73

AIX

November 2001

In this issue

update

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1998 issue, are available separately to subscribers for £16.00 ($24.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aix; you will need to supply a word from the printed issue.

Editors

Trevor Eddolls and Richard Watson

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Configuring AIX to use the socksified IP stack

Letting clients access services on the public Internet can open up large security problems. Without some sort of packet filter, such as a firewall, it would be easy for an external user to attack your internal network.

Even with a firewall in place, it's still good security practice to reveal as little about your internal domain as possible, for instance, your internal IP addresses and hostnames of your clients. If we take an example such as telnet there are several ways of doing this.

First, without taking any special precautions, an internal client telnetting externally over the Internet would reveal its IP address to the outside world, and the firewall would also have to allow external boxes to route through it to reach the client. This is insecure, but from the end user's point of view is at least easy to use, because they just use telnet in the usual way.

The first method we could employ is to use Net Address Translation (NAT). Using this, the firewall would rewrite the outgoing packets so that the IP address was replaced with a public one, and do the reverse when the packet was responded to. Although this is more secure in terms of hiding your internal addresses, the firewall is still essentially routing information through, so the end client is still open to attack; there is also some additional configuration work needed on the firewall.

Another method is proxy telnet. This involves the user using the firewall as a proxy to the Internet. In this case the user would first telnet to the firewall, and then from the firewall telnet over to a host on the Internet. The advantage of this is that there is no connection directly from the client to the external host, but the disadvantage is that the user needs to do an extra step, and additional user configuration is needed on the firewall.

Finally, there is perhaps the combination of the most transparent and secure method, which is to use socks. A socks server runs on a TCP port on the firewall and acts as a transparent proxy to the Internet. A telnet client uses different TCP function calls in order to connect to

a site via a socks server. A client program that is able to do this is referred to as ‘socksified’. The socks server normally runs on port 1080 and, after the initial connection from the client to the socks server, the client sends a packet which lists the authentication method to be used, destination IP address, and port of the server it wants to connect to. The socks server will then perform this function on the client’s behalf. This is illustrated in Figure 1.

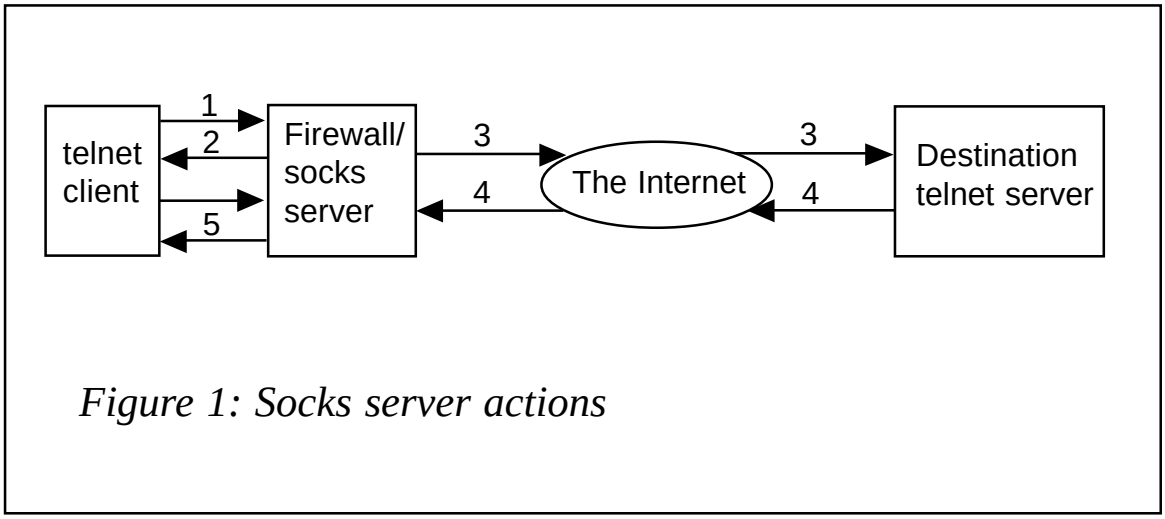

Figure 1: Socks server actions

In Figure 1, the numbers refer to the following:

1 The telnet client makes a normal TCP connection from an ephemeral port to a firewall on port 1080 (a SYN, SYN/ACK, ACK exchange).

2 The client sends a packet encoded with the destination port and IP address it wants to attach to.

3 The firewall makes the connection from its own IP address and ephemeral port to destination port 23 (telnetd) on the destination server.

4 The telnet server answers as usual to the firewall.

5 The firewall takes the packet and passes it back to the client via its connection through port 1080.

Prior to AIX 4.3.3 and above, separate socksified clients were used, such as the shareware products rtelnet and rftp. The problem with

using these was that some client configuration work was needed, and also that the user in question would have to know that they have to use the socksified versions of the clients when communicating across the Internet.

In AIX 4.3.3 and later, a fully socks Version 5 IP stack was introduced. This gives us the ability to configure which destinations require a socksified connection. This is completely transparent to the user, who is able to use the same client program (telnet for example) in order to connect via normal methods and via socks.

The fact that AIX 4.3.3 has a fully socksified stack seems to be little known, although it is documented if you're willing to hunt it out. Before you go ahead and configure the client end of socks, you need a socks 5 server to connect to. This is normally run in conjunction with a firewall, and contains similar rules in terms of authentication (such as allowing the secure network to access 'the world'). From the firewall point of view it also needs multiple rules set up. For example, in order for a telnet connection to be made from your internal network to the outside word, you will need one rule allowing a connection from your secure network to port 1080 on the firewall's secure interface, and another outbound from the firewall's non-secure interface on a port >1024 to port 23 on 'the world'.

In order to configure an AIX 4.3.3 and later client to use socks, you need to create an */etc/socks5c.conf* file. The format of this file is quite simple: you just specify whether any destination hosts or networks should go via a socks server and if so what the IP address of hostname of that socks server is. Alternatively it can be configured so all traffic should go via a socks server unless the destination address falls into your internal network. After the destination address, you can add an optional prefix length to tell socks how many bytes of the address to match.

The following example specifies that all traffic on the internal network (9) doesn't use socks, while everything else will go through the socks server socks.yourdomain.com:

- 9.0.0.0/8 – NONE
- 0.0.0.0/0 – socks.yourdomain.com.

The following example specifies that any traffic for the 160.100 network should use the socks server 9.10.10.10. Traffic to any other destinations will not use socks:

- 160.100.0.0/16 – 9.10.10.10

Lastly, but very importantly, it's necessary to add an environment variable into */etc/environment* specifying the location of the *socks5c.conf* file:

```
SOCKS5C_CONFIG=/etc/socks5c.conf
```

A simple method of checking the connection is to use the **netstat –an** command. Simply check the destination port the client is established with: a socks connection will show port 1080, a TCP client its own port (for example telnetd listens on port 23).

Introducing the Resource Monitoring and Control (RMC) subsystem of AIX 5L

Version 5L of AIX, which was announced earlier this year, introduces completely new components into the features of the AIX operating system. This article will describe the facilities of Resource Monitoring and Control (RMC). RMC has arrived from the universe of IBM SP supporting software; in fact it is part of Reliable Scalable Cluster Technology (RSCT).

The new subsystem enables the system administrator to associate predefined responses with conditions that monitor various system resources. For instance, it is possible to issue an e-mail to the system administrator when a processor goes offline.

Currently RMC has been announced only for the POWER platform.

PACKAGING AND INSTALLATION

The RMC subsystem is installed by default and consists of a single bundle named *rsct.core* containing the filesets (I am omitting message-


```
# lslpp -L '*rsct*'

 Fileset                   Level    State Type Description (Uninstaller)
 ----------------------    -------- ----- ---- ----------------------------
 rsct.basic.hacmp          2.2.0.0  C     F    RSCT Basic Function
                                                 (HACMP/ES) Support)
 rsct.basic.rte            2.2.0.0  C     F    RSCT Basic Function
 rsct.basic.sp             2.2.0.0  C     F    RSCT Basic Function (PSSP
                                                 Support)
 rsct.compat.basic.hacmp   2.2.0.0  C     F    RSCT Event Management Basic
                                                Function (HACMP/ES Support)
 rsct.compat.basic.rte     2.2.0.0  C     F    RSCT Event Management Basic
                                                 Function
 rsct.compat.basic.sp      2.2.0.0  C     F    RSCT Event Management Basic
                                                  Function (PSSP Support)
rsct.compat.clients.hacmp 2.2.0.0  C     F    RSCT Event Management Client
                                                 Function (HACMP/ES Support)
 rsct.compat.clients.rte   2.2.0.0  C     F    RSCT Event Management Client
                                                 Function
 rsct.compat.clients.sp    2.2.0.0  C     F    RSCT Event Management Client
                                                 Function (PSSP Support)
 rsct.core.auditrm         2.2.0.0  C     F    RSCT Audit Log Resource
                                                  Manager
 rsct.core.errm            2.2.0.0  C     F    RSCT Event Response Resource
 rsct.core.fsrm            2.2.0.0  C     F    RSCT File System Resource
                                                 Manager
 rsct.core.gui             2.2.0.0  C     F    RSCT Graphical User
                                                  Interface
 rsct.core.hostrm          2.2.0.0  C     F    RSCT Host Resource Manager
 rsct.core.rmc             2.2.0.0  C     F    RSCT Resource Monitoring and
                                                 Control
 rsct.core.sec             2.2.0.0  C     F    RSCT Security
 rsct.core.sensorrm        2.2.0.0  C     F    RSCT Sensor Resource Manager
 rsct.core.sr              2.2.0.0  C     F    RSCT Registry
 rsct.core.utils           2.2.0.0  C     F    RSCT Utilities
…
 rsct.core.fsrm            2.2.0.0  C     F    RSCT File System Resource
                                                 Manager
 rsct.core.gui             2.2.0.0  C     F    RSCT Graphical User
                                                  Interface
 rsct.core.hostrm          2.2.0.0  C     F    RSCT Host Resource Manager
 rsct.core.rmc             2.2.0.0  C     F    RSCT Resource Monitoring
                                                  and Control
 rsct.core.sec             2.2.0.0  C     F    RSCT Security
 rsct.core.sensorrm        2.2.0.0  C     F    RSCT Sensor Resource Manager
 rsct.core.sr              2.2.0.0  C     F    RSCT Registry
 rsct.core.utils           2.2.0.0  C     F    RSCT Utilities
```

Figure 1: Filesets

only filesets) shown in Figure 1.

All executables and related items are installed into the */usr/sbin/rsct* directory, while the log files and other temporary data are located in */var/ct*. The directory */usr/sbin/rsct/README/* contains various *README* documents that contain the latest product-related information.

The automatic start-up of the RMC subsystem is assured because of the following line in the */etc/inittab* file:

```
# grep rmc /etc/inittab
ctrmc:2:once:/usr/bin/startsrc -s ctrmc > /dev/console 2>&1
```

The subsystem execution can be controlled using SRC commands or its own control command, **/usr/sbin/rsct/bin/rmcctrl**, which is the preferred way to stop and start it. Although the subsystem is implemented as a collection of command line utilities, the best way to configure and control it is to use the Web-based System Manager (WSM). No **smit**-based interface exists at the time of writing.

The RMC subsystem consists of several multithreaded daemons, as shown in the following output:

```
# lssrc -g rsct_rm
Subsystem         Group            PID     Status
 IBM.ERRM         rsct_rm          12394   active
 IBM.AuditRM      rsct_rm          12136   active
 IBM.FSRM         rsct_rm          12650   active

#ps -mo THREAD    -p 12394,12136,12650

IBM.FSRM         rsct_rm          12650   active
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND
COMMAND
    root 12136  4918      - A    0  60  9        *   240001      -   0 /
usr/sbin
/rsct/bin/IBM.AuditRMd
       -     -     -  15483 S    0  60  1        -   418410      -   0 -
       -     -     -  15751 S    0  60  1        -  2400400      -   0 -
       -     -     -  15999 S    0  60  1 7005246c   400410      -   0 -
       -     -     -  16255 S    0  60  1 7005226c   410410      -   0 -
       -     -     -  16513 S    0  60  1 ea002020  8410410      -   0 -
       -     -     -  16771 S    0  60  1 ea0020a0  8410410      -   0 -
       -     -     -  17029 S    0  60  1 ea002120  8c10400      -   0 -
       -     -     -  17287 S    0  60  1 ea0021a0  8c10400      -   0 -
       -     -     -  17545 S    0  60  1 ea002220  8c10400      -   0 -
```

```
    root 12394  4918      - A    Ø  6Ø 1Ø        *   24ØØØ1      -   Ø /
usr/sbin
/rsct/bin/IBM.ERrmd
       -     -     -   4487 S    Ø  6Ø  1 7ØØ6f26c   41Ø41Ø      -   Ø -
       -     -     -   4669 S    Ø  6Ø  1        -   418410      -   Ø -
       -     -     -  12667 S    Ø  6Ø  1 ea0018aØ  8c1Ø4ØØ      -   Ø -
       -     -     -  13443 S    Ø  6Ø  1        -  24ØØ4ØØ      -   Ø -
       -     -     -  13941 S    Ø  6Ø  1 7ØØ6f66c   41Ø41Ø      -   Ø -
       -     -     -  14197 S    Ø  6Ø  1 ea001baØ  8c1Ø4ØØ      -   Ø -
       -     -     -  14965 S    Ø  6Ø  1 ea001d2Ø  841Ø41Ø      -   Ø -
       -     -     -  15223 S    Ø  6Ø  1 7ØØ6e86c   4ØØ41Ø      -   Ø -
       -     -     -  16771 S    Ø  6Ø  1 ea0Ø2Ø1Ø  841Ø41Ø      -   Ø -
       -     -     -  17Ø29 S    Ø  6Ø  1 ea002121Ø  8c1Ø4ØØ      -   Ø -
       -     -     -  17287 S    Ø  6Ø  1 ea0021aØ  8c1Ø4ØØ      -   Ø -
       -     -     -  17545 S    Ø  6Ø  1 ea00222Ø  8c1Ø4ØØ      -   Ø -
    root 12394  4918      - A    Ø  6Ø 1Ø        *   24ØØ1      -   Ø /
usr/sbin
/rsct/bin/IBM.ERrmd
       -     -     -   4487 S    Ø  6Ø  1 7ØØ6f26c   41Ø41Ø      -   Ø -
       -     -     -   4669 S    Ø  6Ø  1        -   418410      -   Ø -
       -     -     -  12667 S    Ø  6Ø  1 ea0018aØ  8c1Ø4ØØ      -   Ø -
       -     -     -  13443 S    Ø  6Ø  1        -  24ØØ4ØØ      -   Ø -
       -     -     -  13941 S    Ø  6Ø  1 7ØØ6f66c   41Ø41Ø      -   Ø -
       -     -     -  14197 S    Ø  6Ø  1 ea001baØ  8c1Ø4ØØ      -   Ø -
       -     -     -  14965 S    Ø  6Ø  1 ea001d2Ø  841Ø41Ø      -   Ø -
       -     -     -  15223 S    Ø  6Ø  1 7ØØ6e86c   4ØØ41Ø      -   Ø -
       -     -     -  27663 Z    Ø  6Ø  1        -   c00001      -   Ø -
    root 1265Ø  4918      - A    Ø  6Ø  7        *   24ØØ1      -   Ø /
usr/sbin
/rsct/bin/IBM.FSrmd
       -     -     -  178Ø5 S    Ø  6Ø  1        -   418410      -   Ø -
       -     -     -  18Ø73 S    Ø  6Ø  1        -  24ØØ4ØØ      -   Ø -
       -     -     -  18577 S    Ø  6Ø  1 ea00242Ø  8c1Ø4ØØ      -   Ø -
       -     -     -  18835 S    Ø  6Ø  1 7ØØ4ea6c   41Ø41Ø      -   Ø -
       -     -     -  19Ø93 S    Ø  6Ø  1 7ØØ4ec6c   4ØØ41Ø      -   Ø -
       -     -     -  19351 S    Ø  6Ø  1 ea0025aØ  841Ø41Ø      -   Ø -
```

RMC CONCEPTS

The two basic notions of conditions and responses form the basis of RMC functionality. To monitor a condition, simply associate one or more responses with it. The system comes preconfigured with 84 conditions and eight responses. The user can use them as supplied, change them, and use them as templates to define his own conditions and responses.

The condition monitors a specific property, such as total percentage

used, in a specific resource class, such as *Host*. You can monitor the condition for one, or more, or all the resources within a monitored property such as Paging Device or Processor utilization. Each condition contains an event expression, which triggers an event, and an optional re-arm expression, which defines an event that will mark the end of an event condition. The event expression is a combination of monitored property, mathematical operators, and some numbers, such as PctTotalPgSpUsed>90 in the case of Paging Space. The re-arm expression is a similar entity, for example, PctTotalPgSpUsed<85.

Responses to various conditions can consist of one or more actions. In order to define an action you choose one of the predefined commands such as *send e-mail*, *log an entry to a file*, or *broadcast a message*; or you can trigger the execution of an arbitrary program or script using the *run program* option. The action can be started by an event, a re-arm event, or both. You can also specify a time window in which an action is active, such as during on-shift on weekdays or always.

The predefined commands are using **notifyevent**, **wallevent**, and **logevent** scripts, located in the */usr/sbin/rsct/bin* directory. These scripts capture events through the Event Response Resource Manager (ERRM) environment variables and notify the user using e-mail, log files, and message broadcasting. It is not advisable to modify the predefined command scripts. However, you can copy these scripts and use them as templates for creating your own scripts to be executed for the *run program* option.

You should note that the **logevent** script uses the **alog** command to log events to the designated log file; therefore, you will have to use the **alog** command in order to read the contents of the files.

When an event expression of a condition is found to be true, an event is triggered, and the ERRM checks all the responses associated with the condition and executes the appropriate event actions. The system then looks for expressions defined for the re-arm expression. Only after the execution of the re-arm event actions will the system start to look for event conditions.

The RMC logs the execution of the actions triggered by event and re-arm events, as well as the success or failure of the commands invoked

by these actions to the audit log. The standard error of the executed command is always recorded; however, the standard output is recorded only if the *redirect command's standard output to audit log* option has been selected for the command in the Action Properties dialog window of WSM. The audit log records can be listed using the **lsaudrec** command or removed from the log file using the **rmaudrec** command. The Audit log Resource Manager (AuditRM) component of the RCM is responsible for the execution of logging activities.

RMC RESOURCES

The system resources monitored by RCM are managed by two daemons – the File System Resource Manager (FSRM) and the Host Resource Manager (HostRM).

The FSRM monitors all local file systems and checks for status online (or offline), the total percentage used, and the number of inodes used in the file system.

The HostRM supports nine different resource classes. The Processor resource class monitors processor utilization by checking idle, user, and kernel, and wait time percentage of processor utilization. The Paging Space resource class checks for Paging Space usage and availability. The Physical Volume class supports monitoring of various utilization parameters of physical disks. The network adapter resource classes (supporting Ethernet, Token Ring, ATM, and FDDI devices) monitor five different options such as error rate.

The Host resource class supports 46 different properties describing various areas of server activity. Among the most important ones are size of system run queue, sizes of various kernel memory pools, and overall utilization of system processors.

The Program resource class monitors the execution states of specific programs. A filter expression, based on a value of a specific field of the **ps** command, can be specified for this resource class. This enables the user to monitor only those programs executing for a particular userid. The predefined condition in this resource class checks whether the **sendmail** daemon is active.

A general configuration change property is supported by all resource classes. Using this property, it is possible to trigger a specific action when the configuration of a device operating under the resource class control changes. The resource classes JFS, PagingDevice, and Processor support the common operational state property.

ENHANCED WSM PLUG-INS

Version 5.1 of AIX 5L supports enhanced RMC related plug-ins. These include:

- Host Overview enhancements.
- Audit log dialog enhancements.
- Conditions plug-in and dialog enhancements.

The Host Overview plug-in provides a convenient summary of a minimal set of vital signs of the system, which are:

- Operating system level
- IP address
- Machine type
- Serial number
- Number of processors
- CPU frequency
- RAM size
- Paging space size
- File systems utilization.

The Host menu of the Host Overview plug-in provides a convenient way to perform critical tasks, such as:

- List top 10 processors
- Delete (Kill) a process
- Expand a Journaled File System

- Increase Paging Space
- Shutdown the server
- Reconnect to RMC system.

The Events plug-in, which displays all the events, re-arm events, and errors that occur during the current WSM session, has been enhanced by inclusion of a new audit plug-in. The audit log dialog can be launched from the Events menu. Using this dialog, it is possible to check for all events and re-arm events that have been recoded by the various monitoring subsystems. Also recorded are the actions that have been taken because of the occurrence of these events.

The Conditions plug-in, which displays a predefined set of standard conditions, has been extended by incorporating the following features:

- Addition of a new Monitored property, showing whether the condition is currently being monitored or not.
- Addition of new icons and menu choices, enabling users to start and stop the monitoring from the Conditions plug-in without going through the monitoring dialog.

COMMAND LINE INTERFACE

In addition to the GUI-based WSM interface, the RMC supports two groups of Command Line-based Interfaces (CLI), which can be utilized by the user in order to create scripts using the subsystem.

The RMC CLI enables the user to manipulate definitions of resources and resource classes. The RMC CLI consists of the following commands:

- **mkrsrc** – defines a new resource.
- **rmrsrc** – removes a defined resource.
- **lsrsrc** – lists (displays) resources or a resource class.
- **lsrsrcde** – lists a resource or resource class definition.
- **chrsrc** – changes the persistent attribute values of a resource or resource class.

- **refrsrc** – refreshes the resources within the specified resource class.
- **lsactdef** – lists (displays) action definitions.

The Event Response Resource Manager (ERRM) command line interface enables the user to control monitoring on the system. The ERRM CLI consists of the following commands:

- **mkcondition** – creates a new condition definition, which can be monitored.
- **rmcondition** – removes a condition.
- **chcondition** – changes any of the attributes of a defined condition.
- **lscondition** – lists information about one or more conditions.
- **mkresponse** – creates a new response definition with one action.
- **rmresponse** – removes a response.
- **chresponse** – adds or deletes the actions of a response or renames a response.
- **lsresponse** – lists information about one or more responses.
- **rmcondresp** – deletes a link between a condition and one or more responses.
- **mkcondresp** – creates a link between a condition and one or more responses.
- **stopcondresp** – stops monitoring a condition that has one or more linked responses.
- **lscondresp** – lists information about a condition and its linked responses, if any.

The following are examples of the responses generated after the execution of some RMC and ERRM commands:

```
#lscondition
Displaying condition information:
Name                                MonitorStatus
"Inetd daemon state"                "Not monitored"
"Sendmail daemon state"             "Not monitored"
```

```
"ATM transmit overflow rate"        "Not monitored"
"ATM transmit error rate"           "Not monitored"
"ATM transmit drop rate"            "Not monitored"
"ATM receive drop rate"             "Not monitored"
"ATM receive error rate"            "Not monitored"
"FDDI transmit overflow rate"       "Not monitored"
"FDDI transmit error rate"          "Not monitored"
"FDDI transmit drop rate"           "Not monitored"
"FDDI receive drop rate"            "Not monitored"
"FDDI receive error rate"           "Not monitored"
"Token ring transmit overflow rate" "Not monitored"
"Token ring transmit error rate"    "Not monitored"
"Token ring receive drop rate"      "Not monitored"
"Token ring receive error rate"     "Not monitored"
"Ethernet transmit overflow rate"   "Not monitored"
"Ethernet transmit error rate"      "Not monitored"
"Ethernet transmit drop rate"       "Not monitored"
"Ethernet receive drop rate"        "Not monitored"

#  startcondresp "/var space used" "Critical notifications" "E-mail root
anytime"
#
lscondresp "/tmp space used"
Displaying condition with response information:

condition-response link 1:
        Condition = "/tmp space used"
        Response  = "E-mail root anytime"
        State     = "Active"
```

SETTING UP AN EFFICIENT MONITORING SYSTEM

The following procedure describes the steps that must be taken in order to set up an efficient SCM-based monitoring system:

1 Check predefined conditions to identify conditions that are relevant to your situation. Use them as provided, customize them, or use them as templates to define your own.

2 Check predefined responses. Customize them to fit your environment and work schedule. For example, the response ‘critical notifications’ has three predefined actions:

- Log events to */tmp/criticalEvents*.
- E-mail the event to the root.

- Broadcast message to all logged-in users any time when an event or re-arm event occurs.

You can modify the response, such as to log events to a different file, page you during non-working hours, and e-mail you only during working hours. With such a set-up, different notification mechanisms can be automatically switched, based on your working schedule.

3 Re-use the responses for conditions. For example, you can customize the three severity responses, *Critical notifications*, *Warning notifications*, and *Informational notifications*, to take actions in response to events of respective severities and associate the responses to conditions depending on severity.

4 Once the monitoring has been set up, your system continues being monitored whether your Web-based System Manager session is running or not. You can use WSM to check the status of the system using the Events plug-in or use the **lsaudrec** command to view the audit log.

REFERENCES

1 *Resource Monitoring and Control Guide and Reference*, SC23-4345

2 *AIX 5L Differences Guide Version 5.1 Edition*, SG24-5765-01

Alex Polak
System Engineer
APS (Israel)

Have you come across any undocumented features in AIX 5L? Why not share your discovery with others? Send your findings to us at any of the addresses shown on page 2.

Tape Manager – part 2

This month we conclude the code for Tape Manager, which controls back-up tapes.

```
  if ( stay_flag==1)
    sprintf(sys_cmd,"%s%s %s","tar -cf
",v_dev_name_n,files_to_be_copied);
  else
    sprintf(sys_cmd,"%s%s %s","tar -cf ",v_dev_name,files_to_be_copied);

  printf("File is being written...\n");
  fflush(stdout);

  sys_rc=system(sys_cmd);
  if (sys_rc!=0) {
    printf("sys_rc=%d\n",sys_rc);
    release_device(0);
    exit(25);
  }

  if ( continue_flag==1 && stay_flag==0 ) {
    exec sql delete from CURRENT_CART
             where KEY=:continue_key;
    if ( sqlca.sqlcode < 0 ) sql_error(24);
  }

  if ( continue_flag==0 && stay_flag==1 ) {
    exec sql select KEY into :dummy
             from CURRENT_CART
             where KEY=:continue_key;
    if ( sqlca.sqlcode < 0 ) sql_error(25);
    if ( sqlca.sqlcode != 100 ) {
      exec sql delete from CURRENT_CART
           where KEY=:continue_key;
      if ( sqlca.sqlcode < 0 ) sql_error(26);
    }
    exec sql insert into CURRENT_CART
        values(:continue_key,:new_serial_no,:v_dev_name,:dev_group_num);
    if ( sqlca.sqlcode < 0 ) sql_error(27);
  }

  exec sql insert into TAPE_FILES
           values(:file_name,:files_to_be_copied,:current_version,
                  :new_serial_no,:seq+1,current
timestamp,:retpd,:share_flag,
                  :tot_size);
```

```
  if ( sqlca.sqlcode < 0 ) sql_error(28);

  release_device(1);

  printf("File %s is successfully loaded into cartridge %d
.\n",file_name,new_serial_no);

}

int main(int argc,char **argv)
{

  int copt,i=0;

  while(1) {
    copt=getopt(argc,argv,"f:v:r:k:nxg:p:cs");
    if (copt==-1) break;
    switch(copt) {
      case 'f' :
              strcpy(file_name,optarg);
              break;
      case 'v' :
              version=atoi(optarg);
              break;
      case 'r' :
              retpd=atoi(optarg);
              break;
      case 'x' :
              strcpy(share_flag,"N");
              break;
      case 'n' :
              new_flag=1;
              break;
      case 'g' :
              dev_group_num=atoi(optarg);
              break;
      case 'p' :
              strcpy(pool_name,optarg);
              break;
      case 'k' :
              strcpy(continue_key,optarg);
              break;
      case 's' :
              stay_flag=1;
              break;
      case 'c' :
              continue_flag=1;
              break;
      otherwise :
              exit(26);
```

```
                break;
      }
    }

    if ( optind < argc ) {
      while ( optind < argc )
        sprintf(files_to_be_copied,"%s
%s",files_to_be_copied,argv[optind++]);
    }
    sprintf(files_to_be_copied,"%s\n",files_to_be_copied);

    exec sql connect to deneme;
    if ( sqlca.sqlcode < 0 ) sql_error(29);

    check_parameters();

    tm_write();

    exit(0);

}
```

TMREAD.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>
#include <string.h>
#include <time.h>

int continue_flag=0;
char v_dev_name_n[22];
int file_name_len;
int random_number;

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
    short nullind;
    short new_serial_no;
    short old_serial_no;
    short seq=0;
    char file_name[100];
    short version=0;
    short current_ver=0;
    short dev_group_num=0;
    short stay_flag=0;
    char v_dev_name[21];
    char v_dev_label[21];
    char continue_key[20];
```

```
EXEC SQL END DECLARE SECTION;

void release_device();

void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  release_device(0);
  exit(1);
}

void release_device(int opt)
{
  int sys_rc;
  char sys_cmd[200];

  exec sql update TAPE_DEVICES set BUSY=0
           where DEV_NAME=:v_dev_name and
                 DEV_GROUP=:dev_group_num;

  exec sql update TAPE_SERIALS set IN_USE='Y'
           where SERIAL_NO=:new_serial_no;

  if ( opt == 0 ) {
    exec sql delete from CURRENT_CART
             where KEY=:continue_key;
  }

  exec sql commit;

  sprintf(sys_cmd,"rm -r %ld",random_number);
  sys_rc=system(sys_cmd);
  if (sys_rc!=0) {
    printf("sys_rc=%d\n",sys_rc);
    exit(2);
  }
}

void get_random_number()
{
  struct timeval tv;
  long msec;
  gettimeofday(&tv, NULL);
  msec=tv.tv_usec;
  srand(msec);
  random_number=rand();
}
```

```
void norew_dev_name()
{
  int lp1;

  strcpy(v_dev_name_n,v_dev_name);
  for ( lp1=0;lp1<strlen(v_dev_name_n);lp1++) {
    if ( v_dev_name_n[lp1]==' ' ) {
      v_dev_name_n[lp1]='.';
      v_dev_name_n[lp1+1]='1';
      v_dev_name_n[lp1+2]='\0';
      break;
    }
  }
}

void tm_read()
{
  FILE *label_file;
  int sys_rc;
  int carts_serial_no;
  char sys_cmd[200];
  char dummy_char;
  int lp1;

  exec sql connect to tmdb;
  if ( sqlca.sqlcode < 0 ) sql_error(1);

  exec sql select CURRENT_VER into :current_ver
           from VERSIONS
           where FILE_NAME=:file_name;
  if ( sqlca.sqlcode < 0 ) sql_error(2);

  exec sql select SERIAL_NO,SEQ_NO into :new_serial_no,:seq
           from TAPE_FILES
           where FILE_NAME=:file_name and
                 VERSION=:current_ver+:version;
  if ( sqlca.sqlcode < 0 ) sql_error(3);
  if ( sqlca.sqlcode==100) {
    printf("Could not find file %s in any cartridge...\n",file_name);
    exit(3);
  }

  exec sql update TAPE_SERIALS set IN_USE='C'
           where SERIAL_NO=:new_serial_no;
  if ( sqlca.sqlcode < 0 ) sql_error(4);

  if ( continue_flag == 0 ) {
    exec sql declare crs1 cursor for
             select DEV_NAME,DEVICE_LABEL
             from TAPE_DEVICES
```

```
          where BUSY=Ø and DEV_GROUP=:dev_group_num and
                DEV_NAME||char(DEV_GROUP)
                  not in ( select DEV_NAME||char(DEV_GROUP)
                           from CURRENT_CART )
          for update of BUSY ;
  if ( sqlca.sqlcode < Ø ) sql_error(5);

  exec sql open crs1;
  if ( sqlca.sqlcode < Ø ) sql_error(6);

  exec sql fetch crs1 into :v_dev_name,:v_dev_label;
  if ( sqlca.sqlcode < Ø ) sql_error(7);

  if ( sqlca.sqlcode==1ØØ) {
    printf("Could not find any available device...\n");
    exit(4);
  }

  exec sql update TAPE_DEVICES set BUSY=1
          where current of crs1;
  if ( sqlca.sqlcode < Ø ) sql_error(8);

  exec sql close crs1;
  if ( sqlca.sqlcode < Ø ) sql_error(9);

  exec sql commit;
  if ( sqlca.sqlcode < Ø ) sql_error(1Ø);

  exec sql select DEVICE_LABEL into :v_dev_label
          from TAPE_DEVICES
          where DEV_NAME=:v_dev_name and
                DEV_GROUP=:dev_group_num;
  if ( sqlca.sqlcode < Ø ) sql_error(11);

  printf("Please use device %s ...\n",v_dev_label);
  norew_dev_name();
}

if ( continue_flag==Ø) {

  get_random_number();
  sprintf(sys_cmd,"mkdir %ld",random_number );

  sys_rc=system(sys_cmd);
  if (sys_rc!=Ø) {
    printf("sys_rc=%d\n",sys_rc);
    exit(5);
  }

  sprintf(sys_cmd,"%ld",random_number );
```

```
    chdir(sys_cmd);

    while (1) {

      printf("Please insert cartridge number %d and press
ENTER...(Seq.no=%d) \n",new_serial_no,seq);
      printf("Press x to quit...\n");
      fflush(stdout);

      dummy_char=getchar();
      if (dummy_char=='x') {
        chdir("..");
        release_device(0);
        exit(6);
      }

      sprintf(sys_cmd,"%s%s%s > /dev/null","mt -f ",v_dev_name,"
rewind");
      sys_rc=system(sys_cmd);
      if (sys_rc!=0) {
        printf("sys_rc=%d\n",sys_rc);
        chdir("..");
        release_device(0);
        exit(7);
      }

      sprintf(sys_cmd,"%s%s%s","tar -xf ",v_dev_name_n," >/dev/null
2>&1");
      printf("Label file is being read...\n");
      fflush(stdout);

      sys_rc=system(sys_cmd);
      if (sys_rc!=0) {
        printf("sys_rc=%d\n",sys_rc);
        chdir("..");
        release_device(0);
        exit(8);
      }

      if ( ( label_file=fopen("./label_file","r") ) == NULL ) {
        printf("There is a problem while reading ./label_file...\n");
        chdir("..");
        release_device(0);
        exit(9);
      }

      fscanf(label_file,"%d",&carts_serial_no);
      fclose(label_file);

      if ( carts_serial_no == new_serial_no ) {
```

```
        break;
      }
      printf("Wrong cartridge is inserted...\n");
    }

    chdir("..");

    if ( seq > 1 ) {
      sprintf(sys_cmd,"mt -f %s fsf %d > /dev/null",v_dev_name_n,seq-1);
      printf("Winding the first empty point of the cartridge...\n");
      fflush(stdout);
      system(sys_cmd);
      if (sys_rc!=0) {
        printf("sys_rc=%d\n",sys_rc);
        release_device(0);
        exit(10);
      }
    }
  }
  else {
  /* check if the new file is on the same cart */
    exec sql select SERIAL_NO,DEV_NAME,DEV_GROUP
             into :old_serial_no,:v_dev_name,:dev_group_num
             from CURRENT_CART
             where KEY=:continue_key;
    if ( sqlca.sqlcode < 0 ) sql_error(12);

    if (sqlca.sqlcode==100) {
      old_serial_no=0;
    }

    if ( old_serial_no == 0 ) {
      printf("You did not use -s flag in the previous operation, you can
not use -c flag...\n");
      release_device(0);
      exit(11);
    }

    if ( old_serial_no!=new_serial_no ) {
      printf("Cartridge number mismatch. previous cart no=%d, current
cart no=%d\n",old_serial_no,new_serial_no);
      release_device(0);
      exit(12);
    }
  }

  norew_dev_name();
  if ( stay_flag==1)
    sprintf(sys_cmd,"%s%s > /dev/null","tar -xf ",v_dev_name_n );
  else
```

```
    sprintf(sys_cmd,"%s%s > /dev/null","tar -xf ",v_dev_name );

  printf("Cartridge is being read...\n");
  fflush(stdout);

  sys_rc=system(sys_cmd);
  if (sys_rc!=Ø) {
    printf("sys_rc=%d\n",sys_rc);
    release_device(Ø);
    exit(13);
  }

  printf("File %s is successfully restored.\n",file_name);

  if ( continue_flag == 1 && stay_flag == Ø ) {
    exec sql delete from CURRENT_CART
             where KEY=:continue_key;
  }

  if ( continue_flag == Ø && stay_flag == 1 ) {
    exec sql insert into CURRENT_CART
        values(:continue_key,:new_serial_no,:v_dev_name,:dev_group_num);
    if ( sqlca.sqlcode < Ø ) sql_error(13);
  }

  release_device(1);

}

int main(int argc,char **argv)
{

  int copt,i=Ø;

  while(1) {
    copt=getopt(argc,argv,"f:v:cg:k:s");
    if (copt==-1) break;
    switch(copt) {
      case 'f' :
              strcpy(file_name,optarg);
              file_name_len=strlen(file_name);
              break;
      case 'v' :
              version=atoi(optarg);
              break;
      case 's' :
              stay_flag=1;
              break;
      case 'c' :
              continue_flag=1;
```

```
                break;
        case 'g' :
                dev_group_num=atoi(optarg);
                break;
        case 'k' :
                strcpy(continue_key,optarg);
                break;
        otherwise :
                exit(14);
                break;
      }
    }

    if ( strcmp(file_name,"")==0) {
      printf("Filename must be entered...\n\n");
      exit(15);
    }

    tm_read();

    exit(0);

}
```

TMLIST.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>
#include <string.h>

int file_name_flag=0;
int exp_date_order_flag=0;
int serial_order_flag=0;

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  short nullind;
  short serial_no;
  short seq_no=0;
  char file_name[100];
  char files_to_be_copied[1000];
  short version=0;
  short ret_pd;
  char cre_date[20];
  char exp_date[20];
  char is_share[2];
  short current_ver=0;
  long v_size_in_kb;
```

```
EXEC SQL END DECLARE SECTION;

void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit(1);
}

void list_by_expdate()
{
  exec sql declare crs2 cursor for
           select FILE_NAME,
                  FILES_TO_BE_COPIED,
                  VERSION,
                  SERIAL_NO,
                  SEQ_NO,
                  CRE_DATE+RET_PD days,
                  SIZE_IN_KB
           from TAPE_FILES
           order by 6 ;
  if ( sqlca.sqlcode < 0 ) sql_error(1);

  exec sql open crs2;
  if ( sqlca.sqlcode < 0 ) sql_error(2);

  printf("       ALL FILES IN THE CARTRIDGES IN ORDER OF EXPIRATION
DATE\n");
  printf("-----------------------------------------\n\n");
  printf("   <------ file name    ver. serial seqno       exp_date
size(KB) \n");
  printf("          <------------------------------ files copied \n");
  printf("-----------------------------------------\n");

  exec sql fetch crs2 into :file_name,
                           :files_to_be_copied,
                           :version,
                           :serial_no,
                           :seq_no,
                           :exp_date,
                           :v_size_in_kb;
  if ( sqlca.sqlcode < 0 ) sql_error(3);

  while (sqlca.sqlcode!=100) {

    printf("%25s %5d  %5d  %5d  %11s
%8ld\n",file_name,version,serial_no,seq_no,exp_date,v_size_in_kb);
    printf("%60s\n",files_to_be_copied);
```

```
    exec sql fetch crs2 into :file_name,
                             :files_to_be_copied,
                             :version,
                             :serial_no,
                             :seq_no,
                             :exp_date,
                             :v_size_in_kb;
    if ( sqlca.sqlcode < 0 ) sql_error(4);
  }
  exec sql close crs2;
  if ( sqlca.sqlcode < 0 ) sql_error(5);
}

void list_by_file_name()
{
  exec sql select CURRENT_VER into :current_ver
           from VERSIONS
           where FILE_NAME=:file_name;
  if ( sqlca.sqlcode < 0 ) sql_error(6);

  if ( sqlca.sqlcode==100)
    current_ver=0;
  else {
    printf("VERSION DEFINITIONS\n");
    printf("-----------\n");
    printf("Current Version Number=%d\n\n",current_ver);
  }

  exec sql declare crs1 cursor for
           select FILES_TO_BE_COPIED,
                  VERSION,
                  SERIAL_NO,
                  SEQ_NO,
                  CRE_DATE,
                  RET_PD,
                  IS_SHARE
           from TAPE_FILES
           where FILE_NAME=:file_name
           order by VERSION;
  if ( sqlca.sqlcode < 0 ) sql_error(7);

  exec sql open crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(8);

  printf("%s FILE IS LOADED IN THE FOLLOWING CARTRIDGES\n",file_name);
  printf("-------------------------------------\n");
  printf("version serial_no seq_no        cre_date       ret_pd is_share
files copied   \n\n");

  exec sql fetch crs1 into :files_to_be_copied,
```

```
                            :version,
                            :serial_no,
                            :seq_no,
                            :cre_date,
                            :ret_pd,
                            :is_share;
  if ( sqlca.sqlcode < 0 ) sql_error(9);

  if ( sqlca.sqlcode==100) exit(2);

  while (sqlca.sqlcode!=100) {

    printf("    %d       %d         %d    %s    %d      %s
%s\n",version,serial_no,seq_no,cre_date,ret_pd,is_share,files_to_be_copied);

    exec sql fetch crs1 into :files_to_be_copied,
                             :version,
                             :serial_no,
                             :seq_no,
                             :cre_date,
                             :ret_pd,
                             :is_share;
    if ( sqlca.sqlcode < 0 ) sql_error(10);

  }
  exec sql close crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(11);
}

void list_by_serial()
{
  exec sql declare crs3 cursor for
           select FILES_TO_BE_COPIED,
                  VERSION,
                  FILE_NAME,
                  SEQ_NO,
                  CRE_DATE,
                  RET_PD,
                  IS_SHARE,
                  SIZE_IN_KB
           from TAPE_FILES
           where SERIAL_NO=:serial_no
           order by SEQ_NO;
  if ( sqlca.sqlcode < 0 ) sql_error(12);

  exec sql open crs3;
  if ( sqlca.sqlcode < 0 ) sql_error(13);

  printf("FOLLOWING FILES ARE LOADED IN CARTRIDGE %d.\n",serial_no);
  printf("————————————————————————————————————\n\n");
```

```
  printf("   <------ file name    ver. seqno       cre_date       ret_pd
size(KB)  \n");
  printf("          <---------------------------- files copied \n");
  printf("-----------------------------------------\n");

  exec sql fetch crs3 into :files_to_be_copied,
                           :version,
                           :file_name,
                           :seq_no,
                           :cre_date,
                           :ret_pd,
                           :is_share,
                           :v_size_in_kb;
  if ( sqlca.sqlcode < 0 ) sql_error(14);

  while (sqlca.sqlcode!=100) {

    printf("%25s %5d  %5d  %11s %5d
%8ld\n",file_name,version,seq_no,cre_date,ret_pd,v_size_in_kb);
    printf("%70s\n",files_to_be_copied);

    exec sql fetch crs3 into :files_to_be_copied,
                             :version,
                             :file_name,
                             :seq_no,
                             :cre_date,
                             :ret_pd,
                             :is_share,
                             :v_size_in_kb;
    if ( sqlca.sqlcode < 0 ) sql_error(15);

  }
  exec sql close crs3;
  if ( sqlca.sqlcode < 0 ) sql_error(16);
}

void tm_list()
{

  exec sql connect to tmdb;
  if ( sqlca.sqlcode < 0 ) sql_error(17);

  if ( exp_date_order_flag==1) {
    list_by_expdate();
  }

  if ( file_name_flag==1) {
    list_by_file_name();
  }

  if ( serial_order_flag==1) {
    list_by_serial();
```

```
  }

}

int main(int argc,char **argv)
{

  int copt,i=0;

  while(1) {
    copt=getopt(argc,argv,"f:s:r");
    if (copt==-1) break;
    switch(copt) {
      case 'f' :
              file_name_flag=1;
              strcpy(file_name,optarg);
              break;
      case 's' :
              serial_order_flag=1;
              serial_no=atoi(optarg);
              break;
      case 'r' :
              exp_date_order_flag=1;
              break;
      otherwise :
              exit(3);
              break;

    }
  }

  tm_list();

  exit(0);

}
```

TMSCRATCH.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>

#include <string.h>

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  short serial_no;
  short seq_no=0;
  char file_name[100];
  char files_to_be_copied[500];
```

```
  short version=0;
  short ret_pd;
  char cre_date[20];
EXEC SQL END DECLARE SECTION;

void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit(1);
}

int main(int argc,char **argv)
{

  FILE *scratched_carts;
  char dummy_char;

  scratched_carts=fopen("./scratched_carts","a");

  exec sql connect to tmdb;
  if ( sqlca.sqlcode < 0 ) sql_error(1);

  exec sql declare crs1 cursor for
           select FILE_NAME,
                  FILES_TO_BE_COPIED,
                  VERSION,
                  SERIAL_NO,
                  SEQ_NO,
                  CRE_DATE,
                  RET_PD
           from TAPE_FILES a
           where SERIAL_NO not in (
                     select distinct(SERIAL_NO) from TAPE_FILES
                     where CRE_DATE + RET_PD days > current timestamp
                                 )
           for update of FILE_NAME;
  if ( sqlca.sqlcode < 0 ) sql_error(2);

  exec sql open crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(3);

  exec sql fetch crs1 into :file_name,
                           :files_to_be_copied,
                           :version,
                           :serial_no,
                           :seq_no,
                           :cre_date,
                           :ret_pd;
  if ( sqlca.sqlcode < 0 ) sql_error(4);

  printf("       SCRATCHED FILES \n");
```

```
  printf("————————————————————————————————————\n\n");
  printf("   <————— file name    ver. serial seqno  cre_date ret_pd\n");
  printf("          <————————————————————— files copied \n");
  printf("————————————————————————————————————\n");

  fprintf(scratched_carts,"       SCRATCHED FILES \n");
  fprintf(scratched_carts,"————————————————————————————————————\n\n");
  fprintf(scratched_carts,"   <————— file name    ver. serial seqno
cre_date  ret_pd\n");
  fprintf(scratched_carts,"          <————————————————————— files
copied \n");
  fprintf(scratched_carts,"————————————————————————————————————\n");

  while ( sqlca.sqlcode!=100)
  {
    exec sql delete from TAPE_FILES where current of crs1;
    if ( sqlca.sqlcode < 0 ) sql_error(5);
    fprintf(scratched_carts,"%d\n",serial_no);
    printf("%\n",serial_no);

    printf("%25s %5d  %5d  %5d  %11s
%5d\n",file_name,version,serial_no,seq_no,cre_date,ret_pd);
    printf("%70s\n",files_to_be_copied);

    fprintf(scratched_carts,"%25s %5d  %5d  %5d  %11s
%5d\n",file_name,version,serial_no,seq_no,cre_date,ret_pd);
    fprintf(scratched_carts,"%70s\n",files_to_be_copied);

    exec sql fetch crs1 into :file_name,
                             :files_to_be_copied,
                             :version,
                             :serial_no,
                             :seq_no,
                             :cre_date,
                             :ret_pd;
    if ( sqlca.sqlcode < 0 ) sql_error(6);

  }
  exec sql close crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(7);

  exec sql commit;
  if ( sqlca.sqlcode < 0 ) sql_error(8);

  exit(0);
}
```

Abdullah Ongul
DBA
Disbank (Turkey)

Local installs – how installp works

INTRODUCTION

If you administer more that two or three AIX servers, then no doubt you have hit the problem of keeping locally-written scripts in sync and up-to-date. Some installations use a **tar** file that is shipped around and un**tar**ed on each system. Some sites simply rely on copying programs after modification. However, IBM provides, as part of AIX, one of the best tools to help you keep multiple AIX systems in sync without having to go looking in file headers for version numbers – **installp**. Only a small amount of understanding is needed to enable you to utilize its powers: this article will help provide that understanding.

But why go to the trouble of creating install images? Won't an aptly named **tar** file do just the same? Well, no, not really. There is extra functionality that **installp** provides. For example, there are two 'exits' allowing you to do some customization both before and after the programs are installed. These exits are simple shell scripts, so you can do just about anything within these scripts. A second good reason is that it does perform some validity checking on the programs it is installing (if you check the appropriate **smit** option or use the *-v* flag on the **installp** command). There are others which I will highlight as appropriate.

This article is not a definitive description of the **installp** command or the install process as a whole. It is intended to give an overview of how an install image is made, provide some scripts to help you make images of your own, and provide pointers as to how to make your systems a little easier to administer. Finally, I will give an example of an install image and how it is made.

THE INSTALL IMAGE

Have you ever watched an install go through and wondered what exactly is going on? Ever seen the message "Adding entries to the ODM" and wondered how it is managed? Well, here is how.

An install image (or .bff file) is simply the product of the **backup** command. You can take any install image, either base fileset or PTF, and look at it by putting it into a temporary directory and executing the command *restore* **–xqdpvf <image name>**. Note: do not do this in the root directory, because it will install the files contained in the image into the appropriate directories as each file within the image is stored as *./<pathname>*, eg *./usr/bin/rm*. It *will* overwrite your current versions on running systems.

You should now have a directory structure within your temporary directory. All the files are 'installed' relative to this directory. Also included are some files down the path *usr/lpp/<image name>/*. The main file is a 'library' file *liblpp.a*. This file contains all the tricks that make the install process what it is. You can extract this file with the **ar** (archive) command.

LIBLPP.A FILES

When the *liblpp.a* file is unarchived, several files will be present in the current directory. All the files normally start *<lpp name>*. The last part of the file name determines the role of the file for the image. Some are compulsory, but most are optional.

Compulsory files:

- al
- size
- inventory.

Optional files include:

- pre_i
- post_i
- copyright
- post_u
- pre_u
- cfg_files

- rm_inv
- fixdata.

Here is a breakdown of the compulsory files and some of the more useful optional files.

al file

The al file is a simple text file containing a list of files within that part of the image (*root/usr/share*). It is the list produced by the **lslpp –f** command, except that each file name or directory is made into a relative path reference (a ‘./’ is added to the front).

size file

The size file is a simple text file listing the directories that will have programs installed into them and a size (in KB) indicating how much space is needed in that directory for the install to complete successfully (unless the *EXTEND file systems if space needed* option in **smit** is checked or the *–X* option is specified for **installp**).

inventory file

The inventory file is another simple text file; however, the format of this file is very important. It is in the **sysck** file format because the information is used by the **sysck** command to validate the installation (the *VERIFY install and check file sizes* **smit** option). This can be very useful. It can help administrators check whether other users have been altering local scripts on-the-fly.

pre_i file

The pre_i file is a shell script that is run prior to the installation of the files within the image. This is useful for adding userids that are to own programs within the image. A similar exit, *pre_u*, exists for removing/rejecting software.

post_i file

The post_i file is another shell script. It is run after the installation of

the files has occurred. This is useful for running configuration programs for the installed files. Some filesets use it to update system files such as **inittab** and **crontab**. For example, if you have several versions of AIX running and one program which is different for each version (eg monitor), I put all versions of monitor in the image (called monitor.4.1 etc) and, using this script, put a link to the appropriate version (depending on the result of the **uname** command). A similar exit, *post_u*, exists for removing/rejecting software.

fixdata file

The fixdata file contains information regarding the problems fixed in the image. It can be handy if a link between problem numbers (ie APAR number) and fix (PTF number) is required. The information is stored in the ODM (*/usr/lib/objrepos/fix*) and is therefore in a fixed format (see example later).

There are 38 files which **installp** looks for (at last count on version 4.3.3). Some are for updating the ODM, some deal with removing/ rejecting the fileset. I have found that a small subset will suffice for most installs and it appears that IBM in its install images uses only a small subset.

STRATEGIES TO TAKE

There are basically two strategies that can be employed when making install images. Firstly, there is the easiest, but not the tidiest. The idea is that each version of your image is a separate entity and does not rely on any that have gone before. So each image contains all the programs, and any customization code contained within the image must cater for being run each time the image is installed (ie checking that a userid exists before adding it etc). This does mean that no history will be kept of each version's install (**lslpp –h**), but it does have the advantage of being easier to produce.

The second method is the one that IBM uses. It involves having a base fileset and a series of delta images, which rely on one or more of the earlier versions. These images require a little more knowledge to produce, but do have the advantage of having an audit trail (lpp

history) and any coding contained needs to be done only once. This article will discuss both mechanisms.

MAKING AN IMAGE (ALL IN ONE)

So, what is needed to make an install image? Well, four things really:

1 *liblpp.a* file.

2 The programs to be installed by this fileset.

3 A directory structure.

4 *lpp_name* file.

I have covered the first two items already. Next is the directory structure.

The programs that make up the image should be placed in a directory that reflects the directory they will finally be installed into, but which is relative to the directory you are working in. So if installing a program */usr/local/bin/test_prog*, in a fileset called *company*, and working in the directory */tmp/install_image*, put it in the directory */tmp/install_image/usr/local/bin*. So a sample directory structure might be:

```
/tmp/install_image/lpp_name
/tmp/install_image/usr/lpp/company/liblpp.a
/tmp/install_image/usr/lpp/company/company.al
/tmp/install_image/usr/lpp/company/company.inventory
/tmp/install_image/usr/lpp/company/company.size
/tmp/install_image/usr/lpp/company/company.copyright
/tmp/install_image/usr/local/bin/test.prog
```

The lpp_name file

The lpp_name file is the one that is used to build the *.toc* file **installp** uses during installs. In fact the contents of this file are copied into the *.toc*, so the format is reasonably important. For images containing everything, the contents will not change (except for the version number).

The first line contains a number (the AIX version number), a letter (usually R), a letter (I), a package name, and a left curly bracket.

The second line contains the image fileset name, version number in full (eg 4.3.3.0), a number, a letter (indicating whether this install requires a reboot to be effective: either *N* or *b*), a letter (usually U), the language (eg en_US), and the title of the fileset.

The third line is normally a left square bracket. Optional lines follow, indicating any filesets and version numbers that need to be installed before or with this one, each one preceded by an asterisk. A percentage sign is on the next line, followed by one or more lines of directories and numbers, which indicate the amount of space required by the files to be installed in the various directories.

Three lines of percentage signs follow. Optional lines are next, stating what has been added to the image (IBM uses this to indicate which APARs are fixed by the code in the image). Finally, the last two lines are the closing, right, square bracket and right curly bracket.

An example of an *lpp_name* file follows:

```
4 R I company {
company.local.base 1.1.0.2 01 N U en_US Companies local scripts LPP
[
%
/usr/local/bin 20
%
%
%
]
}
```

This definition will define a fileset called *company.local.base*, *version 1.1.0.2*, within the package *company*. The image is for AIX version 4, and is an image in its own right (does not depend on version 1.1.0.0). No reboot is required as part of this install, and it uses the *en_US* local information. Files in the image require space in the */usr/local/bin* directory.

Future versions of this image will need only the version number changing (ie 1.1.0.3 etc) and any new directories adding to the space requirements as needed.

MAKING AN IMAGE (PTF STYLE)

To make an image that relies on a base image already having been installed requires a small amount of change from the above *lpp_name* file definition. Also, the directory structure for the image changes slightly.

Below is an example of the same fileset definition except that it relies on the base version (1.1.0.0) having been installed:

```
4 R S company {
company.local.base 1.1.0.2 01 N U en_US Companies local scripts LPP
[
*prereq company.local.base 1.1.0.0
%
/usr/local/bin 20
/usr/lpp/SAVESPACE 20
INSTWORK 20 20
%
%
%
C00001    1 Added script test_script.
]
}
```

The alterations made are:

- On the first line the third character is now *S* not *I*, indicating that this is a PTF not a base fileset
- There is a requirement that the base fileset be installed before this image can be installed. (Note: this is actually not necessary because **installp** will check that the base level is already installed.)
- A line has been added to indicate what was included in the image. This is the abstract from the *fixdata* file in the *liblpp.a* library file.
- Two new entries have been added to the list of directories used by the install process. The *SAVESPACE* entry indicates the amount of work needed when **apply**ing the image as opposed to **commit**ting it. The *INSTWORK* entry indicates the amount of space required during the install. These numbers are added together and appear in the *RESOURCES* section of a **preview** install.

The file structure (assuming */tmp/install_image* is the working directory) would look something like:

```
/tmp/install_image/lpp_name
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/liblpp.a
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/
company.local.base.al
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/
company.local.base.inventory
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/
company.local.base.size
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/
company.local.base.copyright
/tmp/install_image/usr/lpp/company/company.local.base/1.1.0.2/
company.local.base.fixdata
/tmp/install_image/usr/local/bin/test.script
```

The directory structure is based on the following pattern:

```
usr/lpp/<package name>/<fileset name>/<level number>/
```

The install process looks for the *liblpp.a* files in a directory named using the above construct. Without this, **installp** complains about not being able to open the relevant directory.

EXAMPLE

The above is a very brief overview. A full example follows, including a base fileset and a subsequent *PTF* for a package called *company*, using the fileset name *company.local.base*. The base fileset will contain one shell script (xxxx.sh), which is 3135 bytes. As part of the install, the program will be added to **crontab**. (Note: the directory */usr/local/bin* will be created if it doesn't exist.) The PTF will contain one C program, *epoch2date* (from *AIX Update* January 1998, Issue 27, pages 46-47).

Base fileset (company.local.base v1.1.0.0)

.al file:

```
./usr/local/bin/xxxx.sh
```

.copyright file:

```
Copyright Xephon 1999,2001
```

.post_i file:

```
#!/usr/bin/ksh
#
if [[ 'crontab -l | grep -c xxxx' -eq Ø ]]
then
  echo "Ø 12 * * * /usr/local/bin/xxxx.sh" >> /var/spool/cron/crontabs/
root
  crontab /var/spool/cron/crontabs/root
fi
```

.inventory file:

```
/usr/local/bin:
     owner = root
     group = system
     mode = 755
     type = DIRECTORY
     class = inventory,company.local.base
     source =

/usr/local/bin/xxxx:
     owner = root
     group = system
     mode = 555
     type = FILE
     size =3135
     class = inventory,apply,company.local.base
```

.size file:

```
/usr/local/bin 4
```

lpp_name file:

```
4 R I company {
company.local.base 1.1.Ø.Ø Ø1 N U en_US Companies local scripts
[
%
/usr/local/bin 4
%
%
%
]
}
```

Directory structure – all files are relative to the current (working) directory:

```
lpp_name
usr/lpp/company/liblpp.a
```

```
usr/lpp/company/company.local.base.al
usr/lpp/company/company.local.base.inventory
usr/lpp/company/company.local.base.size
usr/lpp/company/company.local.base.copyright
usr/lpp/company/company.local.base.post_i
usr/local/bin/xxxx
```

Command to create *liblpp.a* – run from the base working directory:

```
ar -r -gv usr/lpp/company/liblpp.a usr/lpp/company/company.local.base.al
usr/lpp/company/company.local.base.inventory usr/lpp/company/
company.local.base.size
usr/lpp/company/company.local.base.copyright usr/lpp/company/
company.local.base.post_i
```

Command to create install image (company.local.base.1.1.0.0):

```
cat usr/lpp/company/company.local.base.al > lpp_list
echo "./lpp_name" >> lpp_list
echo "usr/lpp/company/company.local.base/liblpp.a" >> lpp_list
backup -pqvif company.local.base.1.1.0.0 < lpp_list
```

PTF fileset (company.local.base v1.1.0.1)

I will only highlight the changed files required to create a PTF image.

.al file:

```
./usr/local/bin/epoch2date
```

.inventory file:

```
/usr/local/bin/epoch2date:
     owner = root
     group = system
     mode = 555
     type = FILE
     size = 3552
     checksum = "28468     4 "
```

There is no *.post_i* file.

.fixdata file:

```
fix:
     name = C00000I
     abstract = Add program epoch2date
     type = f
```

```
     filesets = "company.local.base:1.1.0.1\n\
"
     symptom = "Added script epoch2date to /usr/local/bin.\n\
"
```

lpp_name file:

```
4 R S company {
company.local.base 1.1.0.1 01 N U en_US Company local scripts
[
*prereq company.local.base.1.1.0.0
%
/usr/local/bin 4
%
%
%
C00001 1 Added script epoch2date
]
}
```

Command to create *liblpp.a:*

```
ar -r -gv usr/lpp/company/company.local.base/1.1.0.1/liblpp.a usr/lpp/
company/company.local.base/1.1.0.1/company.local.base.al usr/lpp/
company/company.local.base/1.1.0.1/company.local.base.fixdata usr/lpp/
company/company.local.base/1.1.0.1/company.local.base.inventory usr/lpp/
company/company.local.base/1.1.0.1/company.local.base.copyright usr/lpp/
company/company.local.base/1.1.0.1/company.local.base.size
```

Command to create image (company.local.base.1.1.0.1):

```
cat usr/lpp/company/company.local.base/1.1.0.1/company.local.base.al >
lpp_list
echo "usr/lpp/company/company.local.base/1.1.0.1/liblpp.a" >> lpp_list
echo "./lpp_name" >> lpp_list
backup -pqvif company.local.base.1.1.0.1 < lpp_list
```

WRAP-UP

You should now have a good understanding of how the install process works and how powerful **installp** is. There is of course scope for enhancing the above scripts/commands, for example by writing a script for producing the *lpp_name* file. With a little thought, it can be used to set up new systems very quickly with local customization.

The above description works fine for AIX Version 4; I have not tried it on Version 5. I cannot guarantee that any of the information will be

correct after future fixes are applied to *bos.rte.install* because it is based on my understanding of the install process at the current time. However, I do use the above format and I have had no problems. My advice is to have a look at IBM's own images: if it works for them, it should work for you.

Phil Pollard
Unix and TSM System Administrator (UK)

November 1998 – November 2001 index

Items below are references to articles that have appeared in *AIX Update* since Issue 37, November 1998. References show the issue number followed by the page number(s). Back-issues of *AIX Update* are available back to issue 37 (November 1998). See page 2 for details.

Call for papers

Why not share your expertise and earn some financial reward at the same time? *AIX Update* is looking to swell the number of contributors who send in technical articles, hints and tips, and utility programs, etc. We would also be interested in articles about performance and tuning. If you have an idea for an article, or you would like a copy of our *Notes for Contributors*, contact the editor, Trevor Eddolls, at any of the addresses shown on page 2.

AIX news

IBM has announced Version 2.0 of its AIXlink/X.25, enhanced to take advantage of the 64-bit kernel scalability offered in AIX 5L Version 5.1. The new version is 64-bit enabled and can be executed in a 32-bit environment and a 64-bit environment on POWER platforms running AIX 5L Version 5.1.

Version 2.0 enhances Network Provider Interface support by supplying CCITT cause and diagnostic codes for X.25 to the user's NPI applications, but you need to recompile the X.25 applications first. Otherwise existing applications can still continue to run on this new version without any modification. The new version supports six existing adapters: two-port Multiprotocol PCI Adapter (supports 64-bit), IBM ARTIC960Hx 4-port Selectable PCI adapter, IBM CoProcessor/1 Adapter for ISA, IBM ARTIC960 Microchannel Adapter, ARCTIC Portmaster Adapter/A for Microchannel, and IBM CoProcessor/2 Adapter for Microchannel.

It supports CCITT 1980, 1984, and 1988 X.25.

For further information contact your local IBM representative.
Web address: http://www.ibm.com.

* * *

IBM Global Services has announced Web-delivered High Availability Services, a set of services that provide Web-based access to information and tools, aimed at improving the availability of pSeries or RS/6000 servers and associated main storage, DASD, attached peripheral devices, and AIX.

At its heart is a Server Availability Self Assessment, a Web-based self-assessment of system availability that includes various process support modules covering change management, problem management, availability management, performance and capacity management, and back-up and recovery management.

For further information contact your local IBM representative.
Web address: http://www-1.ibm.com/services/its/us.

* * *

IBM has launched its rack-mounted eight-way p660 Model 6M1 Unix server, powered by 750MHz RS64 IV processors based on copper and silicon-on-insulator technologies and featuring self-managing and self-healing tools and Capacity Upgrade on Demand capabilities.

Self-healing and self-managing technologies include a built-in service processor for monitoring system operation and taking preventive or corrective action. Dynamic Processor Deallocation, working with AIX, can reassign tasks from a potentially failing processor.

The system also features Chipkill used in mainframes, which eliminates most memory-based system failures and is said to be 100 times more effective than ECCtechniques.

For further information contact your local IBM representative.
URL: http://www.ibm.com/eserver.