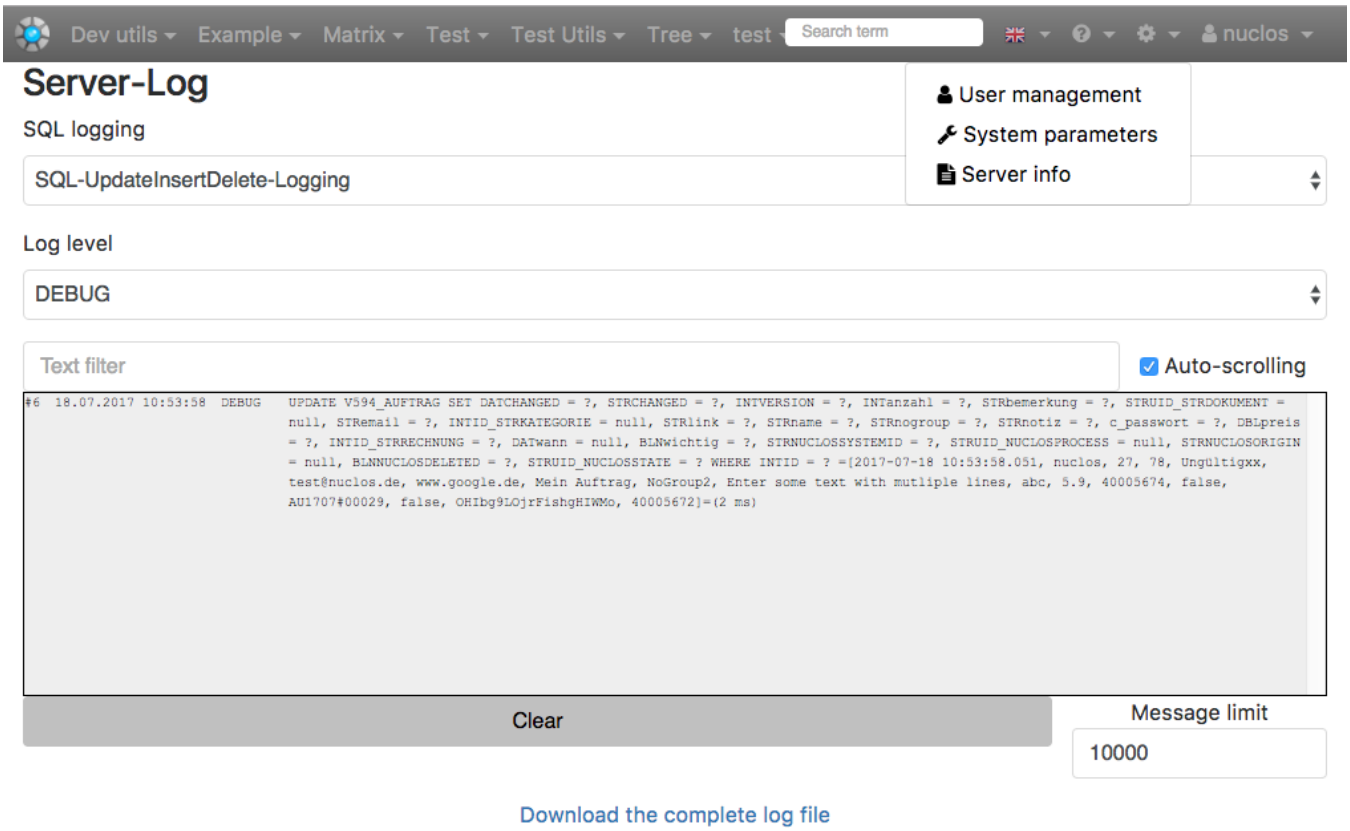
SQL Logging

SQL Logging

Ab 4.19.0 und 4.18.2: Im Webclient kann der Super-User das Logging während des Betriebs an- und ausschalten, rechts unter dem Zahnrad-Dropdown Menüpunkt "Server Info". Log level für den gewünschten Logger auf "DEBUG" setzen:



SQL Timer per REST einschalten:

```
curl --cookie-jar cookies.txt http://localhost:8080/nuclos-war/rest -X POST -H "Content-Type: application/json"
-d '{"username":"'nuclos'", "password":"'"}';
curl --cookie cookies.txt http://localhost:8080/nuclos-war/rest/maintenance/logging/SQLTimer/level -X PUT -H
"Content-Type: application/json" -d '{"level":"'debug'"}';
```

bzw. reines SQL Logging vor der Ausführung ohne Timer mit "SQLLogger" statt "SQLTimer" bzw. mit "SQLUpdate" für das ausschließliche Loggen von "UDPATE/INSERT/DELETE"-Statements. Zum Abschalten des Logging den Level wieder von "debug" auf "info" setzen.

Standard SQL Logging

Um alle SQL Statements zu loggen: So sieht die Konfiguration der log4j2.xml aus, wenn einfache SQLLogging (**vor** dem Ausführen des SQL Statements) eingeschalten werden soll:

log4j2 - Ausgabe des SQL-Logging im Server-Log

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d %p [%c] - %m%n"/>
    </Console>
    <RollingFile name="Logfile" fileName="<Nuclos-Log-Verzeichnis>/server.log"
            filePattern="<Nuclos-Log-Verzeichnis>/server-%i.log" append="true">
      <PatternLayout pattern="%d %p [%c] - %m%n"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="20"/>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Logger name="org.apache.log4j.xml" level="info"/>
    <Logger name="SQLLogger" level="debug"/>
        <Root level="info">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="Logfile"/>
    </Root>
  </Loggers>
</Configuration>
```

Die Datei log4j2.xml ist im "conf" Verzeichnis des Servers zu finden, zusammen mit den Dateien jdbc.properties und server.properties. Alternativ (oder additional) kann noch "SQLTiming" angeschalten werden:

```
    <Logger name="org.apache.log4j.xml" level="info"/>
    <Logger name="SQLTimer" level="debug"/>
```

Damit wird das SQL **nach** der Ausführung des Statements auf der Datenbank geloggt und die Ausführungszeit in Millisekunden mit angeben. Hier ist eine typische Ausgabe des SQLTimers:

```
2018-01-04 18:10:54,844 DEBUG [SQLTimer] - SELECT COUNT (t.INTID) FROM D2SC_FH_LASTPOSITION t WHERE 1=1
=[]=(2 ms)
```

SQL StackTrace Logging

Es kann zusätzlich noch der StackTrace, sowohl vom Server, als auch vom Client, bis zur Ausführung des SQL-Statements geloggt werden. Dafür kann der Textfilter unter "Server Info" im Webclient verwendet werden (siehe oben): Außerdem kann in der Klasse "DataSourceExecuter" ein String-Suchbegriff angegeben werden, entweder über einen Debugger durch setzen des Strings in folgender Methode

```
public static String getDebugSQL() {
        return "SELECT COUNT";
}
```

oder durch setzen aus dem Programm mit der statischen Methoden DataSourceExeceuter.setDebugSQL(String debugSQL)

```
DataSourceExeceuter.setDebugSQL("SELECT COUNT");
```

Dann werden für alle SQL Statements die "SELECT COUNT" beinhaltet zwei erweiterte Stacktraces ausgegeben:

```
2018-01-04 18:10:54,843 DEBUG [ServerStackTrace] - Server Stack Trace:
org.nuclos.server.dblayer.impl.DataSourceExecutor.logStackTrace(DataSourceExecutor.java:179)
org.nuclos.server.dblayer.impl.DataSourceExecutor.beforeExecution(DataSourceExecutor.java:139)
org.nuclos.server.dblayer.impl.DataSourceExecutor.execute(DataSourceExecutor.java:230)
org.nuclos.server.dblayer.impl.standard.StandardSqlDBAccess.executeQueryImpl(StandardSqlDBAccess.java:295)
org.nuclos.server.dblayer.impl.standard.StandardSqlDBAccess.executeQuery(StandardSqlDBAccess.java:274)
org.nuclos.server.dblayer.impl.standard.StandardSqlDBAccess.executeQuery(StandardSqlDBAccess.java:262)
org.nuclos.server.dblayer.impl.standard.StandardSqlDBAccess.executeQuerySingleResult(StandardSqlDBAccess.java:
400)
org.nuclos.server.dal.processor.jdbc.impl.EntityObjectProcessor.countExact(EntityObjectProcessor.java:647)
org.nuclos.server.dal.processor.jdbc.impl.EntityObjectProcessor.count(EntityObjectProcessor.java:625)
org.nuclos.server.common.ejb3.EntityObjectFacadeBean.countEntityObjectRows_aroundBody14(EntityObjectFacadeBean.
java:297)
org.nuclos.server.common.ejb3.EntityObjectFacadeBean$AjcClosure15.run(EntityObjectFacadeBean.java:1)
org.nuclos.server.common.ejb3.EntityObjectFacadeBean.countEntityObjectRows(EntityObjectFacadeBean.java:293)
org.nuclos.server.common.NuclosRemoteInvocationExecutor.invoke(NuclosRemoteInvocationExecutor.java:207)
org.nuclos.server.common.NuclosHttpInvokerServiceExporter.invokeAndCreateResult
(NuclosHttpInvokerServiceExporter.java:13)

2018-01-04 18:10:54,843 DEBUG [ServerStackTrace] - Client Stack Trace:
org.nuclos.client.remote.http.NuclosHttpInvokerProxyFactoryBean.executeRequest
(NuclosHttpInvokerProxyFactoryBean.java:93)
org.nuclos.common.cache.AllEntityObjectsCache.refill(AllEntityObjectsCache.java:125)
org.nuclos.common.cache.AllEntityObjectsCache.register(AllEntityObjectsCache.java:118)
org.nuclos.common.cache.AllEntityObjectsCache.register(AllEntityObjectsCache.java:105)
de.bmw.fdm.common.fh.gaal.LastfallCommonCache.init(LastfallCommonCache.java:76)
org.nuclos.client.main.StartUp$1.run(StartUp.java:388)
```

Hinweis: Der Client Stack Trace ist nur erhältlich, wenn der Server im Entwicklungsmodus (-Dfunctionblock.dev=true) gestartet worden ist.

SQL Logging in separates Log-File

Hier steht, wie man SQL Logging in separates Log-File konfiguriert.

SQL Logging für alte Nuclos-Versionen (bis 4.5)

Hier sind die Logging-Einstellung für Nuclos bis 4.5 zu finden: Altes SQL Logging (Bis Nuclos 4.5)