Mobile DB Monitor

Anleitung und Installationshinweise

infolytics

Mobile DB Monitor – Version 1.1 | Anleitung – Version 1.1

Infolytics AG
Bonner Str. 484–486
DE-50968 Köln
www.infolytics.com

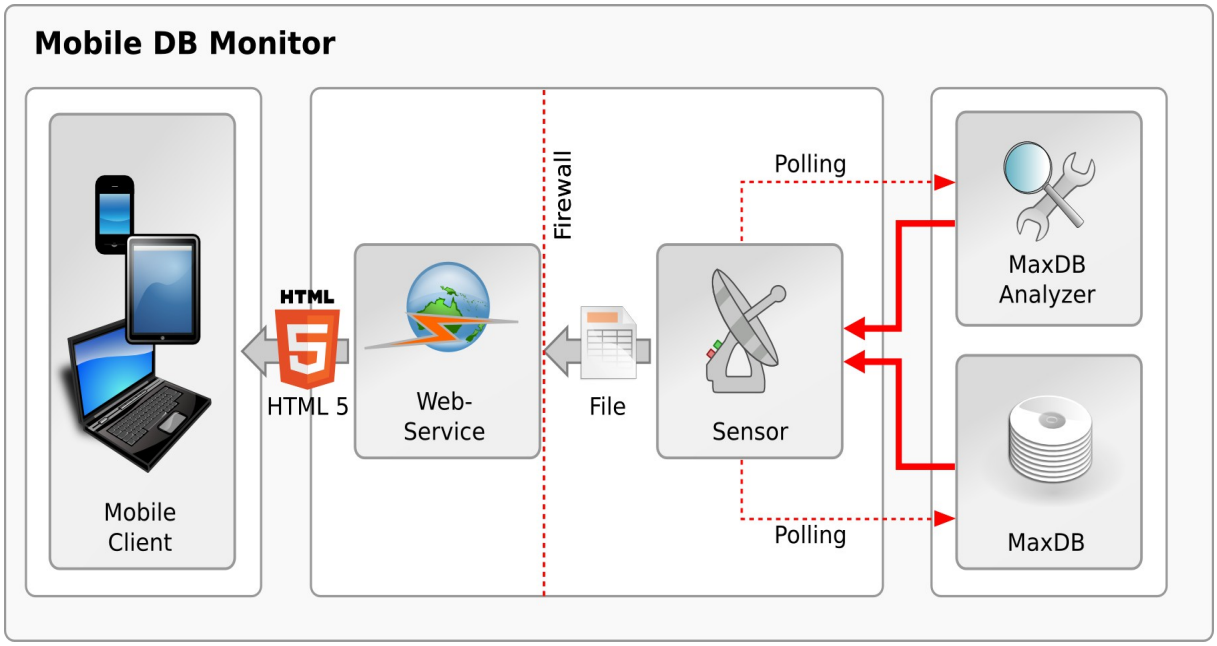
Inhaltsverzeichnis

1 Kurzer Überblick über die Module des MDBM

Dieses Dokument soll Ihnen die notwendigen Schritte für die Installation der einzelnen Module des *Mobile DB Monitor* – kurz: *MDBM* – erläutern.

Dabei handelt es sich um folgende Module:

- das Sensor-Modul
- das Webservice-Modul
- die HTML5-Anwendung



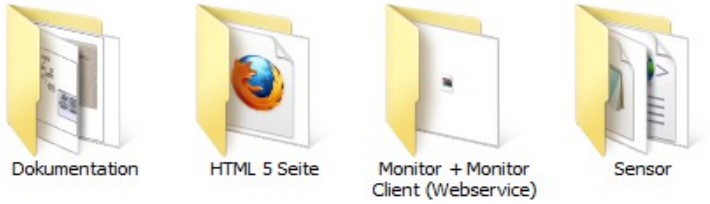
Der *Sensor* sammelt Daten über den Zustand der Datenbank und stellt diese in einer Datei zur Verfügung. Er hat außerdem die Möglichkeit (mittels *Log4J*) E-Mails zu verschicken. Wenn Ihnen dies ausreicht, dann benötigen Sie nur den Sensor.

Das *Webservice-Modul* liest die Datei ein und bereitet die Daten auf. Diese Daten werden dann nach außen durch den Webservice zur Verfügung gestellt. Der Webservice kann dann auf unterschiedliche Weise abgefragt werden. Eine Möglichkeit zur Abfrage bietet die mitgelieferte *HTML5-Seite*, die eigens für den MDBM erstellt wurde.

Um zu verhindern, dass ein Sensor in mehreren Instanzen läuft, wird die Datei `FileLock.lock` erzeugt. Diese sorgt dafür, dass Sie nur eine Instanz des Sensors starten können. Wenn sie eine weitere Instanz des Sensor ausführen wollen, kopieren sie einfach den Ordner, in dem sich der Sensor befindet, an eine andere Stelle und starten ihn von dort.

Hinweis:
Änderungen an den XML-Dateien werden erst nach einem Neustart des Sensors wirksam.

2 Vor der Installation



Nach Ihrer Registrierung auf unserer Download-Seite erhalten Sie Ihren 24 Stunden gültigen Download-Link.

Nach dem Herunterladen der Zip-Datei entpacken Sie diese bitte. Sie sollten nun vier Ordner vorfinden:

- Dokumentation (enthält auch dieses Dokument)
- HTML 5 Seite
- Monitor (Webservice)
- Sensor

3 Das Sensor-Modul

Der Inhalt des Ordners *Sensor* kann an einem beliebigen Ort abgelegt werden.

Dies kann/sollte ein sicherer Bereich in Ihrem Netz sein.

Der Sensor muss nur die Möglichkeit haben, eine Zustandsdsatei namens `State.txt` an einen Ort zu schreiben, auf den der Webservice Zugriff hat (sofern Sie diesen nutzen wollen).

Er benötigt natürlich auch Zugriff auf die Datenbanken, die er abfragen soll.

Wenn Sie *Log4j* entsprechend konfigurieren, kann der Sensor E-Mails verschicken. Wenn Ihnen dies schon ausreicht, ist es nicht nötig, den Webservice und die HTML5-Seite zu installieren.

Wie Sie das Log4J-Framework richtig konfigurieren, wird im Kapitel *Das Log4j-Framework* beschrieben.

Vor dem Starten des Sensor müssen Sie zunächst noch die Datei `def.xml` konfigurieren, damit der Sensor weiß, welche Datenbank er abfragen muss (Siehe hierzu das Kapitel *Konfiguration der Datei def.xml*).

4 Die Konfiguration des Sensors

Die Konfiguration des Sensors erfolgt mittels der Dateien `def.xml` und `log4j.xml`.

Die Datei `def.xml` konfiguriert den Sensor und somit sein Verhalten.

Die Datei `log4j.xml` konfiguriert das Verhalten des Loggers und seiner Appender (d.h. der Dienste, die der Logger zur Verfügung stellt).

Das Logging wird durch eine Standardkomponente (*Log4j*) realisiert.

Sie finden zwei Beipiel-XML-Dateien im Ordner *Dokumentation/Beispiel XMLs*.

Wichtiger Hinweis:
Änderungen an den XML-Dateien werden erst nach einem Neustart wirksam.

4.1 Das Log4j-Framework

4.1.1 Kurze Übersicht über das Log4j-Framework

Log4J ist eine Logging-API für Java. Sie wird über eine XML-Datei konfiguriert und kann verschiedene *Appender* –(Dienste) benutzen, um das Ergebnis des Loggings in einer vom Anwender gewünschten Weise zu verarbeiten.

MDBM nutzt hauptsächlich die *File*- und der *E-Mail*- (SMTP-) Appender.

Weitere Informationen zu Log4J finden Sie unter:

http://de.wikipedia.org/wiki/Log4j

http://logging.apache.org/log4j/

Mittels der Datei `Log4j.xml` können Sie das Log-Verhalten beinflussen.

Vereinfacht gesagt:

- Es gibt *Logger*, die erkennen, wann sie Informationen abgreifen müssen, und es gibt
- *Appender*, die Informationen verarbeiten, die vom Logger ermittelt werden.

Die E-Mail-Appender in der Datei `log4j.xml` sorgen dafür, dass Sie im Fehlerfall – oder wenn ein bestimmter Log-Level erreicht wird – eine E-Mail erhalten.

Wichtig sind die File-Appender, da diese die Logs auf die Festplatte schreiben.

4.1.2 Konfiguration des File-Apenders

Der File-Appender schreibt die Daten, die er vom Logger bekommt, in eine Datei auf dem Filesystem.

Mittels der Datei `log4j.xml` im Verzeichnis *Sensor*.läßt sich der File-Appender konfigurieren.

Hier die wichtigsten Attribute:

Das Attribut `value` des Parameters `file` legt den Pfad fest, in den der Appender schreiben soll. Das Format der Nachricht lässt sich mittels des Parameters `ConversionPattern` festlegen.

```
<appender name="SensorAppender" class="org.apache.log4j.DailyRollingFile-Appender">
  <param name="datePattern" value="'.'yyyy-MM-dd" />
  <param name="file" value="logs/sensor.log"/>
  <param name="Append" value="true" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{ISO8601} %-5p [%t] %c: %m%n" />
  </layout>
</appender>
```

4.1.3 Konfiguration des E-Mail-Appenders

Um den Sensor so einzustellen, dass er Ihnen E-Mails schickt, müssen Sie den E-Mail-Appender entsprechend konfigurieren.

Für weitere Informationen empfehlen wir Ihnen folgenden Link:

http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/net/SMTPAppender.html

Der E-Mail-Appender wird wie folgt konfiguriert:

```
<!-- appender name="AdministratorEmail"
                          class="org.apache.log4j.net.SMTPAppender"-->
  <appender name="AdministratorEmail" class="sensor.report.SensorSMTPAppender">
    <param name="BufferSize" value="512" />
    <param name="SMTPHost" value="Adresse des SMTP-Hostsystems"/>
    <param name="SMTPUsername" value="SMTP-Benutzername"/>
    <param name="SMTPPassword" value="SMTP-Passwort"/>
    <param name="From" value="Email-Versender" />
    <param name="To" value="Adressat" />
    <param name="Subject" value="Fehlermeldung MDBM" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"   Format der Nachricht
        value="[%d{ISO8601}]%n%n%-5p%n%n%c%n%n%m%n%n" />
    </layout>
    <filter class="org.apache.log4j.varia.LevelRangeFilter">
      <param name="LevelMin" value="warn" /> Ab wann greift der Appender
      <param name="LevelMax" value="fatal" /> Bis zu welchem Level greift der Appender
    </filter>
  </appender>
```

4.1.4 Konfiguration eines Loggers

Der Logger selbst ist sehr einfach einzurichten. Sie müssen nur definieren, wann er mitloggen und wann er welchen Appendern „Bescheid sagen“ soll.

Der „Vater“ aller Logger ist der Root-Logger. Von ihm werden *sämtliche* Log-Informationen verarbeitet. Deshalb ist seine Priorität auf `error` gesetzt, damit er nur Fehlermeldungen an seine Appender weitergibt.

```
  <root>                                        Der Name des Loggers hier ist root
    <priority value="error"/>
    <appender-ref ref="AllAppender" />
  </root>
```

Zu beachten ist auch der `sensor.Sensor` Logger:

```
  <logger name="sensor.Sensor" >                Name des Loggers
    <level value="warn" />                     Level, ab wann geloggt wird
    <appender-ref ref="SensorAppender"/>       Name des Appenders
    <appender-ref ref="AdministratorEmail"/>
  </logger>
```

Dieser Logger loggt nur bei Statusänderungen mit. Also z.B. von *Warning* zu *Error*.

Es gibt bei Loggern Hierachien. Wenn Sie selbst einen Logger erstellen und mit ihm einen bestimmten Test überwachen wollen, müssen Sie dies in seinem Namen angeben.

Nehmen wir beispielsweise den Rootlogger. An ihm kommen alle Informationen vorbei.

Wenn ihr Logger `sensor.maxdb.dbtests` heißt, dann überwacht er alle Klassen unter `sensor.maxdb.dbtests`.

Wenn Ihr Logger `sensor.maxdb.dbtests.cpu.CPUTest` heißt, dann überwacht er alle Klassen der CPU-Tests.

Die Hierachie der Package-Struktur:

```
root
sensor.Sensor
sensor.maxdb.dbtests.cpu.CPUTest
sensor.maxdb.dbtests.datavolume.DataVolumeUsedTest
sensor.maxdb.dbtests.dbanalyser.AnalyzerRunningTest
sensor.maxdb.dbtests.dbstate.DBStatetest
sensor.maxdb.dbtests.index.NumberOfBadIndexesTest
sensor.maxdb.dbtests.log.LogOverwriteTest
sensor.maxdb.dbtests.log.LogVolumeUsedTest
sensor.maxdb.dbtests.memory.MemoryTest
sensor.maxdb.dbtests.sessions.SessionCountTest
sensor.maxdb.dbtests.sql.SQLDatabaseTest
sensor.maxdb.dbtests.tables.TableUpStatisticTest
```

4.2 Konfiguration der Datei def.xml

Im nachfolgendem Abschnitt wird Ihnen das Einrichten der Standardtests und die Grundkonfiguration der Datei `def.xml` erläutert.

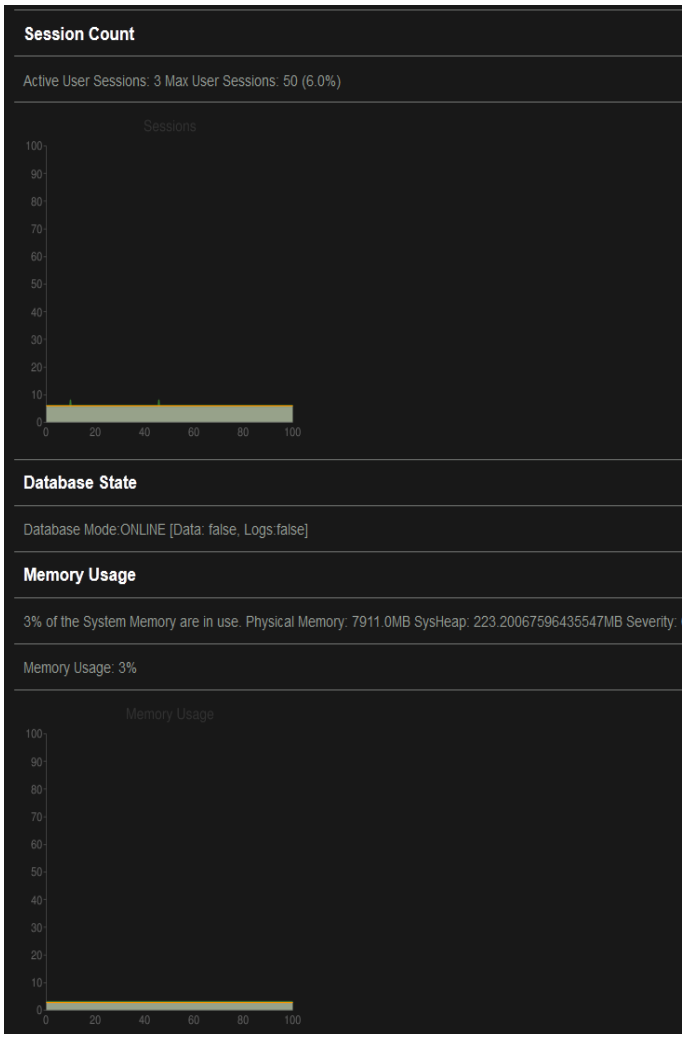
Die Datei `def.xml` in ihrem Sensor-Ordner ist die Standard-XML, die der Sensor benutzt. Sie ist bewußt einfach gehalten und enthält Standardeinträge für die mitgelieferten Tests. Nur die Verbindungsdaten zur Datenbank müssen noch eingetragen werden.

Sie finden zwei Beipiel-XML-Dateien im Ordner *Beispiel XMLs* des Ordners *Dokumentation*.

Mit der Datei `def.xml` bestimmen sie letztendlich auch den Aufbau der HTML5-Seite.

Die Reihenfolge der Tests in einer Connection bestimmt später auch die Reihenfolge der Darstellung der Test auf den Detailseiten einer Connection.

Beispiel der Detailseite:



Anpassung der E-Mail-Endung

Im Attribut `text` des Tags `<EmailHint>` können Sie den Text am Ende der E-Mail festlegen, den der Sensor mittels des Log4j-Framework schickt:

```
<?xml version="1.0" encoding="UTF-8"?>
<TestDefinitions>
<EmailHint text="Just open http://extern.infolytics.com:1080/mdbm/mobile/build/ for further details"></EmailHint>
```

Einbindung eigener Plugin-Klassen in die Weboberfläche

```
<OverviewClasses
Classes="sensor.maxdb.dbtests.dbstate.DBStatetest;sensor.maxdb.dbtests.sessions.SessionCountTest;sensor.maxdb.dbtests.datavolume.DataVolumeUsedTest;sensor.maxdb.dbtests.log.LogVolumeUsedTest/>
```

Das Attribut `OverviewClasses` legt fest, welche Klassen in der Weboberfläche angezeigt werden.

Einbindung eigener Plugin-Klassen

```
<Alias Aliasname="TKTest" Classname="com.infolytics.TKAnlagentest.TKLoader"></Alias>
```

Um zu verhindern, dass bei jedem »Custom«-Test angeben werden muss, um was für einen Test in was für einem Plugin es sich handelt, können Aliasnamen vergeben werden. Dabei muss man angeben, wie der Aliasname lautet und wo der Testloader für diesen Test liegt. Danach kann man diesen Customtest über den Aliasnamen aufrufen.

Aktualisierungsintervall für die Datei State.txt

```
<StateRefreshInterval interval="10000"/>
```

Das Attribut `interval` bestimmt den Intervall, in dem die Datei `State.txt` aktualisiert wird (in Millisekunden).
Es kann passieren, dass während des Schreibens des State-Files das State-Objekt im Speicher gerade die alten Testdaten löscht, weil es neue Testdaten bekommt. In diesem Fall fehlen diese Informationen in der Datei `State.txt` und somit auch in der Webdarstellung. Dies ist aber nicht weiter kritisch, da im nächsten Schreibdurchgang das Statefile alle Werte enthalten sollte. Darum sollten die Intervalle der Tests und des Statefile-Schreiben so gewählt werden, dass diese nicht miteinander kollidieren.

Also z.B für das Schreiben:

```
<StateRefreshInterval interval="10000"/> (10 Sekunden)
```

und für das Testen:

```
<Connection name="Muster_Connection" database="MusterDB" host="190.225.114.231"
```

`user="DBA" password="DBA1" interval="7100"` (7,1 Sekunden)

Die Wahrscheinlichkeit, dass das oben geschilderte Problem auftritt, ist jetzt sehr gering (In 710 Fällen wird das rein statistisch einmal vorkommen). Sollte der Fall eintreten, erscheint die Meldung: `sensor data (created: [Date]) is too old, sensor probably not running.`

Pfad zur Datei State.txt

```
<StateTXTPath path="C:\Users\Martin Hermann\Desktop\Neuer Ordner\"></StateTXTPath>
```

In diesen Pfad wird die Datei `State.txt` geschrieben, die der Webservice später ausliest. Erfolgt hier keine Angabe, wird die `State.txt`-Datei in den Ordner *State* des Sensor-Ordners geschrieben.

Definition der Datenbankverbindung

Zunächst muss die Verbindung definiert werden:

```
<MAXDB>
<Connection name="MAXDB3[LOCAL MAXDB]" database="MAXDB3" host="127.0.0.1" user="DBA"
password="DBA"
       interval="7100" stopOnFirstError="false" closeSessionAfterRun="true">
```

Eine Connection benötigt folgende Attribute:

Attribut	Bedeutung
`name`	ist der Name der Connection
`database`	ist der Name der Datenbank
`host`	ist die IP-Adresse
`user`	ist der Benutzername für die Datenbank (Bitte achten Sie darauf dass der DB-User über DBM-Rechte verfügt)
`password`	ist das Passwort für die Datenbank
`interval`	ist der Interval in dem getestet wird (in Millisekunden)
`stopOnFirstError`	`true` bedeutet: beim ersten Fehler wird der Testjob abgebrochen (auch alle folgenden Tests der laufenden Verbindung)
`Dbanreport`	`true` bedeutet, dass die Datei `DBAN.prt` des MaxDB Analyzers ausgelesen und über den Webservice zur Verfügung gestellt wird. Auch wenn die Datenbank ›tot‹ ist, kann die letzte Version der Datei `DBAN.prt` gelesen werden.
`closeSessionAfterRun`	`true` bedeutet, die Session wird nach dem Test geschlossen und beim nächsten Test wieder neu aufgebaut.

Die MaxDB-Tests

Die MaxDB-Tests sind Teil eines eigenständigen Plugins. Es ist also nicht notwendig, die MaxDB-Tests mit zu installieren, wenn man die Sensor-Applikation installiert. Die MaxDB-Tests werden standardmäßig mitgeliefert. Da die MaxDB-Tests ausgelagert sind, muss angegeben werden, dass es sich um MaxDB-Tests handelt. Ein Alias hierfür ist bereits vorkonfiguriert. MaxDB-Tests können sie aufrufen, indem sie `<MAXDB>` angeben, danach müssen sie nur noch die Verbindungen und deren Tests konfigurieren.

Das Sessionverhalten der MaxDB-Tests lässt sich konfigurieren: die Datenbankverbindung für die Tests kann offen gehalten oder für jeden Test erneut aufgebaut werden.
Beide Verfahren haben Vor- und Nachteile. Wenn die Verbindung jedes mal neu aufgebaut wird, verursacht dies kurzeitig mehr CPU-Last. Wird dagegen die Verbindung offen gehalten, steigt der Bedarf an RAM, dafür hat man aber keine Ausschläge bei der CPU-Leistung.

Der DBAN-Report greift auf die Datei `DBAN.prt` der Datenbank zu. `DBAN.prt` wird vom DB-Analyzer erstellt und enthält die globale Auswertung des letzten DBAnalyzer-Laufs. Sollte die Datenbank ausfallen, können Sie sich noch den letzten DBAN.prt-Bericht ansehen.

Bitte beachten Sie: wenn das Attribut `stopOnFirstError` auf true gesetzt ist und ein Fehler auftritt, werden alle nachfolgenden Tests nicht mehr durchgeführt.

Der DBState-Test ist ein Sonderfall unter den Standard-Tests.
Er wird grundsätzlich bei jeder Verbindung gestartet.
Sie können sein Standard-Verhalten allerdings überschreiben.
Über das Attribut `error` können sie festlegen, ab welchem Datenbankzustand er einen Fehler melden soll.
Das Attribut severityIfFails="OK" bestimmt, welche Severity der Test im Falle eines Fehlschlags liefert.

```
<LogVolumeUsedTest threshold_fatal="90"
              threshold_error="80" threshold_warning="70" showDetailedInfo="true" />

              <SessionCountTest threshold_fatal="90"
              threshold_error="90" threshold_warning="80" />

              <MemoryTest threshold_fatal="40"
              threshold_error="30" threshold_warning="20" />

              <CPUTest threshold_fatal="50" threshold_error="30"
threshold_warning="20" />

              <DataVolumeUsedTest threshold_fatal="50"
              threshold_error="30" threshold_warning="20" showDetailedInfo="true" />

              <LogOverwriteTest severityIfFails="Error" />

              <NumberOfBadIndexesTest threshold_fatal="500" threshold_error="20"
threshold_warning="10"/>

              <AnalyzerRunningTest severityIfFails="OK" />

              <DBANTest severityIfFails="OK" />
```

5 Das Monitor-Modul (der Webservice)

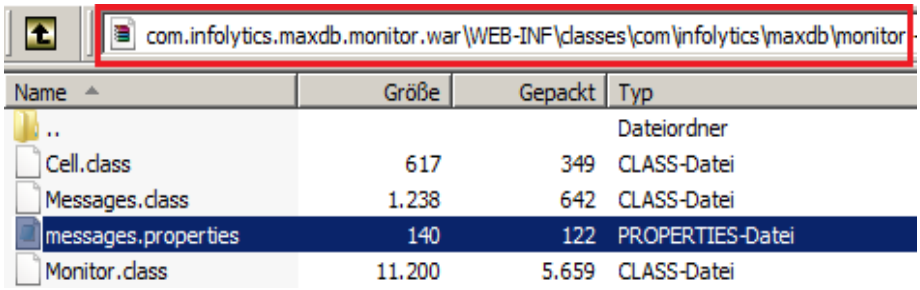
5.1 Konfigurationdes Monitor-Moduls

Im *Ordner Monitor + Monitor Client* liegt eine `.war`-Datei. Diese Datei ist vom Typ *WebArchiv*.

Nun müssen wir noch den Ort definieren, an dem die `State.txt`-Dateien, welche der Sensor liefert, ausgelesen werden können. In dieser Anleitung gehen wir von einer Installation des Webservices unter einem *Tomcat Webserver 7.0* aus. Sie können natürlich auch einen anderen Webserver nehmen. Zum Beispiel den *SAP Netweaver*.

5.2 Anpassen des "State.txt"-Pfades, bevor der Webservice installiert wird

1. Öffnen Sie die Datei mit einem dafür vorgesehenen Programm (z.B. WinRAR)
2. Gehen Sie in dem Programm zu folgendem Pfad:
   com.infolytics.maxdb.monitor.war\WEB-INF\classes\com\infolytics\maxdb\monitor
3. Hier befindet sich die Datei `messages.properties`



4. Öffnen Sie die "messages.properties" mit einem Texteditor.
5. Bitte Ändern Sie die erste Zeile und schreiben Sie dort den Pfad hinein, wohin das Sensormodul seine `State.txt` schreibt.

Zum Beispiel:

Monitor.StateFilePath=C:\\Users\\yourname\\workspace\\com.infolytics.maxdb.sensor\\state

5.3 Anpassen des State.txt-Pfades nachdem der Webservice installiert wurde

Hinweis: Bei dem Beispiel wird von einem Tomcat-server 7.0 ausgegangen.

1. Gehen Sie zum Ordner, in den der Webservice installiert wurde.
   In unserem Beispiel ist das:
   C:\...\tomcat\webapps\com.infolytics.maxdb.monitor\WEB-

INF\classes\com\infolytics\maxdb\monitor

2. Öffnen Sie die `messages.properties`-Datei mit einem Texteditor.
3. Ändern Sie die erste Zeile. Tragen Sie dort den Pfad ein, in den das Sensormodul die `State.txt`-Datei schreibt.

Beispiel:

Monitor.StateFilePath=C:\\Users\\yourname\\workspace\\com.infolytics.maxdb.sensor\\state

5.3.1 Aufbau der messages.properties-Datei

Mittels der Datei `message.properties` wird das Verhalten des Monitors beeinflusst.

Beispiel: Angabe des Pfads, von wo das State-file des Sensors gelesen werden soll.

```
Monitor.StateFilePath=C:\\Users\\Martin
Hermann\\workspace\\com.infolytics.maxdb.sensor\\state\\
```

Beispiel: Nachricht des Monitor wenn es keine Fehler gibt.

```
Monitor.WorksFineMessage=Everything works fine
```

Beispiel: DataTooOldWarning

```
Monitor.SensorDataTooOldWarningAfter=21000
```

Hier stellen sie ein, ab wann die Daten des Statefiles für den Monitor als zu alt angesehen werden. (in Millisekunden).

Der Sensor schreibt standardmäßig alle 20 Sekunden ein neues Statefile. Deshalb ist der Standardwert für den Monitor 21 Sekunden.

5.4 Installation des Monitor-Moduls

An dieser Stelle finden Sie Hinweise für die Installation des Monitor-Moduls.

Für die Installation benötigen Sie einen Webserver.

Als Beispiel benutzen wird in dieser Dokumentation ein *Tomcatserver 7.0* verwendet. Diesen finden Sie unter http://tomcat.apache.org/download-70.cgi

1. Gehen Sie in den Installationsordner des Tomcat-Servers und suchen Sie den Ordner *Webapps*.
   `(...\tomcat\webapps)`
2. Kopieren Sie die .War-Datei in den *webapps*-Ordner.
3. Starten Sie nun den Tomcat neu.
4. Beim Neustart lädt der Tomcat nun die `.war`-Datei.
5. Wenn Sie sich die Ausgabe beim Starten des Tomcat zurückgeben lassen, dann können Sie sehen, dass der Tomcatserver die `.war`-Datei geladen hat.
6. Sie können sich die Ausgabe zurückgeben lassen, indem Sie den Tomcat-Server mittels der zwei `.bat`-Dateien im Tomcat-Ordner neustarten. (`catalina_start.bat` und `catalina_stop.bat`).
7. Um zu testen, ob die Installation funtioniert hat, öffnen Sie Ihren Browser.
8. Besuchen Sie die Seite:
   http://127.0.0.1:8080/com.infolytics.maxdb.monitor/rest/monitor/jobList
9. Sollten Sie den Sensor noch nicht oder falsch konfiguriert haben, dann erhalten Sie folgende Fehlermeldung:

   ```
   C:\Users\yourname\workspace\com.infolytics.maxdb.sensor\state\stateFile.txt
   (Das System kann den angegebenen Pfad nicht finden)
   ```

   Das bedeutet, dass der Webservice die `State.txt`-Datei an Ihrem angegebenen Pfad nicht finden kann.

6 Die HTML5-Seite

Um die Ergebnisse, die der Webservice liefert, darstellen zu können, wird eine weitere Anwendung benötigt. Wir haben dazu eine HTML5 Webseite erstellt.

Um diese zu installieren, benötigen Sie einen Webserver.

Als Beispiel in dieser Anleitung wird ein *Apache Webserver* verwendet, der im *XAMPP Paket 1.7.7* enthalten ist. XAMPP und weitere Informationen dazu erhalten Sie unter:

http://www.apachefriends.org/de/xampp.html

Im Installationsverzeichnis von XAMPP befindet sich der Ordner *htdocs* (z.B. C:\xampp\htdocs).

1. Dort muss zunächst ein Ordner mit dem Namen *mdbm* erstellt werden.
2. Anschließend muss der Inhalt des Ordners *HTML5 Seite* in den Ordner *mdbm* kopieren werden.

6.1 Den Webservice hinter den Apache-Server »verstecken«

Aus Sicherheitsgründen raten wir Ihnen dazu, den Websevice *hinter* den Apache-Server zu setzen. Bei einem Apache- und Tomcat-Server benötigen Sie dazu einen Connector.

Diesen finden Sie unter: `tomcat.apache.org/connectors-doc/`

Die Anleitung für die Installation finden Sie unter:

http://tomcat.apache.org/connectors-doc/generic_howto/quick.html

1. Zunächst muss ein *Worker* definiert werdenen, der die weitergeleiteten Anfragen des Apache-Servers beantwortet.
   Ein Worker ist eine Instanz des Tomcat Servers, welche die weitergeleitete Anfrage des Webservers bearbeitet.
2. Um einen Worker zu definieren, muss eine Konfigurationsdatei erstellt werden.
   Dies wird Ihnen im nachfolgendem Beispiel erläutert:

   Sie erstellen eine Datei mit dem Namen `workers.properties` im Ordner *conf* des Apache-Installationsverzeichnisses mit folgendem Inhalt:

   ```
   #Define 1 real worker using ajp13
   worker.list=worker1
   #Set properties for worker1 (ajp13)
   worker.worker1.type=ajp13
   worker.worker1.host=localhost
   worker.worker1.port=8009
   ```

3. Dann müssen Sie noch den *Apache Tomcat Connector* herunterladen.

4. Die Datei `mod_jk.so` muss in das Verzeichnis *modules* im Apache-Installationsverzeichnis kopiert werden.

5. Jetzt müssen Sie die Konfigurationsdatei des Apache Webservers anpassen.

   Dazu ergänzen Sie folgenden Text in der `httpd.conf`-Datei. Diese Datei befindet sich im Ordner *conf* des Apache-Installationsverzeichnisses:

```
[... httpd.conf ...]
#Load the connect module
LoadModule jk_module  modules/mod_jk.so
# Where to find workers.properties
JkWorkersFile C:\xampp\apache\conf/workers.properties
# Where to put jk shared memory -> apache log
JkShmFile logs/mod_jk.shm
# Log directory of the mod
JkLogFile logs/mod_jk.log
# Set the jk log level [debug/error/info]
JkLog-Level info
# Select the timestamp log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
# Send everything for context /com.infolytics.maxdb.monitor to worker named
worker1 (ajp13)
JkMount /com.infolytics.maxdb.monitor/* worker1
[... httpd.conf ...]
```

6. Nach einem Neustart des Apache- und des Tomcat-Servers sollte der Connector funktionieren.

7. Dies können Sie testen, indem Sie die Log-Datei des Connectors prüfen.

   Diese Datei liegt (bei XAMPP) unter *C:\xampp\apache\logs*. Ihr Name ist `mod_jk.log`

   Wenn der Connector richtig funktioniert dann steht in der Datei:

```
[Di Jan 17 18:02:57 2012][8832:8704] [info] init_jk::mod_jk.c (3252):
mod_jk/1.2.32 () initialized
```

8. Der Tomcat-Webservice sollte jetzt unter

   http://localhost/com.infolytics.maxdb.monitor/rest/monitor/jobList

   erreichbar sein.

6.2 Konfigurieren der HTML5-Seite

Wenn Sie den Webservice nicht hinter einem Apache verstecken, müssen sie eine Anpassung der HTML5-Seite vornehmen. Sie müssen der HTML5-Webseite mitteilen, wo sie den Webservice des Sensor finden kann.

1. Öffnen Sie den Ordner, in dem sich die HTML5-Seite befindet (z.B. *C:\xampp\htdocs\mdbm*).

2. Dort finden sie einen Ordner mit dem Namen `script`.
3. In diesem Ordner finden Sie eine Datei mit dem Namen `mobilemaxdb.js`.
4. Öffnen sie diese Datei mit einem Texteditor (Empfehlung: Notepad++).
5. Ganz oben im Dokument finden sie folgende Zeile:

   ```
   var mobilemaxdb_uri= "/com.infolytics.maxdb.monitor/rest/monitor";
   ```

   Unter dieser Adresse wird auf dem Hostrechner der Website nach dem Webservice gesucht.

6. Wenn Sie den Webservice woanders installiert haben, geben sie bitte den absoluten Pfad an. Also z.B.:

   ```
   var mobilemaxdb_uri= "http://extern.infolytics.com:1080/
                                 com.infolytics.maxdb.monitor/rest/monitor";
   ```

   (Den Zeilenumbruch bitte ignorieren)

7. Wichtig! Hängen sie keinen Schrägstrich an das Ende des Pfades an.

   Also nicht: `/com.infolytics.maxdb.monitor/rest/monitor/";`

7 Test der Installation

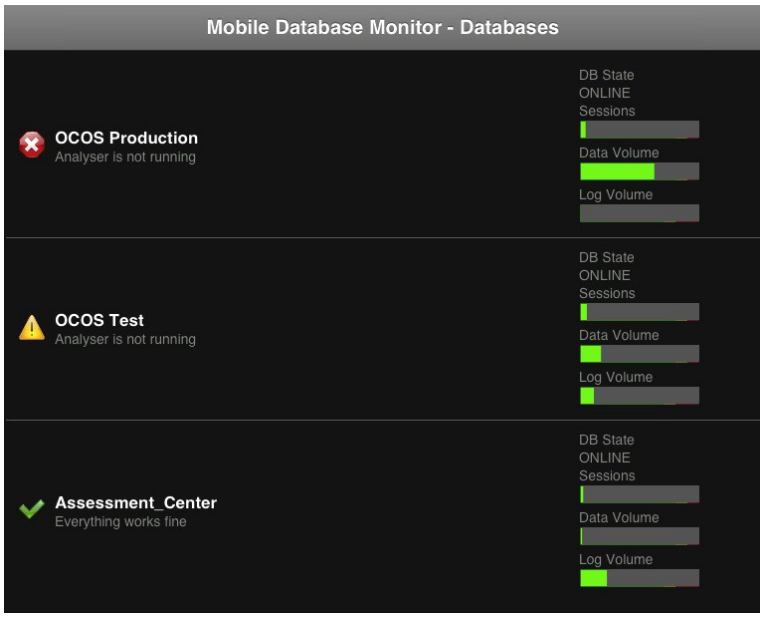
1. Starten Sie den Sensor.
2. Danach starten Sie den Tomcat-Server (falls dieser noch nicht gestartet ist).

   Falls Sie nicht wissen wie man den Tomcat startet:
   Es gibt im Tomcat-Installationsverzeichins eine Datei namens `catalina_start.bat`.
   Führen Sie bitte diese Datei aus.
3. Anschließend starten Sie den Apache-Webserver.

   Dies tun Sie, indem Sie in das Installationsverzeichnis des XAMPP gehen.
   Hier finden Sie eine Datei mit dem Namen `apache_start.bat`
   Führen Sie diese Datei nun aus. Jetzt wird der Apacheserver gestartet

Um die Webseite zu testen, öffnen Sie die URL http://127.0.0.1/mdbm

Wenn alles funktioniert hat, sollte die Seite wie folgt ausehen:



(Darstellung im Safari-Browser. Die Darstellung kann in anderen Browsern abweichen)

8 Die MDBM-MaxDB-Standard-Tests

Bei den MDBM-MaxDB-Standard-Tests werden bestimmte Spalten von Systemtabellen abgefragt.

Die Test sind dabei so konfigurierbar, dass Sie die Schwellwerte, ab wann eine *Warning* oder ein *Error* gemeldet wird, selbst bestimmen können. Zum Benutzen dieser Tests müssen Sie in der Datei `def.xml` die Anmeldeinformationen eines DBM-Users mit eingeben, da es sich bei den abgefragten Tabellen um MaxDB-Systemtabellen handelt.

Die Informationen zu den Tests und wo man die Daten findet, haben wir der Datei `dbanalyzer78.cfg` entnommen. Hier kann man sehen wie der DBAnalyzer die Werte abfragt.

Weitere Informationen zu den Systemtabellen der MaxDB finden sie unter:

> http://help.sap.com/saphelp_nwpi71/helpdata/de/56/4f074152aff323e10000000a155106/frameset.htm

8.1 CPU-Test

Hier wird geprüft, welche CPU-Auslastung auf dem Rechner erzeugt wird.

Dazu wird die Spalte `CPULOAD` der Systemtabelle `MACHINEUTILIZATION` abgefragt.

`CPULOAD` ist ein `CHAR(12)`-Feld und gibt die CPU-Last auf dem Rechner an.

Konfiguration des Tests in der Datei `def.xml`:

```
<CPUTest threshold_fatal="50" threshold_error="30" threshold_warning="20" />
```

8.2 Memory-Test

Hier wird die RAM-Auslastung auf dem Rechner geprüft. Dazu werden die maximale RAM-Auslastung und die maximale RAM-Größe ermittelt. Aus diesen Zahlen wird dann die prozentuale RAM-Auslastung durch die MaxDB errechnet.

Die RAM-Größe steht in der Spalte `PHYSICALMEMORYSIZE` in der Systemtabelle `MACHINECONFIGURATION`.

Die RAM-Belastung durch die MaxDB wird ermittelt, indem man die Spalte `USEDSIZE` in der Tabelle `SYSINFO.MEMORYALLOCATORSTATISTICS` mit der Where-Bedingung `where ALLOCATORNAME = 'SystemHeap'` abfragt.

Konfiguration des Tests in der Datei `def.xml`:

```
<MemoryTest threshold_fatal="40"threshold_error="30" threshold_warning="20" />
```

8.3 LogvolumenUsed-Test

Durch den LogvolumenUsed-Test wird die prozentuale Auslastung der Logvolumes der MaxDB ermittelt.

Dazu wird die Spalte USEDSIZEPERCENTAGE in der Tabelle SYSINFO.LOGSTATISTICS abgefragt.

Die Systemtabelle LOGSTATISTICS beschreibt den Nutzungsgrad der Log-Verwaltung und die Anzahl besonderer Ereignisse beim Logging seit dem letzten Start der Datenbankinstanz. Das Attribut ShowDetailedInfo legt fest, ob die Informationen des Tests auf der HTML5-Detailseite der Datenbankverbindung angezeigt werden sollen.

Konfiguration des Tests in der Datei def.xml:

```
<LogVolumeUsedTest threshold_fatal="90"
         threshold_error="80" threshold_warning="70" showDetailedInfo="true" />
```

8.4 Log-Overwrite-Test

Der Log-Overwrite-Test prüft, ob der Log-Overwrite-Modus der MaxDB aktiv ist. Dafür wird ein Wert aus der Spalte Value der Tabelle DB_STATE abgefragt.

Konfiguration des Tests in der Datei def.xml:

```
<LogOverwriteTest severityIfFails="Error" />
```

8.5 DatavolumenUsed-Test

Durch den DataVolumen-Test wird die prozentuale Auslastung der Datavolumes der MaxDB ermittelt.

Dazu wird die Spalte USEDSIZEPERCENTAGE in der Tabelle SYSINFO.DATASTATISTICS abgefragt.

Konfiguration des Tests in der Datei def.xml:

```
<DataVolumenUsedTest threshold_fatal="50" threshold_error="30"
threshold_warning="20" />
```

8.6 Analyzer-Running-Test

Mit dem Analyzer-Running-Test wird geprüft, ob der MaxDB Analyzer in einer Datenbank läuft.

Dies kann man mit dem Befehl dban_State abfragen.

Konfiguration des Tests in der Datei def.xml:

```
<AnalyzerRunningTest severityIfFails="Error" />
```

8.7 Dbstate-Test

Mit dem Dbstate-Test wird der Zustand der Datenbank abgefragt. Dabei wird geprüft, ob die Logvolumes oder die Datavolumes voll sind. Außerdem wird der aktuelle Status der Datenbank abgefragt. Dafür wird der Befehl db_state verwendet.

Der DBState-Test ist ein Sonderfall. Er wird *grundsätzlich bei jeder Verbindung* gestartet. Sie können sein Standard-Verhalten allerdings überschreiben. Über das Attribut error können sie festlegen, ab welchem Datanbankzustand er einen Fehler melden soll.

Konfiguration des Tests in der Datei def.xml:

```
<DBStatetest error="OFFLINE"/>
```

8.8 Number of Bad-Indexes-Test

Der Number of Bad-Indexes-Test prüft die Anzahl der Indexfehler in einer MaxDB.

Dieser Wert wird aus der Spalte `BADINDEXCOUNT` der Tabelle `SYSINFO.RESTARTINFORMATION` ausgelesen.

Konfiguration des Tests in der Datei `def.xml`:

```
<NumberOfBadIndexesTest threshold_fatal="500" threshold_error="20"
threshold_warning="10" />
```

8.9 Sessioncount-Test

Der Sessioncount-Test liest die maximale und die aktuelle Anzahl von Sessions aus. Die aktuelle Anzahl der Session wird dann mir ihren Schwellenwerten verglichen. Die Werte werden aus der Tabelle `SYSDD.DBM_STATE` ausgelesen. Die maximale Anzahl der Sessions steht in der Spalte `MAXUSERS`, die Anzahl der aktuellen aktiven Sessions in der Spalte `ACTIVE_SESSIONS`.

Konfiguration des Tests in der Datei def.xml:

```
<SessionCountTest threshold_fatal="90"  threshold_error="90"
threshold_warning="80" />
```

9 Werte-Test

Wenn Ihnen die Standard-MaxDB-Tests nicht ausreichen, Sie aber trotzdem bestimmte Werte in einer MaxDB-Datenbank prüfen wollen, können Sie die *SQLDatabasetests* nutzen. Durch die SQL-Databasetests können sie Werte in Spalten in bestimmten Tabellen testen.

Bei den SQLDatabasetests haben Sie die Möglichkeit, ein SQL-Statement mitzugeben. Das Ergebnis dieser SQL-Abfrage wird dann überprüft. Das Ergebnis der SQLDatabasetests kann auf dreierlei Arten geprüft werden:

1. Ist das Ergebnis *größer* als ihr Schwellwert (*GreaterThanThresholdTest*)
2. Ist das Ergebnis *kleiner* als ihr Schwellwert (*LowerThanThrerholdTest*)
3. Weicht das Ergebnis vom *vorherigen* Wert ab (*PreviousValueTest*)

9.1 GreaterThanThreshold-Test

Der GreaterThanThreshold-Test prüft, ob der ihm übergebene Wert größer als der von Ihnen vorgegebene Schwellwert ist. Sollte der Ergebniswert größer als ihr Schwellwert sein, wird eine `ERROR` oder `WARNING`-Meldung ausgeben. Das Verhalten können Sie über die Konfiguration der Datei `def.xml` steuern.

Die Konfiguration eines *GreaterThanThresholdTest* in der Datei `def.xml` sieht wie folgt aus:

```
<ValueGreaterThanThresholdTest Type="Double" SeverityIfFail="Warning"
Threshold="0.001" SeverityMessage="Warnung: Die CPU Auslastung liegt bei {0}
(Schwelle: {1})"/>
```

Attribut	Bedeutung
`Type`	Gibt den Datentyp an, mit dem gearbeitet werden soll. (Integer, Float, Double, String, Biginteger, Bigdecimal, Long, String)
`Threshold`	Ist der Schwellwert, auf den geprüft werden soll.
`SeverityIfFail`	Gibt den Fehlerlevel an, wenn der Test fehlschlägt.
`SeverityMessage`	Ist die Fehlermeldung, die Sie bekommen, wenn ihr Schwellwert überschritten wird. Wobei {0} der Platzhalter für den aktuellen Wert ist und {1} der von Ihnen festgelegte Schwellwert ist.

9.2 ValueLowerThanThreshold-Test

Der ValueLowerThanThreshold-Test prüft, ob der ihm übergebene Wert größer als der von Ihnen vorgegebene Schwellwert ist. Sollte der Ergebniswert kleiner als ihr Schwellwert sein, wird eine `ERROR` oder `WARNING`-Meldung ausgeben. Das Verhalten können Sie über die Konfiguration der Datei `def.xml` steuern.

Die Konfiguration eines ValueLowerThanThreshold-Tests in der Datei `def.xml` sieht wie folgt aus:

```
<LowerValueTest Type="Double"  SeverityIfFail="Warning" Threshold="0.001"
SeverityMessage="Warnung: Die CPU Auslastung liegt bei {0}
(Schwelle:1})"></LowerValueTest>
```

Attribut	Bedeutung
`Type`	Gibt den Datentyp an mit dem gearbeitet werden soll. (Integer, Float, Double, String, Biginteger, Bigdecimal, Long, String)
`Threshold`	Ist der Schwellwert auf den geprüft werden soll.
`SeverityIfFail`	Gibt den Fehlerlevel an wenn der Test fehlschlägt.
`SeverityMessage`	Ist die Fehlermeldung die Sie bekommen wenn ihr Schwellwert unterschritten wird. Wobei {0} der Platzhalter für den aktuellen Wert ist und {1} der von Ihnen festgelegte Schwellwert ist.

9.3 PreviousValue-Test

Der PreviousValueTest prüft, ob sich der aktuelle Wert erst vom übergebenen und später vom letzten Wert unterscheidet. Sollte der aktuelle Wert sich vom übergeben Wert oder vom vorherigen Wert unterscheiden, dann wird eine `ERROR` oder `WARNING`-Meldung ausgeben. Das Verhalten können Sie über die Konfiguration der Datei `def.xml` steuern.

Die Konfiguration eines PreviousValue-Tests in der Datei `def.xml` sieht wie folgt aus:

```
<PreviousValueTest Type="Double"  SeverityIfFail="Warning"  Threshold="1"
SeverityMessage="Warnung: Der Wert a ={0} und nicht gleich mit dem Wert b=
{1})"></PreviousValueTest>
```

Attribut	Bedeutung
`Type`	Gibt den Datentyp an mit dem gearbeitet werden soll. (Integer, Float, Double, String, Biginteger, Bigdecimal, Long, String)
`Threshold`	Ist der Schwellwert auf den geprüft werden soll.
`SeverityIfFail`	Gibt den Fehlerlevel an wenn der Test fehlschlägt.
`SeverityMessage`	Ist die Fehlermeldung die Sie bekommen wenn ihr Schwellwert unterschritten wird. Wobei {0} der Platzhalter für den aktuellen Wert ist

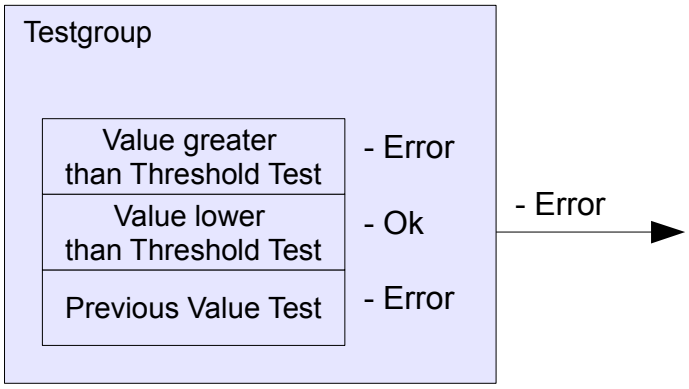
	und {1} der von Ihnen festgelegte Schwellwert ist.

9.4 Testgroups

Testgroups beinhalten SQLDatabase-Tests. Das Gesamtergebnis einer Testgroup hängt von der für die Gruppe definierten Verhaltensweise ab (siehe als Beispiel das Diagramm unterhalb der Tabelle). Durch Testgroups ist es möglich, Tests nicht nur losgelöst voneinander laufen zu lassen, sondern abhängig voneinander. So kann man ein anderes Fehlermeldeverhalten erzeugen und man kann bei SQLDatabasetests mehrere Prüfungen auf einen Wert machen. Außerdem nutzen dann alle Tests den gleichen Wert einer Spalte eines SQL-Statements (siehe Beispiel auf der nächsten Seite).

Folgende Fehlerverhalten von Testgroups sind möglich:

Fehlerverhalten	Bedeutung
`Return Error on first Error or Warning`	Die Testgroup gibt *Error* beim ersten `ERROR` oder `WARNING` zurück
`Return Ok on first Ok`	Beim ersten `Ok` gibt die Testgroup *Ok* zurück.
`Return Error on Majority Error or Warning`	Gibt *Error* zurück, wenn die Mehrheit der Test einen `ERROR` liefert.
`Return Ok on Majority Ok`	Gibt *Ok* zurück, wenn die Mehrheit der Test `Ok` zurückliefert.



Beispiel für die Konfiguration einer Testgroup

```
<SQLDatabaseTest Behaviour="0">
  <SQL>Select cpuload from MACHINEUTILIZATION</SQL>
  <Tests>
    <ValueTestGroup Type="Double">
      <ValueGreaterThanThresholdTest Type="Double" SeverityIfFail="Warning"
         Threshold="0.001" SeverityMessage="Warnung: Die CPU Auslastung liegt
         bei {0} (Schwelle: {1})"/>
      <ValueGreaterThanThresholdTest Type="Double" SeverityIfFail="Error"
          Threshold="50" SeverityMessage="Fehler: Die CPU Auslastung liegt
          bei {0}"/>
    </ValueTestGroup>
  </Tests>
</SQLDatabaseTest>
```

Erläuterung

`Behaviour` Sagt aus, wie sich die Test im Fehlerfall verhalten sollen:

0 – Gibt ERROR beim ersten Fehler zurück, der gemeldet wird.

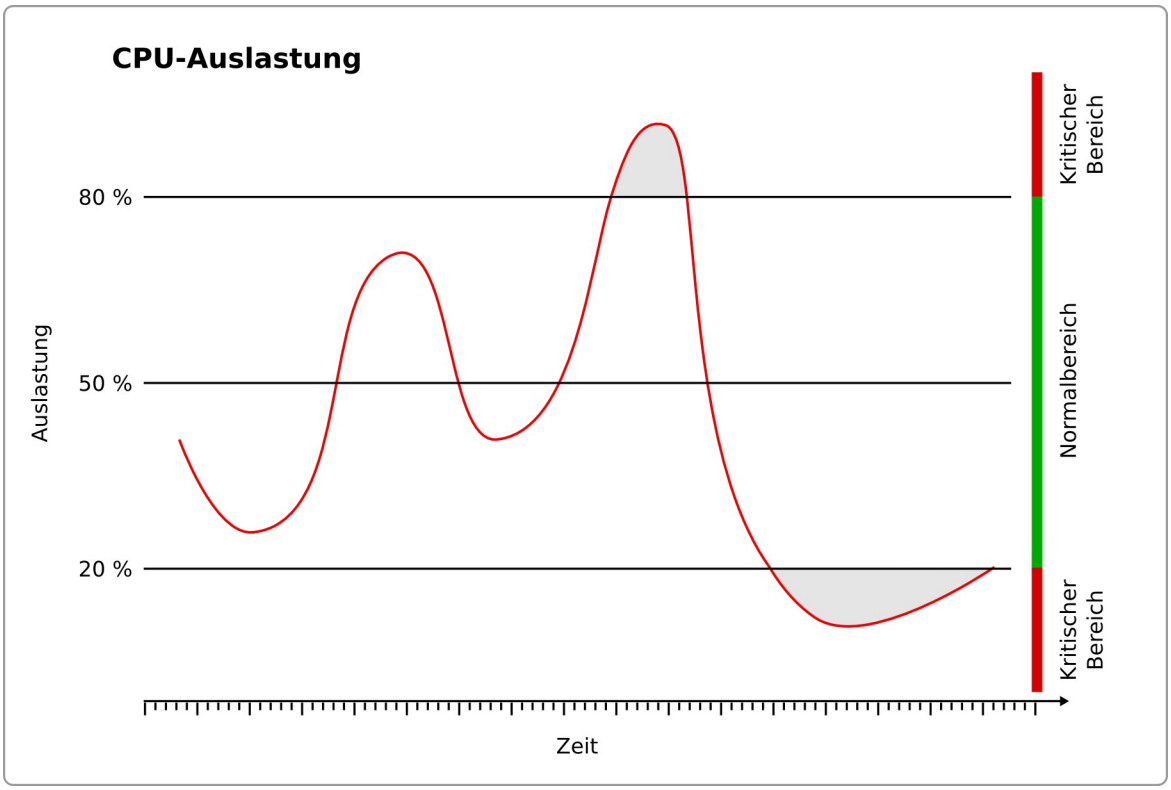
1 – Gibt OK beim ersten OK zurück.

2 – Gibt ERROR zurück, wenn der Großteil der Tests in der Testgroup einen Fehler liefert.

3 – Gibt OK zurück, wenn der Großteil der Tests in der Testgroup ein OK liefert.

10 Wie man einen Wert überwacht

In diesem Teil wird Ihnen erläutert, wie Sie einen Wert in Ihrer Datenbank durch die SQLDatabasetests überwachen können. Als Beispiel haben wir uns dafür die CPU-Auslastung ausgesucht.



Als erstes müssen Sie ihren Normalbereich festlegen. Dieser Bereich wird dann durch zwei Tests eingegrenzt. Einem *GreaterThanThresholdTest* und einem *LowerThanThrerholdTest*.

Als Schwellwert für den *LowerThanThrerholdTest* tragen sie den Wert ein, der nicht unterschritten werden darf.

Als Schwellwert für den *GreaterThanThresholdTest* tragen sie den Wert ein, der nicht überschritten werden darf.

Als Beispielkonfiguration sieht dies dann wie folgt aus:

```
<SQLDatabaseTest Behaviour="0">
  <SQL>Select cpuload as a from MACHINEUTILIZATION</SQL>
  <Tests>
    <ValueTestGroup Type="Double">
      <LowerValueTest Type="Double"  Threshold="20"
            ErrorMessage="Achtung die CPU-Auslastung ist {0} und hat somit ihren
            Schwellwert von {1} unterschritten.">
      </LowerValueTest>
      <ValueGreaterThanThresholdTest Type="Double"  Threshold="80"
            ErrorMessage="Achtung die CPU-Auslastung ist bei  {0} und hat somit
            den Schwellwert {1} überschritten">
      </ValueGreaterThanThresholdTest>
    </ValueTestGroup>
  </Tests>
</SQLDatabaseTest>
```

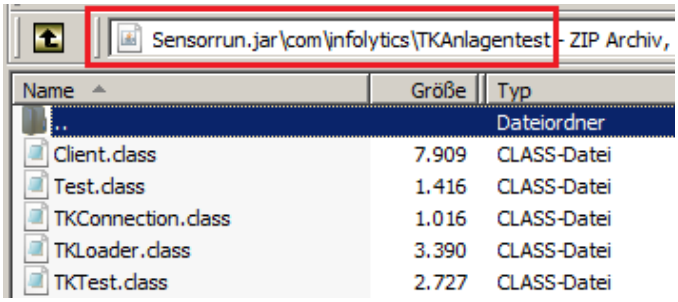
11 Eigene Tests erstellen

Wenn die vorgegebenen Tests nicht Ihren Bedürfnissen entsprechen, haben Sie die Möglichkeit, eigene Tests einzubauen. Allerdings sind dafür Java-Kenntnisse Voraussetzung. Durch selbstdefinierte Test ist es möglich, die Grundfunktionalität des Sensorsprogramms für eigene Problemstellungen zu nutzen. Dazu gehören das regelmäßige Ausführen von Tests und das Behandeln und Bereitstellen der Ergebnisse.

11.1 Dokumentation zum TK-Anlagentest

Praxisbeispiel anhand einer Zustandsüberwachung einer Sipgate-Telefonanlage.

Die Sipgate-Telefone eines Unternehmens verloren ständig ihre Telefon-Accounts. So war ein Telefonieren nach einer gewissen Zeit nicht mehr möglich – die betroffenen Geräte mussten neugestartet werden. Für dieses Problem wurde eine Lösung benötigt.



Darum haben wir im Sensorprogram die Möglichkeit implementiert, eigene Test-Klassen einzubauen. Diese müssen nur in die Datei `def.xml` eingefügt werden. Dabei muss darauf geachtet werden, dass der Name in der Datei `def.xml` dem Namen Ihrer `.Class`-Datei entspricht. Ihre `.Class`-Dateien müssen dann in die Datei `Sensorrun.jar` unter *\com\infolytics\TKAnlagentest* gepackt werden.

Beispiel des TK-Anlagentest in der Datei `def.xml`:

Es gibt zwei Möglichkeiten, den TKAnlagenTest einzubauen:

1. Sie müssen einen Alias vergeben:

```
<Alias Aliasname="TKTest" Classname="com.infolytics.TKAnlagentest.TKLoader"></Alias>
```

Dann können sie den Test mit dem Aliasnamen aufrufen:

```
<TKTest interval="1000" severityIfFails="Error">
      <connection ip="117.245.185.199" password="password" activeAccounts="1"
reboot="false">
      </connection>
</TKTest>
```

2. Altes Verfahren (wird weiterhin unterstützt):

```
<Custom class="com.infolytics.TKAnlagentest.TKLoader" interval="60000">
      <connection ip="117.245.185.199" password="password" activeAccounts="1">
      </connection>
</Custom>
```

Attribut	Bedeutung
`Custom class`	Name und Pfad der benutzerdefinierten Klasse, die geladen werden soll.
`interval`	Intevall, in dem die TK-"Test" durchgeführt werden (in Millisekunden).
`connection`	Damit ist die Verbindung gemeint, die geprüft werden soll.
`ip`	Ist die IP-Adresse des Telefons, das getestet werden soll.
`Password`	Password, um sich am Telefon anzumelden
`ActiveAccounts`	Anzahl der Accounts, die mindestens im Telefon registriert sein müssen.
`reboot`	Legt fest, ob das Telefon neugestartet werden soll, wenn nicht alle Accounts registriert sind. true = neustart false = kein neustart
`severityIfFails`	Legt fest, ob "Error", "Warning" oder "Ok" gemeldet wird, wenn nicht alle Account des Telefons registriert sind. E-Mails werden vom Log4J-Framework erst bei Meldung eines Errors verschickt.

Wichtig beim Erstellen von eigenen Testklassen ist, dass Sie sich an die vorgegebenen Interfaces halten, da sonst Ihre Klasse nicht vollständig in das Sensorprogramm eingebaut werden kann.

Der TKLoader implementiert folgendes Interface:

```
public interface Testloader {
    public List<AbstractTestjob> load(StartElement startElement,
      XMLEventReader reader)throws XMLStreamException;
}
```

Als erstes lädt der TKLoader die Konfigurationsdaten aus der Datei `def.xml`.

Außerdem sollten sie auf die Ergebnisse achten, welche Ihnen Ihr Test zurückliefert.

Im MDBM werden alle Testergebnisse in Info-Objekte geschrieben. Diese wiederum werden in Testresponses geschrieben.

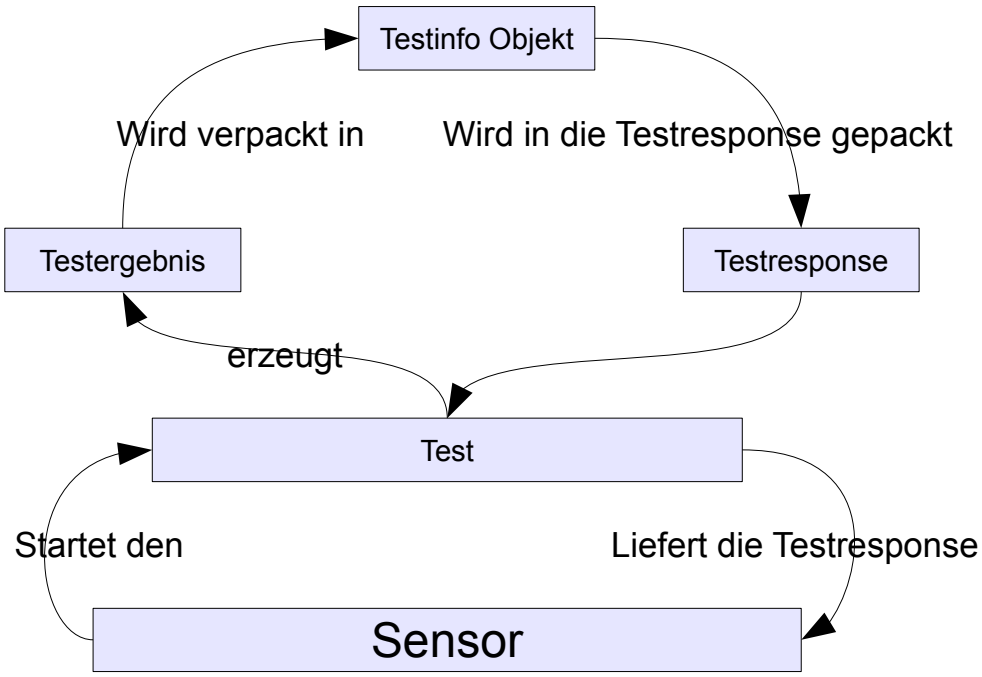
Die *Severerity* gibt den Status des Info-Objektes an. Es gibt die Stufen `Error` (2), `Warning` (1) und `OK` (0).

`Message` ist die von ihnen hinterlegte Nachricht.

```
public interface IStateInfo {
    public int getSeverity();
```

```
    public String getMessage();
    public List<Row> getDetailedInfo();
}
```

Wenn sie möchten, dass ihr Test in der Weboberfläche angezeigt wird, müssen sie außerdem das IoverviewInfo-Interface erfüllen. Ihre Testklasse muss dann auch im <OverviewClasses Classes=-Attribut in der Datei `def.xml` stehen.



Mit `setStateInfo(IStateInfo stateInfo)` werden die Info-Objekte an das Response-Objekt übergeben.

Wichtig ist, dass Ihre eigenen Tests die Klasse `Abstract Testjob` erweitern, damit Ihr Testloader diese Tests dann als Liste von `Abstract Testjob` in der Methode `load` zurückgibt.

Den Quellcode zum TKAnlagentest finden sie im Ordner *TKAnlagentest*.