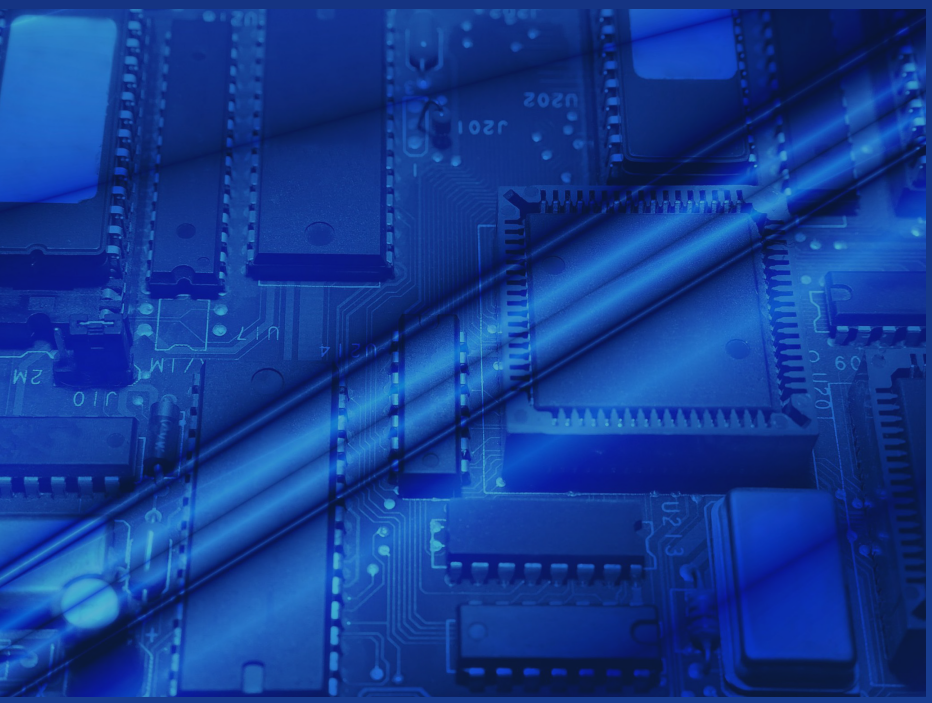
ARGENT
ENCYCLOPEDIA

Unix Monitoring
Overview



Contents

UNIX Monitoring Using Argent Extended Technology

Argent XT is a comprehensive monitoring and alerting solution that can be utilized to manage the UNIX environment.

Argent XT provides a suite of products, as well as flexible architecture options for varied network environments and security requirements.

Argent Guardian provides the ability to monitor all metrics on a UNIX server, through Rules written as fully customizable shell scripts.

Argent Predictor is used to create Graphs and Reports on performance data that is saved through running the Argent Guardian Rules.

Argent Data Consolidator is a comprehensive solution for managing log files, whether they be ASCII files, syslog messages, etc.

Argent SNMP Monitor can be used to create SNMP based rules where SNMP is enabled on a UNIX server.

Argent Monitor for VMware can use the same type of UNIX shell script Rules for customized monitoring of VMware ESX servers.

This manual is provided as an introduction to monitoring UNIX with Argent XT, and the following knowledge is assumed:

- Working knowledge of UNIX systems
- Basic understanding of Bourne Shell scripting
- Basic understanding of XML
- Familiarity with Argent XT concepts

All of Argent's documentation can be found online at:
http://help.Argent.com

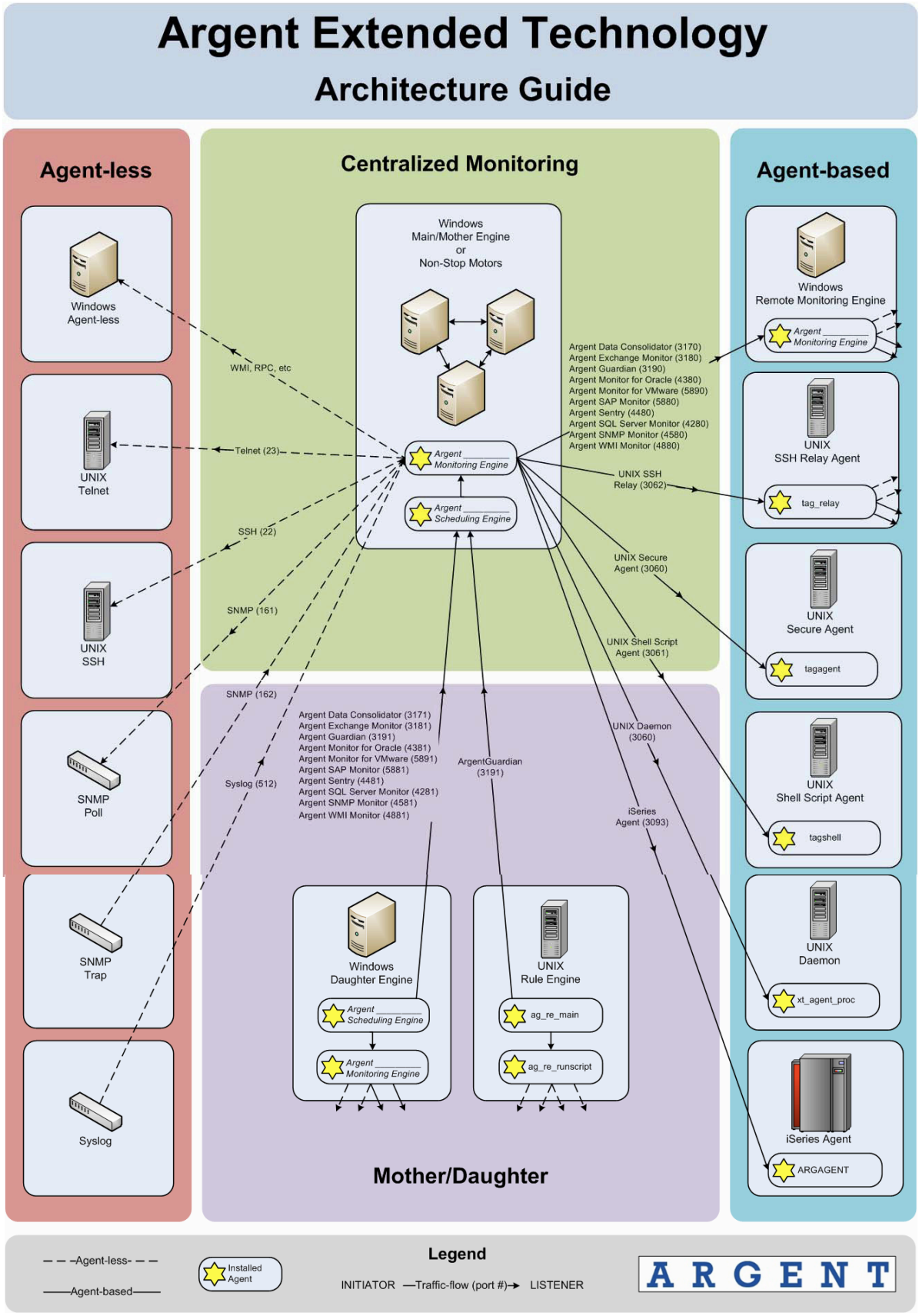
Argent XT offers SEVEN options for monitoring UNIX systems.

- Telnet
- SSH
- SSH Relay Agent
- UNIX Secure Agent
- UNIX Shell Script Agent
- UNIX Daemon
- UNIX Rule Engine

Each of these options provides different features and benefits, depending on your security requirements, network architecture, etc.

Broadly, the Argent XT architecture options can be divided into two categories: agent-less and agent-based monitoring.

Architecture



Architecture – Agent-less

Agent-less monitoring is the easiest to setup as it requires nothing to be installed on the UNIX servers that are going to be monitored. All Rules (shell scripts) are stored on the main Argent XT Windows server and are executed remotely via a Telnet or SSH session.

Telnet

A Windows Monitoring Engine executes the Rule through a Telnet session, and the information is read back for centralized alerting and reporting.

Telnet communication, of course, is completely plain-text, making it not secure.

SSH

SSH, on the other hand, is completely encrypted from end to end, making it secure.

This is achieved from the Argent XT Windows server by utilizing the PuTTy tools PSCP and PLINK.

For a detailed explanation of the steps, see: Appendix A: UNIXSSH.INI

Architecture – Agent-based

SSH Relay Agent

The SSH Relay Agent shifts the SSH connection workload from a Windows Monitoring Engine to a UNIX machine, which generally make SSH connections more efficient.

This requires installing the SSH Relay Agent to a dedicated UNIX server that will monitor the other UNIX servers.

A Windows Monitoring Engine sends the Rule to the SSH Relay Agent, which then makes the connection to the monitored servers via SSH, executes the Rule, then sends the information back.

As an alternative to Telnet and SSH, you can install a local UNIX monitoring agent on the UNIX systems you wish to monitor.

UNIX Secure Agent

The UNIX Secure Agent is a binary executable that is installed on the servers to be monitored.

It is started via the inetd/xinetd super daemon and listens on a dedicated secure channel.

Rules are sent from the Windows Monitoring Engine, executed locally and the information sent back.

UNIX Shell Script Agent

The UNIX Shell Script Agent is just that -- a shell script that is installed on the servers to be monitored.

Similar to the UNIX Secure Agent, it is started via inetd/xinetd and listens on a secure channel.

Being just a shell script allows for user customization and for use on systems where a UNIX Secure Agent binary does not yet exist.

UNIX Daemon

The UNIX Daemon is similar to the UNIX Secure Agent, but does not require inetd/xinetd.

It is installed as its own daemon, listening on a secure channel for Rules that are sent from the Windows Monitoring Engine.

UNIX Rule Engine

The UNIX Rule Engine is similar in concept to the Argent Mother/Daughter architecture.

It retrieves its marching orders from the main Argent server, makes SSH connections to the monitored servers, and sends back the results to the main Argent server.

This provides additional fault tolerance, as well as removing the connection load from a Windows Monitoring Engine. Since each UNIX Rule Engine initiates the connection to the main Argent server, firewall configuration is also simplified, since you only have to open one port.

For more detailed information on the seven options, see also:
http://help.Argent.com/#unix_seven

Argent Guardian

The Argent Guardian is the world's most scalable monitoring solution for all Windows, UNIX, and iSeries applications, monitoring the health and performance of all critical business applications through a unique architecture - Argent provides the same level of monitoring with or without agents, and there is no cost difference.

Applications running on UNIX servers are monitored using a script-based system that is both flexible and popular with UNIX and Linux administrators.

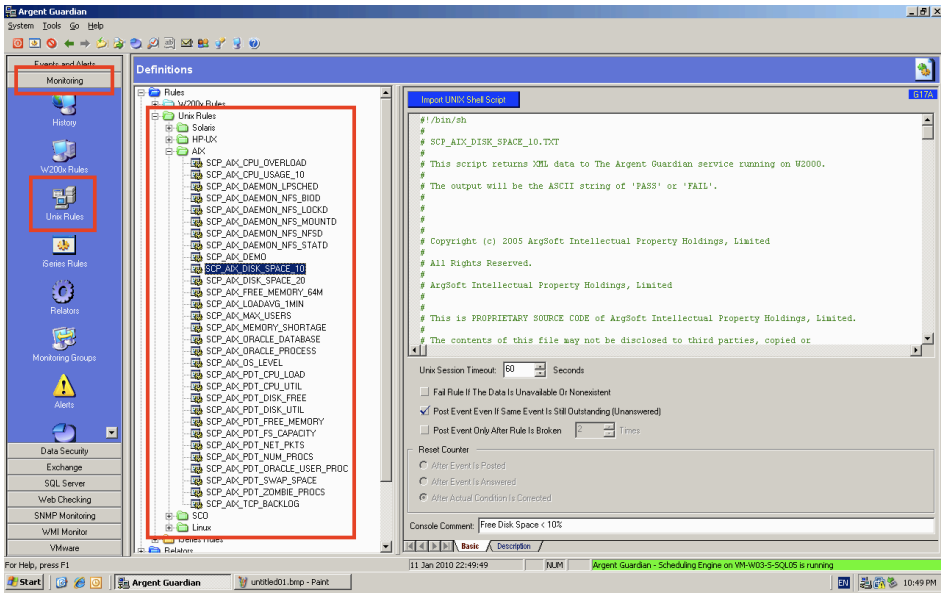
Unix Rules exist for all the popular platforms, including:

- Solaris
- HP-UX
- AIX
- SCO
- Linux

Argent Guardian - Rules

Argent Guardian comes preinstalled with many Rules for monitoring the baseline metrics (CPU utilization, memory, etc) for the various flavors of UNIX.

The existing Rules for UNIX monitoring in Argent Guardian can be accessed via **Monitoring > UNIX Rules > UNIX Flavor**



All Rules are organized via their respective flavor of UNIX, and are further divided between 'Alert' Rules and 'Predictor' Rules.

By default, 'Alert' Rules are written to trigger Alerts when a certain condition has been met – they do not save any data for reporting. This is done via the 'Predictor' Rules, and these are distinguished by the "_PDT_" prefix.

Example:
SCP_AIX_DISK_SPACE_10 is an 'Alert' Rule that triggers Alerts when Free Disk Space is less than 10%.

SCP_AIX_**PDT**_DISK_FREE is the equivalent Rule that saves the % Free Disk Space metric to Argent Predictor for reporting, graphing, trend analysis, etc.

See Also: http://help.Argent.com/#rul_ag_linux_mon

No matter which architectural options are chosen, all Rules for UNIX monitoring are written in the same way – a Bourne shell script.

This allows for customization of existing Rules, as well as writing brand new Rules or adapting existing scripts.

Argent Guardian – "Alert" Rules

Below is an example 'Alert' Rule for monitoring Disk Free Space, similar to the default SCP_LINUX_DISK_SPACE_10, but simplified for readability.

Note that the actual logic of the Rule is entirely customizable – as long as the output of the Rule conforms to the expected XML format, the script can be tailored to suit your needs.

```
#!/bin/sh
# Header information, etc....

STATUS=NOVAL

SUMMARY=NOVAL

COMMENT=NOVAL

# xmlOut() - prints an entire XML output for a command script.
# Used for The Argent Guardian rules that return a PASS/FAIL
# status and summary and comment descriptions.

xmlOut()
{
xmlBegin

xmlStatus

xmlEnd

}

# xmlBegin() - Prints out the definition of the XML format used to
# send status data to The Argent Guardian.

xmlBegin()
{
cat <<!
<?xml version="1.0"?>
<!DOCTYPE TAGResult
[
<!ELEMENT TAGResult (QEResult+)>
<!ELEMENT QEResult (status, summary, comment)>
<!ELEMENT status (#PCDATA)> <!-- (PASS | FAIL) -->
<!ELEMENT summary (#PCDATA) >
<!ELEMENT comment (#PCDATA) >
]>
<TAGResult>
!
} # End of xmlBegin()

# xmlStatus() - Send a status block to The Argent Guardian. The
# status should be 'PASS' or 'FAIL'. The summary explains the result
# and the comment is a generic description of the Unix rule.
```

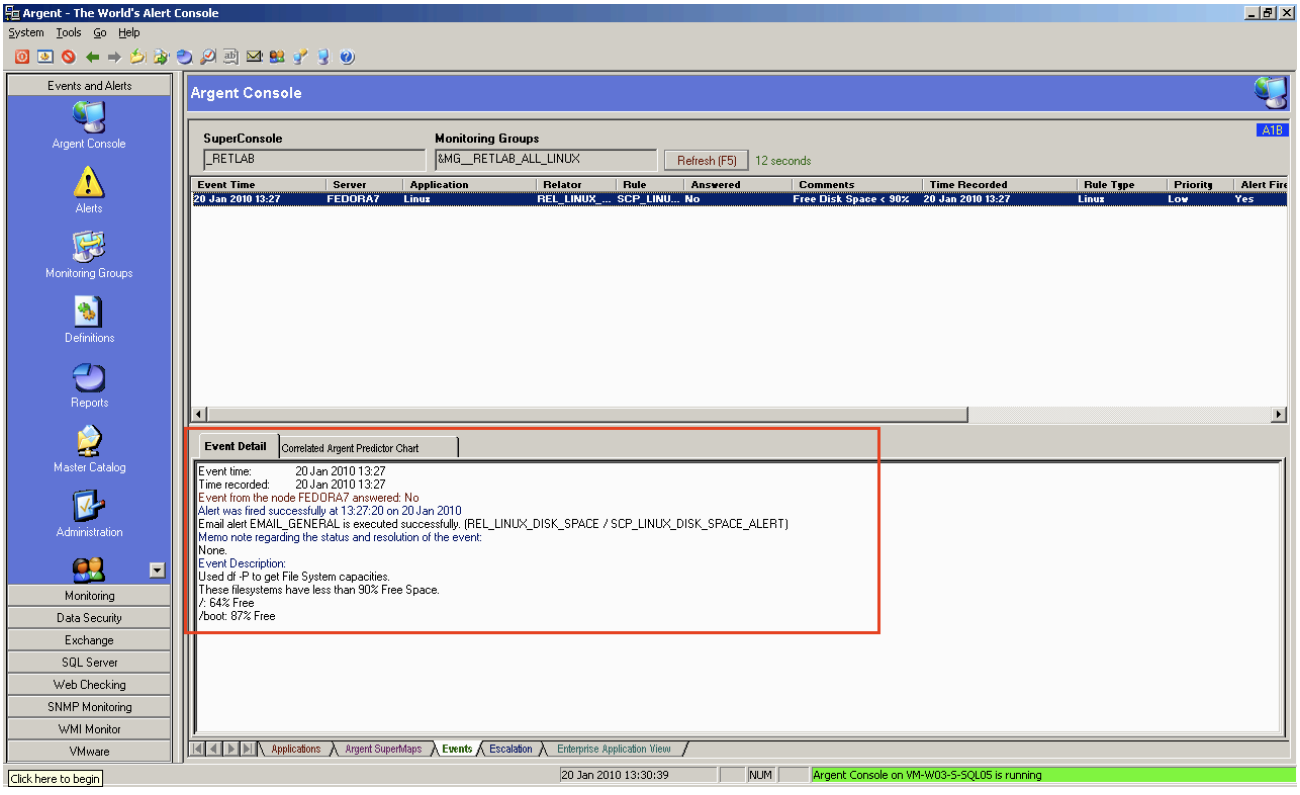
```
xmlStatus()
{
cat <<!
<QEResult>
<status>$STATUS</status>
<summary>$SUMMARY</summary>
<comment>$COMMENT</comment>
</QEResult>
!
} # End of xmlStatus()

# xmlEnd() - Completes the XML data block.

xmlEnd()
{
cat <<!
</TAGResult>
!
} # End of xmlEnd()

############################## Main portion of script ############################

# Set the IFS variable, which is used by the read command to parse input.
# We wish to read the input one line at a time.

IFS='

# Use the df command to get information about the filesystems.

DFCMD="df -P"

#
# The FREE_PCT variable defines the minimum percentage of free disk space
# for each filesystem.

FREE_PCT=90

# Assume the status of the rule is OK

STATUS=PASS

EXIT_CODE=0

SUMMARY="All filesystems have at least ${FREE_PCT}% of their space free"

COMMENT="Used ${DFCMD} to get File System capacities."

# Iterate through all the filesystems

for fs in `eval $DFCMD`; do

        # FS is the actual filesystem, MNT is the mounted directory. Include
        # an 'x' before the line in case the filesystem is blank.
```

```
  FS=`echo $fs | awk ' { print $1 } '`

  MNT=`echo x$fs | awk ' { print $6 } '`

  # Skip the first line of DFCMD, which is the column headers

  if [ "$FS" = "Filesystem" ]; then

    continue

  fi

  # Derive the free space from the capacity used and compare it to
  # the threshold.

  CAPACITY=`echo $fs | awk ' { print $5 } '`

  CAPACITY=`echo $CAPACITY | sed 's/%//'`

  FREE_SPACE=`expr 100 - $CAPACITY`

  if [ "$FREE_SPACE" -lt "$FREE_PCT" ]; then

    # Update the summary string.

    if [ $STATUS = "PASS" ]; then

      SUMMARY="These filesystems have less than ${FREE_PCT}% Free Space."

      EXIT_CODE=1

    fi

    STATUS=FAIL


    SUMMARY="${SUMMARY}
    $MNT: ${FREE_SPACE}% Free "

  fi

done

# Report the findings back to The Argent Guardian

xmlOut

exit $EXIT_CODE
```

Running this Rule provides the following output which is read by the Argent Guardian:

```
<?xml version="1.0"?>
<!DOCTYPE TAGResult
[
<!ELEMENT TAGResult (QEResult+)>
        <!ELEMENT QEResult (status, summary, comment)>
                <!ELEMENT status (#PCDATA)> <!-- (PASS | FAIL) -->
                <!ELEMENT summary (#PCDATA) >
                <!ELEMENT comment (#PCDATA) >
]>
<TAGResult>
        <QEResult>
                <status>FAIL</status>
                <summary>These filesystems have less than 90% Free Space.
                /: 64% Free
                /boot: 87% Free </summary>
                <comment>Used df -P to get File System capacities.</comment>
        </QEResult>
</TAGResult>
```

This XML block is read back to the Argent Guardian, and the following Alerts are sent via the Argent Alert Console.



Argent Guardian – "Alert" Rules, Firing Events For Individual Objects

The Argent UNIX Rules are typically configured to fire a single event for the condition as a whole - the entire Rule's script is treated as a boolean PASS or FAIL.

For example, take the bundled SCP_LINUX_DISK_SPACE_60. This Rule is broken if any file system tested by the Rule's script has less than 60% free space.

But only a single event is fired for all the file systems that have less than 60% free space. As a result, the condition won't be corrected unless all the file systems have more than 60% free space.

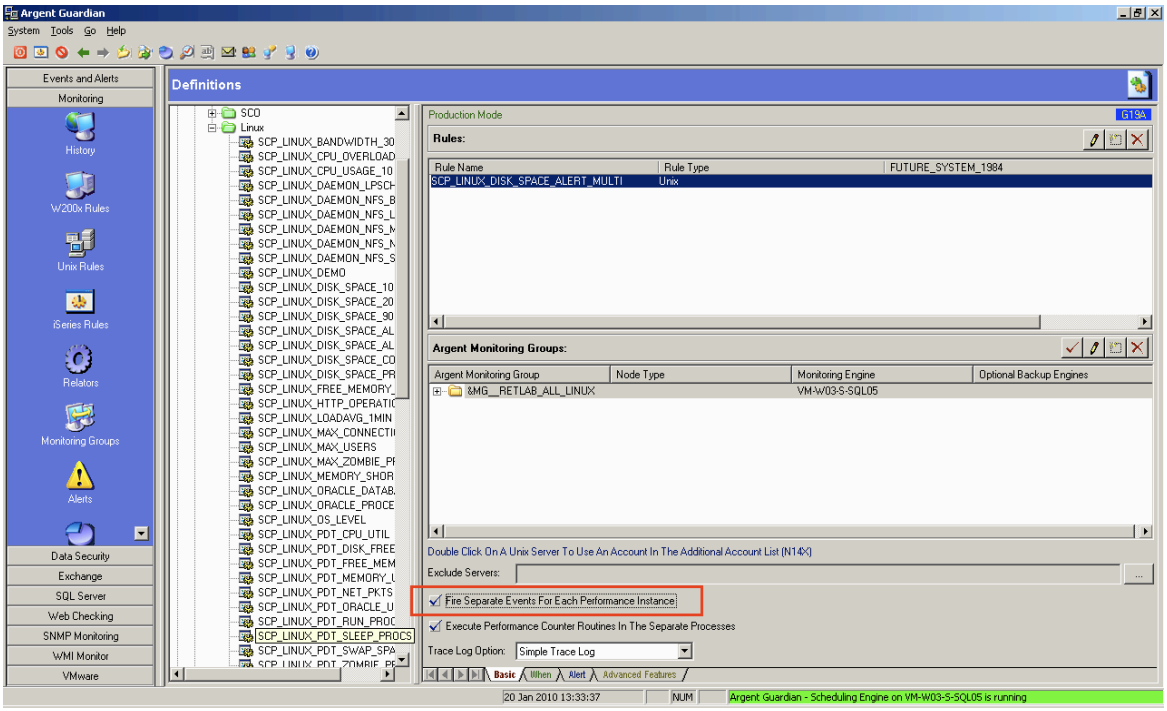
The internal mechanism to identify an event as the same 'XT event' is to compare following fields:

- Node name
- Relator name
- Rule name
- Comparison string

In order to identify individual object, a new tag 'COMPARE' is added. In previous file system sample Rule, the file system name should be used for the COMPARE value.

The logic of the Rule is also changed to output multiple 'QERESULT' tags – one for each broken file system.

Also, the option 'Fire Separate Events For Each Performance Instance' should be checked.



See Also: http://help.Argent.com/#Q716

```
#!/bin/sh

# Header information, etc....

STATUS=NOVAL

SUMMARY=NOVAL

COMMENT=NOVAL

COMPARE=NOVAL

# xmlBegin() - Prints out the definition of the XML format used to
# send status data to The Argent Guardian.

xmlBegin()
{
  cat <<!
<?xml version="1.0"?>
<!DOCTYPE TAGResult
[
<!ELEMENT TAGResult (QEResult+)>
  <!ELEMENT QEResult (status, summary, comment, compare)>
  <!ELEMENT status (#PCDATA)> <!-- (PASS | FAIL) -->
  <!ELEMENT summary (#PCDATA) >
  <!ELEMENT comment (#PCDATA) >
  <!ELEMENT compare (#PCDATA) >
]>
<TAGResult>
!
} # End of xmlBegin(
```

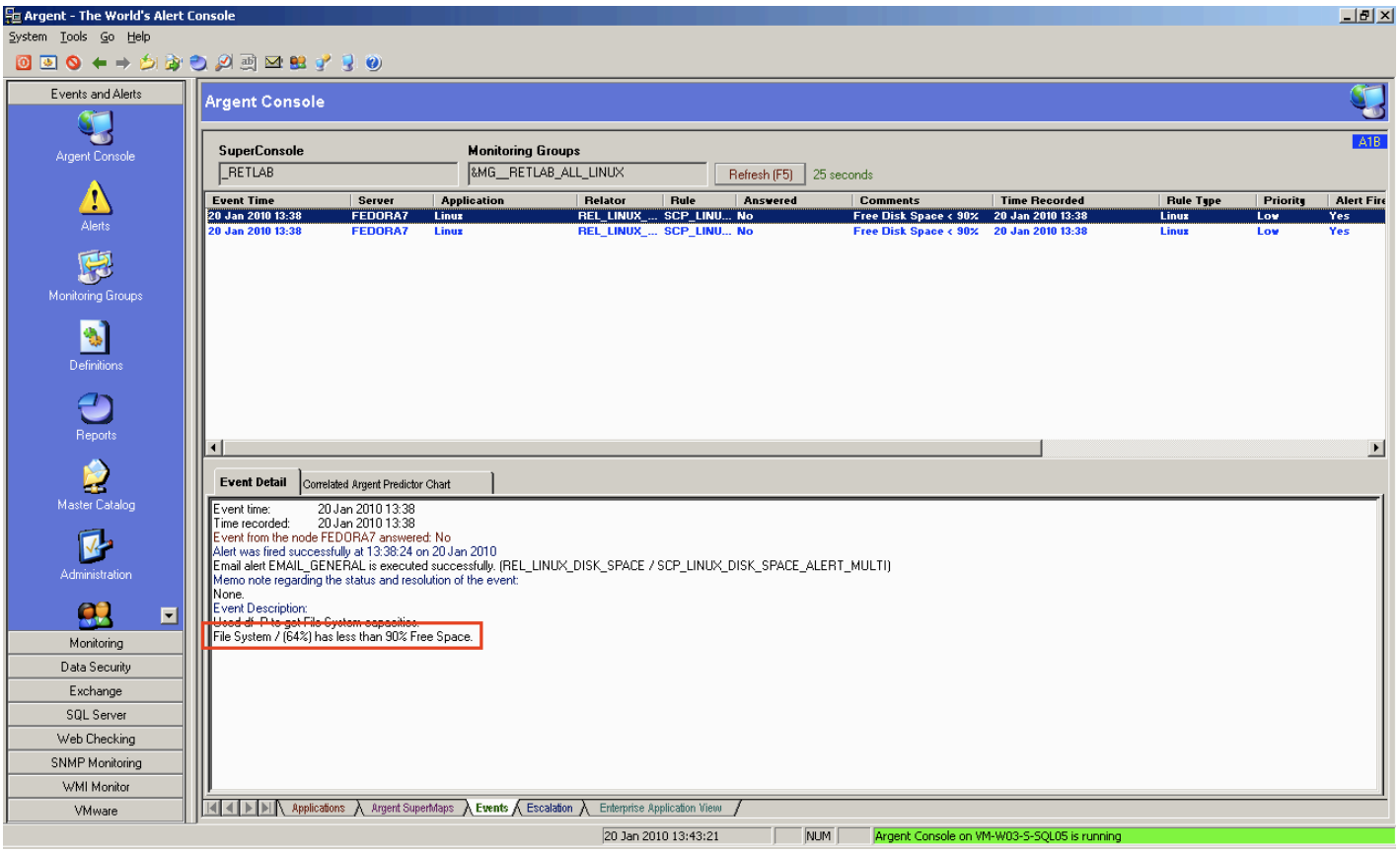
```
# xmlStatus() - Send a status block to The Argent Guardian. The
# status should be 'PASS' or 'FAIL'. The summary explains the result
# and the comment is a generic description of the Unix rule.

xmlStatus()
{
    cat <<!
    <QEResult>
    <status>$STATUS</status>
    <summary>$SUMMARY</summary>
    <comment>$COMMENT</comment>
    <compare>$COMPARE</compare>
  </QEResult>
!
} # End of xmlStatus()

# xmlEnd() - Completes the XML data block.

xmlEnd()
{
    cat <<!
 </TAGResult>
!
} # End of xmlEnd()

############################## Main portion of script ############################

# Set the IFS variable, which is used by the read command to parse input.
# We wish to read the input one line at a time.

IFS='
'
# Use the df command to get information about the filesystems.

DFCMD="df -P"

#
# The FREE_PCT variable defines the minimum percentage of free disk space
# for each filesystem.

FREE_PCT=90

# Assume the status of the rule is OK

STATUS=PASS

EXIT_CODE=0

SUMMARY="All filesystems have at least ${FREE_PCT}% of their space free"

COMMENT="Used ${DFCMD} to get File System capacities."

xmlBegin

# Iterate through all the filesystems

for fs in `eval $DFCMD`; do
```

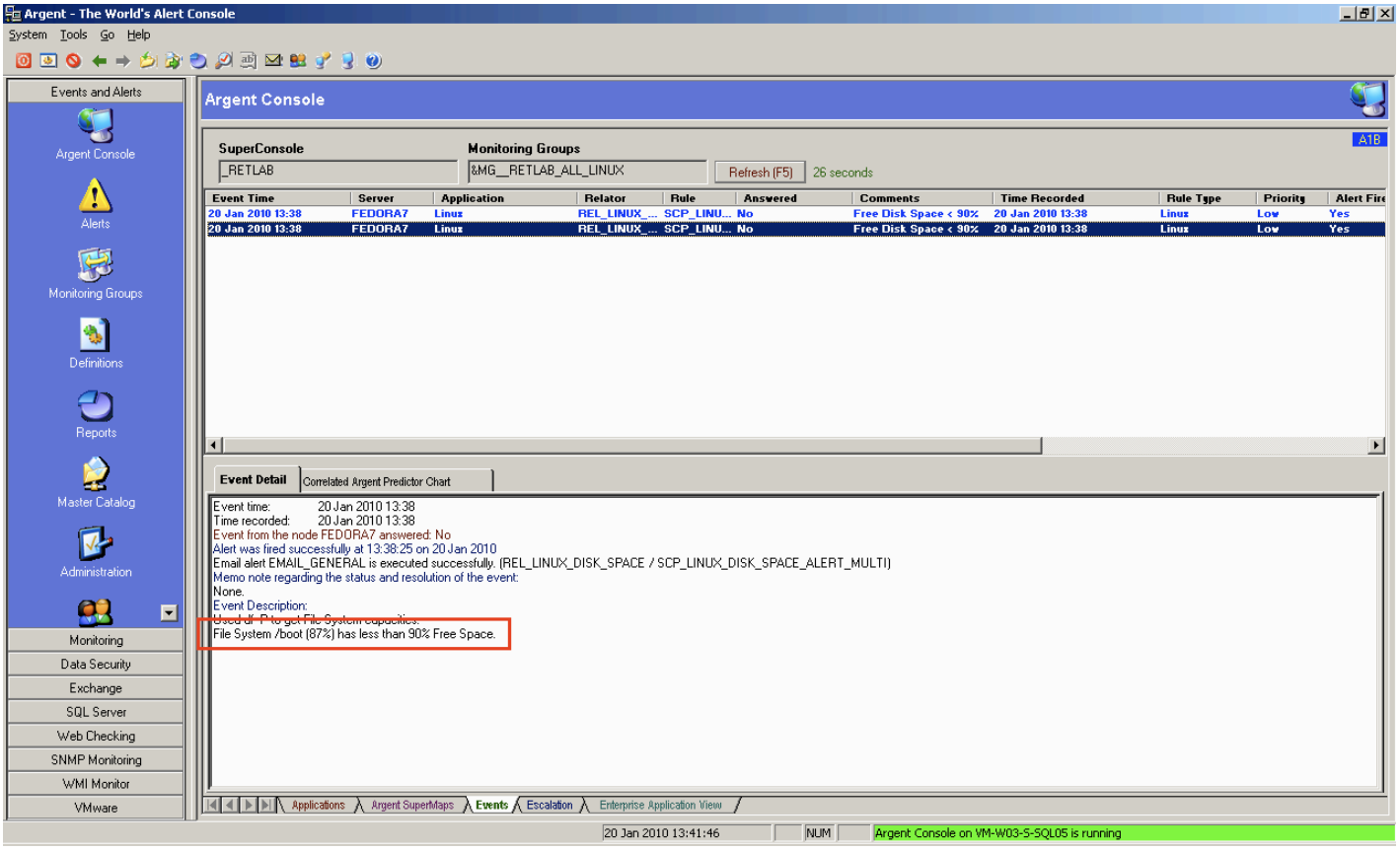
```
# FS is the actual filesystem, MNT is the mounted directory. Include
# an 'x' before the line in case the filesystem is blank.

FS=`echo $fs | awk ' { print $1 } '`

MNT=`echo x$fs | awk ' { print $6 } '`

# Skip the first line of DFCMD, which is the column headers

if [ "$FS" = "Filesystem" ]; then

continue

fi

# Derive the free space from the capacity used and compare it to
# the threshold.

    CAPACITY=`echo $fs | awk ' { print $5 } '`

    CAPACITY=`echo $CAPACITY | sed 's/%//'`

    FREE_SPACE=`expr 100 - $CAPACITY`

    COMPARE=$MNT

 if [ "$FREE_SPACE" -lt "$FREE_PCT" ]; then

    # Update the summary string.

    SUMMARY="File System $MNT (${FREE_SPACE}%) has less than ${FREE_PCT}% Free Space."

    STATUS=FAIL

    EXIT_CODE=1

    xmlStatus
 fi

done

if [ $STATUS = "PASS" ]; then

xmlStatus

fi

# Report the findings back to The Argent Guardian

xmlEnd

exit $EXIT_CODE
```

Notice the change in the output format, there are now multiple 'QERESULT' tags – one for each broken filesystem.

```
<?xml version="1.0"?>
<!DOCTYPE TAGResult
[
<!ELEMENT TAGResult (QEResult+)>
        <!ELEMENT QEResult (status, summary, comment, compare)>
                <!ELEMENT status (#PCDATA)> <!-- (PASS | FAIL) -->
                <!ELEMENT summary (#PCDATA) >
                <!ELEMENT comment (#PCDATA) >
                <!ELEMENT compare (#PCDATA) >
]>
<TAGResult>
    <QEResult>
        <status>FAIL</status>
        <summary>File System / (64%) has less than 90% Free Space.</summary>
        <comment>Used df -P to get File System capacities.</comment>
        <compare>/</compare>
    </QEResult>
    <QEResult>
        <status>FAIL</status>
        <summary>File System /boot (87%) has less than 90% Free Space.</summary>
        <comment>Used df -P to get File System capacities.</comment>
        <compare>/boot</compare>
    </QEResult>
</TAGResult>
```

This corresponds with individual Alerts being raised in the Argent Alert Console for each individual filesystem.





Argent Guardian – "Predictor" Rules

Below is an example 'Predictor' Rule for monitoring Disk Free Space, similar to the default SCP_LINUX_PDT_DISK_FREE, but simplified for readability.

This Rule uses similar in logic to the original Disk Space Alert Rule, but is used only for collecting performance data.

```
#!/bin/sh
# Header information, etc....

OBJECT=NOVAL

COUNTER=NOVAL

INSTANCE=NOVAL

METRIC=NOVAL

# xmlBegin() - Prints out the definition of the XML format used to
# send status data to The Argent Guardian.

xmlBegin()
{
cat <<!
<?xml version="1.0"?>
<!DOCTYPE PDTResult
[
<!ELEMENT PDTResult (QEResult+)>
<!ELEMENT QEResult (status, performance+)>
<!ELEMENT status (#PCDATA)> <!-- (PERFORMANCE) -->
<!ELEMENT performance (object, counter, instance, metric)>
<!ELEMENT object (#PCDATA) >
<!ELEMENT counter (#PCDATA) >
<!ELEMENT instance (#PCDATA) >
<!ELEMENT metric (#PCDATA) >
]>
<PDTResult>
!
} # End of xmlBegin()

# xmlPerformance() - Send a performance block to The Argent Predictor. The
# actual value of the metric is stored in <metric>. The other fields
# are used to describe the metric. Multiple xmlPerformance() calls may be
# made between the xmlBegin() and xmlEnd() functions.
```

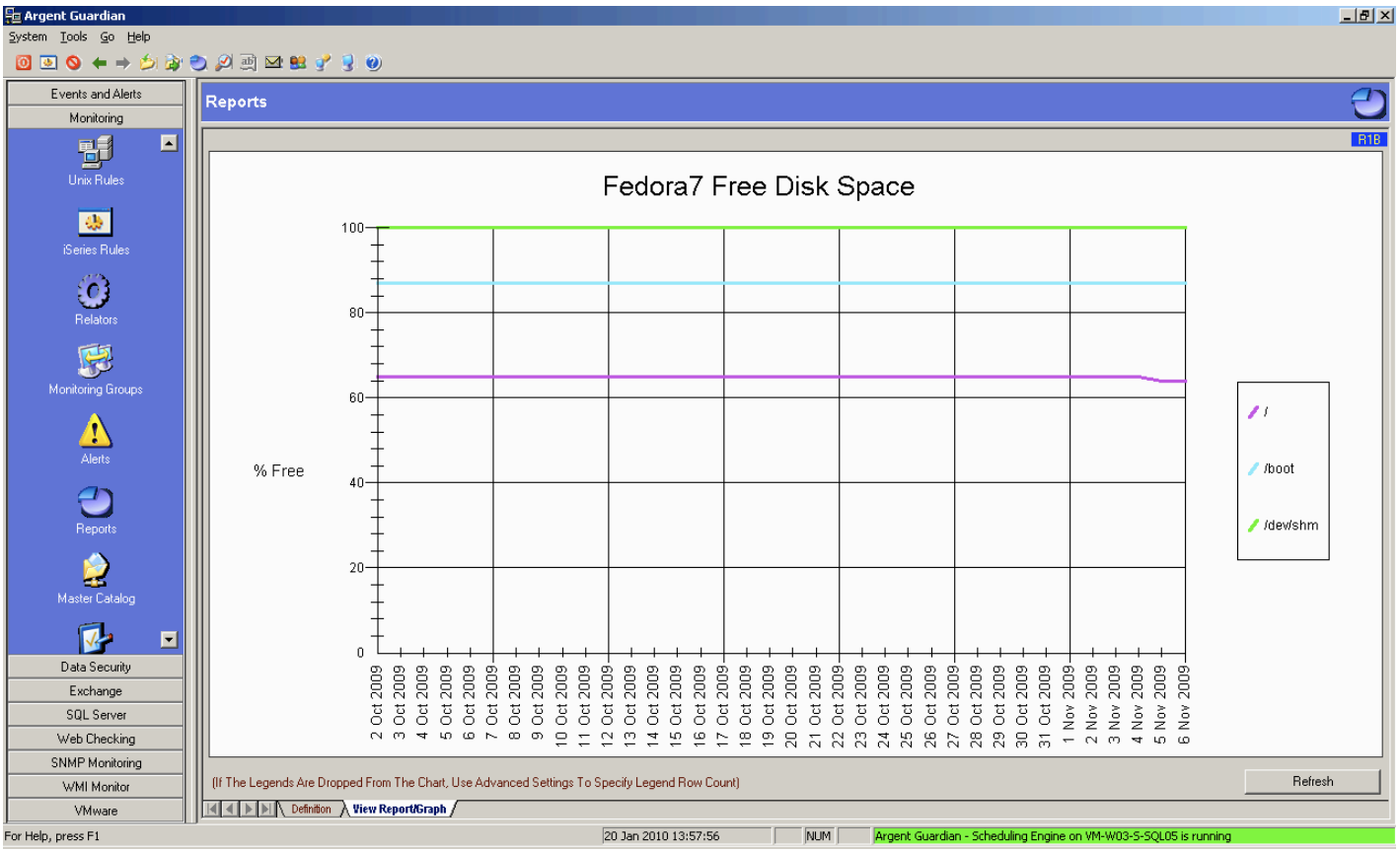
```
xmlPerformance()
{
  cat <<!
  <QEResult>
    <status>performance</status>
    <performance>
      <object>$OBJECT</object>
      <counter>$COUNTER</counter>
      <instance>$INSTANCE</instance>
       <metric>$METRIC</metric>
      </performance>
  </QEResult>
!
} # End of xmlPerformance()

# xmlEnd() - Completes the XML data block.

xmlEnd()
{
   cat <<!
</PDTResult>
!
} # End of xmlEnd()
############################## Main portion of script ############################

# Set the IFS variable, which is used by the read command to parse input.
# We wish to read the input one line at a time.

IFS='
'
OSNAME=`uname`

EXIT_CODE=0

# Define any directories you wish to ignore here. Directory variables
# should start at '1' and increment continuously. The following are
# three example variables (that would need to be uncommented).

# Begin output to The Argent Predictor by sending the XML header.

OBJECT="$OSNAME Filesystem"

COUNTER="Pct Free Disk Space"

xmlBegin

# Use the df commmand to get the free space for each filesystem.

DF_CMD="df -P"

for fs in `eval $DF_CMD`; do
```

```
    # FS is the actual filesystem, MNT is the mounted directory. Include
    # an 'x' before the line in case the filesystem is blank.

    FS=`echo $fs | awk ' { print $1 } '`

    MNT=`echo x$fs | awk ' { print $6 } '`

    # Skip the first line of the df command output, which is the column headers

    if [ "$FS" = "Filesystem" ]; then

        continue

    fi

    # Get the capacity (disk space used) and the free space

    CAPACITY=`echo $fs | awk ' { print $5 } '`

    CAPACITY=`echo $CAPACITY | sed 's/%//'`

    FREE_SPACE=`expr 100 - $CAPACITY`

    # Send performance data to The Argent Predictor

    INSTANCE=$MNT

    METRIC=$FREE_SPACE

    xmlPerformance

done

# Send the XML footer to The Argent Predictor

xmlEnd

exit $EXIT_CODE
```

Notice the change in output, the 'PDTRESULT' tag is used to identify performance data, and each 'QERESULT' tag corresponds to the individual performance instance:

```
<?xml version="1.0"?>
<!DOCTYPE PDTResult
[
<!ELEMENT PDTResult (QEResult+)>
        <!ELEMENT QEResult (status, performance+)>
                <!ELEMENT status (#PCDATA)> <!-- (PERFORMANCE) -->
                <!ELEMENT performance (object, counter, instance, metric)>
                        <!ELEMENT object (#PCDATA) >
                        <!ELEMENT counter (#PCDATA) >
                        <!ELEMENT instance (#PCDATA) >
                <!ELEMENT metric (#PCDATA) >
]>
<PDTResult>
        <QEResult>
                <status>performance</status>
                <performance>
                        <object>Linux Filesystem</object>
                        <counter>Pct Free Disk Space</counter>
                        <instance>/</instance>
                        <metric>64</metric>
                </performance>
        </QEResult>
        <QEResult>
                <status>performance</status>
                <performance>
                        <object>Linux Filesystem</object>
                        <counter>Pct Free Disk Space</counter>
                        <instance>/boot</instance>
                        <metric>87</metric>
                </performance>
        </QEResult>
        <QEResult>
                <status>performance</status>
                <performance>
                        <object>Linux Filesystem</object>
                        <counter>Pct Free Disk Space</counter>
                        <instance>/dev/shm</instance>
                        <metric>100</metric>
                </performance>
        </QEResult>
</PDTResult>
```

This performance data is then stored in the Argent Predictor, and can be used to create Graphs, Reports,provide historical data for trend analysis, etc.



Argent Guardian – Combining "Alert" And "Predictor" Rules

By default, Argent Guardian UNIX Rules are split between 'Alert' Rules and 'Predictor' Rules, but there is no reason that they can't be combined into a single Rule.

With some clever manipulation, the two types of Rules can be combined to send Alerts and save Predictor data.

The example below shows a combined Rule, with options for: sending Alerts; sending Alerts for individual items; and saving Predictor data.

```
#!/bin/sh
# Header information, etc....

# The FREE_PCT variable defines the minimum percentage of free disk space
# for each filesystem.

FREE_PCT=10

# Send Alerts (Y/N)?

ALERT="Y"

# Send individual alerts per instance (Y/N)?

MULTI="Y"

# Save Predictor data (Y/N)?

PREDICTOR="Y"

# Required Argent variables

STATUS=NOVAL

SUMMARY=NOVAL

COMMENT=NOVAL

OBJECT=NOVAL

COUNTER=NOVAL

INSTANCE=NOVAL

METRIC=NOVAL

COMPARE=NOVAL

XML_PERFORMANCE=NOVAL

XML_STATUS=NOVAL
```

```
# xmlOut() - Prints out the definition of the XML format used to
# send status data to The Argent Guardian.

xmlOut()
{
   cat <<!
<?xml version="1.0"?>
<!DOCTYPE CombinedResult
[
<!ELEMENT CombinedResult (PDTResult?, TAGResult?)>
   <!ELEMENT PDTResult (QEResult+)>
         <!ELEMENT QEResult (status, summary?, comment?, compare?, performance?)>
         <!ELEMENT status (#PCDATA) > <!-- (PERFORMANCE | PASS | FAIL) -->
         <!ELEMENT summary (#PCDATA) >
         <!ELEMENT comment (#PCDATA) >
         <!ELEMENT compare (#PCDATA) >
         <!ELEMENT performance (object, counter, instance, metric)>
               <!ELEMENT object (#PCDATA) >
               <!ELEMENT counter (#PCDATA) >
               <!ELEMENT instance (#PCDATA) >
               <!ELEMENT metric (#PCDATA) >
<!ELEMENT TAGResult (QEResult+)>
]>
<CombinedResult>
!

   if [ "$PREDICTOR" = "Y" ]; then

         cat <<!
      <PDTResult>$XML_PERFORMANCE
      </PDTResult>
!

   fi

   if [ "$ALERT" = "Y" ]; then

      cat <<!
   <TAGResult>$XML_STATUS
</TAGResult>
   !

      fi

   cat <<!
</CombinedResult>
!
} # End of xmlOut()

# xmlPerformance() - Send a performance block to The Argent Predictor. The
# actual value of the metric is stored in <metric>. The other fields
# are used to describe the metric. Multiple xmlPerformance() calls may be
# made between the xmlBegin() and xmlEnd() functions.
```

```
xmlPerformance()
{
   XML_PERFORMANCE="${XML_PERFORMANCE}
      <QEResult>
      <status>performance</status>
          <performance>
          <object>$OBJECT</object>
          <counter>$COUNTER</counter>
          <instance>$INSTANCE</instance>
          <metric>$METRIC</metric>
      </performance>
   </QEResult>"
} # End of xmlPerformance()

# xmlStatus() - Send a status block to The Argent Guardian. The
# status should be 'PASS' or 'FAIL'. The summary explains the result
# and the comment is a generic description of the Unix rule.

xmlStatus()
{
   XML_STATUS="${XML_STATUS}
      <QEResult>
          <status>$STATUS</status>
          <summary>$SUMMARY</summary>
          <comment>$COMMENT</comment>
          <compare>$COMPARE</compare>
      </QEResult>"
} # End of xmlStatus()

############################## Main portion of script ############################

# Set the IFS variable, which is used by the read command to parse input.
# We wish to read the input one line at a time.

IFS='
'
# Use the df command to get information about the filesystems.

DFCMD="df -P"

# Assume the status of the rule is OK

OBJECT="UNIX Filesystem"

COUNTER="Pct Free Disk Space"

STATUS=PASS

EXIT_CODE=0

SUMMARY="All filesystems have at least ${FREE_PCT}% of their space free"

COMMENT="Used ${DFCMD} to get File System capacities."
```

```
COMPARE=""

XML_PERFORMANCE=""

XML_STATUS=""

# Iterate through all the filesystems

for fs in `eval $DFCMD`; do

    # FS is the actual filesystem, MNT is the mounted directory. Include
    # an 'x' before the line in case the filesystem is blank.

    FS=`echo $fs | awk ' { print $1 } '`

    MNT=`echo x$fs | awk ' { print $6 } '`

    # Skip the first line of DFCMD, which is the column headers

    if [ "$FS" = "Filesystem" ]; then

            continue

    fi

    # Derive the free space from the capacity used and compare it to
    # the threshold.

    CAPACITY=`echo $fs | awk ' { print $5 } '`

    CAPACITY=`echo $CAPACITY | sed 's/%//'`

    FREE_SPACE=`expr 100 - $CAPACITY`

    if [ "$PREDICTOR" ]; then

      INSTANCE=$MNT

      METRIC=$FREE_SPACE

       xmlPerformance

    fi

    if [ "$ALERT" = "Y" ]; then

       if [ "$FREE_SPACE" -lt "$FREE_PCT" ]; then

          if [ "$MULTI" = "Y" ]; then

             # Update the summary string.

             SUMMARY="File System $MNT (${FREE_SPACE}%) has less than ${FREE_PCT}% Free Space."
```

```
                COMPARE=$MNT
                STATUS=FAIL
                EXIT_CODE=1
                xmlStatus
            fi
            if [ "$MULTI" = "N" ]; then
                if [ $STATUS = "PASS" ]; then
                    SUMMARY="These filesystems have less than ${FREE_PCT}% Free Space."
                    EXIT_CODE=1
                fi
                STATUS=FAIL
                SUMMARY="${SUMMARY}
                $MNT: ${FREE_SPACE}% Free "
            fi
        fi
    fi
done
if [ $STATUS = "PASS" -o $MULTI = "N" ]; then
xmlStatus
fi
    # Report the findings back to The Argent Guardian
xmlOut
exit $EXIT_CODE
```

Notice the output of the combined Rule. The XML DTD has been updated to allow both 'PDTRESULT' and 'TAGRESULT' tags, to save Predictor data and send Alerts, respectively.

```
<?xml version="1.0"?>
<!DOCTYPE CombinedResult
[
<!ELEMENT CombinedResult (PDTResult?, TAGResult?)>
        <!ELEMENT PDTResult (QEResult+)>
                <!ELEMENT QEResult (status, summary?, comment?, compare?, performance?)>
                        <!ELEMENT status (#PCDATA)> <!-- (PERFORMANCE | PASS | FAIL) -->
                        <!ELEMENT summary (#PCDATA) >
                        <!ELEMENT comment (#PCDATA) >
                        <!ELEMENT compare (#PCDATA) >
                        <!ELEMENT performance (object, counter, instance, metric)>
                                <!ELEMENT object (#PCDATA) >
                                <!ELEMENT counter (#PCDATA) >
                                <!ELEMENT instance (#PCDATA) >
                                <!ELEMENT metric (#PCDATA) >
        <!ELEMENT TAGResult (QEResult+)>
]>
<CombinedResult>
        <PDTResult>
                <QEResult>
                        <status>performance</status>
                        <performance>
                                <object>UNIX Filesystem</object>
                                <counter>Pct Free Disk Space</counter>
                                <instance>/</instance>
                                <metric>64</metric>
                        </performance>
                </QEResult>
                <QEResult>
                        <status>performance</status>
                        <performance>
                                <object>UNIX Filesystem</object>
                                <counter>Pct Free Disk Space</counter>
                                <instance>/boot</instance>
                                <metric>87</metric>
                        </performance>
                </QEResult>
                <QEResult>
                        <status>performance</status>
                        <performance>
                                <object>UNIX Filesystem</object>
                                <counter>Pct Free Disk Space</counter>
                                <instance>/dev/shm</instance>
                                <metric>100</metric>
                        </performance>
                </QEResult>
</PDTResult>
<TAGResult>
```

```
                <QEResult>
                        <status>FAIL</status>
                        <summary>File System / (64%) has less than 90% Free Space.</summary>
                        <comment>Used df -P to get File System capacities.</comment>
                        <compare>/</compare>
                </QEResult>
                <QEResult>
                        <status>FAIL</status>
                        <summary>File System /boot (87%) has less than 90% Free Space.</summary>
                        <comment>Used df -P to get File System capacities.</comment>
                        <compare>/boot</compare>
                </QEResult>
        </TAGResult>
</CombinedResult>
```

Argent Data Consolidator

The Argent Data Consolidator provides you with all the archiving and compliance facilities you need, in a single product, regardless of platform - Windows, UNIX, Linux, IBM mainframe, Cisco device or any network device.

Argent can even consolidate logs and files from air conditioning units and UPS power supplies - if the hardware has a log or file Argent can consolidate it.

Argent consolidates your logs and files to one or more central ODBC databases for reporting or analysis. During consolidation you have the option of analyzing each record, and be alerted when anomalies are detected.

Because Argent supports any ODBC backend, you might use Argent to do this:

- Windows Event logs from all 200 Windows servers to SQL Server backend in Stuttgart.
- PST email files from all 20,000 users to an Oracle backend in London.
- Unix SYSLOGs from 350 Solaris machines to a MySQL backend in Palo Alto.
- All UPS and AirCon hardware exception logs and files to a SQL Server backend in Chicago.
- All Cisco logs to a MySQL backend in Sydney.

As you can see the potential with Argent is limitless. Setup in hours not weeks.

Filters

Filters allow you to optionally limit the records to be consolidated - you can consolidate all the records from a Data Source, or you can selectively consolidate.

Alerts

In addition to consolidation and filtering, Argent Data Consolidator Rules can be setup to send Alerts when specific conditions are found in the logs that are being scanned.

For more in depth information, See Also:
http://help.Argent.com/#rul_adc

ASCII Logs

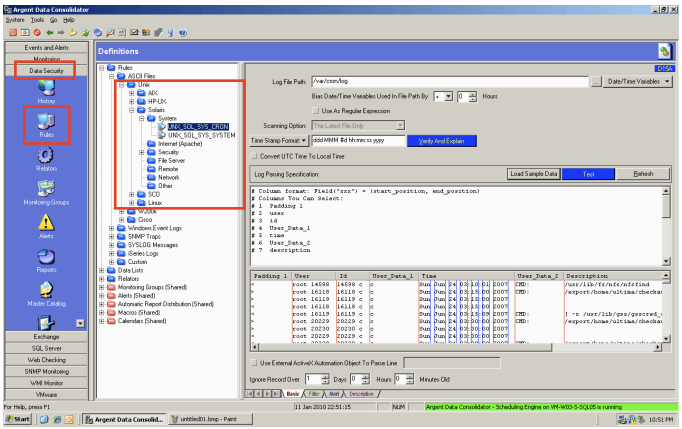
Argent Data Consolidator comes preinstalled with many Rules for parsing common UNIX log files, eg: Cron log, boot log, etc.

The ASCII File Rules let you parse and test any ASCII log file for any computing platform - Solaris, HP-UX, AIX, W200x, Cisco.

The existing Rules for UNIX monitoring in Argent Data Consolidator can be accessed via Data Security > **ASCII Files > UNIX > UNIX Flavor**

As ASCII log files are all different and have different tokens or keywords in different locations in the record, you can create different Rules to parse the data.

Let's look at the sample ASCII File Rule:



Here -- on a single Argent screen - all the criteria are specified:

- The format of the Time Stamp
- The order of the fields
- The offset of the fields

For a detailed description of the Log Parsing Specification, See Also:
http://help.Argent.com/#adc_log_spec

Syslog

SYSLOG Message rules are used to consolidate SYSLOG events.

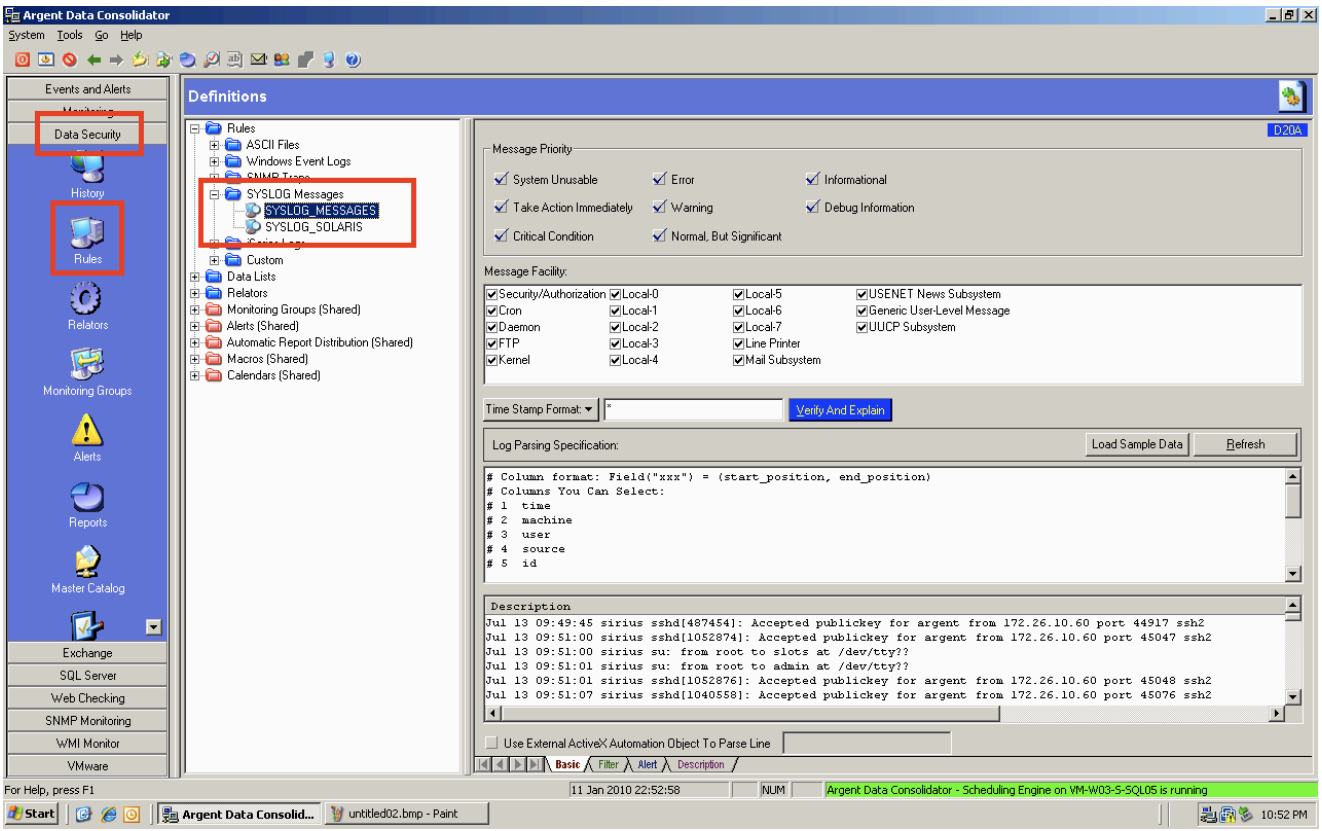
SYSLOG is an event logging protocol (IETF standard http://www.ietf.org/html.charters/syslog-charter.html) running over the network.

Citation from IETF RFC3164:

syslog uses the user datagram protocol (UDP) as its underlying transport layer mechanism. The UDP port assigned to syslog is 514.

Argent acts as a SYSLOG server by listening for the incoming SYSLOG messages on UDP 514, and consolidates them into the central databases you've defined.

The SYSLOG Rule below will consolidate all events according to the selections in the Message Priority section and the Message Facility section.



Once you've defined the Basic tab you can optional add Rule Filters and Rule Alerts.

For further information on setting up SYSLOG Rules, See Also:
http://help.Argent.com/#Q061

Appendix A: UNIXSSH.INI

Argent Guardian includes the ability to do secure agent-less UNIX monitoring using the Secure Shell protocol.

This is achieved from the Windows machine using the PuTTy tools, PSCP and PLINK, using the following techniques:

Argent XT 8.0A-0810 or below

1. Copy script to UNIX machine /tmp directory.
2. Change script permissions to make it executable.
3. Execute the script, reading the output from STDOUT.
4. Remove the file from the UNIX machine /tmp directory.

Argent XT 8.0A-0901-B or above

1. SSH into UNIX machine.
2. From command line, type in the whole script ended with CTRL-D.
3. Read the output of script from STDOUT.

Prior to Argent XT 8.0A-0901-B, the actual commands for copying, running and removing the script could be manually specified via the UNIXSSH.INI file, located in each Argent product directory.

eg: C:\ ARGENT\ArgentManagementConsole\ArgentGuardian\ UNIXSSH.INI

This allowed customers to modify the commands for custom operations, such as authorization using private keys, optimizing the commands into a single operation, etc.

An added benefit of the older method is the ability to create Rules in other languages, eg: Perl.

Unfortunately, as of Argent XT 8.0A-0901-B, this UNIXSSH.INI file is no longer read as the techniques for connecting to UNIX machines have been changed.

Luckily, customers can choose to revert back to the “old”, but slower execution method by changing the registry entry:

HKLM\SOFTWARE\Argent\ArgentGuardian\MonitoringEngine\SSH_ NO_COPY_SCRIPT

Set the value to 0 (ZERO) to revert to using UNIXSSH.INI

This needs to be done on every single Monitoring Engine where the old method wants to be used.

Customizing UNIXSSH.INI

By default, SSH monitoring is a four-stage operation, which is fine for most implementations. However if you are monitoring several hundred UNIX\Linux servers, or simply want to reduce the number of required SSH connections from the Argent server, you will need to update the ‘UNIXSSH.INI’ file.

Contents of default ‘UNIXSSH.INI’:

0 {SSH_BIN}PSCP -P {PORT} -pw {PASSWORD} {SOURCE_ SCRIPT} {USER}@{HOST}:{TARGET_SCRIPT}

0 {SSH_BIN}PLINK -P {PORT} -pw {PASSWORD} {USER}@ {HOST} “chmod +x {TARGET_SCRIPT}”

1 {SSH_BIN}PLINK -P {PORT} -pw {PASSWORD} {USER}@ {HOST} {TARGET_SCRIPT_WITH_PARAM}

0 {SSH_BIN}PLINK -P {PORT} -pw {PASSWORD} {USER}@ {HOST} “rm -f {TARGET_SCRIPT}”

Simply replace the existing file with the optimized file linked below. The optimized file reduces the number of required connections by combining lines two, three, and four into a single execution of PLINK.

Contents of optimized ‘UNIXSSH.INI’:

0 {SSH_BIN}PSCP -P {PORT} -pw {PASSWORD} {SOURCE_ SCRIPT} {USER}@{HOST}:{TARGET_SCRIPT}

1 {SSH_BIN}PLINK -P {PORT} -pw {PASSWORD} {USER}@ {HOST} “chmod +x {TARGET_SCRIPT};{TARGET_SCRIPT_WITH_PARAM};rm -f {TARGET_SCRIPT}”;

UNIXSSH.INI can be further customized, for example, to use private keys for authorization instead of username/passwords:

0 {SSH_BIN}PSCP -P {PORT} –i C:\keys\private.ppk SOURCE_ SCRIPT} {USER}@{HOST}:{TARGET_SCRIPT}

1 {SSH_BIN}PLINK -P {PORT} –i C:\keys\private.ppk {USER}@ {HOST} “chmod +x {TARGET_SCRIPT};{TARGET_SCRIPT_WITH_PARAM};rm -f {TARGET_SCRIPT}”;