Finding and Collecting Availability Measurement Data

January 2002

Mike Bonett
IBM Corporation, Enterprise Systems Management Technology Support
Advanced Technical Support, Gaithersburg, MD
bonett@us.ibm.com

This document describes various data sources, provided by operating system platforms and network protocols, that contain availability information for workloads and components. It also identifies monitoring techniques and products that can also produce and/or report availability measurement data. The intent is to eliminate or reduce manual efforts in gathering information on and reporting the availability of systems, networks, and end-to-end applications.

Preface

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis **without any warranty either expressed or implied.** The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program can be used instead.

The information in this document concerning non-IBM products was obtained from the suppliers of those products or from their published announcements. IBM has not tested these products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

The information in this publication is not intended as the specification of any programming interfaces.

Questions or comments about this publication should be sent via via the Internet to **bonett@us.ibm.com**.

Trademarks

The following are trademarks and registered trademarks of the IBM Corporation or Tivoli Systems:

- ACF/VTAM™
- AIX®
- AS/400 ®
- CICS ™
- DB2 ®
- IBM ®
- IMS/ESA ®
- NetView ®
- OS/2 ®
- OS/390 ™
- PS/2 ®
- RS/6000 ®
- S/390 ™
- System/390 ™
- Tivoli ™
- VTAM ™
- z/OS ™

The following are trademarks and registered trademarks of the respective companies:

- Netware ™ (Novell, Inc.)
- Omegamon ™ (Candle Corporation)
- Omegaview ™ (Candle Corporation)
- Windows ™(Microsoft Corporation)
- Windows 2000 ™(Microsoft Corporation)
- Windows NT ™(MicrosoftCorporation)

Any other company or product names are trademarks or registered trademarks of that respective company.

Acknowledgements

Many thanks to the following individuals who, over the years, have provided valuable experience, information, feedback and sanity checks related to the contents of this document:

- John Bishop
- Bob Campenni
- Linda Cook
- Bob Gelinas
- Allen Gilbert
- Pete Gordon
- Randy Greene
- Rich Grimaldi
- Carl Kindstedt
- Eric Klein
- Kevin Miller
- Mark Nixon
- Doug Orlando
- Dave Petersen
- M. Watanabe

Introduction

Availability Management is the process of ensuring that all components (hardware, software, etc.) that are supporting essential business applications and processes are active and in a state to allow:

- users to access the applications.
- applications to access and manipulate information on behalf of the users.

This document focuses on the **measurement** aspect of availability management. It identifies data sources - either provided from system or network components, or created by using specific monitoring techniques - from which information can be captured and used to determine the availability of components in the information technology (I/T) environment. Complex applications require multiple application, network, and database platforms to work to together to provide proper application functions. Multiple sources of data will be required to determine the true availability of these applications, and the information from these sources will have to be gathered together and related in the proper manner.

The process and technical aspects of identifying and collecting availability data in an efficient manner will be covered. These activities are a subset of the overall process of availability management. Other tasks must also be in place to carry out full availability management, so that effective improvements can be evaluated and implemented. The details of those other activities are beyond the scope of this document; however, as the saying goes, “if you cannot measure it, you cannot manage it”. Measuring availability is critical to the overall availability management process, and this document will provide guidelines and methods for effectively finding and collecting data so that availability can be measured.

The Environment

Business systems, or applications, that are supported by the (I/T) infrastructure are increasingly complex. No longer are they centralized on a single physical platform, and accessed through just one or two connecting devices. Today, the components that support applications:

- Span multiple hardware platforms
- Are connected using multiple networking devices and protocols
- Use multiple types of middleware (databases, message queues, Web Servers, “legacy” transaction systems, and so forth)

At the same time that these applications are using more infrastructure elements, their importance to the business is increasing. Application downtime, or outages, can mean the loss of thousands, or millions, of dollars per hour. Outages also damage business reputation, can result in government or legal action, and generate unwanted publicity. Managing application availability is required to reduce or eliminate these exposures.

Availability Management consists of many tasks. These include:

- Determining the type and level of availability that must be provided.
- Designing the infrastructure and management processes to support availability.
- Monitoring and measuring availability to determine real time status and long term trends.
- Talking actions to correct exposures, or improve the current level availability.

Measuring availability is required for many reasons. Once application service levels - which include when the application must be available for use - are established, there must be a way to measure the actual service level being achieved. If outages occur, their location and length must be known to determine the impact, and to start investigating methods to reduce or eliminate the outage from reoccurring.

Because applications can span the entire I/T infrastructure, availability measurements must be taken all across that infrastructure. If only one or 2 components are measured, this information will not reflect the true state of the application. If only the end user perspective is measured, it will not reflect the reasons end users are encountering problems. Availability measurements must be “end-to-end” - they must account for the following “generic” application structure:

- Users use applications to manipulate data.
- Connections must exist:
  - Between the users and the application.
  - Between the application and the data.
- The entire path must be available to achieve end-to-end application availability.
- Identifying where availability is being affected must occur to begin improving availability.

This paper covers methods of finding and collecting availability measurement data across the I/T infrastructure. It highlights a subset of the end-to-end availability steps that are fully documented in the white paper “Measuring End-to-End Availability: How To Get Started” (available at http://www.ibm.com/support/techdocs, or directly from the author). The steps are:

1. Creating a generic model to assist in identify the end-to-end components that support an application.
2. Determining the relationships among the components that will affect how their measurement data will be used.
3. Identifying and selecting the sources of availability data for the components.
4. Merging and analyzing the data to derive end-to-end availability measurements.
5. Identifying the location and impact of outages in the end-to-end path.

This paper covers step 3 in detail. It identifies the availability data sources to investigate for components. The applicable sources for a particular component will vary, but they will fall into two categories:

1. ***Event sources***, which are produced by operating systems, network protocols, and systems management software.
2. ***Monitoring techniques***, which can be written using scripting languages, or implemented using operating system or system management software functions.

While manual availability measurements can (and are still being) used, there are better reasons for using these the data sources:

- The number of components that will have to be measured will overwhelm manual collection efforts.
- The accuracy of the measurements will come into question. It is not unusual for these measurements to be based on perception or guesses, instead of hard facts.

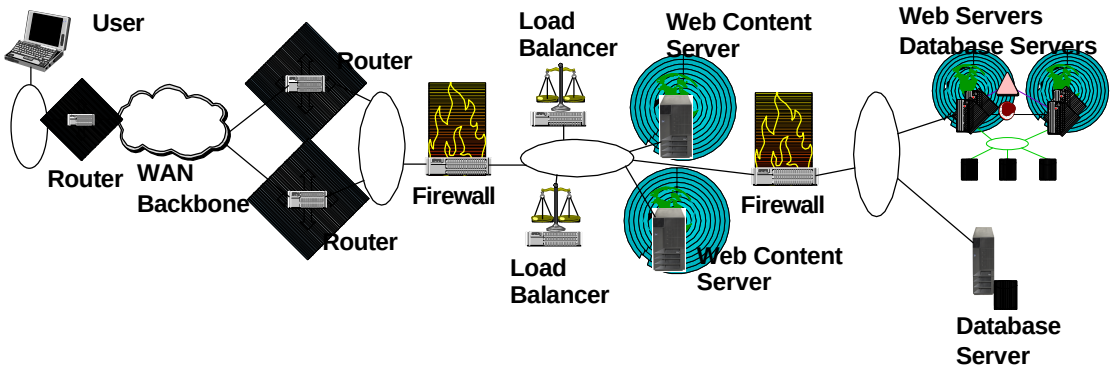
All of the data sources identified in this paper can be accessed using automated techniques. For purposes of this paper, “automation” means any operating system, network, or product function that can be triggered based on a timer or event, can access data or issue a command, and can manipulate the accessed data or command response. For each data source the paper will describe the role automation can play to efficiently collect the desired information.

Getting Started

Finding and collecting availability data cannot begin until the following questions are answered:

- What components do I measure?
- For each component, what data has to be collected?

Consider the following I/T infrastructure. It supports an application with multiple middleware components, and is spread across S/390, LAN, WAN, and distributed environments:



This type of application structure encompasses

- Distributed application logic, residing in both the S/390 and distributed application platform environments
- Distributed data, residing in multiple locations
- Networking and middleware protocols support the information flow across the components

If all hardware and software components identified in this picture are measured, there are at least 30 elements to consider. That is a difficult starting point, especially when trying to establish relationships among the different components, and where to collect data from for each component. Attempting to measure a large number of components without a structured approach can easily lead to collecting lots of meaningless data that does not show the impact beyond just an individual component.

It is much easier to start with a subset of components, and then expand as needed. The best way to identify a subset of components is to start with a model. This is useful for several reasons:

- A model can be applied consistently to multiple application environments.
- A model ensures that all of the key components of an application are identified.
- A model allows for grouping of components; these groups can sometimes be treated as a single component, which helps simplify the data collection and measurement process.

More information on using a model when measuring availability is documented in the ***Measuring End-to-End Availability: How To Get Started*** white paper. The model in that paper defines the following component categories:

- User Platform (components that support the users as part of the application flow)
- User-Application Path (components that connect the users to the Application Platform(s))
- Application Platform (hardware and operating systems that support Application Subsystems and Application Logic required by the application)
- Application Subsystem (middleware that provides services used by the application logic)
- Application (the application logic itself)
- Application-Data Path (components connecting the Application Platform(s) to the application data)
- Data Platform (physical and logical components that provide data access for the application)

Experience has shown that it is best to start out with between three and 10 components types, that are located across the key infrastructure areas of an application - the user location, application logic, data, and connections.

Once the components are identified, measurement information must be collected. This is an area where starting simple will make the effort much easier. Only four data elements are needed for each component. A data source for a component must provide all of them:

1. A component identifier that uniquely identifies this component. This can be a name based on a protocol (e.g. TCP/IP host name or IP address), or a name based on a configuration definition.
2. The component status that is being reported. This can be anything that, for measurement purposes, can be mapped to a "UP" or "DOWN" state for the component. For example, A data source can report that a performance threshold has been exceeded. The result is very poor throughput or response time for the application due to the components' performance. . This can be mapped to a "DOWN" state, if so desired.
3. The date and time that the reported status occurred. These may have to be normalized if the data sources being used are located in different time zones.
4. The status change identifier. This is the specific event or monitoring activity that reported the status change. It can be an event ID, message ID, event source, name of the monitoring software, etc.

From these data elements availability measurements for a component can be derived. For example, the following table shows data elements for 3 different components:

Component ID	**Status**	**Date/Time (YYYY-MM-DD-HH:MM)**	**Status Identifier**
S390_Address_Space_1	Up	2001-04-10-08:00	IEF403I
Router_Interface_1	Down	2001-04-10-12:30	PING
UNIX_Database_1	Down	2001-04-10-13:00	syslog
Router_Interface_1	Up	2001-04-10-15:00	PING

UNIX_Database_1	Up	2001-04-10-16:30	syslog
S390_Address_Space_1	Down	2001-04-10-22:00	IEF450I

The availability of each component for that day can be calculated:

- **S390_Address_Space_1**: SYSLOG messages indicated its status. The component was up for 14 hours (8 AM - 10 PM) and down for 10 hours (midnight- 8AM and 10PM-midnight).
- **Router_Interface_1**: The IP address was PINGed to determine its status. The component was up for 21.5 hours (midnight - 12:30 PM and 3PM-midnight) and down for 2.5 hours (12:30PM-3PM).
- **UNIX_Database_1**: messages in the UNIX syslog indicated its status. The component was up for 20.5 hours (midnight - 1 PM and 4:30 PM - midnight) and down for 3.5 hours (1 PM - 4:30 PM).

A given component, can have multiple sources for this data. The more protocols (system or network) a component participates in, the more different data sources will have to be considered. The component will be represented in each data source. A component that:
- Is defined as a network gateway
- Is directly attached to an operating system platform
- Communicates to components using both SNA and TCP/IP protocol
- Is connected to multiple LAN segments

can be represented in various data sources as:
- An operating system I/O device
- One or more LAN MAC addresses
- An SNA resource
- A TCP/IP node

There will be redundant information when using multiple data sources. Correlating and filtering the data using manual methods is a difficult, if not impossible, task. Automated techniques can be applied to accomplish this. Automation can either identify and discard redundant information, or correlate information for a component from multiple sources - to provide an accurate view of component status.

In general, automation is necessary, once the components and relevant data sources have been identified, to obtain data from the event sources, analyze it, and produce availability measurements. The benefits of using automation include:

- Implementing policies for monitoring and collecting data for specific components can be done in a consistent fashion.
- Human errors can be reduced or eliminated from the availability data gathering process.
- Automation can interface (either directly or by integration with other products) to all points of the application infrastructure.
- Data collection, filtering, and formatting for input to the availability measurement process can be simplified.

Using automation in this process increases the accuracy of the data, while reducing the complexity and effort to collect and analyze the data.

Data Sources Overview

Component availability data will be found in two sets of data sources:

1. **Event Sources**

   Event Sources are where events are placed from operating systems, networking protocols, and management products. Functions that detect component status changes write information to the event source, where it can be extracted for various uses. The event sources that are commonly used are:

   a. Logs
   b. Messages
   c. SNA alerts
   d. TCP/IP traps
   e. Event management products

   These event sources have common characteristics:

   - Some level of programming will be required to extract the desired events
   - The desired events can be captured in real time (when they are written), or extracted later, post-processing. Deciding on when to capture the events depends on the real time availability reporting requirement. If current, up to the minute reports are needed, then real time capture will have to be done. The disadvantage is that this requires more of a programming and processing effort. If daily, weekly, or monthly periodic reports are needed, it will be easier to extract the data on a daily basis.
   - Filtering to obtain the desired events is required. These event sources contain many, many events, and only a small subset of them are needed for availability measurements. Determining which specific events to capture will require the use of event source documentation, component documentation, or working with component experts with knowledge of events related to the components. This can be to most time consuming task in the process.

2. **Monitoring Techniques**

   Event sources will not always contain the desired data. Availability information made be needed in real time, and the event source for a component may not provide that. Or, the event source may contain events indicating when the component status changed to “down”, but does not contain events showing when the component status changed to “up”. For these and other situations, a monitoring technique, which checks the component periodically, can provide the desired information. The possible techniques are:

   a. Heartbeat routines

b. PING commands
c. Remote command execution
d. User simulation
e. Custom monitoring agents

Any one of these monitoring techniques can be used to capture the availability status of a component; each returns a different type or level of information, which will be covered in more detail later in this paper. These techniques can be implemented using standalone programs, but this is not necessary if performance monitoring automation of the component state is already occurring. The necessary information may already be available from these sources, in which case it is just a matter of extracting what is needed.

The monitoring techniques have common characteristics:

a. They are invoked and scheduled using automation. This can be the automation function inherent in an operating system, or an automation function in a product. The monitoring can be triggered on a regular schedule (e.g. Every 5 minutes), or triggered by an event. For example, an event indicating a component status of “down” can be detected by automation and trigger a monitoring technique that monitors the state of the component.
b. The monitoring response must be captured for validation. The response is normally text or a return code. The automation function that invoked the monitoring technique must be able to capture this response and determine the component state based on the response.
c. There is a tradeoff between the monitoring frequency and monitoring overhead. The shorter the monitoring period, the more overhead is incurred. Monitoring a component once a minute uses more system and bandwidth resources than monitoring a component once every 15 minutes. However, this may be necessary when real time information on a component is required. The size of the data used in monitoring technique can be a minimal amount, to lessen the overhead.

There is no one single source or monitoring technique that is “best”. The benefits and drawbacks of each one must be understood, to determine how well they fit within an environment.

Event Logging Facilities

Most operating systems and application subsystems log performance and status events to provide an audit and accounting trail of resource usage. Some of these events provide information on the availability of the operating system, subsystems, applications, and attached components.

This section gives an overview of the event logging functions that exist in the z/OS and OS/390, UNIX (including LINUX), Windows NT/2000, OS/400, Netware, and OS/2 environments. Examples of application logs will also be given. Examples of the data from logs that can be used are given where possible. The latest documentation on the operating system, or the application that uses the logging facilities, should be consulted for the most up to date information.

The following considerations apply when obtaining data from the logging facilities:

- Some platforms maintain separate performance/event and message logs. Availability data can be found in both.
- The information logged may not always be in "text file" format. That is, the information may be recorded in a binary record structure instead of text messages. The operating system platforms provide either documentation of the record structure, or functions to read the binary information and store it as a text file, or both.
- The operating system or application subsystem may only log errors. While this can indicate a workload or component outage, no information is logged that identifies when the workload or component became available. Additional software functions (provided by the application, a resource monitor, resource manager, or developed by the installation) may be required to get the "end of outage" information into the log.
- The logging facilities provide identification of availability status "after the fact". The log information is usually provided after the event has occurred and not in real time. Some platforms provide functions that allow the capturing of information as it is written to the log. In general, if real time information capture is desired, some of the other sources described in this book in other chapters will be easier to use.
- Subsystems and applications can contain their own logging functions and store performance/event information separate from the platform logging information. This can be useful to determine the availability of resources "internal" to the subsystem (transactions, data files, etc.).

Any additional considerations specific to a particular operating system will be identified in its sections.

OS/390 and z/OS

OS/390 and z/OS logs information to the SYSLOG/OPERLOG and Systems Management Facilities (SMF) records; both contain availability related data for OS/390 address spaces and components.

SYSLOG/OPERLOG

The SYSLOG contains text messages issued from both operating system functions and applications running on the platform that choose to use the SYSLOG facility. The OPERLOG can be used, in a parallel sysplex environment, to merge the SYSLOGs from the images in the sysplex into a single log stream. (the OPERLOG is in binary format, but the information can be converted to SYSLOG text format using a utility).

The SYSLOG contains message events that indicate when resources running on, or attached to, the platform changed state. It is normally offloaded to a file when it is full. This file can be processed by a user program to extract the desired information. The **Messages** section of this white paper provides additional detail about messages and the information they contain.

Systems Management Facilities (SMF) Records

OS/390 systems produce SMF records to record system and workload related information that can be used for a variety of purposes. Many installation use products or locally written programs to analyze data in SMF records for billing, performance, auditing, etc.

Many different SMF record types can be produced; parameters in SYS1.PARMLIB member SMFPRMxx control which ones are actually created by the operating system or subsystems. Several of these records indicate the availability status of components such as:

- Address spaces
- Jobs
- JES components
- Attached I/O devices
- SNA network sessions

Each SMF record has a record type; in some cases there are multiple subtypes for a record type. The following table<> <>shows some of SMF record types that can be used as a source of data to indicate the availability status of a component.

Record Type	Component(s)	State	Info	Comments
0(X'00')	OS/390 or z/OS operating system	Available	Indicates date/time of operating system IPL	By obtaining date/time tha the record preceding the IPL record was written (from the SMFxDTE and SMFxTME fields), the time the system incurred a outage can be estimated and the outage length can be calculated.
9(X'09')	I/O device	Available	Indicates date/time device was varied online	This record is written as the result of a VARY ONLINE operator command.
11(X'0B')	I/O device	Unavailable	Indicates date/time device was varied offline	This record is written as the result of a VARY OFFLINE operator command.
30(X'1E')	Address space	Available or unavailable	Contains address space start/termination information (NOTE: consolidates information found in SMF record types 4 and 5; type 30 records are recommended to be used for this data)	Subtype 1 can be used to obtain the time an address space started. Subtype 5 can be used to determine when the job ended (either normal or abnormal termination). Note: Address space start/end times can also be obtained from OS/390 messages IEF403I,$HASP373, IEF404I,$HASP395,and IEF450I.

Record Type	Component(s)	State	Info	Comments
37(X'25')	SNA defined components	Available or unavailable	Contains SNA events and alerts	This record contains NetView hardware monitor events, including alerts, that have passed through the recording filte and are stored in the hardware monitor database. For details on alerts see the "SNA Alerts section of this document.

Record Type	Component(s)	State	Info	Comments
39(X'27')	SNA sessions (LU-LU, SSCP-PU, SSCP-LU, SSCP-SSCP)	Available or unavailable	indicates SNA session start/end times	This record is created by the session monitor component of NetView and will be logged to SMF with the proper NetView definitions. • Subtype 2 can be used to obtain session end time • Subtype 3 can be used to obtain session start time • Subtype 5 can be used to obtain session start and end times. This record is created when a session ends before NetView can write the subtype 3 (session start) record. The record will also provide the primary and secondary SNA information for the session - resource name, resource PU, PU subarea, and domain. Note: The Tivoli Decision Support for OS/390 products uses these records to produce session availability reports.
43(X'2B')	JES2 or JES3 address space	Available	Contains date/time JES2 or JES3 was started	

Record Type	Component(s)	State	Info	Comments
45(X'2D')	JES2 or JES3 address space	Unavailable	Contains date/time JES2 was withdrawn or JES3 was stopped	This record may not always be written; it depends on how JES ended. For some abnorma terminations the JES address space may not be able to write this record.
47(X'2F')	JES2 BSC lines or JES3 BSC/SNA lines	Available	Contains date/time line was started/signed on/logged	This record is created as the result of operator commands to start the line
48(X'30)'	JES2 BSC lines or JES3 BSC/SNA lines	Unavailable.	Contains date/time line was stopped/sign off/logged off	This record is created as the result of operator commands to stop the line
52(X'34')	JES2 SNA lines	Available	Contains date/time line was started/signed on/logged on	This record is created as the result of operator commands to start the line
53(X'35')	JES2 SNA lines	Unavailable	Contains date/time line was stopped/signed off/logged off	This record is created as the result of operator commands to stop the line

For details on SMF record formats and producing and saving SMF records, refer to the publication ***Systems Management Facilities (SMF)***, ***GC28-1628***.

Getting data from SMF

To use SMF as a source of component availability data, the following must be considered:

- The record types that contain status information must be included in the appropriate SMFPRMxx member of SYS1.PARMLIB that is active, or else they will not be recorded. installation uses
- SMF records are most easily processed after they are dumped from the SMF dataset(s) to a sequential dataset. However, real time (as the record is created) access to the data can be obtained by customizing SMF exit IEFU83, which receives control before each record is written to the SMF data set.
- Once dumped to a sequential file, the SMF records can be processed by any programming language. The record layouts and fields are described in the SMF manual. For example, an analysis

program to determine and validate what record types are being written and their contents can be used prior to implementing a data gathering process.
- Macros are provided to allow other products to create SMF record types and write them to the dataset; these records may also be sources of data.
- SMF normally runs whenever the operating system is running. Automation is recommended for offloading the active SMF dataset when it is full, to avoid the loss of records. When this is done, the only time records are not written are during an operating system outage. In this manner SMF also acts as a "heartbeat" to verify if the system was up at a particular time, or to determine the true length of a system outage.

z/VM and VM/ESA

VM logs information via the Programmable Operator (PROP) and the MONITOR functions.

Programmable Operator (PROP)

The PROP virtual machine is normally set up to receive VM system messages. These messages can be logged to a CMS file in plain text, which can them be processed by user programs. The n **VM Messages** section of this document provides additional detail about PROP, VM messages, and examples of message contents.

MONITOR

The MONITOR function generates records that contain information about the performance and status of the VM environment. The MONITOR command is used to control which type of records are recorded and the frequency. The records are stored in a saved segment; virtual machines can connect to the CP *MONITOR System Service to retrieve and process the records.

The records are stored in a binary structured format; the layout of each record is documented in the MONITOR LIST1403 file supplied with VM/ESA and z/VM. The records are grouped into "domains" that correspond to areas of system interest for which performance or event data is collected. The following table summarizes the key monitor records that can be used as a source of data to indicate the availability status of a component.

VM monitor records				
Domain	**Record Number (NAME)**	**Component**	**State**	**Description**
4 (USER)	1 (MRUSELON)	Virtual Machine	Available	Written when a virtual machine is logged on.
4 (USER)	2 (MRUSELOF)	Virtual Machine	Unavailable	Written when a virtual machine is logged off.
6 (I/O)	1 (MRIODVON)	Attached I/O device	Available	Written when an I/O device is varied online.
6 (I/O)	2 (MRIODVOF)	Attached I/O device	Unavailable	Written when an I/O device is varied offline.
6 (I/O)	5 (MRIODATD)	I/O device	Available	Written when an I/O device is attached to a virtual machine.
6 (I/O)	6 (MRIODTDD)	Virtual Machine	unavailable	Written when an I/O device is detached from a virtual machine.

For details on CP Monitor and the Monitor records, refer to the publications ***VM/ESA: Performance*** (SC24-5642) or ***z/VM: Performance*** *(SC24-5292).*

OS/400

The OS/400 history log (QHST) logs system information (system, subsystem, job information, device status) and messages from the QHST message queue. Information sent to the queue are written by the system to the current log version physical file. The log is stored in a database file. If the current log file fills up, a new one is created and becomes the current log file for recording information.

The records in the log file have a structured format with three main sections:

- System date and time
- Record number (a 2-byte field).
- Message data

User programs can process the QHST log files to extract data that indicates the availability of the system, jobs, and I/O devices. The **OS/400 Messages** section of this white paper provides additional detail about OS/400 messages and the types of availability information they contain.

Note:
For details on the OS/400 history log, refer to the ***AS/400 Workload Management Guide (SC21-8078).***

UNIX and LINUX

Variations of the UNIX operating system run on many different operating system platforms. The common ones in the market include:
w AIX (IBM)
w Solaris (Sun Microsystems)
w HP-UX (Hewlett-Packard)

On the z/OS and OS/390 operating systems, UNIX System Services (USS) is available for running UNIX applications under these operating systems.

Linux, a popular derivative of the UNIX operating system, comes in many distributions. The common ones used today include:
- Red Hat
- SuSE
- Caldera
- TurboLinux

Each variation has added extensions to the “base” UNIX operating system functions for optimized execution on the respective hardware platforms, and to provide unique capabilities in the market. This paper will not discuss all of the extensions for all of the variations. The **AIX Messages** section covers some of the enhanced functions in AIX that can provide availability information.

All versions of UNIX and LINUX support a **syslog** function. The syslog is where system messages on the status of hardware and software resources are sent. Information written to the syslog can be directed to any UNIX output device, such as:
- A physical terminal
- A file
- A printer
- The null device (/dev/null) - i.e., it is discarded

Event information is written to the syslog in plain text with no "formal" structure (other than a date and time stamp). The actual content and structure are determined by the operating system function or application that is writing to the syslog. Sometimes this information indicates the availability status of a component. The syslog must be captured to a file for processing to extract that information..

It is also possible, through the use of UNIX commands such as **tail**, to capture information as it is written to the syslog. This is done if the information has to be sent to a process that requires the information in real time.

Documentation for syslog messages is scattered, and usually found among the application documentation. Those wishing to use the syslog as a source of availability data will have to analyze, for the desired platforms, the type and content of messages being written to the syslog, by both the operating system and applications.

Windows NT/2000

Windows NT and Windows 2000 platforms contain a central event logging function to record status information from the operating system and applications. These events can include the status of attached devices, applications, and network connections. They are recorded in three files - the system log, application log, and security log.

The event log contain records that are in a structured binary format. These records contain the following information:

Date	The date the event occurred.
Time	The time (local) the event occurred.
User	The user ID active when the event occurred. This may or may not be reported depending on the type of event.
Computer	The name of the computer where the event occurred.
Event ID	A number identifying the particular event type.
Source	The software function that logged the event. This can be a system function, a driver, or an application.
Type	The Event classification: Success, Information, Warning, Error or Failure. Success or Failure types only appear in the security log; the other types occur in both the system and application log.
Category	Classification of the event by the event source (primarily used in the audit log).
Data	An optional field containing binary data displayed as bytes or words. This optional field is not kept if the record is exported to a text file.

Many types of events are recorded in the event log; these are documented in the Windows NT and Windows 2000 Resource Kits (CDROM help file and database). Applications can also record events to the logs. For example, the NT and 2000 Resource Kits provide the LOGEVENT command, which can be used by batch files to create an event and place it in the application log.

The event logs can be exported to a text file so that the information can be processed by other programs. There are various utilities that can do this.

The **uptime** command, available with Windows NT Service Pack 4 and later, and Windows 2000, reads the event logs to measure the operating system availability; certain customization is required for it to record this information accurately (the **Heartbeat** section of this paper has more details).

The following table shows examples of recorded events recorded that indicate changes in he availability status of a component.

Windows NT Events					
Event ID	**Log**	**Category**	**Component**	**Status**	**Description**
512	Security	System Event	Operating System	Available	Windows is starting up.
513	Security	System Event	Operating System	Unavailable	Windows is shutting down.
6005	Event	None	Operating system functions	Available	The Eventlog has started (a good indication that applications are about to be started).
6006	Event	None	Operating System	Unavailable	The Eventlog has stooped (a good indication that Window is shutting down).
6008	Event	None	Operating System	Unavailable	Records the date and time of the previous shutdown, if it was abnormal.
6009	Event	None	Operating System	Available	Windows has started.
8033	Event	System Event	Operating system functions	Unavailable	(Appears to be the last event written to the log before NT shuts down).
592	Security	Detailed Tracking	Application	Available	A new process has been created (also contains detailed process identification information).

Windows NT Events					
Event ID	**Log**	**Category**	**Component**	**Status**	**Description**
593	Security	Detailed Tracking	Application	Unavailable	A process has ended (also contains process identificatio information).

The Resource Kits for Windows NT and Windows 2000 also contain detailed information on events, and provide a CDROM with useful tools and information.

Netware

Netware Servers log events to a log file in text format; this file can be viewed or printed using Netware utilities. Information written to a Netware 3.11 (or later) server console can be logged to a file using the CONLOG command; this allows console information to be saved so that any availability related information can be extracted at a later time. CONLOG will record messages related to errors, Novell Directory Services (NDS) messages, load/unload of Netware Loadable Modules (NLMs), and RCONSOLE connections and disconnections.

Netware 4.x provides an additional audit logging capability , the AUDITCON utility. AUDITCON can be used to audit and log specific events; these can be filtered by event type, user, file/directory, or volume. AUDITCON logs are stored in binary format. Report files can be generated from the audit log files to allow user programs to extract the desired availability information.

The following table shows some of the event numbers and types AUDITCON reports, which can indicate a change in the availability status of a component.

Netware AUDITCON Events			
Event Number	**Event Name**	**Component(s)**	**Status**
18	A_EVENT_DOWN_SERVER	Server	Unavailable
50	A_EVENT_UP_SERVER	Server	Available
55	A_EVENT_VOLUME_MOUNT	Server disk volume	Available
56	A_EVENT_VOLUME_DISMOUNT	Server disk moun	Unavailable

Further details on AUDITCON can be found in the ***Netware - Auditing The Network*** documentation.

OS/2 (WARP)

First Failure Support Technology/2 (FFST/2) ships with many OS/2 applications. It provides a common message and error logging facility. Messages from applications that support FFST/2 are logged and can be viewed, printed, or saved in a file in text format using the message log formatter utility provided with FFST/2.

Applications that do not have "built-in" FFST/2 support can use this common message logging facility via APIs or a command line program.

If OS/2 LAN Server or Warp Server is installed, the AUDIT function can record the status of users connecting to/disconnecting from the domain, and usage of shared devices. This information can be useful when measuring the availability of OS/2 LAN Server resources and public applications. The audit information can be directed to a text file to allow user programs to process and extract the desired information.

Application Logs

Applications may provide their own event log, specific to the resources used by or provided from the application. Information in these logs can be used to measure the application availability, or the availability of a specific application component.

For example, Web Servers record access activity in their access log. This can provide availability information on resources such as pages, images, or servlets that users request. It can also be used to calculate the Web Server availability (this will be discussed further in the **Heartbeat** section of this paper).

Messages

Messages are events indicating that something has happened, and are usually meant to be seen by a human being. Operating systems and applications produce messages regarding the status of resources they control.

Messages are in a readable text format. The ones relevant for availability purposes will include:
- A timestamp (date and time)
- A component identifier
- Status information (“up”, “down”, “started”, “abended”, “error”, etc.)

Messages have two destinations:

1. A display screen. This can be a “system” console, where messages are centralized across a system. It can also be an “application” display, where messages specific to the application are displayed (this might not be the application itself; it can also be an management application that is monitoring the application).
2. A log, which has been described in the preceding section.

Messages sent to a log can be processed as has been described earlier. Messages sent only to a display screen can be processed in several ways:

1. An automation product may be able to directly receive the messages.
2. An automation product may be able to “screen scrape” the display to capture the message.

Both of these methods can require more work than simply processing a log. The advantage is that the messages can be captured as soon as they occur. This allows availability measurements to be provided in real time, updated as soon as events occur.

The various operating system platforms have different methods of providing access to system and application messages; these methods will be covered in this section. Regardless of the platform, these considerations will apply:

- Some type of automation function, either provided by the operating system or a separate product, can be used to capture the message data and extract the relevant information.
- If the message is not written to the log, the automation function can write it. This supports “centralizing” processing to extract availability information.
- Many more messages are produced than are needed. System and application message documentation will have to be reviewed to determine which messages to capture for availability measurement purposes.

z/OS and OS/390 Messages

OS/390 messages are created when programs use operating system facilities to create WTO/WTOR (Write To Operator/Write To Operator with Reply) requests. Messages can be issued by operating system functions, subsystems, or application programs. Certain messages contain information on the availability status of:

- Platform hardware components
- Operating system components
- Subsystems
- Subsystem components
- Application programs
- Attached I/O devices

After a message is issued, it is processed by several functions:

- Message Processing Facility (MPF)
  The MPF can influence how a message is handled. This includes display suppression, handling action messages, making the message eligible to be automated, and invoking an installation exit.
- Subsystem Interface (SSI)
  After MPF processing, the SSI broadcasts the message to all active subsystems, which can determine what action should be taken. Automated console operations products normally attach to the SSI as a subsystem so that they can access the messages and determine if additional actions should be taken.
- Multiple Console Support (MCS) and Extended Multiple Console Support (EMCS)
  After broadcast on the SSI the message is passed to MCS/EMCS, and displayed at all consoles with a matching message routing code. These consoles can be physical consoles or "logical" consoles (associated with a software task such as NetView or a TSO user ID).
- Hardcopy log
  The final step taken is having the message written to the "hardcopy" log. In reality, this is usually the system log data set - SYSLOG or (if enable in a parallel sysplex) OPERLOG..

There is a message ID associated with every message. Examples of the message IDs used for availability measurement purposes include the following:

IEA371I	The system is being IPLed. The time between this message and the preceding message indicates how long the platform has been down.
IEA389I	The IPL is complete and the operating system can now start processing work.
IEF403I	A unit of work (job/started task) is starting.
IEF404I	A unit of work (job/started task) has ended.
IEF450I	A unit of work (job/started task) has abended.

In addition to the system level messages, applications running under z/OS or OS/390 can issue messages that indicate their specific availability. For example, here are some of the messages issued by the IBM HTTP Server on OS/390 that indicate its availability:

IMW3536I	The HTTP Server is ready to processing URL requests.
IMW3537I	The HTTP Server is terminating but will restart (triggered by an operator reques
IMW3538I	The HTTP Server successfully restarted, and can begin processing URL request again.
IMW3540I	The HTTP Server has stopped processing requests and has begun to shutdown.

The Appendix lists examples of common z/OS and OS/390 messages, the components about which the message is reporting, and the availability state indicated by the message.

Many messages can be used to develop availability measurements for the z/OS or OS/390 platform and attached components. The messages to use that best indicate the status of a particular component have to be identified. If the messages are written to the SYSLOG/OPERLOG, a program can be written to analyze the SYSLOG/OPERLOG and determine which "availability status messages" appear most frequently, and therefore are to be used to create measurements.

If a subsystem or application issues messages that are not normally written to the SYSLOG, more work has to be done to direct the message to the SSI or MCS/EMCS. Some subsystems (such as IMS and CICS) provide exits to allow messages internal to the subsystem to be directed to the SSI and EMCS, where they can be processed by MPF and automation. Other subsystems may require certain initialization parameters to be specified to send messages to the SSI or MCS/EMCS.

Automation functions efficiently filter the desired messages. They interface to the SSI or to EMCS to monitor the message traffic. All messages contain a date and time stamp, information on the source of the message, and within the message text the name or location (I/O address) of the component. Automation can monitor the message contents and extract the desired information in real time. The SYSLOG can also be used to extract the desired message information if post-incident or historical collection is desired.

Additional steps may be needed to derive the component ID. For example, most messages refer to an I/O devices by its unit address. This has to be correlated with other information to obtain, for example, the volume serial of a DASD device. Access to configuration information, especially if it can be done by the automation that collects the availability data, is needed to correlate and provide this level of information.

For details on OS/390 message syntax and contents, refer to the z/OS and OS/390 Messages manuals for the appropriate release of z/OS or OS/390, and supported products, that are being used.

VM and z/VM

Messages in the VM environment are issued by operating system components (CP, CMS, IUCV, for example) as well as by virtual machines. Certain messages contain information on the availability status of:

- Platform hardware components
- Operating system components
- Virtual machines
- Virtual machine components
- Real devices
- Virtual devices

VM has a single "system operator" virtual machine where system messages are sent. Messages from other virtual machines can be routed to it, if this is required.

The Programmable Operator Facility (PROP) is provided by VM to assist in handling messages in the VM environment. PROP would normally be run in the system operator virtual machine to intercept messages sent to that virtual machine. It can also receive messages sent to that virtual machine's console from another virtual machine by other means (CP SMSG, NetView, etc). PROP can log the messages into a file and can take actions based on a message entry in the active routing table (including suppression, logging, programmed response) or route the message to a logical operator - which can be an automation program (for example, NetView).

The Appendix lists some of the VM CP messages, affected components and the indicated availability status.

Many messages that can be used to measure availability of various VM platform and attached components. Messages specific to resources within a virtual machine will also have to be considered. Additional work may be needed to get messages within a virtual machine directed to the CP system operator, or to PROP. Some virtual machines subsystems will need certain parameters defined for this to take place.

The messages that best indicate the availability status of a component will have to be selected. The message can be either a VM system message or a message issued by a virtual machine. PROP can log messages it receives into a CMS file. This file can be analyzed to determine which "availability status messages" tend to appear most frequently, and therefore are to be used to indicate the state of a component.

Automation functions can provide some efficiency for filtering the desired messages. They interface to PROP (so that PROP can route messages to them) to see the message traffic. All messages contain a date and time stamp, information on the source of the message, and within the message text the name or location (I/O address) of the component. Automation can monitor the message contents and extract the

desired information in real time. The message log file created by PROP can be processed for post-incident or historical data collection.

Additional steps may be needed to derive the component ID. For example, most messages refer to an I/O devices by its unit address. This has to be correlated with other information to obtain, for example, the volume serial of a DASD device. Access to configuration information, especially if it can be done by the automation that collects the availability data, is needed to correlate and provide this level of information..

For details on the topics discussed in this section, please refer to the following manuals for the release of VM/ESA or z/VM that is being used:

- Planning and Administration
- System Messages and Codes
- CP Command and Utility Reference

IBM Communications Server SNA Messages

The IBM Communications Server produces messages for both the SNA and IP functions on S/390. This section focuses on VTAM messages related to components that participate in the SNA network.

Devices are logically defined as SNA network physical and logical units (PUs and LUs). These units represent hardware components, applications, and users:

- Components such as gateways and cluster controllers are usually defined as physical unit nodes.
- Applications and users are usually defined as logical unit nodes (an end user intelligent workstation can be defined as a physical unit, with each possible SNA session connection as a logical unit).
- Connections can be defined as lines (physical), cross-domain (logical) or inter-network paths (logical).

Unsolicited VTAM messages - messages that indicate an unexpected change in the state of a component - go to the defined primary program operator application program (PPO). This is normally Tivoli NetView for OS/390, or an equivalent product. This is defined in the NetView VTAM APPL definition, and allows NetView to received these messages. If NetView (or its equivalent) is not active, the messages will flow to the normal message facility for the operating system..

IBM Communications Server uses the message facilities of the operating system, so the same considerations for using messages on the appropriate platform apply. Automation products that interface to the message stream via NetView or (if unsolicited messages flow to the operating system) the operating system message interface (SSI or Extended Consoles for OS/390, PROP for VM, OCCF for VSE) can capture and filter the desired messages. The messages contain the information needed for availability measurements - a time stamp, and message text containing a component identifier and

availability status. Automation products can extract the required availability information from these messages.

Communications Server will also generate or forward SNA alerts to NetView to report component status changes. Information from both alerts and messages may be needed to provider complete status information. NetView can capture both sources of information, so it is a good candidate for use in collecting these message/alert indicators. NetView can invoke real time processing, or log the information for post-incident or historical processing.

Status information collected by NetView can also be reflected in NetView's Resource Object Data Manager (RODM); this is another function that could be used to get availability status information.

The Appendix identifies some Communications Server messages, affected components, and the indicated availability status. For details on message syntax and contents, refer to the IBM Communications Server or VTAM Messages manuals for the appropriate release of Communications Server or VTAM that is being used.

OS/400 Messages

Messages in the OS/400 operating system flow between users or programs using ***message queues***. Each user, program (represented as a batch or interactive job) and display station has an associated message queue; messages are that are sent to a user or display are directed to the appropriate message queue, where they can be displayed on a terminal or processed by a program.

The system operator queue (for system messages or messages directed to the operator) is named QSYSOPR. Optionally, a message queue named QSYSMSG can be created; certain messages will be directed to it instead of, or in addition to, the QSYSOPR message queue.

Certain OS/400 messages contain availability status information for:

- Operating system components
- Subsystems
- Jobs
- Files
- I/O devices
- Network components (links, controllers, workstations)

OS/400 CL programming can be implemented to monitor message queues, including QSYSOPR and QSYSMSG. When messages are received, the program can take actions (reply to a message, invoke a program, execute a command, forward a message, etc.) based on the message attributes. For example, a program could be invoked to record the availability status of a component when a particular message containing or indicating that information occurs.

The QHST history log contains information on system events. It logs a high-level trace of system activities such as system, subsystem and job information, device status, and system operator messages. All messages written to the QSYSOPR message queue go to QHST. The QHST log can be processed to determine which messages occur that should be used to indicate component availability status, or processed as a source of availability data, either through online commands or by writing it to a file and extracting information from the file.
.
When dealing with multiple systems, the OS/400 operating system can convert messages to SNA alerts and forward them to a focal point. Availability monitoring and data collection could be consolidated by having the appropriate messages sent as alerts to the focal point, where they could be processed. Many messages are shipped as "alertable"; additional messages can also be defined to create alerts if they occur.

The Appendix identifies examples of OS/400 messages that are useful for availability measurements. For more details on OS/400 messages and syntax, please refer to manuals for the appropriate OS/400 release, such as:

- AS/400 Basic System Operation, Administration, and Problem Handling (SC41-5206-04)
- AS/400 System Operation (SC41-4203)
- OS/400 CL Programming (SC41-5721)
- OS/400 Workload Management (SC41-5306)

AIX Messages

Messages in the AIX environment are issued by operating system components and applications. Some messages contain availability information for:

- AIX operating system and components
- Attached I/O devices
- Network interfaces
- Subsystems
- Processes

Messages are directed to, and can be found in, two locations:

1. System log (syslog)
   This is the “traditional” method of logging messages in UNIX operating systems Messages from the operating system or applications, using either the syslog() function or the logger command, can be directed to one or more files based on:
   - Message source - function or product process that sent message
   - Message priority (emergency, alert, critical, error, warning, notice, information, or debug)
     Normally messages sent will go to the system console (represented by the /dev/console file); the output from the console device is called the syslog.

2. Error log
   System error messages are logged in the AIX error log (/dev/error file). Error logging is normally enabled when the system is initialized. Commands are provided to clear the error log or to generate error reports from the data. Errors are classified by:
   - Error ID
   - Timestamp
   - Error type (PERM, TEMP, PERF, PEND, UNKN)
   - Error Class (hardware, software, created by a command message)
   - Resource Name
   - Resource Type
   - Resource Class

   Detailed description and product information will also be included in the logged information..

Information from the two sources can be consolidated in various ways. For example:

1. Syslog messages can be sent to the error log by specifying it as one of the destination files for all or certain sources of and/or severity level messages.
2. Error log messages can be sent to the error log by creating an error notification object that will send the message to the syslog using the logger command. The object can be customized to send all or only certain classes of messages.

Syslog messages can be processed directly from the file that they have been written to. If they are being sent to the console, the **swcons** command can direct the console output (syslog) to a file. This file can then be processed to extract the appropriate information.

The **errpt** command reads the error log and puts the data into a readable file that can be processed to extract the appropriate information.

If real-time data capture is desired, automation functions can be used to accomplish this:
- For the error log, create an error notification object with an associated method (a program or command language script). When an error log message is created, the object will be invoked and the message will be passed to the program/script, which can examine the message contents and take appropriate actions. For example, this can be used to send an SNMP trap whenever a particular type of error occurs.
- For error log messages, have desired messages made "alertable" via the **errupdate** command. Alertable errors are presented as SNA alerts to SNA network management software (such as Tivoli NetView for OS/390, assuming there is a properly configured connection from the AIX platform to NetView).
- For the syslog, have a program monitor the file the desired messages are be written to and take appropriate analysis and action(s) as message lines are received.

The Appendix contains some of AIX error log messages that contain component and status information useful for availability measurements. For details on AIX messages, syslog, and error log, refer to the manuals for the appropriate level of AIX, including:

- AIX General Programming Concepts
- AIX System Management Guide
- AIX Problem Solving Guide and Reference

Alerts

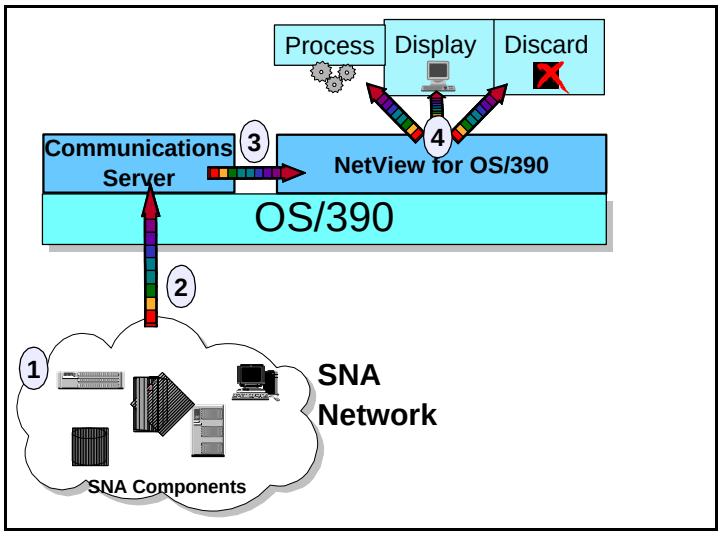
The Systems Network Architecture (SNA) communications protocol contains ***SNA Management Services Units***. and ***Network Management Vector Transports (NMVTs)*** that are commonly known as **alerts**. They are notifications sent to report a change in a SNA component. Generated alerts can indicate component availability status changes, and provide availability measurement information. The types of components reported on include:

- Applications known to Communications Server via LU definitions
- SNA network devices (communicators controllers, lines, cluster controllers, etc.)
- Gateway devices that communicate between SNA and non- SNA entities
- Software that issues SNA alerts (applications, performance monitors, network management software, etc.)

An alert is a notification from a component to an SNA network management product that some aspect of a component has changed. Tivoli NetView for OS/390 is commonly used as the SNA management product that receives alerts. For the AS/400 and e-server iSeries platforms, the OS/400 operating system contains network definition and management functions that include sending and receiving SNA alerts.

The alert may be viewed as good, bad, or indifferent by the installation. The notification is unsolicited (meaning it occurs without anything querying for it) and "one way" (meaning it cannot be directly replied to). Since alerts may be caused by all sorts of conditions, many more alerts are generated than are actually needed to determine the availability of SNA components in the environment.

For S/390 networks, the path an alert takes from the component to the management product (which is also called the ***alert focal point***) shown in the following picture:



1. A component, or software that manages the component, generates an alert (based on the component and software alerts can be created for specific situations).
2. The alert goes to Communications Server (VTAM).
3. Communications Server (VTAM) sends the alert to the Alert Receiver focal point (usually NetView).
4. The alert focal point takes action on the alert. The action can be:
   - Discarding it
   - Displaying it on an operator terminal (for NetView, on the NetView Hardware Monitor)
   - Logging to a log file (For OS/390, to SMF record type 37(X'25'))
   - Invoking automated actions or procedures (for NetView, via the NetView Automation Table)

An address space running on the same OS/390 image as the NetView program can issue alerts directly to the NetView address space using the NetView Program-to-Program Interface (PPI).

For iSeries and AS/400 networks, the alert flow is similar, except that the alert generation and focal point functions are part of the OS/400 operating system (no separate SNA protocol product or NetView is needed). An OS/400 can be the focal point for any SNA alerts in the network, and can log them in a database. Alerts can also be forwarded to and received from Tivoli NetView for OS/390.

An alert contains one ***major vector*** and one or more ***subvectors***; each subvector may have several ***subfields*** .The contents of these vectors, subvectors, and subfields can be checked and used by the NetView automation table. There are also NetView programming functions, usable in NetView REXX or CLIST language automation routines, to access the alert contents.

Valid values for the vectors and fields are documented in the ***SNA FORMATS*** manual. Products can provide their own alerts, and installations can implement customized alerts as well. The major vector types that may contain component availability information are:

X'0000' Alert (most alerts use this major vector)
X'0001' Link problem
X'0002' Resolution alert (a problem has been resolved)

The subvectors to be examined are:

X'05' This is a usually a list of the resources from the component having the problem up through the network that sent the alert to NetView. The information includes:
- Resources names - the SNA network names (PU, LU, line etc. From configuration information).
- Resource type codes - codes that indicate what type of component this is (line, controller, adapter, etc.). NetView supplies a table (which can be updated) that is used to associate resource type codes with generic component descriptions.

X'10' Identifies one or more products. It usually contains information about the product the alert occurred on. It may contain inventory type information such as machine type, serial number, etc.

X'51' If the alert is for a LAN attached component, this subvector contains information on the LAN link connection (adapter address, ring/bus identifier, bridge identifier, etc.).

X'92' Contains detailed alert information:

- A code indicating the event type:
  - Permanent loss of availability
  - temporary loss of availability
  - performance impacted
  - permanently affected resource
  - impending problem
  - unknown
  - Bypassed
  - redundancy lost
- The Alert ID (4 bytes)
- A description code (documented in the SNA Formats manual; installations can add additional codes). Categories include:
  - Hardware
  - Software
  - Communications
  - Performance
  - Congestion
  - Microcode
  - Operator
  - Specification
  - Intervention Required
  - Notification
  - Security
  - Undetermined

X'93' Probable Cause Code(s) that denotes possible causes of the event.

X'94' Probable User Cause Code(s) that denotes probable user actions that may have been taken that caused the error, and recommended actions to take to try to resolve the problem.

X'95' Probable Install Cause Code(s) that denotes probable causes of the problem due to install activities (initial installation or setup of the component).

X'96' Probable Failure Cause Code(s) that denotes condition(s) that have resulted in the failure of a resource, and recommended actions to take to resolve the problem.

Action(s) invoked for an alert will vary based on the contents of the alert, and how the installation chooses to handle/filter alerts based on those contents.
Here is an example of an alert, as displayed on Tivoli NetView for OS/390:

```
HCBNV       BONETT
            +--------+
 DOMAIN     |  PHON  |
            +--------+

DATE/TIME: RECORDED - 12/06 15:10

 EVENT TYPE: UNKNOWN

 DESCRIPTION: SOFTWARE PROGRAM ABNORMALLY TERMINATED

 PROBABLE CAUSES:
    APPLICATION PROGRAM

 APPLICATION PROGRAM TEXT:
    TEST ALERT
UNIQUE ALERT IDENTIFIER: PRODUCT ID - 5642010 ALERT ID - 03728157


   NMVT - 41038D 0000000000
   MAJOR VECTOR 0000 - 004F 0000
   SUBVECTOR 92
0B920000 12200003 728157
   SUBVECTOR 10
1010000D 110E0A00 40F5F6F4 F2F0F1F0
   SUBVECTOR 93
04931001
   SUBVECTOR 31
15310201 02110321 110C30E3 C5E2E340 C1D3C5D9 E3
   SUBVECTOR 03
11030301 09C2D6D5 C5E3E340 40D7C8D6 D5
   SUBVECTOR 97
06970481 1012
```

Getting Data From Alerts

Using automation, such as NetView (or a product that interfaces with NetView), is the best option for capturing availability data. However, analysis of the alerts that can be generated for a component is required to identify the ones that carry meaningful availability information. Both standard alerts (document in the SNA Formats manual) and custom alerts (documented in the individual component manuals) exist.

Once the desired alerts are determined, the following actions on the S/390 platform are necessary to obtain availability information:

- Define the NetView alert filters so that the desired alerts are passed to the automation table. Filtering is based on a combination of alert event type and the component generating the alert.

- Define the NetView automation table to check for values in specific subvectors and subfields of an alert, to determine when automation routines are to be invoked to access the information contained in the alert. This is necessary if further data examination or analysis has to occur.
- If availability data collection automation is being done outside of NetView, NetView can act as an "agent" to convert the alert data to a format the outside automation function work with (message, global variable, object repository, TCP/IP trap, etc.).

For the AS/400 environment, the actions to take include the following:

- Messages sent to the QSYSOPR message queue or the QHST history log can be converted into alerts. A subset of OS/400 messages are predefined to be "alertable"; any message can be defined to be alertable by updating a parameter in the message description.
- At the remote site where the alert is generated, or at the focal point, the alerts can be filtered so that only the desired ones are processed.
- Alerts can be directed to a data queue; a program can monitor the data queue and take action when an alert arrives, thus allowing further processing and/or automation based on the detailed contents of an alert.

Is it very possible that alerts ***will not*** be generated when problem conditions return to normal. For example, an alert may be generated when a component outage or severe performance problem occurs. When the component returns to an available, or normal, state, no alert is generated. Addressing these situations requires using a monitoring technique (discussed in the “Monitoring Techniques” chapter” in the following manner:

1. When the alert that indicates an outage is received, an automation routine invokes the desired monitoring technique.
2. When the monitoring technique determines that the component has returned to the normal state, it records that information to a log file or a message and then stops monitoring the component. This avoids unnecessary monitoring overhead.

For further details on alert syntax, and using automation to detect and process alerts, use the following manuals:

- SNA Formats
- Tivoli NetView Automation Guide (for the appropriate Tivoli NetView software level)

Traps

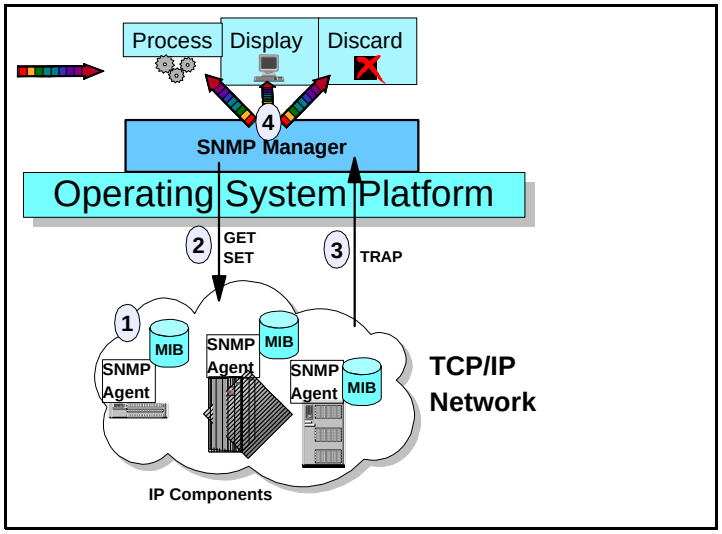
TCP/IP traps can be used when the components that are being monitored support Transmission Control Protocol/Internet Protocol, and have an SNMP agent. These components include:

- UNIX based application platforms (AIX, Linux, HP-UX, Solaris, etc.).
- Application platforms that support TCP/IP (including z/OS, OS/390, VM, z/VM, AS/400, Windows platforms, OS/2, and Netware).
- Devices that interconnect TCP/IP components and networks (switches, bridges, routers, etc.).
- Gateway devices that communicate between TCP/IP and non-TCP/IP environments.
- Software that can interface with SNMP to issue traps.

The Simple Network Management Protocol (SNMP) is used to support the management of TCP/IP networks. The major components of SNMP consists of the following:

- **Managed Node:** a component in the TCP/IP network that needs to be monitored/controlled. Each managed node must have a TCP/IP network address, and run an **SNMP agent** software that receives, processes, and sends SNMP requests and notifications. A single physical component can have multiple TCP/IP network addresses, both physical (multiple physical interfaces) and logical (multiple addresses per physical interface).
- **Management Information Base (MIB):** a MIB is required for SNMP management. It is a set of objects (with a standard naming method) that represent component information or status (for example, number of hard drives or current CPU utilization). An SNMP agent has a default MIB; components can also have component-specific MIBs. The MIB contains both object and trap definitions.
- **Management Station**: a platform that runs SNMP manager software (such as Tivoli NetView or HP OpenView) to processes management information from SNMP agents on the managed nodes.
- **Management Protocol**: the communication between the management station and the managed node, to obtain information about, or change some attribute of, the managed node. There are 3 major functions supported by the management protocol:
  - **GET** the value of one or more MIB variables. The management station must be authenticated by the managed station before information is returned.
  - **SET** the value of one or more MIB variables. This results in a change of state in the managed node - assuming (for security reasons) that the SNMP agent software on managed platform allows MIB variable changes to physically change the managed node.
  - **TRAP:** this is an unsolicited notification from the managed node to the management station that something has changed on the managed node.

The follow figure depicts the communication flow when using SNMP:



1. Each component in the IP network to be managed has a SNMP agent and MIB.
2. The SNMP Manager software issues GET/SET commands to retrieve monitored information, or to set a monitored object value (which in turn may cause some physical change in the component).
3. When certain conditions occur, the SNMP agent sends a TRAP to the SNMP Manager (a trap can be sent to multiple SNMP Managers).
4. The SNMP Manager can process the trap (for example, store it, or, if it has automation, invoke an action such as running a program), display it on its console, or discard it.

A TCP/IP trap is very analogous to an SNA Alert, and can be used as a source of availability data for TCP/IP components. the easiest way to work with traps is via the SNMP management software that is used to monitor and capture traps.

Some management stations can convert a trap into an alert. This is useful if:
- There is an SNA network.
- Tivoli NetView is installed on a S/390 system.
- S/390 automation is being used to collect availability data.

This is one way to consolidate availability data.

A trap contains the following values:

enterprise	The MIB object ID assigned to the vendor implementing the agent. This MIB variable uniquely identifies the agent.
agent-address	The TCP/IP network address that generated the trap.

generic-trap	A number from 0-6 that indicates trap type: • 0: The SNMP agent is reinitializing and may be resetting M variables. • 1: The SNMP agent is reinitializing but MIB variable value are not being changed. • 2: The SNMP agent has detected a ***linkdown*** condition. A network interface monitored by this agent has been disablec • 3: The SNMP agent has detected a ***linkup*** condition. A network interface monitored by this agent has been enabled • 4:. The SNMP agent received a message that could not be • authenticated. • 5: If the agent is running on a platform that is also running exterior gateway protocol (a TCP/IP protocol used for routing information between groups of networks and gateways), it has detected the loss of a neighboring platform that is also running EGP. • 6: The SNMP agent has issued an ***enterprise-specific*** trap. This is a trap defined specific to this component type (usual by the component vendor).
specific-trap	May contain more information about what caused the trap. For example, for enterprise-specific traps this can contain a number uniquely identify this defined trap.
time-stamp	Elapsed times (hundredths of seconds) from the time the agent was last initialized to the time the trap was generated.
variable-bindings	A of name=value pairs, where ‘name’ is a MIB object and ‘valu is the object’s value, that is included in the trap information. Th allows enterprise-specific traps to send MIB data as part of the trap.

Enterprise specific traps are defined in the component MIB. Here is an example trap definition:

```
enterprise3174StatusCodeChange TRAP-TYPE
     ENTERPRISE ibm3174EnterpriseTrap
     VARIABLES { gen3174SscChanges }
     DESCRIPTION
             " The enterprise3174StatusCodeChange trap indicates
               that the 3174's system status code has changed.
               This could be any addition or removal of system status.
               The table gen3174SscTable contains the current status
               codes in the system status queue.
               Note: this trap is generated only at the end of each
               time period where the value of gen3174SscChanges is
               different from the previous period."
     ::= 1
```

RFC1215 can be used to determine what all the fields mean. The key items here are:

- This is an enterprise-specific trap, since the ENTERPRISE section contains a specific identifier (MIB object).
- The specific trap type is 1.
- The variable-bindings list will contain the object ID represented by ‘gen3174SscChanges’ (defined elsewhere in the MIB) and the values associated with that object.

Generic-trap types 2,3, and 6 can be used to indicate the change in the availability status of a component. Trap type 6 will require further investigation, to determine which of these enterprise-specific traps contain relevant availability information. Be aware that the SNMP agent periodically polls the network interfaces (based on values that can be customized) to determine their status - which means that some traps may not be generated at the time the status actually occurs.

Getting data from traps

Almost all TCP/IP operating system and component implementations include, as part of the TCP/IP application suite, SNMP agent software. Some may include SNMP management software, but these are more likely found incorporated into vendor products that support SNMP management. Theoretically, almost any point on the TCP/IP network can be the management station, or one of several management stations. Placement of the management station will depend upon SNMP traffic volume and trap processing requirements. The choice of the appropriate platform depends on various technical, performance, political, and financial considerations that are far beyond the scope of this document).

Use TCP/IP network management software products to gather and analyze information from a trap. The potential traps that could be issued for a component can be determined from the MIB for that component. Some management software products may also provide a ‘command-line” program that can generate a trap, useful when using a monitoring technique (described in the ‘Monitoring Techniques’ chapter) on a component.

Once the desired traps are identified, one (or more) management stations should be defined to receive the trap. Trap transmission is "connectionless'; that is, once the agent sends the trap, nothing comes back to indicate that a managing system has received it. Since it is an unsolicited notification, the management station has nothing to indicate that a trap may be arriving. This should be taken into consideration when planning where the traps will be received.

After the trap is received by the managing station, the SNMP Manager should have the capability to do one or more of the following:

- Record the status in a file or database, either directly, or, have enough automation capability to start a local program when a particular type of trap is received that saves the appropriate information. This allows later collection or processing of the data to measure and report availability, either on the SNMP manager platform or at another location the information can be forwarded to.

- Convert the trap into an alert, if a SNA network exists and it is desired to do so. The management software should have access to service point functions (either directly or interfacing to another product) that allow the conversion from a trap to an alert that is forwarded into the SNA network (an example is Tivoli NetView on AIX interfacing with Tivoli NetView for OS/390 to accomplish this task).

Is it very possible that, as in the case with alerts, traps ***will not*** be generated when problem conditions return to normal. A trap may be generated when a component outage or severe performance problem occurs. When the component returns to an available, or normal, state, no trap is generated. Addressing these situations requires using a monitoring technique (discussed in the "Monitoring Techniques" chapter) in the following manner:

1. When the trap that indicates an outage is received, an automation routine invokes the desired monitoring technique.
2. When the monitoring technique determines that the component has returned to the normal state, it records that information to a log file or a message, or generates a unique trap on its on, and then stops monitoring the component. This avoids unnecessary monitoring overhead.

Event Management Products

Event Management products can provide an easier path for using the data sources that have been described, or can create their own data sources for availability events. These products are able to:

- Access log, message, alert and trap data sources, to extract the desired information.
- Provide agents, or use other products as agents, to create new event sources.
- Filter information coming from the event sources, so that only the data relevant for what is being measured is obtained.
- Correlate events coming from multiple sources, which may reflect status on the same component, to eliminate redundant information.
- Invoke automation, triggered by events or by a schedule, to invoke monitoring techniques against resources.
- Store captured data in a format that is easily accessible from other programs, particularly reporting programs.

Products providing these functions are very useful in collecting availability data. The more data sources they can access, the better they can help consolidate data for a variety of components. Their agents, or interfaces to other products, can create new event sources. They reduce or eliminate the amount of "custom coding" necessary to access information in the event sources. They can create, or feed, a repository against which measurement reports can be run; they may even have enough automation function to extract the "raw" data from the event source and format it in a consistent manner for reporting, regardless of where the data originated.

Examples of these products from IBM include:

- **Tivoli Enterprise Console (TEC)**
  Tivoli Enterprise Console provides adapters to capture events from a variety of sources, such as:
  - Log files
  - Traps (via SNMP Managers such as Tivoli NetView or HP OpenView)
  - Tivoli Distributed Monitoring events
  - OS/390 messages and SNA alerts (via Tivoli NetView for OS/390)
  - Tivoli Management products for specific applications (DB2, WebSphere, Web Services, Notes, Oracle, etc.)
  - Third party products

  TEC also provides the automation to correlate, store, and extract specific data from these sources.

- **Tivoli NetView for OS/390**
  Tivoli NetView for OS/390 captures events from:
  - z/OS and OS/390 messages
  - SNA alerts
  - Traps
  - 3270 applications (via screen scraping)

  - TEC events (via Tivoli Enterprise Console)

  Tivoli NetView for OS/390 also provides the automation to correlate, store, and extract specific data from these sources.
- **IBM Director**

  IBM Director provides agents to capture availability and performance events from Intel workstations, and the automation to store and extract data from these events.

There are also products from other vendors that can perform these functions.

Monitoring Methods

Overview

There will be times when the data sources described in the earlier sections of this white paper will not provide the availability information needed to create measurements. When those sources are not adequate, monitoring of the component to detect availability status will be required.

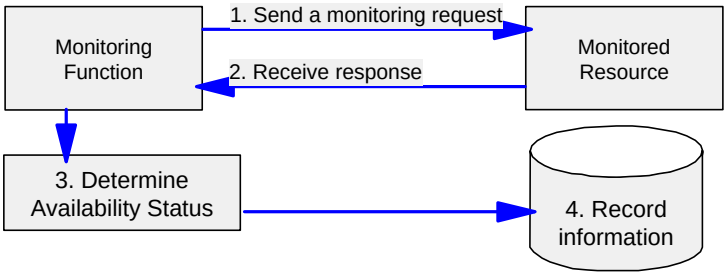
An active monitoring method or technique can be used to capture this status. This does not need to be a separate task if performance monitoring or automation of the component state is already occurring. The necessary information may already be available from this activity; it is just a matter of extracting what is needed.

This section discuss the monitoring methods that can be used. They vary in complexity; in general, the more complex a method is to implement, the more detailed, specific measurement data it can provide. The monitoring methods that will be covered are:

- Heartbeats
- PINGs
- Remote command execution
- End user simulation
- Custom monitoring agents

These methods will be covered from a “standalone” implementation standpoint. However, these functions may also be part of existing or planned management or monitoring software for an particular environment.

All of the methods are used in a similar manner, as shown in the following diagram:



1. The monitoring method is invoked and sends a request to the monitored resource.
2. The monitored resource returns a response to the monitoring function.

3. Based on receiving or not receiving a response, or the content of a received response, the monitoring function determines the availability status of the resource.
4. The monitoring function records the status in a repository.

Common considerations when implementing any of these techniques include the following:

- The technique must be invoked or scheduled using automation functions (provided by the operation system platform or a product). This provides consistent usage and eliminates human errors from the process.
- The response returned from a monitoring request must be captured for validation. Ideally the automation function that invoked the monitoring provides functions to analyze whatever data is returned, if that is needed to determine availability status.
- There is a tradeoff between monitoring interval size and monitoring overhead. The smaller the monitoring interval, the more accurate measurements will be, and the more quickly a status change can be discovered. However, this uses more "overhead" than a larger monitoring interval. Components that are very critical to the availability of an application should be monitored often, perhaps once or twice a minute.
- The technique can be used in conjunction with message/alert/trap events, when a verification of "return to normal" state is needed. This is most efficiently done by having the event that indicates a "down" or "problem" state trigger automation invoking the appropriate monitoring technique. The technique runs at a regular interval, sending requests to and checking the responses (or lack of response) from the resource. When the technique receives a response that indicates normal operation, it records this (as an event), and stops monitoring.
- Products already installed in the environment may already contain these monitoring functions. While most are easy to program in one's favorite complied or interpreted language, there is no reason to duplicate the effort if the function already exists in a product.

Heartbeat

A ***heartbeat*** is perhaps the simplest type of monitoring technique to implement. It is a function that runs continuously and periodically records or reports the status of the component on which it executes. As long as it is running and able to record heartbeats, the component is available The heartbeat function must always be started when the component is started, and is only stopped when the component is stopped. This ensures an accurate reflection of the component's availability.

The heartbeat can report status (a "pulse", so to speak) in two ways:

1. Local - recorded in data storage on the component (e.g. log file).
2. Remote - sent to another component, or monitoring function running elsewhere, and recorded there.

Availability reporting normally uses a metric of minutes. The heartbeat reporting interval must be no greater than a minute to best support this metric

When a component restarts after an outage, the time of the last recorded pulse indicates when that outage occurred. Calculating the time between that record pulse, and the time the heartbeat program starts, provides an outage measurement length. For example, when a component starts, the heartbeat function determines that the current time is 16:00, and the time the last pulse was recorded was 14:00. Therefore, the outage length was 2 hours, or 120 minutes.

Clustered environments, such as Parallel Sysplex or Highly Available Clustered Multiprocessing (HACMP) use a heartbeat function among the systems that participate in the cluster. The heartbeat is used to indicate to other cluster members that the system is still running. If a system's heartbeat is missing after a certain interval, the other members will assume that system is no longer operating, and will invoke actions to ensure the work supported by the cluster continues to run with minimal or no interruption.

A heartbeat program is best used when the component can do the following:

- Automatically start programs or defined functions after it is started.
- Invoke the heartbeat function as one of the first functions that is started after the platform starts it processing.
- Record the heartbeat pulse to a file or other storage medium on the local platform, or send real time output to a remote operating system platform.

Appendix A lists examples of "roll your own" heartbeat functions using REXX, a shell script, and a Java servlet, that can be implemented on any platform that supports that programming language. Each of these programs does the following:
1. When started, retrieves the last recorded pulse.
2. Calculates the outage length, or stores the outage start and stop times for processing by another program to for outage calculation.
3. Records a heartbeat in a local file every minute.

In addition, certain platforms may have a heartbeat function built in. For example, Windows NT (Service Pack 4 and later) and Windows 2000 have a program called **uptime.** This program enables a heartbeat function that records status to the registry, and calculates the platform availability using that heartbeat. Here is an example of output from the **uptime** command:

```
              Total Reboots: 235
     Mean Time Between Reboots: 3.39 days
             Total Bluescreens: 1
    Total Application Failures: 0

------------------------------------------------


Since 7/26/01:

           System Availability: 85.3382%
                  Total Uptime: 124d 4h:14m:35s
                Total Downtime: 21d 8h:1m:43s
                 Total Reboots: 30
     Mean Time Between Reboots: 4.85 days
             Total Bluescreens: 0
    Total Application Failures: 0
```

Certain operating system platforms and application subsystems provide "indirect" heartbeat functions. If there is a function that continuously, at least once a minute, records information to data storage with a time stamp, and is always running when the platform runs, it can be used to determine platform or application subsystem. availability. Any gaps of information greater than 1 minute indicate that an outage has occurred. Examples:

- On z/OS and OS/390, the SYSLOG and SMF are normally always being written to while the operating system is running. By processing these records, and looking for intervals of greater than one minute where no records were being recorded, operating system availability can be determined.
- Web Servers record incoming requests in a log. If a Web Server regularly receives multiple requests per minute, any periods that show no requests for more than a minute are likely indicators of a Web Server (or web server platform) outage. By processing the log and looking for these gaps in the recorded activity, outage times and availability of the web server can be calculated.

Using Heartbeats for availability monitoring and measurement

When using a heartbeat function, there are several considerations to keep in mind:

- heartbeats indicate the ability for the platform to execute. They do not indicate if specific functions needed by an application are running, or their responsiveness (Depending on how the function is implemented it may be possible to modify it to provide this function).
- Actions to calculate the availability must be taken.. A heartbeat recorded locally must be retrieved and processed. A heartbeat sent remotely must be captured and stored to use in subsequent calculations.
- If an indirect heartbeat is created, the mechanism that records the information used for the indirect heartbeat must always be running.

PING

A ***ping*** checks the status of a component from a remote location. It is a simple function that tries to get any response from the monitored component. The response, regardless of the contents, indicates that the monitored component is function.

A PING function is usually incorporated within a communications protocol. The most well known of these is the TCP/IP PING function. Almost every operating system platform and networking device communicates using TCP/IP, so this function is well suited to use in availability monitoring. Other ping functions exist in other protocols. For example, APING is a program found in Advanced Peer-to-Peer Networking (APPN) environments. APING allows one APPN node to determine if another APPN node is active.

An availability monitoring technique using PING follows these steps:

1. The monitoring function issues the PING to the monitored resource (it may set the numbers of pings to send, the data size of the ping request, the amount of time to wait before a timeout occurs, etc.).
2. The monitoring function waits for the responses to return.
3. If responses are received before the timeout value, the monitoring function assumes that the component is active. It may also set a criteria, such as:
   a. The percentage of ping requests that must be returned for the component to be considered available
   b. The response time that the request(s) must be received in for the component to be considered available.

   If the criteria are met, the component is considered available; otherwise, it is not available.
4. If no responses are returned - they all time out - the component is considered unavailable.

The following example shows the PING command being issued from a Windows NT desktop, to monitor a LINUX system. It will send 10 requests, with a time out threshold of 3 seconds (3000 milliseconds):

```
H:\>ping -n 10 -w 3000 hslsuse

Pinging hslsuse [9.82.131.240] with 32 bytes of data:

Reply from 9.82.131.240: bytes=32 time=20ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=11ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
Reply from 9.82.131.240: bytes=32 time=20ms TTL=253
Reply from 9.82.131.240: bytes=32 time=10ms TTL=253
```

All ping requests were responded to, with response times varying from 10 to 20 milliseconds. If a resource is not available, the response would look as follows:

```
H:\>ping -a -n 10 -w 3000 sms-tm390

Pinging smts-tm390 [9.82.131.251] with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

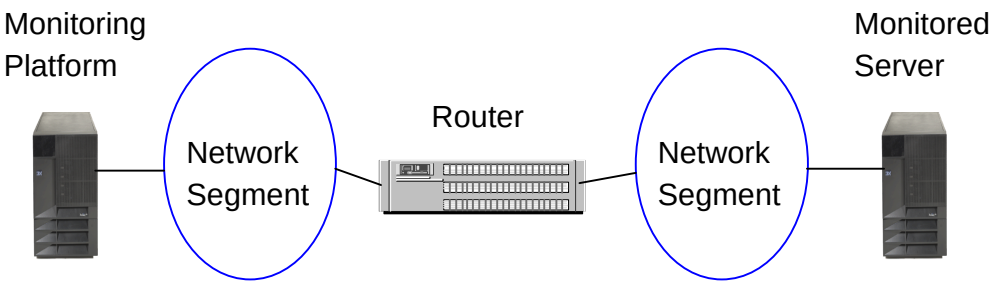
The ping requests were never returned, or the timeout threshold was reached before they could be returned, so the component is considered unavailable.

Using PING for availability monitoring and measurement

When using a ping function, there are several considerations to keep in mind:

- It can indicate that the component is active and able to communicate using the protocol that issued the ping. It may not indicate that the specific application is active. The TCP/IP PING function will show if a platform is active and running TCP/IP, but will not show that a web server function on that platform is active. Getting that level of information requires a customized ping function (either roll-your-own or contained within a management product) that, for example, will also send a request to the port used by the application, and determine if a response was received back from that port.
- A ping timeout can mean one of two things:
  1. The component is unavailable
  2. Another component in the path between the monitoring function and the path is unavailable.

  The following diagram shows a monitoring function running on an application platform that monitors a network component:



If the monitoring platform pings the monitored server and the ping timeouts, either the server is down, or the router - the major component between the monitoring platform and the monitored server - is down. The availability monitoring must understand the networking topology to take appropriate actions to ensure accurate measurements. In this example, if the monitoring detects the server is down, it should also check the router status, to determine where the actual unavailability is occurring.

- Programs that use this monitoring technique must be able to capture the information returned by a ping function, and analyze the contents. This is especially true if the availability criteria includes attributes such as percentage of successful pings, or the response time of the ping requests. A program that is part of an automation function is the best way to implement this requirement.

Remote Commands

A ***remote command*** extends the monitoring performed by a ping by issuing a specific command (or set of commands) and looking for a particular set of responses to be returned. This is similar to "PING" in that a command is issued from one platform and a response received, but is more powerful because the command(s) can be requests for information (such as "show all programs that are executing") or requests that an action be taken (such as "stop function X").

Remote commands provide the capability monitor both platform/device availability and specific resource availability, for a resource related to that platform or device that can be monitored or controlled by commands.

Remote command monitoring can be implemented in two ways:
1. Commands are sent directly from the monitoring function to the monitored component.
2. Commands are sent to an intermediate location, which is connected to the monitored component. The command runs at this location, but can gather information form the connected monitored component.

Remote commands require software on each platform to support command transmission, receipt and execution, and response transmission. Commands used for availability monitoring purposes must be non-interactive, line mode commands. That is, the command is issued, one or more command response lines are returned, and the command ends. The command response (depending on how remote execution is implemented) can be captured in some fashion (within variables in a command procedure, return code, written to a file, etc.) so that further processing actions can be taken based on the command results.

Examples of remote command functions are described in the following paragraphs.

REXEC (TCP/IP)

The TCP/IP ***rexec*** function sends a command from one TCP/IP platform to another, with the response being returned to the originating TCP/IP platform. The originating system must be running a REXEC client program, and the destination platform must be running a REXEC server.

ROUTE (z/OS and OS/390 Sysplex)

Within a sysplex the **ROUTE** command sends a command from one sysplex image to another. Any command that can be entered on a z/OS or OS/390 console can be used. This includes both commands related to the operating system and commands for applications running in z/OS or OS/390.

RMTCMD (Tivoli NetView for OS/390)

The NetView **RMTCMD** command sends a command from one z/OS or OS/390 image to another; the two images do not have to be in a sysplex but both must be running NetView, have a network connection (SNA or, if running Tivoli NetView V1R4 or later, TCP/IP) between them, and have the appropriate NetView definitions that enable NetView-to-NetView communication. Any NetView, network, or operating system/subsystem command can be sent, and the response is returned to the NetView operator or command procedure that issued the command.

SBMRMTCMD (iSeries and AS/400)

The iSeries and AS/400 Distributed Data Management (DDM) function provides a Submit Remote Command (***SMBRMTCMD***) function to send commands to another iSeries or AS/400 system. APPC/APPN is the protocol used to send the command(s) and return the command response(s).

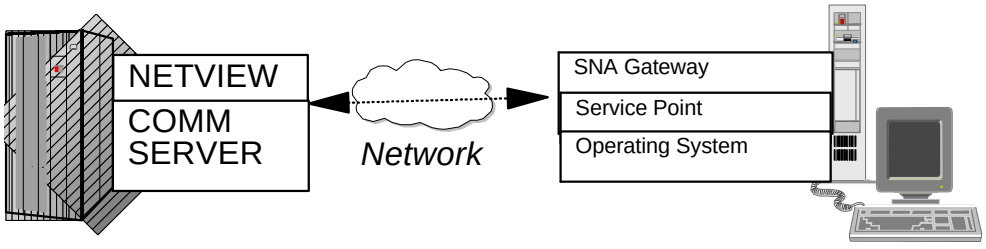
Remote Command Service (Windows NT, Windows 2000)

The Windows NT Server and Windows 2000 Server resource kits provide a set of programs called the ***remote command service***. This allows the sending of commands from one NT or 2000 system for executing on another. The monitored system must be running the server portion of the remote command service.

RUNCMD (Tivoli NetView for OS/390)

The ***RUNCMD*** function of Tivoli NetView on the S/390 platform (z/OS, OS/390, VM, VSE) provides a way to send commands from NetView to network connected non-S/390 components and receive responses. This supports monitoring the status of many different types of components and platforms.

The following is a picture of the major RUNCMD components:



- NetView - for the RUNCMD processor

- SNA gateway - for the SNA networking protocol between NetView and the Service Point
- Service Point - the main interface between NetView/SNA and the non-SNA platform. The application:
  - accepts the RUNCMD request from the SNA Network
  - interfaces with the local platform to perform the appropriate actions to carry out the request
  - returns the command response to the SNA network via the SNA gateway

The SNA Gateway and Service Point Application functions can be separate products or contained in a single product. Examples of products that support RUNCMD include:

- IBM Communications Server
- AIX NetView Service Point
- NetView Remote Operations Manager/Agent for AS/400
- Netware for SAA
- Ciscoworks Blue Native Service Point
- Microsoft SNA Server

A NetView automation procedure can issue a RUNCMD request that contains a command unique to a Cisco router. The Ciscoworks service point function executes the command on the router, and returns the result to NetView. The automation procedure can examine the response to this command and determine what further actions have to be taken.

Using Remote Commands for availability monitoring and measurement

When using remote commands, there are several considerations to keep in mind:

- Remote commands go further than PINGs; they can verify that specific workloads are running, and are able to respond to commands.
- Command security issues are critical. Sending the wrong type of commands to a component can inadvertently increase unavailability for that component. If a remote command process is used, it must meet the appropriate security standards that are in place.
- Any remote command function implemented should be part of automation, or should be able to be invoked by automation, so that results can be efficiently captured, analyzed, and used in the availability measurement process.

User Simulation

The best way to determine the availability of an application from a business perspective is to determine if it performing exactly as the user expects. This is done by implementing ***user simulation*** monitoring. This technique uses software functions to simulate user interactions with an application function.

- A program (or 'script', as it is commonly called) interacts directly with the application. This can be done in several ways:
- If the application provides an application programming interface (API), the program can directly invoke application calls via the API.
- If the application has a 3270 interface, the program can use functions such as the Enhanced High Level Language Application Programming Interface (EHLLAPI) to issue key strokes to the application and perform “screen scraping” to capture text from the application.
- If the application can be accessed using a browser, a Java program can invoke methods that can access the application URL, invoke URL or HTTP requests, and capture the response.
- If the application has a graphical interface, products exist that can record a sequence of mouse clicks and/or keystrokes against the application, and then play them back at regular intervals.

The simulation program performs functions such as:
- Signing on to the application
- Entering one or more application-specific commands
- Determining if information was returned within a specified interval
- Validating any information that is returned

User simulation can measure application availability and response time as an application user would see it. It will provide more detailed information that the other monitoring techniques, but is also more difficult to implement, since it must be set up for each application function to be measured.

Using a programmed script is more difficult than the other monitoring techniques because of the amount of maintenance that is required to maintain its accuracy. Any change in the application - such as screen layout, web page design, or URL changes - will likely require that the script be updated. However, if changes in the monitored application are synchronized with simulation program, user simulation will continue to provide very accurate results and availability information from the perspective of the application user.

The program or script runs on a workstation that is best located at or as close to the physical location of end users. While the simulation usually can be implemented using automation software running on the same platform as the application it is monitoring, this method will not accurately identify bottlenecks or problems in the path between the users and the application. Wherever it runs, it can either capture the information and immediately forward the relevant information to a central location, or store it on the workstation for later uploading to or retrieval into a common repository.

Using User Simulation for availability monitoring and measurement

- If a programming language is used, it must support the available methods for accessing the applications as described above. The most common languages used for this are C/C++, REXX, and Java. All run on a variety of operating system platforms. C /C++ and REXX can directly use EHLLAPI functions (provided by a communications program such as IBM Communications Server). Java provides methods that can be used to programmatically access and use web based applications, and both C/C++ and REXX support socket level programming that can be used to do the same..
- There are products that support designing and implementing user simulation functions without the need for programming. Many of these are categorized as “Application Test” or “Stress Test” products. They should, at a minimum, be able to:
  - Create or interface to an emulated user application session, or use APIs that provide access to the application across a network.
  - Support sending keystrokes and pointing device clicks to the emulated session, or API calls to the application.
  - Scan for or capture data displayed in the emulated session (known as "screen scraping"), or capture the application responses from the API calls.
  - Time the interval between the last user keystroke/click or API input call and the application response.
  - Store a recording of the interactions so that they can be executed at regular intervals (by the product itself, or by a program or script invoked by a platform automation function).
- When user simulation determines that either the application has not responded within the desired interval, two situations may have occurred:
  1. The application is unavailable
  2. Another component in the path between the monitoring function and the application is unavailable.

  User simulation by itself will not be able to narrow it down to a particular component. Its measurements must be correlated with other availability measurements, described earlier in this paper, to determine what component(s) are causing the problem. This identifies the component(s) that are the cause of the applications unavailability.
- The workstations where the user simulation functions are running must be as highly available themselves as possible. They should be dedicated for monitoring and measurement purposes. They should be placed within the network topology to give the perspective of different groups of users. If users reside in multiple geographic sites, each site should have at least one user simulation workstation; monitoring in this fashion will identify availability exposures that are unique to a particular site.

Custom Monitoring Agents

Many management and monitoring products provide agents - code that runs on the monitored hardware or interfaces to the monitored software - that can be used to provide the status of the monitored component. These ***custom monitoring agents*** connect to the managing software, which usually runs on a separate physical operating system platform.

The monitoring agents implement one or more of the monitoring techniques described earlier in this section. They may be implemented using open protocols, or using proprietary methods specific to the monitor or management software product.

The agent reports status to the management portion of the software. The availability status notification can be done in two ways:

1. Direct: the agent has the ability to explicitly report to the manager when a function in the monitored component changes status. The agent generates a notification (message, alert, trap, or proprietary datastream) to the managing software.
2. Indirect: The agent does not explicitly notify the manager of availability status. The manager expects to hear from the agent at some regular interval (for example, the agent is providing performance information about the monitored product). If the agent reports but has no data, or doesn't report at all, the manager can use this as an indication that the component is no longer available until it starts receiving data again. The managing software may record these status changes in a file, or create a notification (message, alert, or trap) that other software can detect.

Using Custom Monitoring Agents for availability monitoring and measurement

Considerations for using custom monitoring agents depend upon the type of monitoring the agent supports. The agent will implement at least one of the heartbeat, ping, remote command, or user simulation monitoring techniques; whichever are implemented will have the considerations as described earlier in this section for that technique.

The greatest hindrance to using monitoring agents is lack of skills on or knowledge about the agent - or management software that uses the agent - in terms of its use within the availability management process. The most important activity of a measurement project can be investigating the management software and associated agents already deployed in the installation for availability functions. The following questions are useful in determining if the management software and associated agents can be used to monitor and measure availability:

- Is the agent fully enabled on the components it is monitoring?
- Does the agent provide status notification to management software?
- Do the status notifications indicate the availability state of the component?
- Can this status notification be captured by or forwarded to other software for consolidation?
- Can the status notification be reported in log, message, alert, or trap format?
- Can the agent interface to management software that can support many different types of agents?

Data Capture and Monitoring Products

An installation can, as outlined in the preceding sections, implement their own routines for capturing availability data, either from data sources or by writing monitoring techniques. However, there are many products than can be used to produce, capture, and collect availability data for I/T components. In most cases these products use a subset of the system functions and monitoring methods that have been described earlier in this document. Some products also provide additional functions that can be used to capture availability information.

The variety of technologies used in modern applications makes it almost impossible to find a single product that can capture all required data from all sources on all components. With the number of components, the number of protocols, and the different technologies , one product will not have all the functions to interface to every type of component that has to be measured.

There will be products that can “aggregate” availability event information from other products. Instead of trying to capture information for each component, these “aggregation” products will interface to many products that management or monitor specific components, and can extract the desired availability information from them. They can act as "consolidation points"; that is, other products can forward component availability information to them, and they can consolidate or aggregate the information into a single view or repository.

Products containing data capture functions will fall into five major categories:

1. ***Event Management***. These products provide ability to capture "events" - an event being anything to indicate the change in status of a particular resource- from multiple sources. The events can be captured from the sources described earlier in this paper - logs, messages, alerts, and traps. The event management product can also provide agents to capture events from additional sources. Since availability status events are a subset of all events, these products can be an excellent source of availability status information. Event management products can also provide automation to extract the desired information from the event and store the information in an appropriate repository for further analysis and reporting.
2. ***Automation.*** These products can interface to the message flow of the platform they execute on, and may be able to integrate with a network manager to capture alert, trap, or other network information. Some products focus on remote console access/automation by using a workstation to directly connect to the console port of a component, and running software on the workstation that can capture information from the console and take a programmed action. The products may also have data manipulation functions to extract the required availability information and format it to be processed by a report generator program, or feed the data directly into a database.
3. ***Performance Monitoring***. A component must be available before its performance can be measured. Performance monitors report on the performance health of the component, as well as performance problems the component encounters that may be causing delays that are causing unavailability to users, or which may be to a component unavailable state. The absence of

performance data from a component (or some aspect of the component) can be an indication that the component (or some aspect of the component) is unavailable.

4. ***Application Monitoring and Management***. These products focus on the management of specific applications, or application response time. They provide agents that interface with the application (or are even incorporated into the application code directly) to monitor and control its health. They can provide application availability, performance, and response time information. In some cases the products can forward this information to automation or enterprise management products.
5. ***Enterprise (Systems and Network) Management***. The "management" aspect as related to availability means "what is used to monitor the state of this resource, and what is used to control it (change its state)?" A management product that performs these functions can capture the changes that have occurred, so that both current and historical availability data is accessible. These products many contain some or all of Event Management, Automation, Performance Monitoring, and Application Monitoring functions.

Most installations have at least one (and usually more than one) product that falls into one or more of these categories, and which can be potentially used as part of this process. Further investigation of the usefulness of product must include considerations such as:

- The ease of use of the availability-related functions.
- The capability to provide availability related information real time, and /or from a log or file.
- The ease of integrating the product within an automated process for monitoring, collecting, and reporting availability information.
- The skills needed to implement the functions, and the commitment of the installation to build and maintain skills on the product for the long term.

Example Products

The products mentioned in this section are listed to illustrate the type of capabilities that exist. This is not meant to be a complete list. They are included to provide guidance for those unsure of the type of products that should be considered. For every product mentioned there may be several others that provide similar capabilities.

Only the product functions relating to availability monitoring or data capturing are described. Covering all the functions a product provides is beyond the scope of this document. All of the products have general information documents that can be obtained from IBM or Tivoli.

AS/400 Management Central

AS/400 Management Central is part of the OS/400 operating system (V4R3 and later) that extends the iSeries and AS/400 Operations Navigator function to manage multiple iSeries and/or AS/400 systems using TCP/IP. It provides these availability data capture and monitoring related functions:

- Allows remote commands to be issued to AS/400 systems to determine their status.

- Monitors system performance for iSeries and AS/400 systems, and allows thresholds to be set that, when exceeded, can log the event, and do the same when the threshold returns to normal.
- Thresholds can invoke commands when they are exceeded or return to normal, such as sending an alert or trap.

CICSPlex System Manager (CICSPlex SM)

CICSPlex SM is part of the CICS Transaction Server product. It provides a "single point of control" for multiple CICS address spaces across multiple z/OS, OS/390 , VSE, and distributed platforms. It contains a system availability monitoring function that detects when a CICS address space becomes unavailable (due to stall, shutdown, address space/transaction dump, MAXTASKS, etc). Notification when these conditions occur, or when an existing condition returns to normal, can be done via a message or alert.

IBM Communications Server (AIX, Windows NT/2000, OS/2 Warp, Linux)

The IBM Communications Server product provides SNA gateway server functions that include a NetView Service Point application. It can accept RUNCMD invocations from Tivoli NetView for OS/390 for execution on the Communications Server platform (or attached clients), and return the results of the commands to NetView. It can also issue alerts to NetView for certain server or client workstation status changes.

IBM Director

IBM Director is a workstation management product provided with IBM Netfinity xSeries Servers. It provides the following availability data related functions:

- Logging or sending alerts on status changes such as system and application startup and shutdown. Alerts that are logged can be exported to a text file for processing.
- Receiving events from managed workstations running the Universalbility Management Services (UMS) agent. These events can then be sent as traps to any SNMP manager.
- A heartbeat function to detect workstation availability status changes.
- Sending remote commands to managed workstations for execution.
- Setting threshold for a variety of status and performance monitors, which can be set to trigger automated actions.

System Automation for OS/390 (SA for OS/390)

SA for OS/390 is an automation product that runs within the Tivoli NetView for OS/390 environment. It provides automation and control of z/OS and OS/390 software and hardware components, primarily address spaces, ESCON devices, and hardware. For the components it automates it will issue status

messages, which can be captured from the SYSLOG or from automation routines, so that the appropriate availability information can be captured.

System Manager for AS/400 and Managed System Services for AS/400

These products provide automation and performance monitoring for stand alone or SNA interconnected AS/400s. The availability related data functions include:

- Issuing messages or alerts based on performance exceptions from one or more connected AS/400 systems. These can be issued both when the exception(s) occur and when the exception condition returns to normal.
- Sending remote commands to AS/400 systems.
- Providing customer monitoring through Managed System Services for AS/400 functions.

Teleprocessing Network Simulator (TPNS)

TPNS runs on z/OS or OS/390 and it used to develop scripts that perform user simulation against application functions. These scripts can be used to verify if the application is available; the application status can be recorded for future retrieval and processing.

Tivoli Application Performance Management (TAPM)

TAPM provides availability and performance measurement at the application transaction level. It can be used to implement user simulation monitoring. The availability data related functions include:

- The Application Response Measurement (ARM) API to monitor application response time and availability.
- Tools to monitor availability and response time of client server and web based applications.
- An agent that captures the monitoring output, logs the results, and sends a notification to Tivoli Enterprise Console and Tivoli Distributed Monitoring.

Tivoli Business Systems Manager (TBSM)

TBSM monitors and controls components in the z/OS, OS/390, and distributed environments. It can interface directly to availability data sources (such as z/OS and OS/390 SMF logs and console messages), and integrate with products that perform event management and performance monitoring (such as Tivoli NetView, Tivoli Enterprise Console, and Omegamon Monitors) to obtain availability related information. TBSM can also issue commands against any monitored component.

TBSM can interface with a large number of data sources and management products, and can aggregate the events and show components from a hierarchical (physical connectivity) view, and a business system (application connectivity) view. The events and views are stored in a database. Because it can

build business views and store the related event information in a repository, it can provide real time views and long term reports on both component and application availability.

Tivoli Distributed Monitoring

Tivoli Distributed Monitoring provides monitoring and automated notification for resources on various application platforms. The availability data related functions include:

- Agents that reside on the monitored platforms to check system, resource, and application-specific performance or thresholds.
- Logging events, or sending events to Tivoli Enterprise Console, when the availability status of a monitored system, resource, or application changes is threatened or changes.
- Taking a predefined action , such as running a command, based on a defined threshold.
- Support for monitoring agents that are developed by the installation.

Tivoli Enterprise Console (TEC)

TEC is an event management and automation product for capturing and correlating system and network events. The availability data capture related functions include:

- Capturing and correlating status events from a variety of sources, such as:
  - Distributed System and Network management products (Tivoli NetView, HP OpenView BMC Command Post, etc.).
  - Tivoli Distributed Monitoring
  - Log files
  - Tivoli Manager for... Products
  - SNMP traps
  - Tivoli NetView for OS/390
- Correlating events from these sources to better determine which outage events are causes and which are symptoms of other outages.
- Invoking programs against the captured availability information, to transform the data into a common format for report processing.
- An automation engine to carry out automated actions based on a notification or a set of notifications received.
- Integration with Tivoli NetView for z/OS and OS/390. TEC can send events to NetView, and NetView can send messages and alerts to TEC.

Tivoli NetView (AIX ,NT/2000 platforms)

Tivoli NetView, running on AIX, NT, or 2000 platforms, provides network management of TCP/IP resources. It monitors IP resources, IP network topology, and is a SNMP Manager. Its availability data capture functions include:

- Providing event automation for traps received from TCP/IP SNMP components in the network. The product can invoke programs or command procedures based on the content of a trap.
- Invoking programs or procedures to process captured information into a desired format and store results in a log or file for later processing..
- Providing two-way communication with Tivoli NetView for OS/390:
  - Sends Tivoli NetView on OS/390 TCP/IP topology status information.
  - Interfaces with the AIX NetView Service Point product to accept RUNCMD UNIX or TCP/IP command invocation from Tivoli NetView on OS/390, execute them, and return the result.
- Providing an application platform and APIs to support applications and provide management (including monitoring/capture of availability information) for non-SNMP managed components for both UNIX and non-UNIX environments.

The NetView Mid-Level Manager component of Tivoli NetView provides detailed performance exception and event forwarding/consolidation for SNMP components and AIX, SUN, HP, and NCR UNIX environments:
- Notifications can invoke command/automated responses.
- Notifications can be forward as traps to Tivoli NetView.

Tivoli NetView for z/OS and OS/390

Tivoli NetView for z/OS and OS/390 provides Enterprise Management, Automation, and Event Management and Correlation for the z/OS and OS/390 platforms, including network connecting SNA, APPN, and TCP/IP components.

Note: The NetView products for VM and VSE have been stabilized to the NetView for MVS V2R3 level of functions. For the iSeries and AS/400 platforms, a lot of the NetView function is built into the OS/400 operating system; these functions can be used to managed interconnected iSeries and AS/400s systems, and devices that are attached to them (local or remote).

The availability related data functions of Tivoli NetView for z/OS and OS/390 include:
- Automation for system messages and SNA alerts; availability status indicators from these sources can be captured and analyzed by automation procedures.
- SNMP management functions to receive SNMP traps and, optionally, convert them to SNA alerts.
- Converting SNA alerts to SNMP traps.
- The RMTCMD function to execute remote commands on another network connected NetView z/OS or OS/390 platform.
- The RUNCMD function to execute commands on a non-SNA platform, via a Service Point application.
- Automation procedures that can process information into a desired format and store it:

  - In a file for later processing.
  - Directly into a problem management/help desk product, as Tivoli Information Management, via the NetView Bridge function.
  - In its Resource Object Data Manager (RODM), which can contain object representation of components that other products can access and use for various purposes, including availability status.
  - In a DB2 database for later processing.
- APIs such as the NetView Program-to-Program interface (PPI) to allow further integration with other applications.
- Integration with Tivoli Enterprise Console (TEC). TEC can send events to NetView, and NetView can send messages and alerts to TEC.

Tivoli NetView Performance Monitor (NPM)

NPM provides performance monitoring of SNA and TN3270 TCP/IP sessions into a z/OS or OS/390 host. The availability data capture functions it provides includes:

- Setting performance or availability thresholds against the resources it monitors. When these thresholds are exceeded (or when exceeded thresholds return to normal), NPM can create an event and log it in its repository, or generate an console message or SNA alert that can be captured and analyzed by an automation product.
- Recording information on SNA session start/stop times (including LU 6.2 sessions).
- Interfacing with NetView and LAN Network Manager to analyze token ring segment and utilization statistics, and generating alerts when performance exceptions are detected.

Tivoli NetView Performance Monitor for IP (NPM/IP)

NPM/IP provides performance monitoring of TCP/IP sessions (TELNET, FTP, HTTP, etc.) into the z/OS or OS/390 TCP/IP stack, and of remote TCP/IP components from z/OS or OS/390. It can set performance or availability thresholds against the monitored monitors. When these thresholds are exceeded, NPM/IP can create an event and log it in its repository, or generate a console message that can be captured and analyzed by an automation product.

Tivoli Web Component Manager (TWCM)

TWCM provides availability and performance monitoring of Web Server components, primarily the HTTP Server and associated Application Server, such as WebSphere Application Server. The availability data capture functions include:

- Monitoring the HTTP Server and Application Server functions and generating events when their availability changes.

- Monitoring HTTP Server and Application Server performance attributes against defined thresholds., and generating events when those thresholds are violated, or when violated thresholds return to normal.
- Forwarding events to the Tivoli Enterprise Console.

Tivoli Web Services Manager (TWSM)

TWSM provides availability and performance monitoring of Web Server applications. The monitoring is done outside of the web server, so any web server platform can be monitored. The availability data capture functions include:

- Monitoring web server pages to detect broken links and missing pages, which can indicate availability problems.
- Monitoring web server end user response time against defined thresholds.
- Providing user simulation functions to issue web application transactions and capture the transaction availability and response time information, and compare it against defined thresholds.
- Generating events to Tivoli Enterprise Console when thresholds are exceeded, and when they return to normal.

Product Mappings

The following table lists the products described in the previous section and, for each product, identifies the specific data source or monitoring technique the product supports or interfaces with. This type of table is useful to build when evaluating products being considered for this task; it can easily show the scope a product can be used within the process of finding and collecting availability measurement data.

Product	Logs	Messages (sends or detects)	Alerts (sends or receives)	Traps (sends or receives)	Heartbeat	PING	Remote commands	RUNCMDS (sends or responds to)	User simulation	Custom monitoring agents
AS/400 Management Central	Yes			Both			Yes			
CICSPlex SM		Sends	Sends		Yes		Yes			Yes
IBM Comm Server			Sends					Responds		
Microsoft SNA Server			Sends					Responds		
Netware for SAA			Sends	Sends				Responds		
IBM Director	Yes			Sends		Yes	Yes			Yes
Omegaview			Sends							Yes
System Manager for AS/400	Yes		Both				Yes			Yes
TPNS									Yes	
Tivoli Application Performance Mgmt	Yes				Yes				Yes	Yes
Tivoli Business Systems Manager	Yes	Detects		receives (via TEC)						Yes
Tivoli Distributed Monitoring	Yes			Sends		Yes				Yes
Tivoli Enterprise Console	Yes	Both		Receives						Event Adapters
Tivoli NetView	Yes		Sends	Both	Yes	Yes	Yes	Responds (via AIX Service Point)		Yes
Tivoli NetView for OS/390	Yes	Both	Both		Yes	Yes	Yes	Sends		
Tivoli NetView Performance Monitor	Yes	Sends	Sends							
Tivoli NetView Performance Monitor for IP	Yes	Sends				Yes				
Tivoli Web Component Manager	Yes	Sends (to TEC)	Sends (via TEC)		Yes		Yes			Yes
Tivoli Web Services Manager	Yes		sends (via TEC)	sends (via TEC)		Yes	Yes		Yes	Yes

Reporting

Once the availability data has been captured, two steps have to be done:
1. The data must be formatted into a common record layout, regardless of where it was captured.
2. Reporting logic must be applied against this data to create the desired reports.

Creating a Common Record Layout of the Captured Data

A common layout was described earlier in this paper:

1. A component identifier that uniquely identifies this component.
2. The component status that is being reported. This can be anything that, for measurement purposes, can be mapped to a "UP" or "DOWN" state for the component.
3. The date and time that the reported status occurred.
4. The status change identifier.

Each availability status event captured should have this information (or, the information can be provided by the mechanism that captured the event - an automation product, for example). Programming logic is then applied to extract the appropriate information and place it in the above layout.

Here are two examples of the process:

Example 1: The component is a z/OS or OS/390 sysplex image. It is considered available within the sysplex when the following message appears in the SYSLOG:

```
01219 01:22:16.97  IXC418I SYSTEM SYSA IS NOW ACTIVE IN SYSPLEX PRODPLEX
```

The information to be extracted is:
- The component name (SYSA)
- The component status (since the IXC418I means it is available, the status will be “UP”)
- The date and time the reported status occurred (01219 is cycle date 219 in 2001, which is August 7th).
- The status change identifier (The message ID, IXC418I)

An automation product that captures the message when it is issued, or a program that scans the SYSLOG for this message, then extracts the information so that the common layout is:

```
20010812 0122 SYSA UP IXC418I
```

The programming logic has reordered the fields to the sequence date-time-component ID - availability status - status change identifier. It has also reformatted the date and time values to formats that are easier for report processing.

Example 2: The component is a HTTP Server running on a Linux platform with a host name of HASL13. It is monitored by sending a remote command to it **(ps -ef | grep httpd).** If the HTTP server process is up, the following response is expected (as one line)

```
root       640     1  0  2001 ?        00:00:03 /opt/IBMHTTPServer/bin/httpd
-f /opt/IBMHTTPServer/conf/httpd.conf
```

The information needed for the common layout is not directly contained in the response. Much of it will have to be derived:

- The component name (to uniquely identify this process, the name hasl13.httpd.80 is assigned).
- The component status (since the command response indicates the httpd process it is running, the status will be “UP”).
- The date and time the reported status occurred (the date and time the remote command request was issued will be used).
- The status change identifier (the host name of the platform where the monitoring is running is HASL02, so the identifier name assigned will be hasl02.rmcd).

The function performing the remote command monitoring would derive the additional information needed, and either create the common format, or write the information to a file, where it can later be processed into the common format. In either case, the end result appears as:

```
20010812 0122 hasl13.httpd.80 UP hasl02.ping
```

The goal of using a common layout is the make all availability status information, regardless of the source, have the same format and fields. This allows efficient report processing of the data.

Once in a common record layout, the data can be stored in a repository of choice. This can be a sequential file, a relational database, or a reporting program repository.

Applying Reporting Logic Against the Formatted Data

A number of options exist for creating reports from the data. They range from custom programs, to spreadsheets, to SQL queries, to reporting products. Which is used depends upon the skills in an installation and long term usage of the report. In the short term a program or spreadsheet may be able to produce reports quickly, but as reporting needs changed and more flexibility is needed (such as correlating availability data with other data), a reporting product, or report function within a product, may be a better option.

Reporting products provide functions to combine and manipulate the data to produce meaningful reports. Some specialized systems management products contain a a reporting function that may be flexible enough to use for this purpose. One of advantage of using a reporting product is that multiple types of data from multiple processes can be kept in a single repository. This allows,

in addition to component and application availability reporting, any trends with other management data (such as problem, performance, or change) can be investigated.

Other products may produce availability reports for specific environments. For example, a product may only provide availability reports for SNA network components.

Availability reporting software can run on any operating system platform. When the information to be processed is not created on the same platform, procedures must be established to move the information to the platform with the reporting software, or allow remote data access from the platform where the reports are being created. Additional products may be required to do this.

Depending on the types of reports desired, the reporting function may have to do further data transformations beyond the common format described above. For example, suppose the availability data for a component has been formatted into this common format:

```
20010306 1700 COMPONENT_ID_1  DOWN  * SOURCE_A
20010306 1800 COMPONENT_ID_1  UP    * SOURCE_A
20010309 1530 COMPONENT_ID_1  DOWN  * SOURCE_A
20010309 1610 COMPONENT_ID_1  UP    * SOURCE_A
```

This format is sufficient to provide a report for the component over a particular time range.

Suppose a report is needed to show the amount of available or unavailable minutes for the component on a daily basis. Transforming the data into the following format will enable that type of report to be produced:

```
20010306 COMPONENT_ID_1 UP   0000 1659 1020
20010306 COMPONENT_ID_1 DOWN 1700 1759   60
20010306 COMPONENT_ID_1 UP   1800 2359  360
20010307 COMPONENT_ID_1 UP   0000 2359 1440
20010308 COMPONENT_ID_1 UP   0000 2359 1440
20010309 COMPONENT_ID_1 UP   0000 1529  930
20010309 COMPONENT_ID_1 DOWN 1530 1609   40
20010309 COMPONENT_ID_1 UP   1610 2359  470
```

The transformation has created a record for each component available and unavailable period for each day, and the length of that status period. Using this format an availability report for a particular day, or range of days can be created.

Data Accessibility

It is easy to look at measuring and reporting on availability as an isolated task. However, availability is influenced by, and influences, other processes such as problem management, change management, performance management, configuration management, etc. The data available from those processes can shed more light on why the current availability is where it is and, if necessary, where efforts to improve

availability can be focused. Therefore, availability measurement data should not be isolated from data created by other systems management processes. The data captured by the methods discussed in this document will be primarily component names and outage start/stop times. Information such as outage status (who is working on it or who has been notified) and outage root cause have to be obtained from other processes.. This moves the availability measurement process beyond simply reporting availability to analyzing trends and identifying actions to take to avoid or minimize outages (which is why availability is managed in the first place).

Placing this data alongside of problem, change, and performance data in the same logical repository allows information from these processes to be correlated show cross-impacts such as:

- outage categories and root causes
- performance trends and the resulting availability impact
- change activity and the resulting availability impact

This will better identify activities to address both short and long term availability improvement actions. It is more difficult to do when the measuring and reporting of availability is implemented as a separate, isolated process.

Reporting Products - Examples

Selecting a product for reporting availability measurements is influenced by the following:

- The existing use of the product for reporting systems management information.
- The programmability of the product to analyze and manipulate data.
- The ease of producing reports from different sources of data at the desired frequency. Some basic reports include:
  - Component and application availability for a time interval
  - Availability interrupts within a time interval
  - Component availability impact on application availability for a time interval
- The usefulness of "canned" availability reports provided by the product.
- The ability to create both individual component reports and end-to-end availability reports (availability that is derived from the status of multiple components).
- The flexibility of data interchange with other systems management processes, to get a view of what actions are impacting availability, and the impact of availability on business functions.

Spreadsheets and relational database products are also options for creating reports. These require programming skills in the spreadsheet macro language or SQL to create the reports. In addition, external programs may have to be written to perform any needed data transformations before importing the data into a spreadsheet or relational database.

The following products are examples of what can be used to get started with reporting availability. This is not an exhaustive list. It is presented as a guide to illustrate the type of products and offerings that are available and that should be investigated. It also does not cover the full range and breath of individual

product functions, but highlights how the product can be used to provide availability measurement reports.

Tivoli Decision Support for OS/390

Tivoli Decision Support for OS/390 (previous names were Performance Reporter for MVS and Enterprise Performance Data Manager) is a data analysis and reporting product. It collects data from availability sources (SMF, logs, output from other products, files, etc.) and stores it in a DB2 repository, where data elements can be related together to generate various historical and trend reports (performance, problems, accounting, etc). Many types of canned reports, including component availability reports (using data from systems, networks, or user-created sources) are provided. It also provides programming function to create installation-specific reports.

Tivoli Decision Support

Tivoli Decision Support interfaces to many of the Tivoli management products to produce various reports on the resources those products manage. Information from these products can be integrated with data from external sources to produce various reports, including availability related reports. Guides are provided for creating availability reports from Tivoli Enterprise Console event data.

Tivoli NetView for OS/390 Automated Operations Network (AON) component

AON can produce availability reports on the network components it is monitoring and automating.

Tivoli Service Desk for OS/390 (INFOMAN)

Tivoli Service Desk for OS/390 (INFOMAN) supports Problem, Change and Configuration Management processes. Based on the type and content of problem information being collected, availability reports can be developed using the Tivoli Service Desk for OS/390 reporting functions.

Putting It All Together

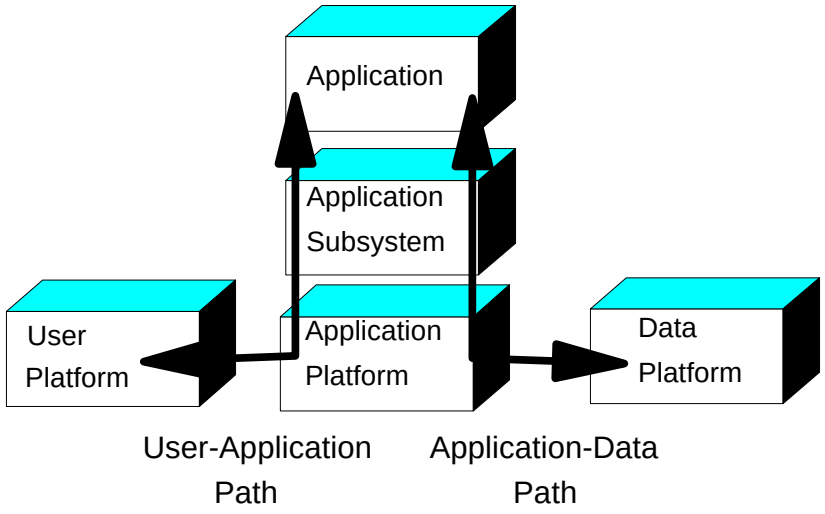
The preceding sections have identified many data sources and monitoring techniques for obtaining component availability data. However, a frame of reference is still needed to understand how to apply this to realistic environments. There is no single "right" answer for
how to go about doing this, but the guidelines and examples in this chapter will provide information that can be built upon to address specific environments.

The more components that are monitored, the better a true picture of availability, particularly end-to-end availability, can be determined. It is helpful to understand the types of components should be monitored, and where they are "logically" located, so that an appropriate data collection or monitoring technique can be used. The white paper "**Measuring End-to-End Availability: How To Get Started",** covers the overall method in greater detail. The foundation to establish includes the following:

- Use the premise that users use applications to manipulate data. Applications can only do what is requested of the users, and the users cannot get to the requested data without the application.
- Paths must exist between the users and the application components, and the application and data components. These paths themselves are made up of components, which must be monitored to ensure that the paths are working as desired.
- Take the application view of the environment - that is, do not look at measuring all of the components, because that will seem overwhelming.. Look first at the critical applications, and then at the key components those critical applications require.

Data Source and Monitoring Technique Selection Guidelines

The following diagram depicts a simple generic model that can be applied to any application environment. The model categories help identify the components that provide application availability to end users and data availability to applications (The details of this model, and the overall steps to for measuring end-to-end availability - for which finding and collecting data is one step - will not be covered here. This information is contained in the "**Measuring End-To-End Availability: How To Get Started"** white paper). Each component will map to one or one of the seven categories that are depicted here:



For each category, certain data sources and monitoring techniques will be more applicable to others. The following table depicts this on a scale of 1 to 4:

Model Area	Logs	Msgs	Alerts	Traps	Heartbeat	PING	Remote Cmds	RUN-CMD	User Sim.	Custom Monitoring
End User Platform	3	3	1	1	2	1	2	2	1	1
User to Application Path	2	2	1	1	4	2	3	3	4	3
Application Platform	1	1	3	3	1	1	2	2	2	4
Application Subsystem	1	1	3	3	3	2	1	1	2	2
Application	4	3	2	2	3	3	1	1	1	1
Application to Data Path	2	1	1	2	4	2	3	3	4	3
Data Platform	2	1	1	2	4	3	2	2	4	3

The table recommends the appropriateness of a data source or monitoring technique (columns) for components mapped to the model categories (rows), relative to other techniques. The numbers should be interpreted as follows:

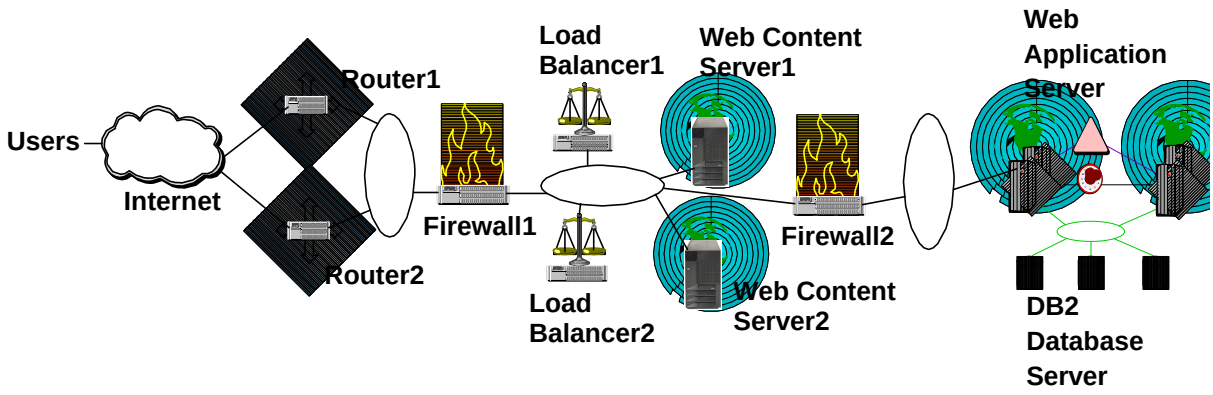
- **1:** This is a very likely source of availability data, and should be investigated.
- **2:** This is a somewhat likely source of availability data.
- **3:** This is a unlikely source of data; it is possible, but may require a lot of effort.
- **4:** This is a very unlikely source of availability data; it should only be tried if nothing in a higher ranked source of data for the component can be found.

The table is a guide, showing the relative merit or ease of using a particular source or monitoring technique for components within a particular model category. With enough effort almost any source could be used for any component. The table numbers can change based on the context of the environment they are applied to - in other words, your mileage may vary.

An installation can access the data sources or use the monitoring techniques via the functions provided in the appropriate operating system or network protocol. Or, they can investigate monitoring and management products, since these are likely to contain the monitoring methods and provide access to the data sources.

Application Example

The following diagram depicts the end-to-end components that support an application flow commonly seen today - e-business, or the integration of web and classic I/T technologies to build or enhance business applications:



1. The users access the application through the internet.
2. Router1 and Router2 connect to the internet.
3. Firewall1 acts as a secure barrier between the internet and the demilitarized zone (DMZ), where the web content servers are.
4. Load Balancer1 and Load Balancer2 spread the user requests among the Web content servers.
5. Web Content Server1 and Server2 receive the user requests and provide static HTTP content. Dynamic content is obtained by connecting to Web Application Servers running in z/OS parallel sysplex.
6. Firewall2 acts as a secure barrier between the DMZ and the Trusted intranet.
7. The Web Application Server runs in multiple address spaces in the parallel sysplex. As long as one image in the sysplex is running, the Application server can receive requests from the Web Content Servers, and retrieve information from the DB2 database.

8. The DB2 database subsystem runs in data sharing mode in the sysplex. As long as one image in the sysplex is running, the DB2 database can receive requests from the Web Application Server, and provide the appropriate data.
9. The communications protocol used across the application is TCP/IP.

The application components span multiple operating systems and the network. The management products in use in this environment are:

- Tivoli NetView for OS/390, running Systems Automation for OS/390, for automation and event management of the parallel sysplex.
- Tivoli NetView, to monitor the network IP resources.
- Tivoli Enterprise Console (TEC), with Tivoli Distributed Monitoring (DM), to monitor the distributed operating system platform resources.

Using these products, and customized monitoring techniques, the following table describes how availability information for each of the major components is collected:

Component	**Model Category**	**Source or Technique**	**Details**
Router1, Router2	User-Application Path	SNMP traps	NetView will monitor the routers and forward status traps to TEC.
Firewall1 Platform	User-Application Path	PINGs	NetView will monitor the firewall platform and forward status traps to TEC.
Firewall1 Application	User-Application Path	Tivoli DM process monitoring	Tivoli DM will monitor th firewall application (process), and forward status changes to TEC.
Load Balancer1, Load Balancer2 Platforms	User-Application Path	PINGs	NetView will monitor the load balancer platforms and forward status traps to TEC.
Load Balancer1, Load Balancer2 Applications	User-Application Path	Tivoli DM process monitoring	Tivoli DM will monitor th load balancer application (process), and forward status changes to TEC.

Component	Model Category	Source or Technique	Details
Web Content Server1, Web Content Server2 platforms	Application Platform	Heartbeat	A Heartbeat program running on the platform will record startup and shutdown times in a file. The file will be uploaded daily into the data repository.
HTTP Servers on Web Content Servers	Application Subsystem	Tivoli DM process monitoring	Tivoli DM will monitor th HTTP Server process, and forward status changes to TEC.
Servlet on Web Content Servers	Application	User Simulation	Tivoli NetView on OS/390 will issue a command to the servlet at 1 minute intervals; the command response (or lack of one) will indicate the Servlet is available or unavailable). The results will be placed into the data repository.
Firewall2 Platform	Application-Data Path	PINGs	NetView will monitor the firewall platform and forward status traps to TEC.
Firewall2 Application	Application-Data Path	Tivoli DM process monitoring	Tivoli DM will monitor th firewall application (process), and forward status changes to TEC.

Component	Model Category	Source or Technique	Details
Parallel Sysplex	Application Platform	Logs (SYSLOG)	The SYSLOG will be processed daily to collect the messages that show when the sysplex was available (at least 1 system active) or unavailable (no systems active). The process will calculate the availability times and store the information in the data repository.
Web Application Server Address Spaces	Application Subsystem	Messages	System Automation for OS/390 is automating the startup and shutdown of the Web Application Server. SA for OS/390 status messages will be saved and the availability information extracted and placed into the data repository.
Servlets on Web Application Server	Application	Remote Command	Tivoli NetView on OS/390 will issue a command to the servlet at 1 minute intervals; the command response (or lack of one) will indicate the Servlet is available or unavailable). The results will be placed into the data repository.
DB2 subsystem (Address Spaces)	Data Platform	Messages	System Automation for OS/390 is automating the startup and shutdown of the DB2 subsystem; its status messages will be saved and the availability information extracted and placed into the data repository.

Component	Model Category	Source or Technique	Details
DB2 Database	Data Platform	Remote Command	Tivoli NetView on OS/390 will issue a command to DB2 requesting the status of the database at 1 minute intervals; the command response (or lack of one) will indicate if the database is available or unavailable. The results will be placed into the data repository.

Functions to create the formatted data layout to run reports can be implemented in two ways:

1. Before the information is sent to the repository. In this example, these functions would reside
   - At the TEC, for the availability status events it receives.
   - At Tivoli NetView for OS/390, for the messages and remote commands it issues.
   - On the parallel sysplex, when the SYSLOG is processed.
   - On the Web Content Servers, where the Heartbeats run.
2. Where the data repository is. The repository would receive the data as forward to it, and functions would then create the common layout, taking into consideration where the data was sent from.

Once the data was in a common format, the reporting function of choice could be used to create availability reports.

Summary

There are many existing sources of data, produced by systems, networks, and monitoring techniques, that can be used to determine the availability of individual components. Putting individual component availability measurements together will provide the ability to accurately measure end-to-end availability - the availability experienced by application users.

It is not impossible to collect this data, but careful planning data collection is needed. The use of automation will provide great benefits when collecting information from a wide range of sources, and will eliminate much of the manual drudgery and potential errors associated with collecting the data.

No single product will do everything; some products will help aggregate data source collection or monitoring from many sources. Again, careful planning is needed to identify the applications, the key components, the tools or techniques to measure them, the products to use. and the process to gather, relate, and report the measurements. Customization of products and data will be required.

This process does not normally require investment in additional products . From the authors' experience, most installations already have all or nearly all the products they need to start collecting the data. However, the product functions needed may not be enabled, are not accessible by the organization that would have to use them, or have not been taught to the organization. These issues must be addressed before collection can begin. At times, due to these issues, the installation has elected to invest in additional software primarily for the purpose of collecting availability data.

The measurements based on the collected data may show lower availability than what was reported before. This happens because:

- Previous manual measurements were not capturing all of the outage incidents accurately.
- With additional components being measured, true overall availability will be less that individual component availability.

For example, suppose an application requires three key components. Each component reports availability of 98%. In the simplest scenario (components are completely independent of each other, and no backup/redundant components exist), availability of the application would be 94.1%. If the application availability measurement was previously based on a single component, the new measurement will be lower... but more accurate. In addition, the focus is on improving availability. Accurate measurements will accurately identify where improvement actions are needed. They will be better suited for the availability management process, to assist in identifying and addressing causes of unavailability.

Finally, this is only one - and not the only - action to take to move closer to improving availability. Finding and collecting availability data is a subset with **Availability Management**. Identifying sources of component data, collecting the data, and reporting the measurements does not, by itself, improve availability (as stated above, it may show availability to be worse than it was thought to be). However, it will provide valuable information for guiding and focusing on

where availability improvements are needed, as well as reveal the effectiveness of actions taken to improve availability.

Appendix

The following examples are contained in this section:

- z/OS and OS/390 system and subsystem messages
- z/VM Messages
- Communications Server Messages
- OS/400 Messages
- AIX Messages
- Sample REXX heartbeat program
- Sample UNIX shell script heartbeat program
- Sample Java Servlet heartbeat program

The messages are not meant to be an exhaustive list, but to illustrate the type of availability information that is available. Always check the messages documentation for the
version of the operating system or product being used, for the most accurate and detailed information.

z/OS and OS/390 Messages

Message ID	Component	State	Description
IEA371I	Platform	Available	The system has started to IPL. The time stamp o the preceding messages are a good indication of when the system became unavailable.
IEE389I	Operating System	Available	The operating system is ready to being executin workloads.
IXC418I	Sysplex image	Available	An image joins the sysplex.
IXC101I	Sysplex image	Unavailable	An image is being removed from the sysplex.
IXC105I	Sysplex image	Unavailable	Ian image has been removed from the sysplex.
IEF403I	Address space	Available	An address space is started
IEF404I	Address space	Unavailable	An address space has ended.
IEF450I	Address space	Unavailable	An address space has abended.
DFHSI1517	CICS address space	Available	A CICS address space is ready to process commands.
DFHKE1799	CICS address space	Unavailable	A CICS address space has been shut down
DFS994I	IMS control reigon address space	Available	An IMS control region is ready to being processing requests.
DFS629I	IMS control reigon address space	Unavailable	An IMS control region abends.
DSNR007I	DB2 subsystem	Available	A DB2 subsystem is ready to process applicatio requests
DSN3104I	DB2 subsystem	Unavailable	A DB2 subsystem has been shut down.
IMW3536I	HTTP Server address space	Available	The HTTP Server is ready to process requests.
IMW3541I	HTTP Server address space	Unavailable	The HTTP Server has been shut down.
IEE302I	Device	Available	The device is online and can be used.
IEE303I	Device	Unavailable	The device is offline and cannot be used.

z/VM Messages

Message	Component	Status	Description
AAA LOGON AS BBB	Virtual Machine	Available	Virtual machine AAA is active; BBB i its CP console.
AAA LOGOFF AS BBB	Virtual Machine	Unavailable	Virtual machine AAA is no longer active; AAA was its CP console.
AAA Varied ONLINE	Device	Available	Device AAA is online.
AAA VARIED OFFLINE	Device	Unavailable	Device AAA is offline.
AAA ATTACHED TO BBB BY CCC	Real or virtual device	Available	Device AAA has been attached to virtual machine BBB (CCC is the user, if done by someone other than the operator).
AAA DETACHED BBB BY CCC	Real or virtual device	Unavailable	Device AAA has been detached from virtual machine BBB (CCC is the user, if done by someone other than the operator).
HCP450W	Virtual machine	Unavailable	The virtual machine has loaded a disabled wait state PSW and has stopped processing.
HCP961W	Operating System	Unavailable	The system has been shut down.

z/OS and OS/390 Communications Server Messages

Message ID	Component	Status	Description
IST093I	SNA node	Available	The network resource is available
IST105I	SNA node	Unavailable	The network node is no longer active.
IST020I	VTAM address space	Available	VTAM is ready for processing.
IST133I	VTAM address space	Unavailable	VTAM is terminating.
EZB6473I	TCP/IP address space	Available	TCP/IP is ready for processing.
IST590I	SNA switched PU node (including LAN attached SNA gateways)	Available or Unavailable	Depending on the contents of th message text, this indicates whe the connection to the PU was established or terminated.

OS/400 Messages

Message ID	Component	Status	Description
CPF0934	Operating system	Available	Operating system IPL has completed; work cannow be processed.
CPF1124	Job	Available	A job has started processing.
CPF1164	Job	Unavailable	A job has completed processing.
CPF0927	Subsystem	Unavailable	A subsystem has ended and is no longer available to support jobs.
CPF5908	Controller, workstation (SNA PU)	Available	A session has been established with the controller or workstation.
CPI5935	Controller, workstation (SNA PU)	Unavailable	A controller (SNA PU) has failed; the system will try to re-establish contact to the controller.
CPF5909	Communications Link	Available	Communications is active and sessions can be established using the link.
CPI593D	Communications Link	Unavailable	Communications is active and sessions can be established using the link.
CPF6784	Device	Available	Device is available for use.
CPF6783	Device	Unavailable	Device is not available for use.

AIX Error Log messages

Message	Component	Status	Description
0F27AAE5 SOFTWARE PROGRAM ABNORMALLY TERMINATED	Process	Unavailable	An active process has ended due to an error.
22E93753 LINK ERROR	Communications link	Unavailable	An error has been received on a communications link; the lin
2BFA76F6 System shutdown by user	Application Platform	Unavailable	The system has been shut down.
9359F226 Physical volume is now active	Disk volume	Available	A volume that was previously inactive is now active and can be used.
A668F553 DISK OPERATION ERROR	Disk file	Unavailable	A problem was encountered during a disk read/write/seek operation.
DB3E3DFD CSMA/CSD LAN COMMUNICATIONS LOST	Ethernet LAN segment	Unavailable	The platform is unable to communicate across an Ethernet LAN segment.

Sample REXX Heartbeat Program

This program is an example of how heartbeat monitoring can be implemented for any platform that supports REXX.

```
/* EXEC TO CREATE SYSTEM HEARTBEAT TO TRACK REAL TIME STATUS

   SYNTAX: HBEAT SYSID TMINS TTYPE TSLOG AVADSN

           * SYSID  = SYSTEM ID (EXAMPLE: FOR MVS WE USE SMF ID)
           * TMINS  = FREQUENCY OF WRITING TIMESTAMPS
           * TTYPE = "IPL" - IF STARTED AFTER SYSTEM IPL
                     "AUTO" - IF STARTED BY  AUTOMATION
                     BLANK - PROBABLY STARTED MANUALLY
           * TSLOG = Name of "last timestamp" file
                     (ON z/OS or OS/390, DD Name of "last timestamp" file)
           * AVADSN = Drive and directory to keep log files
                     (On z/OS or OS/390, DD Name of availability log file)

    USAGE: THIS PROGRAM, ONCE STARTED, ALWAYS RUNS OR UNTIL A PLATFORM
           OUTAGE. THE PLATFORM SHOULD AUTOMATICALLY START THIS PROGRAM.
*/
ARG SYSID TMINS TTYPE TSLOG AVADSN  .

if right(AVADSN,1) <> '\' then AVADSN=AVADSN||'\'
AVADSN = AVADSN||"av"||left(date('S'),6)||".log"

say "starting TIMESTMP on" sysid
DO  FOREVER

    /*** STEP 1: create current date/time stamp: sysid yymmdd hhmm ***/

    TEMPSTAMP = TIME
    NEWDATE=DATE('S')
    NEWTIME=LEFT(SPACE(TRANSLATE(TIME(),' ',':'),0),4)
    CTS = SYSID newdate newtime

    /*** STEP 2: read old timestamp and update timestamp log
    with new timestamp  ***/

       OLDTS=Linein(TSLOG)
       Y=lineout(TSLOG,CTS,1)
       Z=lineout(TSLOG)

    /*** STEP 3: If ttype = IPL or AUTO, update availability log
    with new timestamp  ***/

    if (ttype = "IPL") | (ttype= "AUTO") then do
       avarec.1=left(word(oldts,1),40) "DOWN" word(oldts,2) word(oldts,3),
                "* HEARTBEAT_PROGRAM"
       avarec.2=left(sysid,40) left(ttype,4) newdate newtime "* HEARTBEAT_PROGRAM"

       X=lineout(AVADSN,avarec.1)
       Y=lineout(AVADSN,avarec.2)
       Z=lineout(AVADSN)

       ttype = ttype||"FLAGOFF"
    end
```

```
    /*** STEP 5: wait for TMINS minutes before continuing loop ***/

    /*** NOTE: For Windows NT/2000, segment wait time to allow for
         shutdown interrupt                                    ***/



    sleeptime = tmins*60
    sleep_segment = sleeptime / 5 /* Allow shutdown check every five seconds */
    do j = 1 to sleep_segment
       call SysSleep 5
       end

END
EXIT 0
```

Sample Shell Script Heartbeat Program

This shell script is an example of how heartbeat monitoring can be implemented for any UNIX based platform, including LINUX.

```
#!/bin/ksh
#
# HEARTBEAT availability monitoring program from the UNIX environment
#
# Syntax: hearbeat system_name interval start_type timestamp_file avail_file
#
# echo script name: $0
if [[ $# -ne 5 ]]
then
    echo SYNTAX: hearbeart system_name beat_interval start_type timestamp_file
availability_file
    exit 1
fi
scriptname=$0
sysname=$1
interval=$2
start_type=$3
tstamp_file=$4
ava_file=$5
echo
temp1=`date`
logger Starting HEARTBEART on $sysname at $temp1
#
#
#
while true
do
###############################################################################
curdate=`date +%Y%m%d`
newtime=`date +%H%M`
newts=$curdate" "$newtime" "$sysname
#echo time: $newtime
##
## If timestamp file exists, read it and write out new timestamp
##
test -r $tstamp_file &&  read oldts < $tstamp_file
echo $newts > $tstamp_file
#echo old time stamp: $oldts
##
## If type = IPL or AUTO, update availability log with both  old (down)
## and new (UP) Timestamp
##
if [ $start_type = "IPL" ]
then
     set -- $oldts
     outtype1="DOWN"
     outtype2="UP"
     printf "%-10s %-5s %-9s %-6s\n" $sysname $outtype1 $1 $2 >> $ava_file
     printf "%-10s %-5s %-9s %-6s\n" $sysname $start_type $curdate $newtime >> $ava_file
     start_type=$start_type"FLAGOFF"
fi
##
## Now pause for beat_interval (convert to seconds)
##
```

```
let beat_interval_secs="$interval * 60"
sleep $beat_interval_secs
temp1=`date`
done
```

Example Java Servlet “UpTime” Heartbeat Program

This is an example of how heartbeat monitoring can be implemented for any Web Server that supports a Java servlets via a servlet engine . The servlet engine (which can be part of a bigger application server product, such as IBM WebSphere Application Server) is normally started and shut down when the associated HTTP Server is started and shut down. This servlet code would be loaded at servlet engine startup. It will log the time that it started. HTTP requests to the server that invoke the servlet return the length of time the server has been running since it was last started. When the servlet engine shuts down, it will log the time of the shutdown.

```
/*UPTIME SERVLET - Does the following:

  - When loaded, writes a message to the System Log
  - When invoked, returns the amount of time since loaded (if loaded at
    WAS startup, this is the amount of time WAS has been active
  - When destroyed, writes a message to the System Log. If only destroyed
    when WAS stops, indicated that WAS is down
*/

import java.io.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UpTime extends HttpServlet
{
 long StartupTime;
 Date startdate;
 DateFormat df;

 public void init(ServletConfig config) throws ServletException
        {

        /* Get and save the current date and time */
         super.init(config);
         df = new SimpleDateFormat("yyyy.MM.dd HH:mm");
         StartupTime=System.currentTimeMillis();
         startdate = new Date(StartupTime);


         /* Write a message indicating the servlet engine start */

         try
            {

             System.out.println("Servlet engine active at " + df.format(startdate));
            }
         catch(Exception e)
            {
             System.out.println("UpTime init error: " + e.getMessage());
            }
         }

 public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
        {
```

```
         /* Respond to each request with the time stamp of the servlet engine
initialization
            and the length of time it has been running */

         long Ctime, Dtime, htime, mtime, stime;

         Ctime = System.currentTimeMillis();
         Dtime = Ctime - StartupTime;
         htime = Dtime / 3600000;
         mtime = (Dtime - (htime * 3600000)) / 60000;
         stime = (Dtime - ((htime * 3600000) + (mtime *60000))) / 1000;

         res.setContentType("text/html");
        PrintWriter out = res.getWriter();
         out.println("<HTML><BODY><P>");
         out.println("Servlet engine started at: " + df.format(startdate));
         out.println("<P>");
         out.println("Servlet engine uptime(ms): " + Dtime);
         out.println("<P>");
         out.println("Servlet engine uptime(h:m:s): " + htime +":"+ mtime + ":" +stime);
         out.println("</BODY></HTML>");
        }

 public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
        {
         doGet(req,res);
        }

 public void destroy()
        {

        /* This is only destroyed when the Servlet engine is stopped. Log the time
           of the stoppage. */

         Date stopdate = new Date(System.currentTimeMillis());


         try
            {
             System.out.println("Servlet engine stopping at " + df.format(stopdate));
            }
         catch(Exception e)
            {
             System.out.println("UpTime destroy error: " + e.getMessage());
            }
         }

}
```