Exploring servmon and itmquery functionality in NetView 7.1.4

Jane Curry, Skills 1st Limited
jane.curry@skills-1st.co.uk

Abstract

This paper examines some of the new functionality shipped with IBM Tivoli NetView 7.1.4. servmon provides a way to monitor processes and applications using the NetView network manager. itmquery offers an integration point between NetView and the IBM Tivoli Monitoring suite of products .

Introduction

IBM Tivoli NetView 7.1.4 was released in October 2003. It provided a number of enhancements, including:

- Closer integration with the Tivoli Enterprise Console 3.9 for more effective root-cause problem analysis
- Better integration with Tivoli Enterprise Data Warehouse to include support for Simple Network Management (SNMP) performance data
- A new daemon, **servmon**, to monitor processes and applications
- A new utility, **itmquery**, to provide integration between the systems management capabilities of IBM Tivoli Monitoring 5.1.1 (ITM), and the network management capabilities of NetView

This paper examines the latter two utilities in detail.

In general, the comments made in this paper apply to all versions of NetView 7.1.4 Unix platforms, and to NetView 7.1.4 on Windows. Where exceptions apply, this will be made clear.

Background to servmon development

The NetView product has had the **nvsniffer** utility for many years. nvsniffer is a command utility that tests whether a system has an application running on a particular TCP port. It is customized with a configuration file, /usr/OV/conf/nvsniffer.conf by default, which specifies what port to test for, on what systems. nvsniffer works by creating a new boolean NetView object field to represent whether a system supports the particular port, or not. In addition, a NetView SmartSet can automatically be created, based on whether the port is supported. For example:

```
isTelnet|23|TelnetServers|Telnet Server|||*
```

in /usr/OV/conf/nvsniffer.conf directs the nvsniffer command to test all nodes in the NetView object database (the final “*” in the line above), to discover a service on port TCP 23 (the second field in the line above). If nvsniffer can connect to this port, create a new field in the NetView object database called isTelnet (the first field), and create a SmartSet called

TelnetServers (third field), with a SmartSet label of "Telnet Server" (fourth field). Fields are separated with the pipe (vertical bar) symbol.

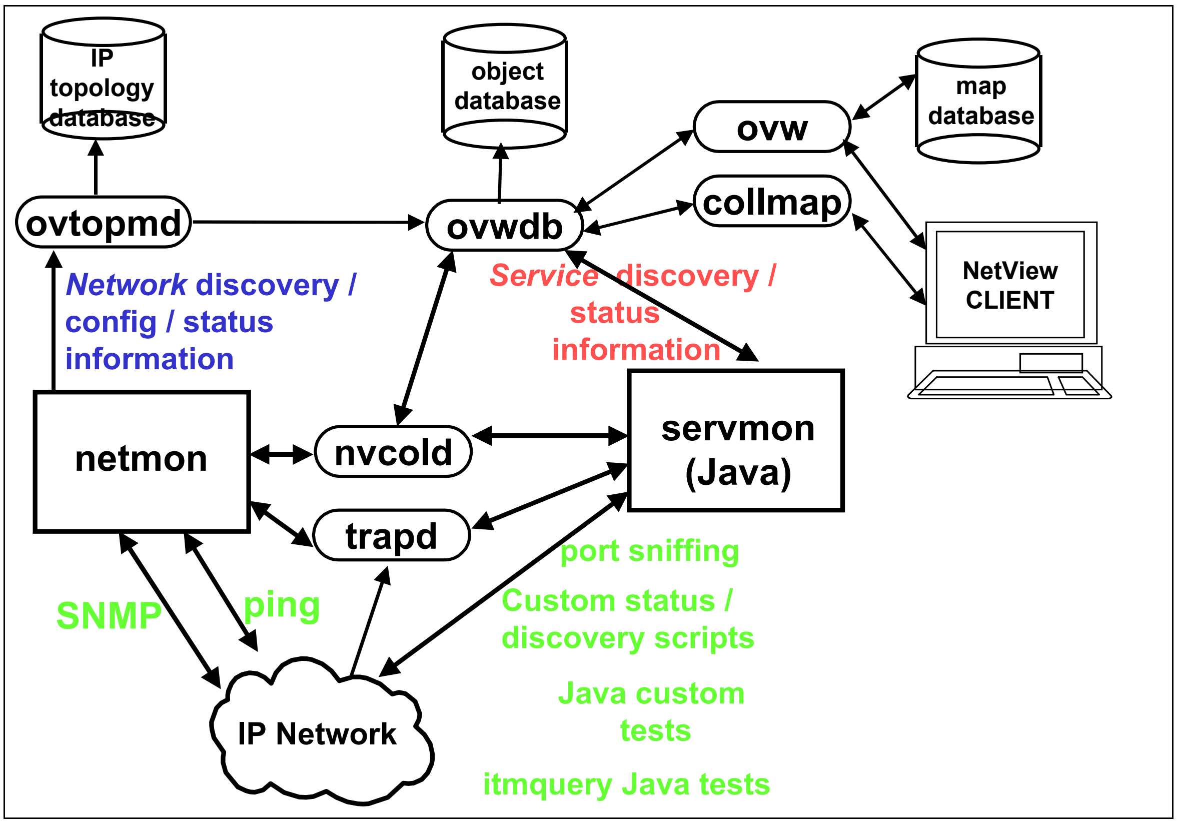
nvsniffer can be used in a slightly different way, if required. Instead of detecting services on a specified port, a discovery script can be supplied in field five and a status script can be supplied in field six. The port field can then be used to pass parameters to these scripts, if desired.

nvsniffer has some significant limitations. It is a command, not a daemon, so needs configuring to run periodically to maintain accurate status and discovery information. This has been possible recently with the "-r" parameter of the nvsniffer command, or by using the Unix cron facility or Windows scheduler.

Configuring the servmon daemon

NetView 7.1.4 has introduced the servmon daemon to replace nvsniffer - the Release Notes suggest that nvsniffer users now migrate to servmon. It is a Java-based daemon rather than a command, which starts automatically with the other NetView daemons. (If required, automatic startup of servmon can be disabled using the **serversetup** utility and following the Configure -> Set Options for daemons -> Set options for topology, discovery and database daemons -> Set options for servmon daemon path).

servmon responds to asynchronous events that it receives from NetView's trapd and nvcold daemons.



servmon.conf

The servmon daemon is configured with the default configuration file, /usr/OV/conf/servmon.conf, which has a very similar format to the old nvsniffer.conf. The format for each entry is:

```
[Node Field]|[TCP Port]|[Service SmartSet]|[Service Label Name]|[Discover
Test]|[Status Test]|[Discover Node Selection Criteria]|[Status Interval]
```

where:
Node Field
is the name of the Boolean field to create on a node object if the service is successfully discovered. A value must be specified for this parameter. The value cannot contain a space. If the discovery attempt of a service against a node is successful, this field is set to TRUE.
TCP Port
is either a comma-separated list of the TCP port numbers to test on each node, or arguments for a custom plug-in, launchable test if one is specified. A value must be specified for this field unless a custom test is specified for both the Discover test and Status test fields.
Service SmartSet
is the name of the service SmartSet to create if a service is discovered. If a value is not specified, a service SmartSet is not created. The value cannot contain a space. If the *Node Field* value is set to TRUE, the node automatically is made a member of this SmartSet. This field is also used as the node selection criteria for monitoring the status of the service. If a value is not specified for this field, the status of the service defined by this entry is **not** checked.
Service Label Name
is the symbol label name assigned to the service object created as a result of the service being discovered. A value must be specified unless the *Service SmartSet* field is not specified. The value can contain a space.
Discover Test
is the name of the optional, custom plug-in module to use to discover a service. Specify the full path of the launchable application. The same launchable application can be specified as the one that is specified in the Status Test field. The use of Java JAR file custom extensions is reserved for use only by the Tivoli NetView product.
Status Test
is the name of the optional, custom plug-in module to be used to check the status of a service that has already been discovered. Specify the full path of the launchable application. The same launchable application can be specified as the one that is specified in the Discover Test field. The use of Java JAR file custom extensions is reserved for use only by the Tivoli NetView product.
Discover Node Selection Criteria
specifies the nodes to test during service discovery. Specify either the name of any existing SmartSet, or an asterisk (*) to specify all managed IP nodes. The value cannot contain a space.
Status Interval
specifies how often (in minutes) to check the status of this service on nodes known to have this service. The maximum value that can be specified is 44640 minutes. A value of 0 specifies that no status checking should be performed for this service.

General points about servmon.conf:

- Comments can (and should!) be used. Comment lines start either with # or with // - trailing comments are *not* allowed
- If a service is ever removed from the node, servmon removes the designated boolean field from the node after it has determined that the service has been unavailable for the interval specified by the "Service Down Delete Interval" in the xnmsnmpconf dialogue on NetView/Unix, or the "Delete Services down for ..." interval in the nmpolling dialogue on NetView/Windows
- Either forward or backward slashes can be used when specifying pathnames to custom plug-in modules on Windows
- Remember that custom plug-in modules are run on the NetView server, not on the destination nodes being tested
- The Discover Node Selection Criteria field should be specified using a SmartSet whenever possible; otherwise all nodes in the NetView object database will be tested for that particular service
- Each configuration file entry must be on a single line
- servmon logs to /usr/OV/log/servmon.log by default. The level of logging is controlled in /usr/OV/conf/servmon-log4j.properties.
- After changing servmon.conf, the servmon daemon must be stopped and restarted

Check the servmon.conf file itself for comments on how to use the file. The *man servmon* command on a Unix system also provides further information. The servmon daemon and configuration file are documented in the new NetView 7.1.4 Administrator's Guide for Unix and in the new NetView 7.1.4 User's Guide for Windows.

Other configuration utilities for servmon

In addition to the servmon.conf configuration file, servmon is also controlled by two new parameters in the xnmsnmpconf utility (Options -> SNMP Configuration menu item) on NetView on Unix or the nmpolling utility for NetView on Windows. The following parameters cannot be set for individual nodes or collections of nodes, only for the Global Default:

• Service Discovery Interval	default is 12 hours
• Service Down Delete Interval	default is 7 days

These values are saved in /usr/OV/conf/service_polling.conf. Note that short intervals in these parameters can produce a very heavy load on the NetView system. For performance reasons, it is recommended that the Service Discovery Interval should be at least 1 hour.

The Options -> Topology/Status Polling Intervals menu also has a servmon parameter. The **Discover New Services** check box determines whether servmon performs discovery or not.

Changes to this flag and the two parameters modified by xnmsnmpconf / nmpolling, are stored in /usr/OV/conf/service_polling.conf. Any changes to this file result in a CSP_EV NetView event - a Change Service Polling Interval trap. This trap is configured by default to be LogOnly in NetView, so can only be seen by viewing /usr/OV/log/trapd.log. If you are watching /usr/OV/log/servmon.log, it sometimes takes several minutes before entries are reported that reinitialize the monitors.

Differences between nvsniffer.conf and servmon.conf

The differences between nvsniffer.conf and servmon.conf are:

- An extra, final field specifying the frequency of status checks. Note that this is inactive if *either* this value is 0, *or* the SmartSet field is not specified.
- The location of the configuration file is /usr/OV/conf/servmon.conf by default and can be modified using the “-c” parameter to the servmon command. This can be specified in /usr/OV/jars/start_servmon.sh which configures the servmon daemon startup. .
- Service objects are created in the NetView object database for each service discovered on a node, when the Service SmartSet field is specified.
- A third option is available for Status and Discovery plug-in modules. Custom Java class files can be used, but they are only supported for IBM internal use and are not documented. There are a number of samples provided in servmon.conf using these Java plug-ins.
- The nvsniffer discovery process has no simple mechanism for deleting references to services that have been down for a given period. The Service Down Delete Interval, customized using xnmsnmpconf / nmpolling, provides this for servmon.

Custom plug-in modules for servmon discovery and status scripts

Custom plug-in tests can consist of any thread-safe, re-entrant launchable script or executable file. servmon launches the exact syntax specified in the Status Test or Discovery Test field, including all arguments designated in the TCP Ports field (with the comma separator characters removed).

Custom launchable tasks can be any executable program or script that:

- Can receive and parse the character string node name and command arguments (if any) that servmon "passes" to the script or program
- Can return an integer value upon completion, which must be one of:
  - 2 = Normal status (the test succeeded)
  - 3 = Marginal status (the test partially succeeded)
  - 4 = Critical status (the test failed)
  - 0 = the test cannot be performed (servmon treats this value the same as the value 4 - (Critical/failed))

Values returned that are outside this range are treated as 4 (failure). A discovery test is considered successful if either the Normal or Marginal status values are returned.

The calling sequence for custom launchable tasks launched by servmon is as follows:

```
    <task launch syntax> <node name> <arguments ...>
```

where:
<task launch syntax> is the value of the Discover Test field when discovery of the service is being attempted, or the value of the Status Test field when an existing service is being monitored for status.

<node name> is the name of the node that the test is being performed on (servmon automatically passes the node name as the first parameter)

<arguments> are the values taken from the TCP Ports field (minus the comma delimiters)

Further information is provide by the **man servmon** command and a sample plugin is provided in /usr/OV/prg_samples/launchport.

Events generated by servmon

A number of different events are generated by servmon:

58916975	SUP_EV	Service Up
58916976	SDWN_EV	Service Down
58916977	SMAR_EV	Service Marginal
58916978	SACK_EV	Service Acknowledged (Only applicable to NetView/Win)
58916979	SUACK_EV	Service Unacknowledged (Only applicable to NetView/Win)
58916980	MSER_EV	Service Managed (applicable when a user manages a service icon (NetView/Windows) or when a user manages a node containing the service (all platforms))
58916981	USER_EV	Service Unmanaged (applicable when a user unmanages a service icon (NetView/Windows) or when a user unmanages a node containing the service (all platforms))
58982417	SADD_EV	Service Added
58982418	SDEL_EV	Service Deleted
58982422	ATADD_EV	Service Attribute Added (used when a new service attribute is added to the NetView object database but a service object is not created; that is, when a service SmartSet is not specified in servmon.conf)
58982423	ATDEL_EV	Service Attribute Deleted (used when a service *attribute* is deleted from the NetView object database, but not when a service *object* is deleted)

Only two of these events are configured by default with a Tivoli Enterprise Console (TEC) event class - the Service Up and Service Down events; both map to the TEC_ITS_SERVICE_STATUS class. Within the TEC slot mapping, the servicestatus field is set to 1 for the Service Up event, and is set to 2 for the Service Down event.

In spite of this configuration, the default NetView ruleset shipped for forwarding events to TEC (TEC_ITS.rs) does *not* contain these two events. They must be added to this ruleset (or any other ruleset that is used) to forward these NetView events to TEC.

Note that TEC 3.9 *does* provide service correlation rules but such rules are fired on receiving TEC_ITS_NODE_SERVICE_IMPACT events, which are generated by the State Correlation Engine (SCE) functionality in TEC 3.9 / NetView 7.1.4.

servmon service objects

Service objects are created in the NetView object database for each service discovered on a node, when the Service SmartSet field is specified. Service objects contain status (such as Normal or Critical). On Windows systems, the status of a service object can contribute to the overall IP status of a node; on NetView for Unix systems, the service status does *not* contribute to a node's overall IP status.

The Selection Name of these service objects is of the format:

```
   <Selection Name of Node>:<SmartSet Name>.CF
```

A typical entry in the NetView object database for a service object would look like this (use ovobjprint to view such information):

```
OBJECT: 9111

      FIELD ID    FIELD NAME        FIELD VALUE
      10          Selection Name    "poppet.skills-1st.co.uk
:Skills_Unix_ITM_DMXProcess.CF"
      163         Service Status    Critical (4)
      164         Down Time         1074706838
      220         TopM Node ID      262
```

This is a service object representing the Skills_Unix_ITM_DMXProcess SmartSet on the node poppet.skills-1st.co.uk. The service being monitored is in a critical status and went down at time 1074706838 since the epoch (January 1st 1970). This service object is contained within the object 262, which is the object for node poppet.skills-1st.co.uk.

Note that you can use perl to help translate epoch times. The following example translates the above epoch time to GMT time:

```
   perl -e 'print scalar gmtime 1074706838'
```

Note that a service object is not created if no SmartSet field is specified for that entry in servmon.conf. In this case, a service attribute is created in **/usr/OV/databases/servmon/service_attributes.properties**. Each entry of the service_attributes.properties file contains:

- The node's selection name
- The *{Node Field}* value
- The discovery timestamp

When the service attribute has been *Critical* for the configured maximum, the respective entry is removed from the *service_attributes.properties* file, and the *{Node Field}* field is removed from the node (it is never set to FALSE).

Increasing debugging information for servmon

servmon logs to /usr/OV/log/servmon.conf by default. The level of logging is controlled in /usr/OV/conf/servmon-log4j.properties and provides basic INFO logging. To increase the level of debugging, this file can be modified as follows:

- Change line 7 from:
  - ❖ log4j.rootCategory=INFO, A1, R to
  - ❖ log4j.rootCategory=DEBUG, A1, R
- and change the last line from:
  - ❖ log4j.category.com.tivoli.netview.servmon=MIN#com.tivoli.netview.log4j.NvLevel to
  - ❖ log4j.category.com.tivoli.netview.servmon=DEBUG#com.tivoli.netview.log4j.NvLevel

Observations on using servmon

Note that by default, servmon will wait 10 seconds for each TCP Port test or custom plug-in Status or Discovery test. This value can be changed using the -t <seconds> parameter to servmon, which can be configured in /usr/OV/jars/start_servmon.sh. Increasing this time should only be done with care as it will increase the demand on the number of threads that servmon uses (15 by default). However, if the time interval is too short for a test (or the machine is very busy), and the test does not complete, the result will be as though the test failed. Timeout failures can be seen in /usr/OV/log/servmon.log with a "Timer expired" message.

Unmanaging / managing a node that has services monitored on a Unix NetView, does not appear to generate Service Unmanaged / Service Managed traps.

A major glitch with servmon on Unix NetView is that the documentation indicates that service icons will be created at the interface level of nodes where servmon has discovered services. This is *only* true for Windows NetView, *not* for Unix NetView (although the service objects are created in the NetView object database)..

itmquery

itmquery is a new utility shipped with NetView 7.1.4 to provide integration between NetView and the IBM Tivoli Monitoring (ITM) 5.1.1 suite of applications. It is a Java utility that is *either* activated using the itmquery command *or* it can be used under-the-covers by the servmon daemon.

Configuring itmquery

itmquery has two configuration files, both in /usr/OV/conf:

- **itm_servers.conf** lists the IBM Tivoli Monitoring servers that you want to monitor and the account information for each server. The information is used by the itmquery function, and by the service monitor function of servmon to perform IBM Tivoli Monitoring attribute discovery tests. Do not use an editor to change this file. Instead, use either the **serversetup** utility or use the **itmquery --add-server** or **--remove-server** parameters to modify this file. Parameters required are:
  - ❖ TCP/IP address or resolvable name of the IBM Tivoli Monitoring server
  - ❖ The port that the oserv runs on - normally 94
  - ❖ A user name that is both defined to the IBM Tivoli Monitoring server Operating System *and* is associated with a Tivoli Administrator with sufficient roles to run ITM commands
  - ❖ The password associated with the above user
- **itm_attributes.conf** is used to specify the names of services that the **itmquery** function should search for when it queries IBM Tivoli Monitoring endpoints to determine which services are installed on the endpoints. The servmon daemon also uses this file for service discovery purposes. Use a text editor to modify this file. The file itself provides helpful configuration information. Fields to specify are:
  - ❖ A user-specified product name

- ❖ A perl regular expression to match against an ITM Resource Model (RM) name
- ❖ For example,
- ❖ Standard_Unix_ITM|^DMX.*$
- ❖ specifies a product name of Standard_Unix_ITM which looks for any ITM Resource Model starting with the characters DMX - this includes all the standard Unix RMs.

Note that both these files are shipped as owned by root and have unix file protection bits set to 600. This means that any non-root user cannot successfully run an itmquery command (they simply get a blank response, *not* an error message).

Also note that the documentation refers to "IBM Tivoli Monitoring servers". It appears that it is adequate to specify the TMR Server or any other single Tivoli Managed Node which has ITM installed on it, even if Endpoints running an ITM engine are assigned to different endpoint gateways (and therefore different ITM gateways).

Useful itmquery commands are:
- itmquery --verify-server-info
- itmquery --dump-endpoints

There are two logfiles for itmquery in /usr/OV/log:
- msgItmquery.log
- traceItmquery.log

The level of logging is controlled by /usr/OV/conf/itmquery-log4j.properties (change the trace threshold and category entries from **MIN** to **DEBUG**).

Current limitations with itmquery

There are currently some distinct limitations with itmquery, some of which are documented in itm_attributes.conf and in the NetView 7.1.4 Admin Guide.

- The only entries NetView's servmon daemon and itmquery utility currently support are WAS, MQ and DB2. Other lines are supplied in itm_attributes.conf but are commented out (i.e. "Oracle" through "WebLogic"); they have NOT been verified to work by IBM and are not officially supported with this release of NetView. However (says the documentation in itm_attributes.conf) , these entries probably would work without problems and a future NetView release will most likely officially support more of these entries.
- If you wish to take the next step, and have servmon automatically set a capability field on all nodes that are recognized to be running services configured in itm_attributes.conf, you have to add an entry like this to the servmon.conf file (it should all be on one line)
  - ❖
    ```
    isStandard_Unix_ITM|Standard_Unix_ITM|Standard_Unix_ITM|Standard_Unix
    _ITM|/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.Discovery
    Monitor|/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.Discov
    eryMonitor|ITM_Endpoints|5
    ```
- Please note the following about this servmon entry:
  - ❖ The first field, "isStandardUnixITM", is the boolean NetView object database field that will be set for nodes discovered to have a standard Unix ITM Resource Model

    - the second field within the entry, "Standard_Unix_ITM", **must** match an associated "Standard_Unix_ITM" entry within itm_attributes.conf
    - servmon does **not** *always* automatically create a SmartSet for service attribute entries, so if you want to have a "Standard_Unix_ITM" SmartSet (as specified in the third field above), you may have to manually create this SmartSet on your own using the SmartSet editor or the nvUtil / smartsetutil command line utilities. The third field in the example above may be ignored from the point-of-view of creating a SmartSet (however it, and the fourth field (the SmartSet label) may be necessary to specify the nodes to monitor if a Status Test field is supplied).
    - The Discover Test and Status Test fields (fields five and six) use the same Java custom plug-in script (remember that Java plug-ins are only supported for IBM Internal use)
    - Field seven is the Discover Node Selection Criteria, specifying the SmartSet of nodes to perform discovery against. A SmartSet called ITM_Endpoints already exists.
    - The eighth field specifies how frequently the Status Test should be run against the set of nodes defined by the Service SmartSet field (field three)
- The ITM_Endpoints SmartSet can be created by uncommenting and modifying the line already supplied in servmon.conf (all on one line):

    -
      ```
      isITMEndpoint||ITM_Endpoints|ITM
      Endpoints|/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.Disc
      overyMonitor|/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.D
      iscoveryMonitor|*|5
      ```
    - This SmartSet should automatically find any node supporting ITM

- It appears that itmquery commands sometimes hang - they often seem to take an inordinate length of time to run when they do complete. Sometimes itmquery commands produce some relevant output and then output the command usage (as though it had lost some of the command part way through execution)
- In order for servmon Status Tests that use Java itm classes to run successfully, the patch for PMR 43850 must be applied - this is included in NetView 7.1.4 Fixpack 1.
- **itmquery --dump-endpoints** never seems to find ITM Resource Models running on endpoints within a TMR with an AIX TMR Server, even though the command correctly finds the endpoints in that TMR. This means that any servmon entries using Java itm classes for Discovery and Status tests fail. **itmquery --dump-endpoints** does appear to work correctly for endpoints in a TMR with a Windows TMR Server. Looking at /usr/OV/log/traceItmquery.log, it appears that the failing commands never complete. This is caused by a problem in the ITM resource manager (tmnt_rm) and is solved by applying ITM 5.1.1 Fixpack 6 (which is the equivalent of ITM 5.1.2).
- The itmquery commands that succeed in examining ITM Resource Models only appear to check whether a Resource Model has been sent to an endpoint - it does not appear to check its status. This can be tested using the following command to stop all Resource Models in a particular ITM profile. **itmquery --dump-endpoints** still reports successful on these stopped Resource Models
    - wdmeng -e <endpoint> -p <profile>#<region> -stop
    - wdmeng -e <endpoint> [-p <profile>#<region>] <resource model> -stop
-

An example of custom plug-in scripts for use with servmon

The servmon daemon has three different ways it can detect services:

- TCP port sniffing
- Custom plug-in Discovery and Status Test scripts, which may or may not be written in Java (but Java scripts are only supported for IBM Internal use)
- Using itmquery by way of IBM-supplied Java plug-in scripts for Discovery and Status tests

The servmon.conf file, as shipped, provides the following line to detect the NetView Jetty webserver on ports 8080 or 80, using a Java custom plug-in (it should all be on one line):

```
isService_Jetty|8080 80 Jetty|Jetty|Jetty|
/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpD
iscoveryMonitor|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpSe
rviceTests.HttpDiscoveryMonitor|*|20
```

A simpler, better supported way of achieving the same effect is as follows:

```
isPortService_Jetty|8080 80|PortJetty|Port Jetty|||*|20
```

Shellscript to perform similar functionality to itmquery

Given that itmquery currently has some limitations, and the fact the Java plug-in scripts are not documented and not supported for general use, it would be useful to have a shellscript which performs similar checking to the itmquery-style of Java Status and Discovery Test scripts.

The script shown below is *not* as flexible as itmquery in that:

- It only works within 1 TMR
- The NetView machine must have the Tivoli Framework and ITM 5.1.1 installed to provide access to the wdm commands
- No use is made of itm_servers.conf

An advantage of this script is that it does test the status of ITM monitors on an endpoint, to ensure that they not only exist, but also are Running.

A possible problem arises if the naming convention throughout the entire Tivoli environment is not consistent. servmon will pass the Selection Name of a node from the NetView object database, to servmon, as the first parameter. This Selection Name will depend on the naming convention in place on the NetView system when the node was discovered. If that was Domain Name System (DNS), then the node name will be a fully qualified domain name (fqdn). If NetView was using /etc/hosts with shortnames, then the node name will be a short name. If neither mechanism could resolve the node name on discovery, the Selection Name field will be the IP address.

The Tivoli TMR environment *should* be using the same name-to-address resolution mechanism - DNS is very strongly recommended for all names. However, endpoint names (labels) may not be the same as the fqdn of a node passed by NetView. Therefore the script assumes that endpoints have labels that at least start with the short hostname and compares Selection Name node names and endpoint names that have both been truncated to the first dot.

```
#!/bin/ksh
# Script to replace some of the itmquery function shipped with NetView 7.1.4
#
# It is assumed that the NetView system has ITM 5.1.x installed on it
# to provide access to the wdmlseng command. If this cannot be the case,
# something like ssh will be needed to run wdmlseng commands
# on remote systems, having queried the itm_servers.conf file
#
# No use is made of the /usr/OV/conf/itm_servers.conf file in this script
# One implication of this is that only the local TMR is supported.
#
# Two parameters are required - the name of a node to check and the
# name (or partial name) of the Resource Model (RM) to check for.
# The node name is not used when running this script standalone.  When
# called by servmon, servmon passes a node name as the first parameter.
#
# This script can be used both as a discovery and as a status script
# in servmon.conf
# servmon expects exit codes as follows:
#  2 = Normal   3 = Marginal  4 = Critical  0 = test cannot be run
#
# Source the Tivoli environment, if it exists
#
#set -x
if [ -f /etc/Tivoli/setup_env.sh ]
then
 . /etc/Tivoli/setup_env.sh
fi
#
RET_CODE=4
# get Selection Name passed by NetView
HOST="$1"
RM="$2"
#
# Find each Alive endpoint for each ITM MN gateway
#   and for each, see if the host matches a valid endpoint
#   and whether the specified RM exists and is Running, eliminating duplicates
#
for i in `wdmmngcache -m all -l | grep -v ManagedNode | grep -v \| \
   | grep -v + | grep -v Unreachable | grep -v DMEngineOff  | cut -f1 -d " " `
do
  shorthost=`echo "$HOST" | cut -f1 -d . `
  shortendpoint=`echo "$i" | cut -f1 -d . `
# Check whether short hostname is same as a shortname of a valid endpoint
# If so, check for running RM on Endpoint and return status code
  if [ "$shorthost" = "$shortendpoint" ]
  then
    RUN=` wdmlseng -e "$i" | grep "$RM" | grep Running| sort -u | cut -f1 -d : `
     if [ "$RUN" != "" ]
     then
        echo "$RUN" running on "$i" Endpoint
        RET_CODE=2
     else
        echo failure $RUN
        RET_CODE=4
     fi
  fi
done
exit $RET_CODE
```

The script uses the **wdmmngcache** command to find all ITM gateways and for each ITM gateway it gets a list of all endpoints whose status is *not* either Unreachable or DMEngineOff. For each endpoint, it then truncates the name to the first dot, compares it against the short name of the node passed by NetView, and checks to see if the Resource Model (specified by the second parameter to the script) exists and is Running on the endpoint. Success results in an exit return code of 2 (Normal); failure results in an exit return code of 4 (Critical).

Commands used to stop, start and test ITM monitoring are:

- `wdmcmd -stop -e <endpoint>`   to stop the ITM engine
- `wdmcmd -restart -e <endpoint>`   to start the ITM engine
- `wdmlseng -e <endpoint> -verbose`   to display status of ITM engine

Configuring servmon to use a custom plug-in shellscript

Configure /usr/OV/conf/servmon.conf with single-line entries for each Resource Model or partial name of a Resource Model, that you wish to monitor, providing the RM name in the second field. Don't forget to stop and start servmon to ensure the new configuration is read.

```
isSkills_Unix_ITM_DMXProcess|DMXProcess|Skills_Unix_ITM_DMXProcess|Skil
ls_Unix_ITM_DMXProcess|/usr/OV/conf/skills/skills_itmquery.sh|/usr/OV/c
onf/skills/skills_itmquery_status.sh|ITM_Endpoints|5

isSkills_Win_ITM_Services|TMW_Services|Skills_Win_ITM_Services|Skills_W
in_ITM_Services|/usr/OV/conf/skills/skills_itmquery.sh|/usr/OV/conf/ski
lls/skills_itmquery_status.sh|ITM_Endpoints|5
```

These examples specify different Discovery and Status Test scripts, though actually the discovery script, **skills_itmquery.sh**, has simply been copied to **skills_itmquery_status.sh**.

The speed of service discovery is controlled by the Service Discovery Interval specified in xnmsnmpconf / nmpolling. The nodes that are checked to try and discover these services are those in the SmartSet ITM_Endpoints. Nodes where these services are discovered are added to the SmartSet Skills_Unix_ITM_DMXProcess (if the DMXProcess Resource Model is discovered) and Skills_Win_ITM_Services (if the TMWServices Resource Model is discovered). Status checking of the services is carried out every 5 minutes on the nodes in the Skills_Unix_ITM_DMXProcess and Skills_Win_ITM_Services SmartSets respectively.

Since these scripts may take a relatively long time to run, /usr/OV/jars/start_servmon.sh has been modified to include a "-t 20" parameter to allow the scripts to run for 20 seconds.

Issues can be tracked by checking /usr/OV/log/servmon.log - if necessary use the information provided earlier to increase the level of debugging.

Generating service icons on Unix NetView systems

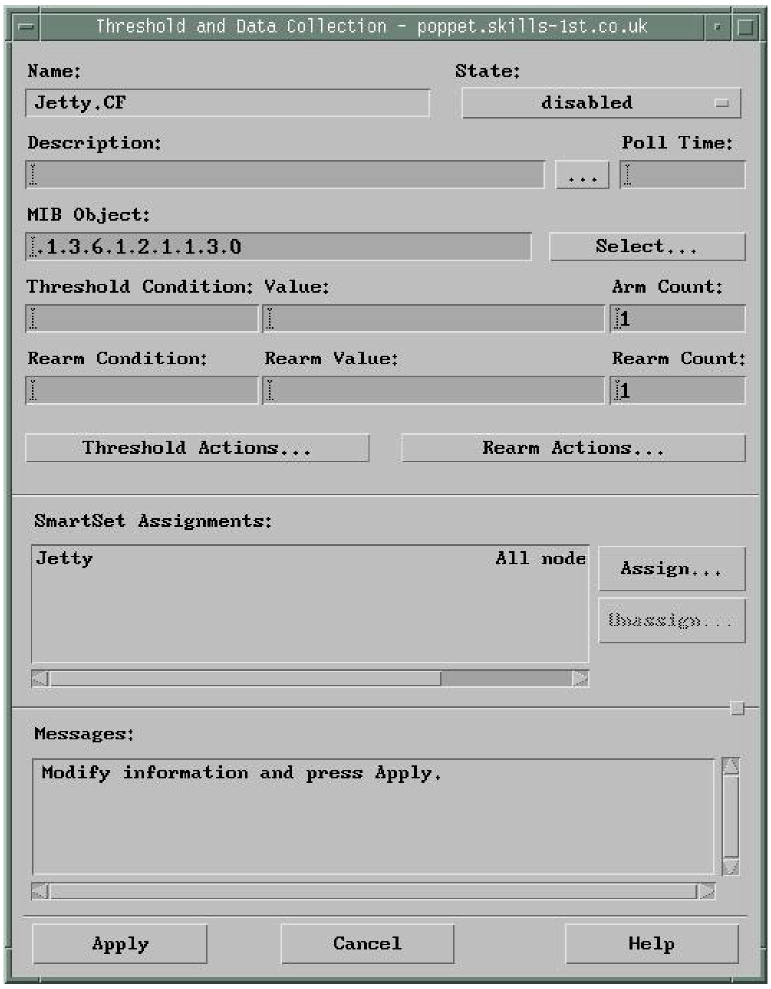
A major drawback with servmon on Unix NetView systems is that the documentation suggests that service icons should appear at the interface level for any node where a service has been discovered. Although this works on Windows NetView systems, it does *not* work for Unix systems. This section describes a solution to this. Note that it is *not* applicable to NetView on Linux as it utilises the Agent Policy Manager (APM) which is not supported on Linux.

Creating NetView icons using the Agent Policy Manager (APM) and MLM

All versions of NetView ship with implementations of the Mid Level Manager (MLM) which is an extra SNMP subagent, designed for offloading some of NetView's work to a distributed system or systems. MLM is available for AIX, Solaris, HP-UX and Windows (not Linux). In the samples shown here, the AIX MLM was installed on the same system as NetView.

Regardless of whether any MLMs are deployed, NetView on AIX and Solaris has a daemon called C5d, which is *not* started by default with all other NetView daemons. C5d implements the Agent Policy Manager which permits a simple way to customise a local or remote MLM, by filling in values in a table in a Graphical User Interface (GUI). The Agent Policy Manager daemon can be customized to start with the other NetView daemons, using the **serversetup** utility. The APM Configuration tool can be started either from the Tools menu or from the Tool Palette.

The APM can be used to create policies for Data Collection and Thresholding of SNMP data. The APM Configuration Tool allows specification of what SNMP variable to sample, whether to store and/or threshold values, threshold and rearm conditions, and actions to run on threshold and rearm events. Once the monitor is specified, the targets for this monitoring are denoted by selecting one or more NetView SmartSets. (In many ways, the APM Configuration panel is analogous to a Tivoli Profile Manager - in the top half of the window, you specify *what* you want to monitor; in the bottom half, you specify *where* to deploy this monitoring).


Threshold and Data Collection - poppet.skills-1st.co.uk

Name: Jetty.CF

State: disabled

Description:

Poll Time:

MIB Object: .1.3.6.1.2.1.1.3.0

Threshold Condition:

Value:

Arm Count: 1

Rearm Condition:

Rearm Value:

Rearm Count: 1

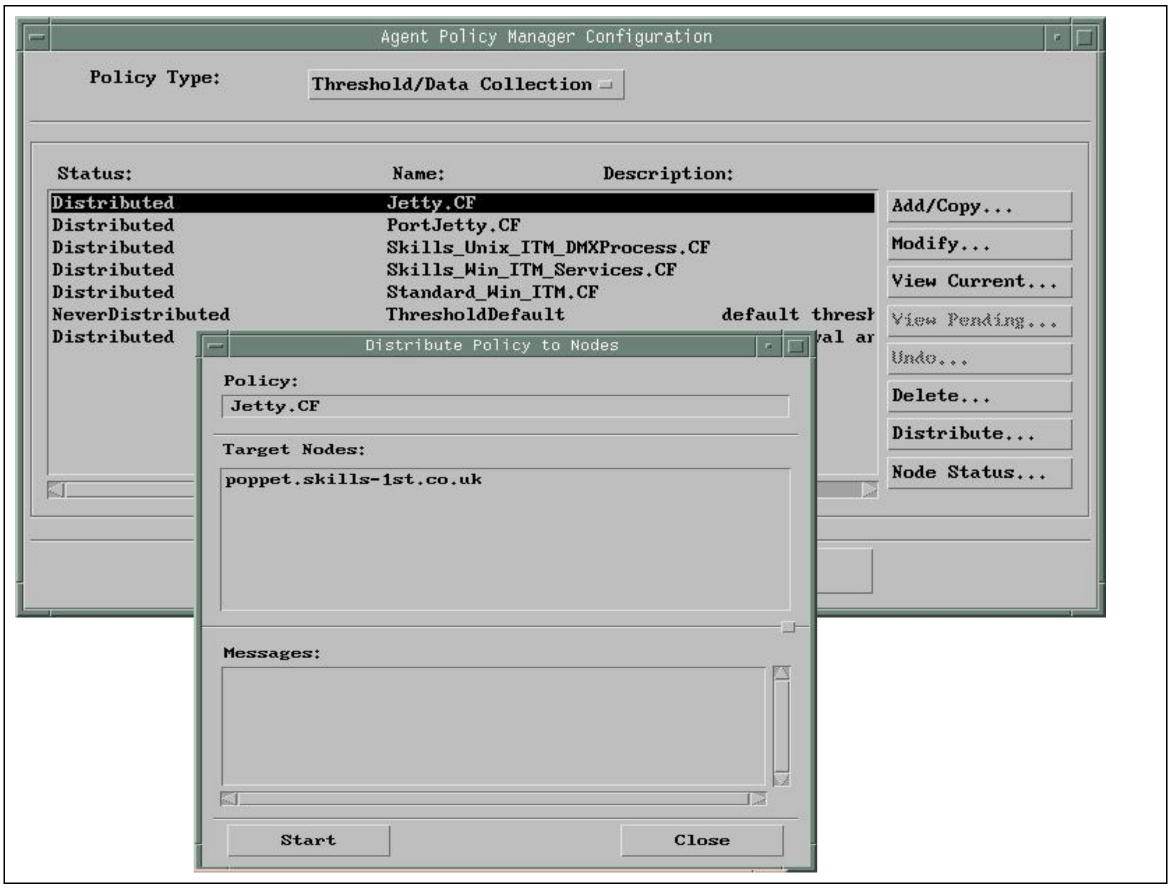
SmartSet Assignments:

Jetty All node

Messages:

```
Modify information and press Apply.
```

When the APM is complete, it is Applied and can then be Distributed (again, just like a Tivoli Profile is distributed to a number of target subscribers). Remember that the purpose of APM is to customize one or more MLMs. When the APM is distributed, the C5d works out which MLM(s) need to be configured in order to distribute this monitoring to the nodes specified by the SmartSets in the APM. The targets may be managed by a number of different MLMs throughout the enterprise. In the screenshot below, the only MLM that needs to be configured to distribute to all nodes in the SmartSet Jetty, is the MLM on poppet.skills-1st.co.uk.



```
Status:              Name:                          Description:
Distributed          Jetty.CF
Distributed          PortJetty.CF
Distributed          Skills_Unix_ITM_DMXProcess.CF
Distributed          Skills_Win_ITM_Services.CF
Distributed          Standard_Win_ITM.CF
NeverDistributed     ThresholdDefault               default thresh
Distributed                                         al ar
```

Policy:
Jetty.CF

Target Nodes:
poppet.skills-1st.co.uk

Messages:

One of the extra benefits of APM is that, for values customizing an MLM's Threshold and Collection table, an icon is created on all NetView maps, at the interface level, for each node where that APM is deployed (that is, you get extra icons in all the nodes specified by the SmartSet, not in the node representing the MLM). This mechanism can provide icons exactly where we need them for servmon! Although we do not need to monitor actual SNMP values, we *do* want the icons.

In the first screenshot above, the State of the monitor is **disabled**, the SNMP MIB variable to sample is a dummy **.1.3.6.1.2.1.1.3.0** (MIB-2 sysUpTime), no threshold or rearm values or actions are specified, but the threshold and rearm counts are set to **1** (otherwise MLMs on Windows complain!). The **Jetty** SmartSet is specified as the target assignment.

servmon and the NetView object database

servmon can create SmartSets based on whether a service is supported. In the example in the previous section, APM is using a SmartSet called **Jetty** which has been created by a line in servmon like this:

```
isService_Jetty|8080 80 Jetty|Jetty|Jetty|
/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpD
iscoveryMonitor|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpSe
rviceTests.HttpDiscoveryMonitor|*|20
```

When servmon first discovers this service on a node, it automatically creates a new field in the NetView object database, whose name is the concatenation of the node and the service, joined by a “:” (colon), with “.CF” on the end. For example:

```
        poppet.skills-1st.co.uk:Jetty.CF
```

This object is created, even though servmon cannot create an associated icon.

By using the ovobjprint command and redirecting the output to a file, it is possible to see that the object representing the node poppet.skills-1st.co.uk has Object Id 262 and a large number of attributes, including the **isServiceJetty** attribute:

```
OBJECT: 262

      FIELD ID    FIELD NAME              FIELD VALUE
      10          Selection Name                "poppet.skills-1st.co.uk"
      11          IP Hostname                   "poppet.skills-1st.co.uk"
      14          OVW Maps Exists               2
      15          OVW Maps Managed              2
       .          .....................         ...
      518         isTest                        TRUE
      5061        isITMEndpoint                 TRUE
      5062        isService_IBM_DB2             TRUE
      5063        isService_Jetty               TRUE
      6091        isStandard_ITM                TRUE
      9103        isSkills_Unix_ITM_DMXProcess        TRUE
      12094       isPortService_Jetty           TRUE
```

The object for the SmartSet called Jetty that is created automatically by servmon, is as follows:

```
OBJECT: 5072

      FIELD ID    FIELD NAME              FIELD VALUE
      10          Selection Name                "Selection Name5072"
      150         isCollection                  TRUE
      151         nvCollectionName              "Jetty"
      152         nvCollectionDescription       "All nodes that are Jetty"
      153         nvCollectionRule            "("isService_Jetty" ="TRUE")"
```

There is also an entry for the *icons* representing the SmartSet Jetty. This is called **Jetty.CF**:

```
OBJECT: 5073

      FIELD ID    FIELD NAME              FIELD VALUE
      10          Selection Name                "Jetty.CF"
      14          OVW Maps Exists               2
      15          OVW Maps Managed              2
```

The object that represents the service on a node is as follows. Note that the **TopM Node ID** field, i.e. the “containing” field, is the object ID for the node poppet.skills-1st.co.uk. This service object also has fields for Service Status and, if Critical, the time that the node went down (as time in seconds since January 1st, 1970).

```
OBJECT: 5081

      FIELD ID    FIELD NAME              FIELD VALUE
      10          Selection Name         "poppet.skills-1st.co.uk:Jetty.CF"
      163         Service Status                Critical(4)
      164         Down Time                      1074843634
      220         TopM Node ID                   262
```

These objects and fields are all created automatically by servmon and the Service Status and Down Time fields are updated automatically, based on input from Service Up / Service Down traps received via servmon and trapd. All we need is an icon for this service object that changes when the service status changes.

Combining servmon and APM

To create icons that represent the service objects, create a Threshold/Data Collection APM as described in the APM section above, and shown in the screenshots. The *name* of the APM *must* be the name of the SmartSet defined in the third field of the servmon.conf entry, with ".CF" appended (the APM Configuration tool will prevent creation of an APM with an identical name to the SmartSet). The SmartSet Assignment in the APM should also be the same SmartSet defined in the third field of servmon.conf - in other words, create a service icon for all nodes in the SmartSet defined by an entry in servmon.conf.

Although this mechanism may appear clumsy and non-automatic at first, once the first APM has been created it is trivial to copy it, simply modifying the APM name and the SmartSet assignment.

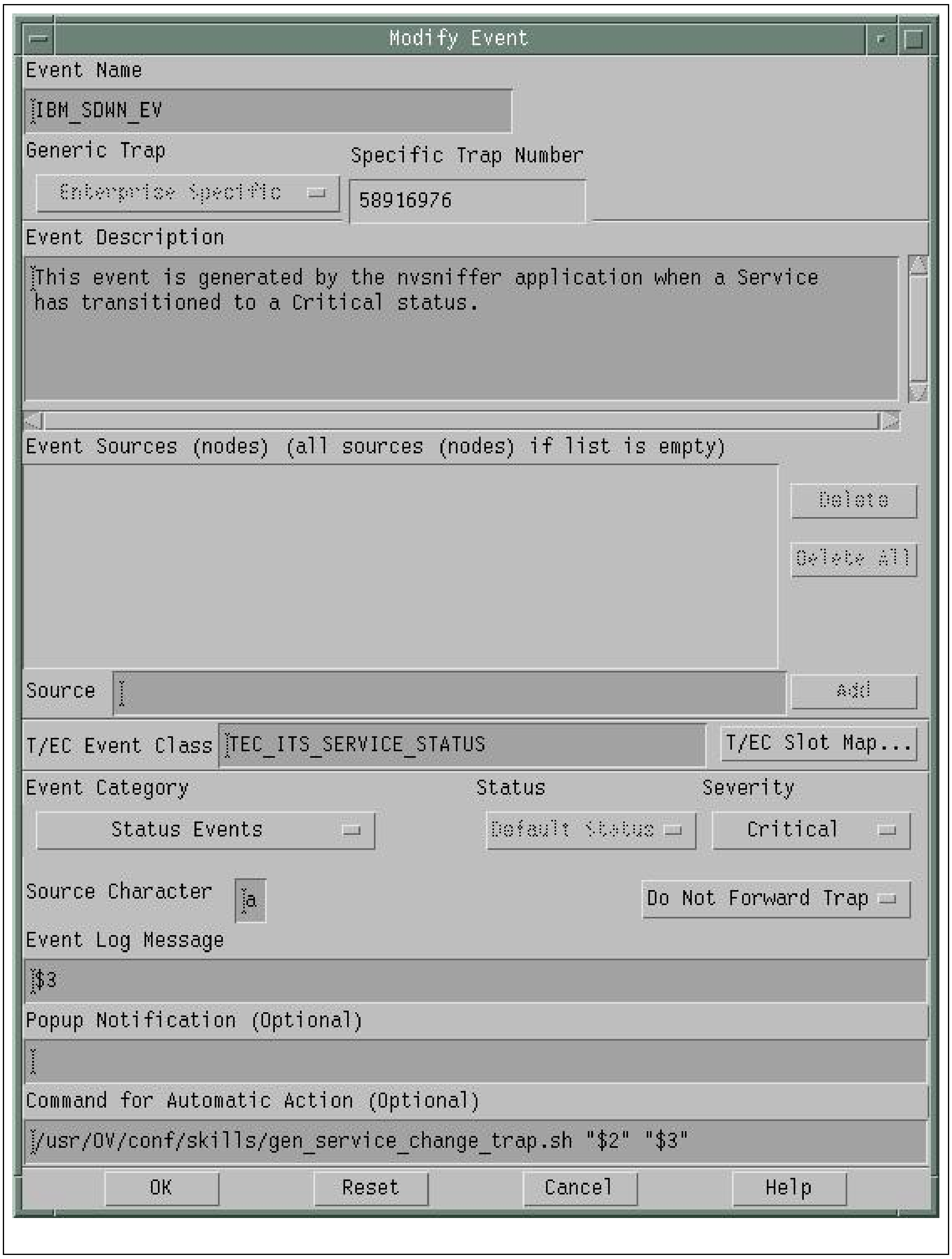
Rather than creating a new object for this APM service object, APM adds extra fields to the existing <nodename>:<SmartSet>.CF object, representing the C5d attributes (Object: 5081 in the example above).

```
OBJECT: 5081

      FIELD ID    FIELD NAME               FIELD VALUE
      10          Selection Name         "poppet.skills-1st.co.uk:Jetty.CF"
      14          OVW Maps Exists              2
      15          OVW Maps Managed             2
      17          isSMAPPL                     TRUE
      21          smDeleteObject               FALSE
      22          C5status                     3
      23          C5mlmObjectId                262
      24          C5nodeObjectId               262
      25          C5resourceObject             2
      26          C5resourceName               "Jetty.CF"
      28          isC5ManagedResource          TRUE
      163         Service Status               Critical(4)
      164         Down Time                    1074868834
      220         TopM Node ID                 262
```

Note that this Object 5081 now exists in 2 Maps and is still contained within Node ID 262.

The last piece of customization is to persuade the service icons to change colour as Service Up/Down events are received. The icon does not inherit status from the Service Status field, but the NetView Status Change trap (58916871) could be used to change the status of the service icon when a Service Up/Down event is received. Reconfigure these service status events to run a small shellscript that generates a NetView change status trap.



Modify Event

Event Name: IBM_SDWN_EV

Generic Trap: Enterprise Specific

Specific Trap Number: 58916976

Event Description: This event is generated by the nvsniffer application when a Service has transitioned to a Critical status.

Event Sources (nodes) (all sources (nodes) if list is empty)

Source:

T/EC Event Class: TEC_ITS_SERVICE_STATUS

Event Category: Status Events

Status: Default Status

Severity: Critical

Source Character: a

Do Not Forward Trap

Event Log Message: $3

Popup Notification (Optional):

Command for Automatic Action (Optional): /usr/OV/conf/skills/gen_service_change_trap.sh "$2" "$3"

The **gen_service_change_trap.sh** script is shown below. The second and third variables from the original service trap are passed to the script - the second variable is the hostname of the node sending the service trap; this variable is checked for "\" escape characters and then has the service name and ".CF" appended to it. It is used to form the Selection Name of the icon to be changed.

The third variable from the original service status trap, has 3 words:

- Service
- < The name of the service that has changed status >
- < The value of the status of the service>

This second variable is split into its component words by the script to determine the service that has changed and the status it has changed to. A status of User2 (purple) has been used for a service status of anything *other* than Normal, while Normal status (green) has been used for a service status of up. User2 has been chosen as that status is defined to contribute up the NetView map hierarchy so a node that contains a purple service icon, along with other green icons, will itself be Marginal (yellow).

```
#!/bin/ksh
#
# Send trap using the snmptrap supplied with NetView in /usr/OV/bin
# Trap here is NetView change status trap
# Source should be passed as $1 - fully qualified domain name
# $2 is 3 word string where:
#  1st word is "Service" , 2nd word is service name, 3rd word is service status
#
#
#set -x
MANAGER=`hostname`
ENTERPRISE=.1.3.6.1.4.1.2.6.3.1
# If domain name has "." and "-" they will be escaped, so unescape
SOURCE=`echo "$1" | sed "s:\\\\\\::g"`
# Split out $2 into 3 words, $1, $2, $3
set $2
SERVICE="$2"
SERV_STATUS="$3"
#
GENTRAP=6
SPECTRAP=58916871
TIMESTAMP=1
TRAPVAR=.1.2.6.1.4.1.2.6.3.1.1.2.0
SELECTION="$SOURCE":"$SERVICE".CF
if [ $SERV_STATUS = Normal ]
then
 STATUS=Normal
else
 STATUS=User2
fi
#
MESSAGE="Status changed on $SELECTION to $STATUS"
#
/usr/OV/bin/snmptrap $MANAGER $ENTERPRISE $SOURCE $GENTRAP $SPECTRAP
$TIMESTAMP  \
 $TRAPVAR Integer 14                        \
 $TRAPVAR OctetString $SELECTION           \
 $TRAPVAR OctetString "$MESSAGE Object status is"     \
 $TRAPVAR OctetString $STATUS
#
```

The final result should be that, when a service status event is received from a node, the service icon within the node should turn purple for “bad news” and green for “good news”. Once testing is complete, it may be advisable to customise the Change Status trap to be of Category LogOnly so that NetView users do not see it.

Observations using APM, MLM and servmon

There are a number of points to be aware of when installing and using MLM and the APM:

- MLM code is shipped on the same CDs as NetView, under an MLM directory
- Before installing an MLM, ensure that community names for external addresses and loopback addresses are correct both in SNMP agents and in NetView's ovsnmp.conf. MLM configuration is achieved using SNMP SET commands so readWrite access is required to any SNMP agent on an MLM.
- Beware that the AIX MLM does not install properly on an AIX 5.x system - the /usr/OV/bin/smmlm_smit install script needs modifying with an extra line (around line 144), containing an entry for AIX 5.x - simply copy the line for AIX4:
    - 4) arch=AIX4;;
    - 5) arch=AIX4;;
    - esac;;
- If a service object is deleted by servmon when the Service Down Delete Interval parameter has detected a service down for a given period, then the whole service object is deleted, including any service icon that was created with APM. If the APM Configuration tool is used to delete a service icon associated with a servmon service object, then the C5 fields are removed from the service object and the *icon* is deleted but the basic service *object* remains with the original four fields, as created by servmon.
- Occasionally the C5d daemon creates extra icons with Object Id numbers appended to the name. Try using the **Node Status -> Resynch** option from the APM Configuration tool to remove these extraneous icons. Typically these icons *cannot* simply be deleted using the **Edit -> Delete Object** menus. If the APM Tool is used to delete the icon then deleting any remaining, associated extraneous icons with the **Edit -> Delete Object** menu always seems to work, but this means that the APM has to be recreated and redistributed. If all else fails, back up the NetView databases, and use the **ovwdbdmap -d <Object Id>** command, having first determined the Object Id using ovobjprint or the **Tools -> Display Object Information** menu.

Summary

The purpose of this document was to gather together disparate information available for the new NetView 7.1.4 servmon and itmquery utilities and to demonstrate practical uses of them. Current limitations are documented and workarounds proposed to produce a system that combines network and service management.

The team that wrote this paper

Jane Curry is an independent Tivoli consultant and instructor, specializing in the Tivoli availability products - Framework, NetView, TEC and ITM. Previously, she spent 11 years in IBM working in both presales and postsales, systems and network management roles.