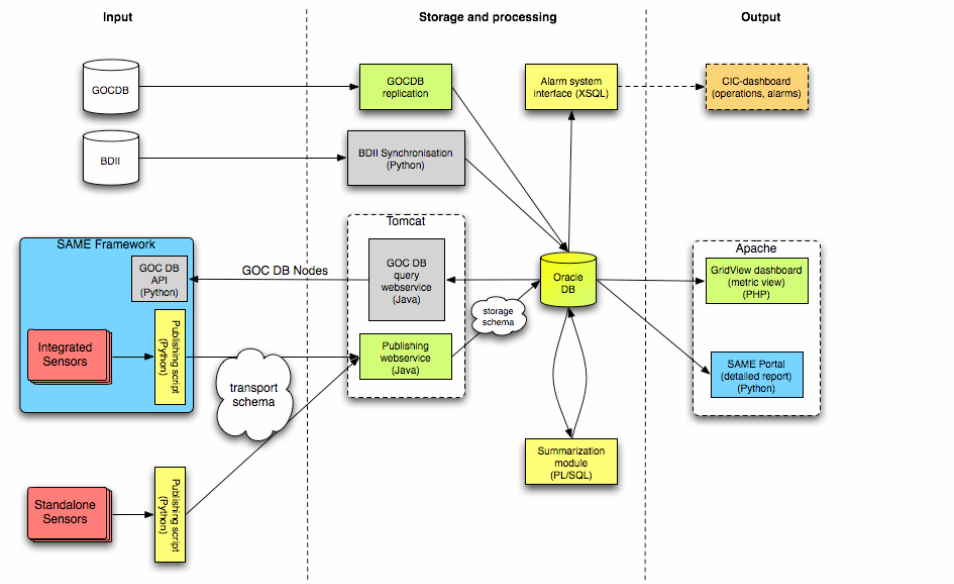
ADDING NEW TESTS AND PUBLISHING RESULTS WITH SAM

D. VICINANZA – CERN/IT
domenico.vicinanza@cern.ch



Second draft version – November 7th, 2006

TABLE OF CONTENTS

INTRODUCTION

Service Availability Monitoring (SAM, previously known as SAME, Service Availability Monitoring Environment) is a monitoring tool conceived as a SFT extension. It has been written mainly in Python (submission framework, logging, error reporting, configuration files), re-using most of the SFT code, with the only exception of the web service API framework.
The heart of the testing and publishing system is an Oracle database, which is connected to the SAM sensors (both the ones integrated into the framework and the standalone ones) through the Tomcat web services. These services include (GOC DB) query and publishing web services implemented in Java or using servlets.
In addition to that, the Oracle DB interacts with the top level BDII using a Python script.

The monitoring procedures are carried out using sensors which regularly publishes the results for all monitored service nodes and that are integrated within the SAM framework mainly following two different approaches:

- Creating a standalone sensor (daemon or cron job) which tests all essential service nodes and publishes the results to SAM using publishing scripts or webservice APIs
- Integrating test script (sensor) with the SAM framework.

The test workflow goes across two stages called test phase and publishing phase. During the test phase, simple sensors performs the tests, then publishing the results, while more complex sensors (like the CE or the FTS ones) carries out basic tests, publish some results and submit some long term test jobs. The publishing phase is optional for the simplest sensors. More complicated sensors (CE, FTS), during this stage, check the status of long term test jobs and publishing the results
Within the SAM testing infrastructure, to seamless integrate sensors, it has been developed a special package called SAM Submission Framework that allows collecting service sensors and providing them a uniform way to:

- discover list of service nodes to test
- schedule and execute tests
- publish results to the central SAM Oracle database

The services currently monitored (i.e. already available) by SAM are: SE, CE, gCE (gliteCE), LFC, SRM, FTS. Some others are still under development, namely: gRB (gliteRB), MyProxy, VOMS, RGMA. Finally other sensonrs have been developed as standalone ones, namely: Top-level BDII, Site BDII, RB. Here follows some details:

CE, gCE

- job submission - UI->RB->CE->WN chain
- version of CA certificates installed (on WN!)

- version of software middleware (on WN!)
- broker info - checking edg-brokerinfo command
- UNIX shells environment consistency (BASH vs. CSH)
- replica management tests - using lcg-utils, default SE defined on WN and a selected “central” SE (3-rd party replication)
- accessibility of experiments software directory - environment variable, directory existence
- accessibility of VO tag management tools
- other tests: R-GMA client check, Apel accounting records

SE, SRM

- storing file from the UI - using lcg-cr command with LFC registration
- getting file back to the UI - using lcg-cp command
- removing file - using lcg-del command with LFC de-registration

LFC

- directory listing - using lfc-ls command on /grid
- creating file entry in /grid/<VO> area

FTS

- checking if FTS is published correctly in the BDII
- channel listing - using glite-transfer-channel-list command with ChannelManagement service
- transfer test (in development): submitting transfer jobs between SRMs in all Tier0 and Tier1 sites (N-N testing), checking the status of jobs

GStat:

- site-BDIIs: accessibility (response time), sanity checks (partial Glue schema validation)
- top-level BDIIs: accessibility (response time), reliability of data (number of entries)

RB:

- jobs submitted through all important RBs to selected “reliable” CEs, measuring time of matchmaking

SAM testing agents are called “sensor”; a sensor is a collection of scripts which are conceived to check the functionalities of a certain service.
Going to the SAM portal home page, the list of sensors is on the left column (see Fig. 1).

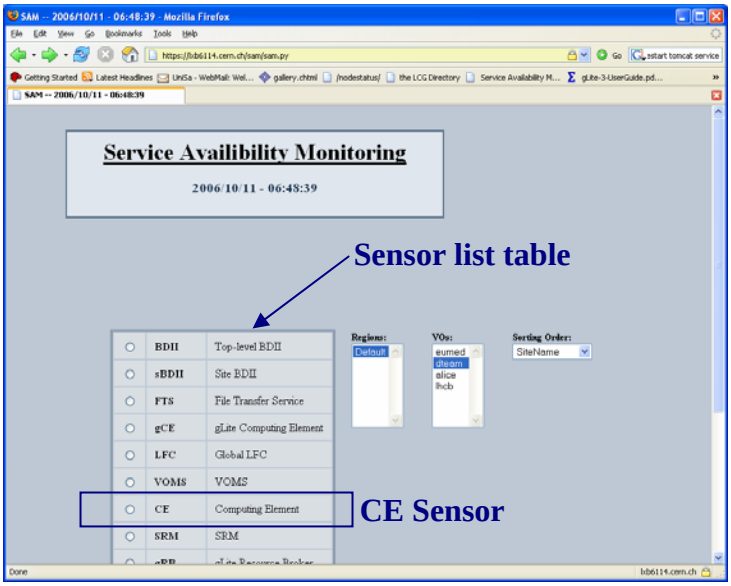


Fig. 1 – SAM portal home page

By clicking on a sensor, for example SE, a detailed view of the tests appears, as if it visible in Fig. 2

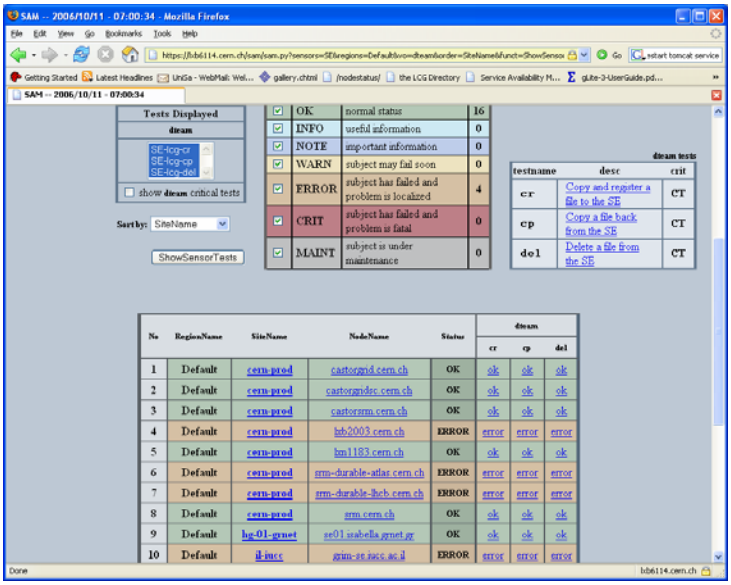


Fig. 2 – SAM SE Sensor page

1. Running tests belonging to a certain sensor

1.1 Getting started

To launch a test, we have to issue firstly a *voms-proxy-init* command with voms extension and choosing an appropriate duration[1]:

```
[vicinanz@ui01]$ voms-proxy-init --voms dteam -valid 25:00
Your identity: /C=CH/O=CERN/OU=GRID/CN=Domenico Vicinanza 6589
Enter GRID pass phrase:
Creating temporary proxy .......................Done
Contacting  lcg-voms.cern.ch:15004 [/C=CH/O=CERN/OU=GRID/CN=host/lcg-
voms.cern.ch] "dteam" Done

Creating proxy ................................. Done
Your proxy is valid until Thu Oct 12 11:21:26 2006
```

Then we can simply start the tests by issuing the *same-exec* command, which is the *same/client/bin* directory. The syntax is (we are, from now on, supposing that SAM is installed in the *$HOME* directory):

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec --help
Usage:
same/client/bin/same-exec [-c <config_file>] [<operation>]
<sensor_name> [<filter>]

If operation is not specified it defaults to '--submit'

Operations supported:
--submit
--publish
--status
--cancel
--nodetest

Filter should be in the follwoing format:
attr1=val1,val2,... attr2=val3,... ...
```

The only compulsory parameter is the <sensor name>; it corresponds to the name of the directory in the *same/client/sensors* folder (see next sections)

[1] It depends on the test scripts they are going to be executed. Some tests like Gilbert Grosdider's ones requires a proxy which lasts for at least 24 hours, that is the why in the example which follows it has been used the *–valid* option with 25h as parameter.

1.2 Running sensors without special publishing requirements

Let us start by running a sensor without special publishing requirements, like the SE one. To run the tests belonging to this sensor one has to write:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec SE
operation: submit
Sensor: SE
Node attrs: ['sitename', 'nodename', 'inmaintenance']
Node filter: {'status': ['Certified'], 'serviceabbr': ['SE'], 'type':
['Production'], 'ismonitored': ['y']}
Sensor args:

Matching nodes: 4
Executing prepare script
----------------------------------------
prepare-FTS script started.
got nodes:      4 nodes.map
publishing test definitions... DONE
----------------------------------------
Prepare script finished successfully
Executing check script for all nodes
----------------------------------------
lxb0724.cern.ch: started
lxshare0297.cern.ch: started
lxb1921.cern.ch: started
lxb2036.cern.ch: started
lxshare0297.cern.ch: publishing
lxb2036.cern.ch: publishing
lxb0724.cern.ch: publishing
lxshare0297.cern.ch: finished
lxb2036.cern.ch: finished
lxb0724.cern.ch: finished
lxb1921.cern.ch: publishing
lxb1921.cern.ch: finished
----------------------------------------
Executing check scripts finished
```

A deeper view about how the “Matching nodes” (i.e. the nodes under test) are determined, is given in the Appendix.
It is possible to have information about the execution status of the sensor by issuing the command *same-exec* with the *–status* option, as in the following example:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec --status SE
operation: status
Sensor: SE
Executing prepare script
----------------------------------------
prepare-FTS script started.
got nodes:      4 nodes.map
publishing test definitions... DONE
----------------------------------------
Prepare script finished successfully
```

1.3 Running sensors with special publishing procedure

Some of the SAM tests publish the results right after their execution (see next sections for details), while other ones (typically more complex sensors like CE) require a separated publishing procedure, accomplished by a special script in the sensor directory (called *prepare-XYZ* for the sensor XYZ) by issuing the command

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec --publish CE
operation: publish
Sensor: CE
Executing prepare script
----------------------------------------
prepare-CE script started.
----------------------------------------
Prepare script finished successfully
```

The list of the tests which in fact require a separated publishing is available having a look at the *same/client/cron/same-cron.sh* (as "*publish_sensors*", in this example "CE FTS gCE"):

```
[vicinanz@ui01 ]$ more same/client/cron/same-cron.sh

#!/bin/bash --login
log=$HOME/same-cron.log
myproxy=myproxy-fts.cern.ch
publish_sensors="CE FTS gCE"
sensors="LFC SE SRM FTS CE gCE"

echo "" >> $log
echo "" >> $log
echo -n "----------------------[ " >> $log
date >> $log
myproxy-get-delegation -s $myproxy -S < $HOME/.myproxy_passphrase >>
$log 2>&1

for sensor in $publish_sensors ; do
echo "" >> $log
echo "----------" >> $log
echo "Publishing $sensor sensor: " >> $log
echo "" >> $log
$HOME/same/client/bin/same-exec --publish $sensor >> $log 2>&1
done

for sensor in $sensors ; do
echo "" >> $log
echo "----------" >> $log
echo "Executing $sensor sensor: " >> $log
echo "" >> $log
$HOME/same/client/bin/same-exec $sensor >> $log 2>&1
Done
```

Summarizing, to correctly launch those tests, the sequence of commands is:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec CE
```

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec --publish CE
```

Before publishing, to be sure that everything is ready (the test scripts are completed and the results are ready), it could be useful to check the status of the execution by issuing the command:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec --status CE
```

1.4 Running sensors on a particular node (using the nodename parameter)

The standard execution of the tests associated to a certain sensor is carried out by SAM by gathering (in a way described in detail in the Appendix) a list of all the nodes which satisfy all the requirements specified in the *same.conf* file for that sensor.
To launch a sensor on a certain node the syntax is the following one:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec <SENSOR NAME>
nodename=<NODE NAME>
```

Where <SENSOR NAME> is one of the available sensors located in the same/client/sensors directory, and whose list could be also accessed by querying the Oracle database:

```
[vicinanz@ui01 ~]$ same/client/bin/same-query serviceabbr
BDII
CE
FTS
LFC
MyProxy
RB
RGMA
SE
SRM
VOMS
gCE
gRB
sBDII
```

<NODE NAME> is one of the nodes actually available to run the sensor (i.e. registered for the right VO and actually running the tested service).

As an example we can have a look at the following command which launches the SE sensor on a well determined storage element, namely n002.grid.cs.um.edu.mt:

```
[vicinanz@ui01 ]$ same/client/bin/same-exec SE
nodename=n002.grid.cs.um.edu.mt

operation: submit
Sensor: SE
Node attrs: ['sitename', 'nodename', 'inmaintenance']
```

```
Node filter: {'status': ['Certified'], 'serviceabbr': ['SE'],
'nodename': ['n002.grid.cs.um.edu.mt'], 'voname': ['eumed'],
'type': ['Production'], 'ismonitored': ['y']}
Sensor args:

Matching nodes: 1
Executing prepare script
----------------------------------------
prepare-FTS script started.
got nodes:      1 nodes.map
publishing test definitions... DONE
----------------------------------------
Prepare script finished successfully
Executing check script for all nodes
----------------------------------------
n002.grid.cs.um.edu.mt: started
n002.grid.cs.um.edu.mt: publishing
n002.grid.cs.um.edu.mt: finished
----------------------------------------
Executing check scripts finished
```

We can focus on the message “Matching nodes: 1”, which confirms that the test is executed on just one node and we can observe the sequence of steps:

```
----------------------------------------
n002.grid.cs.um.edu.mt: started
n002.grid.cs.um.edu.mt: publishing
n002.grid.cs.um.edu.mt: finished
----------------------------------------
```

which actually keeps track of the execution of the SE tests on the required node.
We can also notice that the selected storage element (n002.grid.cs.um.edu.mt) appears in the list of nodes which supports the SE tests (so, it is actually a storage element), as we can easily observe in the output of the following query:

```
[vicinanz@ui01 ]$ same/client/bin/same-query nodename serviceabbr=SE

lxb2003.cern.ch
cclcgseli01.in2p3.fr
se.ulakbim.gov.tr
castorgridsc.cern.ch
grid005.ct.infn.it
aliserv6.ct.infn.it
se01.isabella.grnet.gr
ccsrm.in2p3.fr
n002.grid.cs.um.edu.mt
srm-durable-lhcb.cern.ch
grim-se.iucc.ac.il
gridse.roma3.infn.it
castorgrid.cern.ch
castorsrm.cern.ch
cclcgseli02.in2p3.fr
se1.cnrst.magrid.ma
srm.cern.ch
grid007g.cnaf.infn.it
```

```
srm-durable-atlas.cern.ch
lxn1183.cern.ch
se01.grid.cynet.ac.cy
```

1.5 The .same directory

The results of the test scripts run for each sensor are stored in a special directory which is located in the home SAM directory (following our convention, it is the user home directory) called *.same*.
Let us inspect this directory, looking for the results of the SE tests launched on the node n002.grid.cs.um.edu.mt

First of all we move to the .same directory:

```
[vicinanz@ui01 ]$ cd .same
[vicinanz@ui01 .same]$ ls
CE  cloneCE  FTS  gCE  LFC  SE  SRM
```

The files listed in the .same directory are related to the sensors which have been run at least once (i.e. the first time a sensor is launched, a directory is created). Let's move to the SE folder to inspect the tests on the storage elements:

```
[vicinanz@ui01 .same]$ cd SE

[vicinanz@ui01 SE]$ ls
envName  nodes  nodes.map  same.conf  testFile.txt
```

The nodes.map file contains the list of nodes reached by the tests, i.e. the ones which appear in the list "*Executing check script for all nodes*":

```
[vicinanz@ui01 ] same/client/bin/same-exec SE
…
Matching nodes: 9
…
Executing check script for all nodes
----------------------------------------
se1.cnrst.magrid.ma: started
grid007g.cnaf.infn.it: started
n002.grid.cs.um.edu.mt: started
se.ulakbim.gov.tr: started
se01.isabella.grnet.gr: started
gridse.roma3.infn.it: started
grid005.ct.infn.it: started
grim-se.iucc.ac.il: started
se01.grid.cynet.ac.cy: started
...
----------------------------------------
[vicinanz@ui01 SE]$ more nodes.map
ma-01-cnrst     se1.cnrst.magrid.ma     N
infn-cnaf       grid007g.cnaf.infn.it   N
hg-01-grnet     se01.isabella.grnet.gr  N
```

```
mt-01-uom       n002.grid.cs.um.edu.mt  N
tr-01-ulakbim   se.ulakbim.gov.tr       N
cy-01-cynet     se01.grid.cynet.ac.cy   N
infn-roma3      gridse.roma3.infn.it    N
infn-catania    grid005.ct.infn.it      N
il-iucc         grim-se.iucc.ac.il      N

[vicinanz@ui01 SE]$ more nodes.map |wc -l
  9
```

The testFile.txt is a test text file created by the SE scripts to be transferred on the storage element and the *same.conf* file is the configuration file used while launching the test.

The nodes directory contains a list of subfolders, one per each tested node:

```
[vicinanz@ui01 SE]$ cd nodes

[vicinanz@ui01 nodes]$ ls
aliserv6.ct.infn.it   ccsrm.in2p3.fr         lxn1183.cern.ch
srm.cern.ch
castorgrid.cern.ch    grid005.ct.infn.it     n002.grid.cs.um.edu.mt
srm-durable-atlas.cern.ch
castorgridsc.cern.ch  grid007g.cnaf.infn.it  se01.grid.cynet.ac.cy
srm-durable-lhcb.cern.ch
castorsrm.cern.ch     gridse.roma3.infn.it   se01.isabella.grnet.gr
cclcgseli01.in2p3.fr  grim-se.iucc.ac.il     se1.cnrst.magrid.ma
cclcgseli02.in2p3.fr  lxb2003.cern.ch        se.ulakbim.gov.tr
```

To access the results for the n002.grid.cs.um.edu.mt storage element, it is just needed to enter the directory:

```
[vicinanz@ui01 nodes]$ cd n002.grid.cs.um.edu.mt

[vicinanz@ui01 n002.grid.cs.um.edu.mt]$ ls
SE-lcg-cp.result  SE-lcg-cr.result  SE-lcg-del.result  testFile.lfn
testFile.txt
```

and have a look at the output files:

```
[vicinanz@ui01 n002.grid.cs.um.edu.mt]$ more SE-lcg-cp.result
voName: eumed
envName: SE-1162483223
testName: SE-lcg-cp
nodeName: n002.grid.cs.um.edu.mt
timestamp: 1162483230
status: 50
detailedData: EOT
Checking if a file can be copied back from n002.grid.cs.um.edu.mt
<pre>
++ pwd
+ lcg-cp -v --vo eumed lfn:SE-lcg-cr-n002.grid.cs.um.edu.mt-1162483224
file:/home/domenico/.same/SE/nodes/n002.grid.c
s.um.edu.mt/testFile.txt
lcg_cp: No such file or directory
```

```
Using grid catalog type: lfc
Using grid catalog : lfc.isabella.grnet.gr
+ result=1
+ set +x
</pre>
Checking if files are identical
<pre>
+ diff testFile.txt /home/domenico/.same/SE/testFile.txt
+ set +x
</pre>
EOT
```

In particular, we can observe, in this example the line

```
status: 50
```

which means that the execution of the test was unsuccessful.

1.6 Customizing the timeout (test and overall sensor)

The timeout (maximum allowed execution time) for each test of a sensor and for the whole sensor is set in the *same/client/etc/same.conf* file in the sections [submission] and [scheduler]:

```
[vicinanz@ui01 ]$ more same/client/etc/same.conf
# Default configuration for SAME
…

[submission]
#vo=dteam
vo=eumed
test_timeout=240

[scheduler]
max_processes=10
default_timeout=300
shell=/bin/sh
….
```

The *test_timeout* allows to set (in seconds) the maximum execution time allowed to each test of a certain sensor (in this example it has been set to be 4 minutes).
The *default_timeout* is the overall execution time limit (in seconds) of all the tests of a sensor, and in this case it has been set to 5 minutes.

2. Adding new sensors by wrapping existing test scripts

2.1 Sensor directory structure

Listing the directory *same/client/sensors* it is easy to find that each sensor has its own directory organized as follows:

- A "check" script
- A "prepare" script
- A file containing the sequence of execution of the single scripts of the test, called *test-sequence.lst*
- A *tests* directory containing the scripts sources
- Eventually, some other additional files (JDL scripts, scripts helping to return information about the status of the test execution)

Let us imagine having a shell script called *myscript.sh* installed in our SAM client machine, in the *$HOME* directory and let us imagine to integrate this test within a SAM sensor. We basically have two possibilities:

- add new tests to a certain existing sensor (i.e. it already exists a directory in the *same/client/sensors* corresponding to the service to test)
- add a new same sensor (i.e. it is necessary to create a new directory in the same/client/sensors)

Let us consider the first case. We shall take as en example the *gRB* sensor, for which there is just an empty directory.

2.2 Naming convention

Creating a new sensor, let is name be *XYZ*, one has to fulfill the following naming convention:

- the directory in the same/sensor folder have to be named *XYZ*
- inside *XYZ* it has to be created a directory named tests containing all the scripts to be executed.
- the check and prepare script have to be named respectively: *check-XYZ* and *prepare-XYZ*
- the list of the scripts to be launched when the sensor is executed have to be put in the a file called *test-sequence.lst* in the *XYZ* folder
- each test in the tests directory has to be named with a string starting by the name of the sensor, in our example: *XYZ-<name>*
- each test in the tests directory has to have a description/configuration file called *XYZ-<name>.def*

As an example we can consider the SE sensor (the one which tests the storage elements, using three scripts). Listing the *same/client/sensor/SE* directory we have:

```
[vicinanz@ui01 SE]$ ls
check-SE  prepare-SE  tests  test-sequence.lst
```

We can inspect the content of the *test-sequence.lst* file:

```
[domenico@ui01 SE]$ cat test-sequence.lst
SE-lcg-cr
SE-lcg-cp
SE-lcg-del
```

and the content of the *tests* directory:

```
[domenico@ui01 SE]$ ls tests
SE-lcg-cp  SE-lcg-cp.def  SE-lcg-cr  SE-lcg-cr.def  SE-lcg-del  SE-lcg-
del.def
```

The names listed in the *test-sequence.lst* file have to match with the ones of the corresponding script in the *tests* directory.

2.3 Writing the check-XX, prepare-XX script and test-sequence.lst

We firstly enter the directory *gRB* and create the file *check-gRB*. This is the content of a standard *check-XX* script which is normally suitable for general purpose sensors with no special publish requirements (no separated publishing phase) adapted

```
[domenico@ui01 gRB]$ cat check-gRB
#!/bin/bash

siteName=$1
nodeName=$2
inMaintenance=$3
envName=`cat $SAME_SENSOR_WORK/envName`

if [ "x$1" == "x--publish" ] ; then
    shift
    echo "$nodeName: published" >&2
else
    echo "$nodeName: started" >&2
    mkdir -p $SAME_SENSOR_WORK/nodes/$nodeName
    cd $SAME_SENSOR_WORK/nodes/$nodeName
    rm -f *.result
for test in `cat $SAME_SENSOR_HOME/test-sequence.lst` ; do
    same-run-test -v $SAME_VO -t $test -n $nodeName –e
    $envName -d $SAME_SENSOR_HOME/tests $nodeName >
    $test.result
done
    echo "$nodeName: publishing" >&2
    same-publish-tuples TestData *.result >&2
    echo "$nodeName: finished" >&2
fi
```

We can focus the attention of the loop:

```
for test in `cat $SAME_SENSOR_HOME/test-sequence.lst` ;
```

This script would take the *test-sequence.lst* file and it will execute one by one the scripts whose name is in the list. Each test will write a *<testname>.result* file and at the end of the execution of all the scripts (to have a more efficient transaction) the results will be published on the database with the command:

```
same-publish-tuples TestData *.result
```

This is the standard way adopted for simple tests that do not require any special publishing.
If we would need to publish results right after the execution of each single test (it could be useful when there is a list of very slow tests and we want to look at the results as soon as they finish), the previous script can be adapted by moving the publishing command (*same-publish-ntuples*) within the for cycle as follows:

```
[domenico@ui01 gRB]$ cat check-gRB
#!/bin/bash

siteName=$1
nodeName=$2
inMaintenance=$3
envName=`cat $SAME_SENSOR_WORK/envName`

if [ "x$1" == "x--publish" ] ; then
    shift
    echo "$nodeName: published" >&2
else
    echo "$nodeName: started" >&2
    mkdir -p $SAME_SENSOR_WORK/nodes/$nodeName
    cd $SAME_SENSOR_WORK/nodes/$nodeName
    rm -f *.result
for test in `cat $SAME_SENSOR_HOME/test-sequence.lst` ; do
    same-run-test -v $SAME_VO -t $test -n $nodeName –e
    $envName -d $SAME_SENSOR_HOME/tests $nodeName >
    $test.result
    echo "$nodeName: publishing" >&2
    same-publish-tuples TestData $test.result >&2
    echo "$nodeName: finished" >&2
done
fi
```

The other script we have to write is the *prepare-gRB*. In the same manner we can edit this file using this standard template:

```
[vicinanz@ui01 gRB]$ cat prepare-gRB
#!/bin/bash
if [ "x$1" == "x--publish" ] ; then
```

```
    echo "This sensor does not support publish operation!" >&2
else
    echo -n "got nodes:" >&2
    wc -l nodes.map >&2
    cd $SAME_SENSOR_WORK
    echo -n "publishing test definitions... " >&2
    same-publish-tuples TestDef $SAME_SENSOR_HOME/tests/*.def >&2
    envName=gRB-`date +%s`
    cat <<EOF | same-publish-tuples TestEnvVars
envName: $envName
name: dummy
value: none
EOF
    echo $envName > $SAME_SENSOR_WORK/envName
    echo "DONE" >&2
fi
```

Finally let us create the of the *test-sequence.lst* file just putting into it the name of the SAM script we are going to create in the *scripts* directory. Let us consider to add just one test to the gRB sensor, called gRB-mytest.

```
[vicinanz@ui01 SE]$ cat test-sequence.lst
gRB-mytest
```

2.4 Writing the test and test definition files

Once we added *gRB-mytest* in the *test-sequence* file we have to create a couple of files in the *tests* directory named:

- *gRB-mytest*
- *gRB-mytest.def*

The content of the .def file is quite simple:

```
[vicinanz@ui01 gRB]$ cd tests
[vicinanz@ui01 tests]$ more gRB-mytest.def
testName: gRB-mytest
testAbbr: mytest
testTitle: mytest script wrapped into the SAM environemt
testHelp: http://...
EOT
```

As it is possible to see, the file contain some definitions, namely each of them contains:

- testName: name of the test (it goes into the “Tests Displayed” table in the SAM sensor webpage)
- testAbbr: abbreviated name of the test (it goes in the sensor result table, as column name)
- testTitle: brief description of the test

As a comparison this is the *SE-lcg-cr.def* content, one of the SE sensor tests, which start checking the storage element by using the lcg copy-and-register command (*lcg-cr*).

```
[domenico@ui01 tests]$ more SE-lcg-cr.def
testName: SE-lcg-cr
testAbbr: cr
testTitle: Copy and register a file to the SE
EOT
```

Having a look at Fig. 3, it is possible to see where these parameters (testName: SE-lcg-cr and testAbbr: cr) appear in the SE sensor webpage:

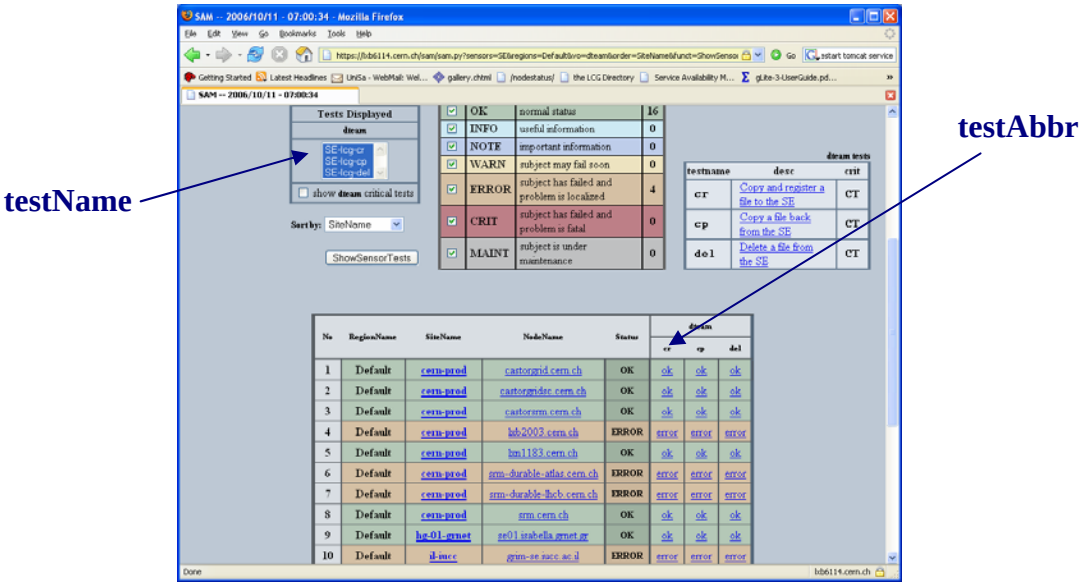


Fig. 3 – SAM SE Sensor page

Let us come to the *gRB-mytest* script now. We suppose to have a script named *myscript.sh* in our *$HOME* directory and let us suppose that this script would write an output file called *output.xml* in the local directory, and that we have prepared a simple parser called *parser* to extract the status of the test (i.e. a parser which exit with a status code "0" if the results are fine, "2" in case of warnings, "3" if it fails, "1" if it happens something that prevent the execution of the test, so the status is "untested").

The first thing we have to do is moving the *myscript.sh* from our *$HOME* to the *same/client/sensor/gRB* directory. This path, i.e. the local path of the sensor will be available within the SAM scripts thanks to the environment variable: *$SAME_SENSOR_HOME*. The output file will be so available at *$SAME_SENSOR_HOME/parser*.

Please note the check on the existence of the file made at the line:
[-f "$OUT_FILE"] || exit $SAME_ERROR

```
[vicinanz@ui01 tests]$ more gRB-mytest
#!/bin/bash

export OUT_FILE="output.xml"
$SAME_SENSOR_HOME/myscript.sh
[ -f "$OUT_FILE" ] || exit $SAME_ERROR
mv $OUT_FILE $SAME_SENSOR_HOME
$SAME_SENSOR_HOME/parser $SAME_SENSOR_HOME/$OUT_FILE
CODE=$?
exit $CODE
```

As a comparison this is the *SE-lcg-cr* content:

```
[domenico@ui01 tests]$ more SE-lcg-cr
#!/bin/bash
export LFC_HOME=/grid/$SAME_VO/SFT
export LCG_CATALOG_TYPE=lfc
timestamp=`date +%s`
nodeName=$1
echo "Checking if a file can be copied and registered to $nodeName"
echo "<pre>"
lfn="lfn:SE-lcg-cr-$nodeName-$timestamp"
echo $lfn >testFile.lfn
set -x
lcg-cr -v --vo $SAME_VO file:$SAME_SENSOR_WORK/testFile.txt -l $lfn -d
$nodeName
result=$?
set +x
echo "</pre>"
if [ $result  == 0 ] ; then
    exit $SAME_OK
else
    exit $SAME_ERROR
fi
```

2.5 A deeper insight: the $test.result file of our script

Running a certain test, SAM will execute the command:

```
same-run-test -v $SAME_VO -t $test -n $nodeName –e $envName -d
$SAME_SENSOR_HOME/tests $nodeName > $test.result
```

The *$test.result* file will contain all the information required for publishing the test results to the server. This is, for example, the content of *gRB-mytest.result* file, created by SAM while executing out *gRB-mytest* script:

```
voName: dteam
envName: gRB-1160731486
testName: gRB-mytest
nodeName: lxb2032.cern.ch
timestamp: 1160731487
status: 10
```

```
detailedData: EOT
I am the script!
<pre>
[OK]
</pre>
EOT
```

As it is possible to see, it is conceptually divided in two parts, a first one containing:

- voName: dteam
- envName: gRB-1160731486
- testName: gRB-mytest
- nodeName: lxb2032.cern.ch
- timestamp: 1160731487
- status: 10

and a second one containing the output of the test (the one addressed to the standard output device, the printout in this case) enclosed in two “EOT” strings:

- detailedData: EOT
  I am the script!
  <pre>
  [OK]
  </pre>
  EOT

The status code corresponds to OK. The full list of all the codes is specified in the *same/client/etc/same.conf* file:

```
…
[statuscode]
ok=10
info=20
notice=30
warning=40
error=50
critical=60
maintenance=100
…
```

(other details will be provided later on in the document, where the XML to SAM parser is described)

3. Screenshot from the new gRB sensor (with the gRB-mytest)

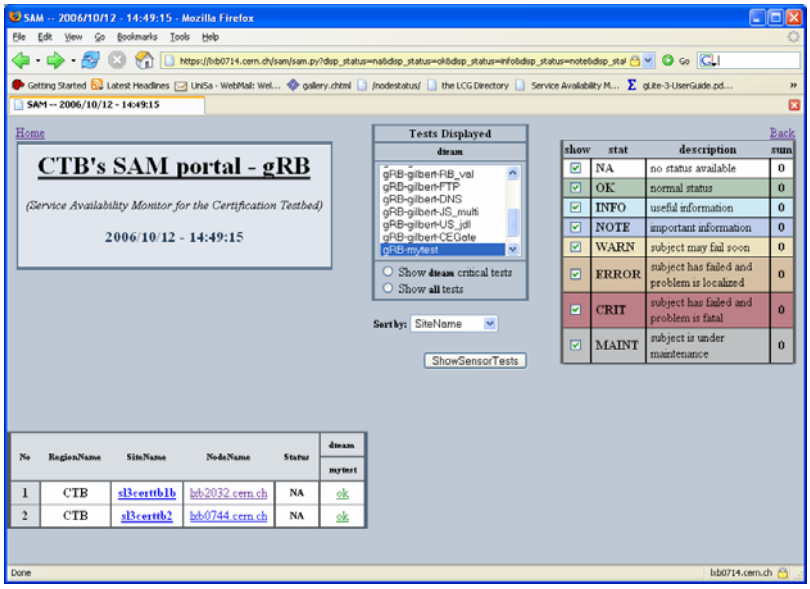


Fig. 4 – SAM gRB Sensor page with the new script

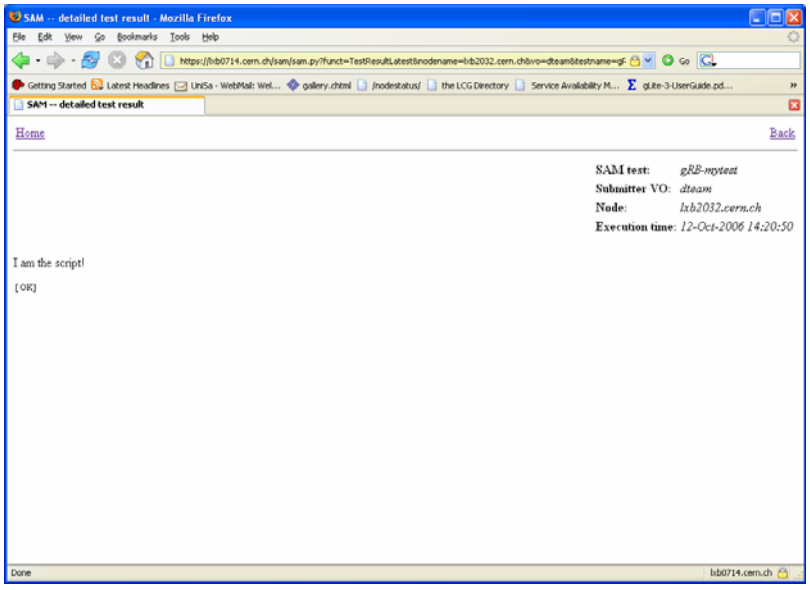


Fig. 5 – SAM output page of the new script

4. An example of myscript.sh

This is an example of a toy *myscript.sh* (the simplest one) which creates a valid xml file to be processed by the parser which follows in the next section. Their location is the same of the parser, the *same/client/sensor/gRB* directory)

```
[vicinanz@ui01 gRB]$ more myscript.sh
#!/bin/sh
echo "I am the script!"
RESULT="OK"
OUTPUT_FILE="output.xml"
touch $OUTPUT_FILE
echo "prova" >> $OUTPUT_FILE
echo "<html>" >> $OUTPUT_FILE
echo "  <body>" >> $OUTPUT_FILE
echo "    <test>mytest script" >> $OUTPUT_FILE

echo "      <result>[OK]</result>" >> $OUTPUT_FILE

echo "    </test>" >> $OUTPUT_FILE
echo "  </body>" >> $OUTPUT_FILE
echo "</html>" >> $OUTPUT_FILE
```

Finally this is the output of the script:

```
[vicinanz@ui01 gRB]$ more output.xml

<html>
  <body>
    <test>mytest script
      <result>[OK]</result>
    </test>
  </body>
</html>
```

5. The XML to SAM parser

5.1 The parser code

Here follow an example of XML to SAM parser which would return SAM compatible status (in our example it is called just *parser* and it is located in the *same/client/sensor/gRB* directory):

```
#!/usr/bin/perl -w
##############################################################################
##
# Script to convert XML log file to SAM-compatible output
# Author: Andrey Kiryanov <Andrey.Kiryanov@cern.ch>
# adapted by Domenico Vicinanza <Domenico.Vicinanza@cern.ch>
#
# $Id:$
#
##############################################################################
#
use strict;

my ($xml, $test, $code, %res);
my %map = (
'[OK]' => [0, $ENV{SAME_OK}],
'[FAIL]' => [3, $ENV{SAME_ERROR}],
'[WARNING]' => [2, $ENV{SAME_WARNING}],
'[XFAIL]' => [3, $ENV{SAME_ERROR}],
'[UNRESOLVED]' => [2, $ENV{SAME_WARNING}],
'[UNTESTED]' => [1, $ENV{SAME_INFO}],
'[UNSUPPORTED]' => [1, $ENV{SAME_INFO}]);

sub chopsection($$) {
        if($_[0] =~
s/^((.|\n)*?)<\s*($_[1])\s*>\s*((.|\n)*?)\s*<\s*\/\3\s*>/$1/i) {
                return $4;
        } else {
                return undef;
        }
}

my $in_file = shift;
die "Usage: xml2sam <XML filename>\n\n" unless $in_file;
open(IN, "<$in_file") || die("Cannot open input file: $!\n");
while(<IN>) {
        $xml .= $_;
}
close(IN);

#Strip comments and determine whether input file is valid XML log
$xml =~ s/<!--\s(.|\n)*?\s-->//g;
$xml = chopsection($xml, "body");
die("Input file is not a valid XML log (body section missing)\n")
unless defined $xml;
```

```
# Process the TEST sections
print "<pre>\n";
while($test = chopsection($xml, "test")) {
        undef %res;
        $res{summaryData} = chopsection($test, "result");
        if(! defined $res{summaryData}) {
                warn "Warning: XML log has no RESULT section\n";
                $res{summaryData} = "";
        }
        $code = $res{summaryData} if $res{summaryData} and (! defined
$code or $map{$code}[0] <
$map{$res{summaryData}}[0]);
}
print "$code\n";
print "</pre>\n";
exit $map{$code}[1] if defined $code and defined $map{$code}[1];
```

5.2 The SAM exit codes code

The exit codes the parser is referring to (see last line) are the one reported into the *same/client/etc/same.conf* file, in the section named [statuscode]

```
…
[statuscode]
ok=10
info=20
notice=30
warning=40
error=50
critical=60
maintenance=100
…
```

These values are then available as environment variable named respectively:
$SAME_OK
$SAME_INFO
$SAME_NOTICE
$SAME_WORNING
$SAME_ERROR
$SAME_CRITICAL
$SAME_MANTAINANCE

This allows us to write the parser coding the status [OK], [FAIL], … with the corresponding environment variable ($ENV{SAME_OK}, $ENV{SAME_ERROR}, …):

```
'[OK]' => [0, $ENV{SAME_OK}],
'[FAIL]' => [3, $ENV{SAME_ERROR}],
'[WARNING]' => [2, $ENV{SAME_WARNING}],
'[XFAIL]' => [3, $ENV{SAME_ERROR}],
```

```
'[UNRESOLVED]' => [2, $ENV{SAME_WARNING}],
'[UNTESTED]' => [1, $ENV{SAME_INFO}],
'[UNSUPPORTED]' => [1, $ENV{SAME_INFO}]);
```

APPENDIX: The SAM query command and the Matching nodes

Among the SAM command there is a query command which looks for data in the Oracle database according to some filters entered as arguments by the user. For example let us consider the following command which lists all the site names (this example refers to the CERN Certification Testbed BDII):

```
[vicinanz@ui01 ~]$ same/client/bin/same-query sitename
sl3certtb2
sl3certtb1a
sl3certtb1b
```

These are other examples: the first one is a query selecting all the nodes belonging to the site *sl3certtb1b.*

```
[vicinanz@ui01]$ same/client/bin/same-query nodename
sitename=sl3certtb1b
lxb2016.cern.ch
lxshare0297.cern.ch
lxb2033.cern.ch
lxb1917.cern.ch
lxb1737.cern.ch
lxb2034.cern.ch
lxb1762.cern.ch
lxb1928.cern.ch
lxb1782.cern.ch
lxb2011.cern.ch
lxb1905.cern.ch
lxb1941.cern.ch
lxb2032.cern.ch
```

The second one is a query which lists all the nodenames and sitenames tagged as belonging to the “Production” cluster:

```
[vicinanz@ui01 ~]$ same/client/bin/same-query sitename nodename
type=Production
sl3certtb2      lxb1751.cern.ch
sl3certtb1a     lxb2018.cern.ch
sl3certtb1a     lxb2019.cern.ch
sl3certtb1a     lxb2017.cern.ch
sl3certtb1b     lxb2034.cern.ch
sl3certtb1b     lxb1917.cern.ch
sl3certtb1b     lxb2032.cern.ch
sl3certtb1b     lxb1782.cern.ch
sl3certtb1b     lxshare0297.cern.ch
sl3certtb1b     lxb2033.cern.ch
sl3certtb2      lxb2035.cern.ch
sl3certtb2      lxb1915.cern.ch
sl3certtb1b     lxb2016.cern.ch
sl3certtb1a     lxb2020.cern.ch
sl3certtb1b     lxb1928.cern.ch
```

```
sl3certtb1b     lxb2011.cern.ch
sl3certtb2      lxb0743.cern.ch
sl3certtb2      lxb0730.cern.ch
sl3certtb1a     lxb1921.cern.ch
sl3certtb1b     lxb1737.cern.ch
sl3certtb1b     lxb1905.cern.ch
sl3certtb1b     lxb1941.cern.ch
sl3certtb2      lxb0724.cern.ch
sl3certtb2      lxb2036.cern.ch
sl3certtb2      lxb0744.cern.ch
sl3certtb1a     lxb1909.cern.ch
sl3certtb1b     lxb1762.cern.ch
```

Finally this is the way to query the list of the available sensors (the ones which appear in the SAM web page):

```
[vicinanz@ui01 ~]$ same/client/bin/same-query serviceabbr
BDII
CE
DPM
FTS
FTS2
LFC
MyProxy
RB
RGMA
SE
SRM
SandBox
VOMS
gCE
gRB
sBDII
```

In one of the previous sections we saw that a standard *check-XYZ* file starts with the lines:

```
[domenico@ui01 gRB]$ cat check-gRB
#!/bin/bash

siteName=$1
nodeName=$2
inMaintenance=$3
```

The main goal of this paragraph is to something more about these three variables. When one issues the command

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec <sensor-name>
```

SAM takes from the *same/client/etc/same.conf* file some info from the section [sensors]:

```
[vicinanz@ui01 ~]$ more same/client/etc/same.conf
```

```
# Default configuration for SAME



[DEFAULT]
# Settings for locations
workdir=%(home)s/.same
logdir=%(same_home)s/var/log
resdir=%(same_home)s/var/results
secresdir=%(same_home)s/var/results-secure
webdir=%(same_home)s/web
cachedir=%(same_home)s/var/cache

# Logging levels:
# CRITICAL, ERROR, WARNING, INFO, DEBUG, NOTSET
# Logging level for the log file
loglevel=INFO
# Logging level for console messages
verbosity=CRITICAL

[sensors]
common_attrs="sitename nodename inmaintenance"
common_filter="type=Production status=Certified ismonitored=y"
CE_filter="serviceabbr=CE"
gCE_filter="serviceabbr=gCE"
FTS_filter="serviceabbr=FTS"
FTS_attrs="sitename nodename inmaintenance tier"
SE_filter="serviceabbr=SE"
SRM_filter="serviceabbr=SRM"
LFC_filter="serviceabbr=LFC voname=dteam"
host-cert_attrs="nodename serviceabbr"
host-
cert_filter="serviceabbr=FTS,gCE,LFC,VOMS,CE,SRM,gRB,MyProxy,RB,SE,RGMA
"
…
```

More precisely SAM composes a query to retrieve the name of the nodes where the tests will run according to the

- parameters coming from *common_attr*
- parameters coming from *common_filter*
- eventually parameters coming from *XYZ_filter* (where XYZ is the sensor, if it exists a special set of parameters for that sensor.)

The filters will build the parameters through which the same-query will select the nodes to test (to pass to the sensors). As an example we can consider the SE case. The parameters are:

- *sitename nodename inmaintenance* coming from *common_attr*
- *type=Production status=Certified ismonitored=y* coming from *common_filter*
- *serviceabbr=SE* coming from *SE_filter*

So the query launched by SAM main program to gather the list of nodes addressee of the SE tests is:

```
[vicinanz@ui01 ~]$ same/client/bin/same-query sitename nodename
inmaintenance type=Production status=Certified ismonitored=y
serviceabbr=SE
sl3certtb1a     lxb1921.cern.ch N
sl3certtb2      lxb0724.cern.ch N
sl3certtb1b     lxshare0297.cern.ch     N
sl3certtb2      lxb2036.cern.ch N
```

The output is a list of three data, *sitename*, *nodename* and a Boolean variable called *inmanteinance* which will be mapped to the *check-SE* script as $1, $2 and $3:

```
[vicinanz@ui01 ~]$ more same/client/sensors/SE/check-SE
#!/bin/bash

siteName=$1
nodeName=$2
inMaintenance=$3
…
```

When we are going to run the SE sensor, all the tests contained in the *same/client/sensors/SE/test-sequence.lst* file will be executed on the four machines coming out from the same-query command:

```
[vicinanz@ui01 ~]$ same/client/bin/same-exec SE
operation: submit
Sensor: SE
Node attrs: ['sitename', 'nodename', 'inmaintenance']
Node filter: {'status': ['Certified'], 'serviceabbr': ['SE'], 'type':
['Production'], 'ismonitored': ['y']}
Sensor args:

Matching nodes: 4
Executing prepare script
----------------------------------------
prepare-FTS script started.
got nodes:      4 nodes.map
publishing test definitions... DONE
----------------------------------------
Prepare script finished successfully
Executing check script for all nodes
----------------------------------------
lxshare0297.cern.ch: started
lxb0724.cern.ch: started
lxb2036.cern.ch: started
lxb1921.cern.ch: started
lxshare0297.cern.ch: publishing
lxshare0297.cern.ch: finished
lxb2036.cern.ch: publishing
lxb2036.cern.ch: finished
lxb0724.cern.ch: publishing
lxb0724.cern.ch: finished
```

```
lxb1921.cern.ch: publishing
lxb1921.cern.ch: finished
----------------------------------------
Executing check scripts finished
```

If the sensor gRB we are developing there are no specific restriction/filters, we can avoid to specify any particular string, so they will be used for the node matching just the *common_attr* and *common_filters* information:

```
[vicinanz@ui01 ~]$ same/client/bin/same-query sitename nodename
inmaintenance type=Production status=Certified ismonitored=y
sl3certtb1a     lxb1921.cern.ch N
sl3certtb1a     lxb2018.cern.ch N
sl3certtb1b     lxb2016.cern.ch N
sl3certtb1b     lxb1941.cern.ch N
sl3certtb2      lxb0724.cern.ch N
sl3certtb2      lxb2035.cern.ch N
sl3certtb1b     lxb1762.cern.ch N
sl3certtb2      lxb0730.cern.ch N
sl3certtb1b     lxb2011.cern.ch N
sl3certtb2      lxb0743.cern.ch N
sl3certtb1b     lxb1905.cern.ch N
sl3certtb1b     lxshare0297.cern.ch     N
sl3certtb2      lxb2036.cern.ch N
sl3certtb1b     lxb2034.cern.ch N
sl3certtb1a     lxb1909.cern.ch N
```

Additional information: workplan

Service	Responsible	Class	Comments
SRM 2.1	Piotr Nyczyk Dave Kant	C	DONE, simple sensor implemented by Piotr and integrated with SAM framework
LFC	James Casey	C/H	DONE, sensor integrated with SAM framework
FTS	Piotr Nyczyk Gavin McCance	C	DONE, sensor integrated with SAM framework
CE	Piotr Nyczyk	C	DONE, monitored by SFT today
RB	Dave Kant Sergio Andreozzi	C	DONE, standalone sensor publishing to SAM
Top-level BDII	Min-Hong Tsai	C	DONE, standalone sensor (GStat) publishing to SAM
Site BDII	Min-Hong Tsai	H	DONE, standalone sensor (GStat) publishing to SAM
MyProxy	Maarten Litmaath	C	sensor to be written and integrated with SAM framework
VOMS	Valerio Venturi	C	sensor to be written and integrated with SAM framework

The Service Class is a set of parameters which share the same service level objectives. It provides an easy way of describing the high level parameters required for a service: C stands for Critical and H for High (more information available at: https://uimon.cern.ch/twiki/bin/view/LCG/ScFourServiceDefinition#Service_Class.

6. References

SAM reference website (information about sensors and metrics assignment, architecture, data schema, submission framework, metrics)	http://goc.grid.sinica.edu.tw/gocwiki/Service_Availability_Monitoring_Environment
SAM list of sensors and all the details about their development	http://goc.grid.sinica.edu.tw/gocwiki/Service_Availability_Sensors
SAM monitoring data schema	http://goc.grid.sinica.edu.tw/gocwiki/Monitoring_Data_Schema
SAM submission framework	http://goc.grid.sinica.edu.tw/gocwiki/SAME_Submission_Framework