--Index Seek(OBJECT:([x3erpv11].[SEED].[ADOVALAUS].[ADOVALAUS_ADU1] AS [ADU_]), SEEK:([ADU_].[PARAM_0]=[@P2] AND [ADU_].[CODUSR_0]=[@P1]) ORDERED FORWARD) !--RID Lookup(OBJECT:([x3erpv11].[SEED].[ADOVALAUS] AS [ADU_]), SEEK:([Bmk1000]=[Bmk1000]) LOOKUP ORDERED FORWARD)
[EventClass]=10 RPC:Completed [StartTime]=2021-03-01 17:34:55 [EndTime]=2021-03-01 17:34:55 [Duration]=14263 exec sp_cursorexecute 1073742362,@p2 output,@p3 output,@p4 output,@p5 output,N'ADMIN',N'AUTOSEL' select @p2, @p3, @p4, @p5

The trace-files will record multiple RPC:Starting, Execution Plan and RPC:Completed events for each execution of a SQL Cursor – for example, Cursor 1073742170

[EventClass]=11 RPC:Starting [StartTime]=2021-03-01 17:34:55 exec sp_cursorexecute 1073742170,@p2 output,@p3 output,@p4 output,@p5 output,N'ZA002'
[EventClass]=68 Execution Plan [StartTime]=2021-03-01 17:34:55 Nested Loops(Inner Join, OUTER REFERENCES:([Bmk1000])) !--Index Seek(OBJECT:([x3erpv11].[SEED].[BPCUSTOMER].[BPCUSTOMER_BPC0] AS [BPC_]), SEEK:([BPC_].[BPCNUM_0]=[@P1]) ORDERED FORWARD) !--RID Lookup(OBJECT:([x3erpv11].[SEED].[BPCUSTOMER] AS [BPC_]), SEEK:([Bmk1000]=[Bmk1000]) LOOKUP ORDERED FORWARD)
[EventClass]=10 RPC:Completed [StartTime]=2021-03-01 17:34:55 [EndTime]=2021-03-01 17:34:55 [Duration]=8514 [CPU]=15 [Reads]=5 [Writes]=0 exec sp_cursorexecute 1073742170,@p2 output,@p3 output,@p4 output,@p5 output,N'ZA002'
[EventClass]=10 RPC:Completed [StartTime]=2021-03-01 17:34:57 [EndTime]=2021-03-01 17:34:57 [Duration]=367 [CPU]=0 [Reads]=3 [Writes]=0 exec sp_cursorexecute 1073742170,@p2 output,@p3 output,@p4 output,@p5 output,N'ZA011'
[EventClass]=10 RPC:Completed [StartTime]=2021-03-01 17:34:57 [EndTime]=2021-03-01 17:34:57 [Duration]=407

```
[CPU]=0
[Reads]=3
[Writes]=0
exec sp_cursorexecute 1073742170,@p2 output,@p3 output,@p4 output,@p5 output,N'ZA002'
```

These three Events will record Inserts, Updates and Deletes as well:

```
[EventClass]=10 RPC:Completed
[StartTime]=2021-03-01 17:35:04
[EndTime]=2021-03-01 17:35:04
[Duration]=315
[CPU]=0
[Reads]=5
[Writes]=0
exec sp_prepexec @p1 output,N'@P1 int,@P2 nvarchar(81),@P3 nvarchar(13),@P4 nvarchar(6),@P5
nvarchar(21),@P6 tinyint,@P7 datetime2,@P8 nvarchar(6),@P9 nvarchar(6),@P10 datetime2,@P11
datetime2,@P12 binary(16)',N'INSERT INTO X3.AFCTCUR (UPDTICK_0, FCT
_0, UID_0, USR_0, LOGIN_0, MODULE_0, CREDAT_0, CREUSR_0, UPDUSR_0, CREDATTIM_0, UPDDATTIM_0,
AUUID_0)  VALUES
(@P1,@P2,@P3,@P4,@P5,@P6,@P7,@P8,@P9,@P10,@P11,@P12)',1,N'SAIWRKPLN',N'1231',N'ADMIN',N'admin
',1,'1753-01-01 00:00:00',N'ADMIN',N'ADMIN','20
21-03-01 16:35:04','2021-03-01 16:35:04',0x6FB4F3C07CC83B4F9C79AFC74568B90F
```

Conclusions

When to use the different Tracing Mechanisms?

sage

- Engine Trace
  - Use this to show what code is being executed.
  - Use it in conjunction with SQL Profiler/Extended Events.
- Flamegraph
  - Use this to see the code-flow “at a glance”.
- Timing Trace
  - Use this to see what blocks of code are taking the time.
- SQL Trace
  - Use this instead of SQL Profiler/Extended Events if they are not available.

Conclusions

Suggestions on how/when to use the different Tracing Mechanisms

- Engine Trace
  - Use this to show what code is being executed.
  - Use it in conjunction with SQL Profiler/Extended Events.
- Flamegraph
  - Use this to see the code-flow “at a glance”.
- Timing Trace
  - Use this to see what blocks of code are taking the time and should be used in conjunction with SQL Profiler to see if particular SQL statements have long durations.
- SQL Trace
  - Use this instead of SQL Profiler/Extended Events if they are not available.
  - This only records SQL statements – you cannot see which piece of code is executing the statements – so it is of interest when you are investigating exactly which data is involved with the Function.

Appendix

Appendix 1: Another example of an Engine Trace

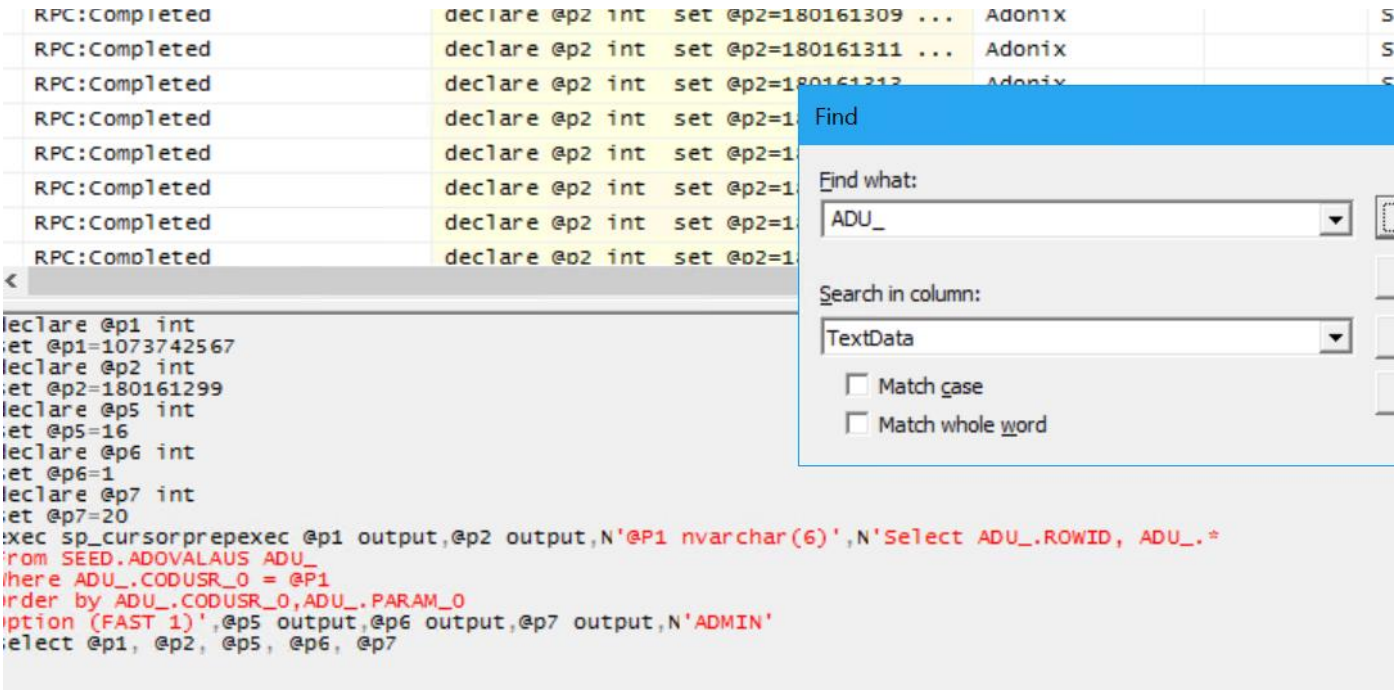
Engine Traces

- Engine Traces are good for investigating what code a function runs through, along with the basic SQL statements.
- It would be unusual to use Global-level tracing as all sessions are recorded and it would be difficult to manage and decide which tra-file to look at.
- Best to use Function-level Tracing if possible.
- Using Flag value 7 means you can see both the code and SQL statements – if you run SQL Profiler at the same time, then you can figure-out the parameter-values for the SQL in the tra-file
- Use SQL Profiler in conjunction with Engine Trace to try to discover the full SQL-statements and put them in context
- Enable Tracing for the shortest time possible.

Search for “select” in the x3diary tra-file

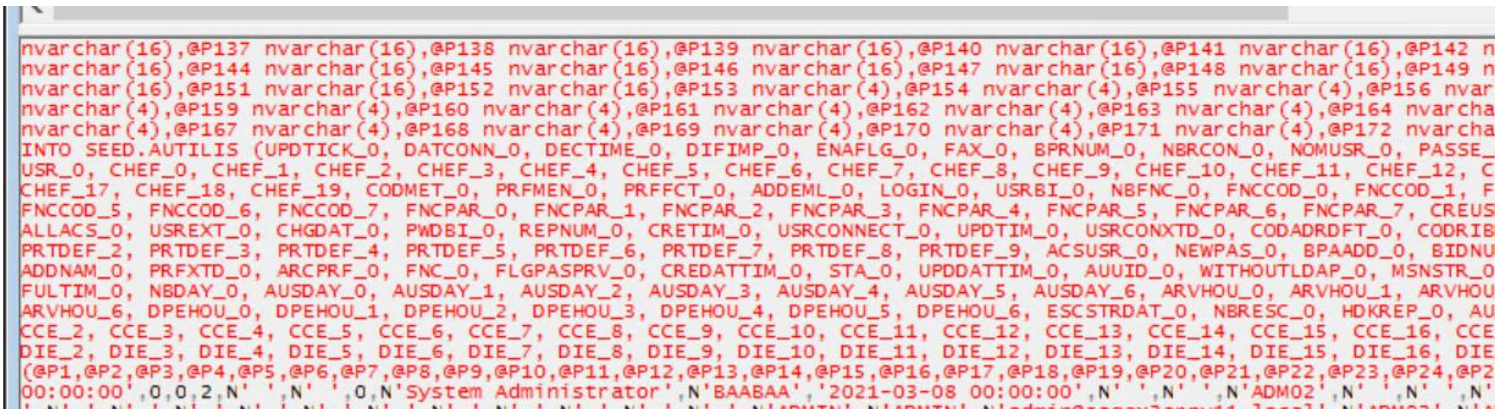
```
<channel 2>@X3.TRT/SUBAUS$adx(1892)                                                    | | | | | | | | | | | If
<channel 2>@X3.TRT/SUBAUS$adx(1895)                                                    | | | | | | | | | | | Read
<channel 4>Execution SQL on Read clause in @X3.TRT/SUBAUS$adx at line 1895, tick : 55703
<channel 4>Select ADU_.ROWID, ADU_.* From SEED.ADOVALAUS ADU_ Where ( ADU_.CODUSR_0 = ? And ADU_.PARAM_0 = ?
<channel 4>Query end, tick : 55708
<channel 2>@X3.TRT/SUBAUS$adx(1897)                                                    | | | | | | | | | | | If
<channel 2>@X3.TRT/SUBAUS$adx(1900)                                                    | | | | | | | | | | | If
```

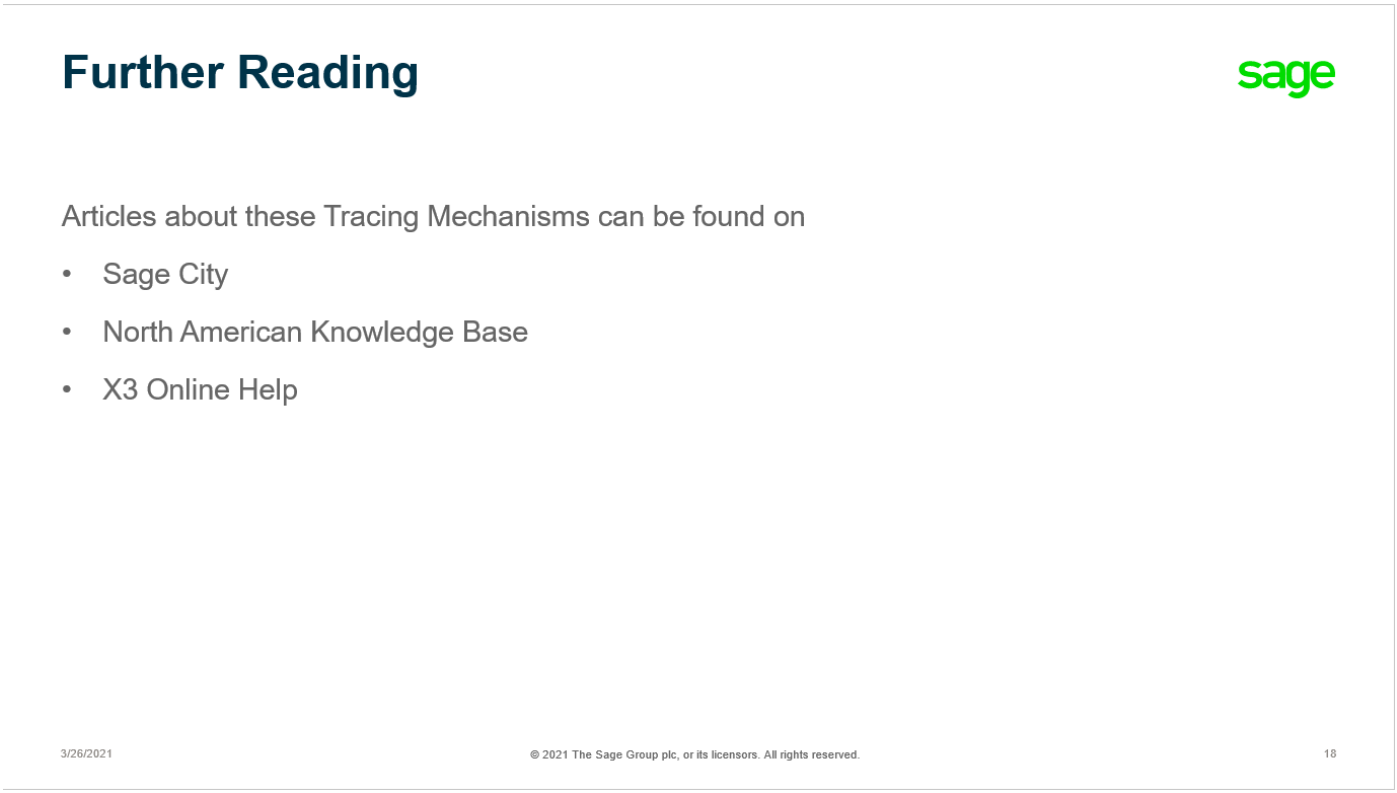
Then search for “ADU_” which is the alias for the SEED.ADOVALAUS table – this may show you the value of PARAM_0 in the above statement.



When looking for writes/re-writes in a x3diary tra-file search for Write/Rewrite. Unfortunately, the tra-file doesn't record which table is being updated, so you may have to refer to the code related to the Write

Look for INSERT in SQL Profiler:





Appendix 2: Further Reading

Engine Traces

Sage Knowledge Base Articles

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=73801 : How to run an Engine Trace in Sage X3?
https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=81615 : How do I run an Engine Trace from within a function (Openog/Closelog)?

Examples

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=109472 : Traceability results are incomplete.
https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=106695 : Address Overflow Error upon launching Sales Deliveries.

There are also KBs on how to initiate an Engine Trace with X3 Code:

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=83553

Sage City Articles

How to enable Runtime Logging to create a debug trace in V7? - Sage X3 Support - Sage X3 - Sage City Community
Sage X3 Support North America - Running an Engine Trace - YouTube
https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=97757

Online Help

https://online-help.sageerpx3.com/erp/12/ticket/how-to-generate-a-sql-server-trace/
https://online-help.sageerpx3.com/erp/12/staticpost/openlog/

Timing Traces

Sage Knowledge Base Articles

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=37655 : How to troubleshoot performance in Sage X3? (Run Activation Timing Trace utility)
https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=79316 : How to activate the timing trace for my program or subprogram in my code?

Examples

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=73759 : How to improve performance for any stock transaction.

Online Help

https://online-help.sageerpx3.com/erp/12/ticket/how-to-activate-the-timing-trace-for-a-progsubprog/

SQL Tracing

Sage Knowledge Base Articles

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=102484 : How to generate a SQL trace for one user session.
https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=81494 : How to run a SQL Profiler trace for one Sage ERP X3 User.

Online Help

https://online-help.sageerpx3.com/erp/12/ticket/how-to-generate-a-sql-server-trace/

Flamegraph

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=79383

https://online-help.sageerpx3.com/erp/12/staticpost/flamegraph-presentation/

General Tracing Tips

Sage Knowledge Base Articles

https://support.na.sage.com/selfservice/viewdocument.do?noCount=true&externalId=76348 : How to troubleshoot slow performance for Sage X3.

SQL Profiler / Extended Events

Please note that SQL Profiler is now deprecated and the advice is to use Extended Events in its place

https://docs.microsoft.com/en-us/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-ver15 : SQL Server Profiler
https://docs.microsoft.com/en-us/sql/relational-databases/extended-events/extended-events?view=sql-server-ver15 : Extended Events Overview

Thank you