# Finding and Collecting Availability Measurement Data for the IBM HTTP Server and WebSphere Application Server for z/OS and OS/390

**Mike Bonett**
**IBM Advanced Technical Support, Gaithersburg, MD**
**bonett@us.ibm.com**
**October 2001**

Many I/T environments are finding that the z/OS and OS/390 operating system platforms are prime candidates to support e-business applications. These applications rely on the technical foundation provided by the IBM HTTP Server for OS/390 and the WebSphere Application Server for OS/390. These products become part of the end-to-end path for an application, to allow users to submit requests to application logic, and application logic to retrieve data for presentation back to the user.

Monitoring and measuring the availability of the IBM HTTP Server and WebSphere in the OS/390 and z/OS environment must be done as part of proper management of the applications or business systems supported by these components. This paper identifies the data that is available to support these tasks, and how to collect it from two sources of measurement data:

1. ***Event Sources***, locations where products can directly place status information. This includes:
   a. Product logs
   b. SMF
   c. System messages
   d. SNA alerts
   e. TCP/IP traps
2. ***Monitoring Techniques***, which can be used (either directly or as part of a product function) to monitor a product component to determine its status. These include:
   a. Heartbeats
   b. "Ping" functions
   c. Remote commands
   d. User simulation
   e. Custom monitoring agents

Detailed descriptions of these sources can be found in the White Paper ***Finding and Collecting Availability Measurement Data***, which is available from the author.

Each event source or monitoring technique has advantages and disadvantages for both products. The skills and policies within a particular I/T environment will determine which sources can be accessed or which monitoring techniques can be deployed. Information from the appropriate event sources and

 

monitoring techniques provides, in various levels of detail, data on IBM HTTP Server and WebSphere Application Server availability.

The information in this white paper is based on the following product levels:

- IBM HTTP Server for OS/390 V5R3
- WebSphere Application Server V3.5 Standard Edition for OS/390
- WebSphere Application Server V4.0 for Z/OS and OS/390.

References to OS/390 in the paper also apply to z/OS.

## Architecture Basics

Before measurement starts, what is being measured has to be defined. For the HTTP Server and the different WebSphere levels, their configuration and architecture influences how they are measured.

### IBM HTTP Server

The IBM HTTP Server can run in two modes: standalone and scalable server.

In standalone mode, each IBM HTTP Server environment is an independent address space. Requests are received by the server on the defined port, and serviced by the address space. The standalone address space, and the resources within that address space, are the focus of the availability measurements.

In scalable mode, each IBM HTTP Server environment runs as a collection of address spaces. Within each collection is a queue manager address space, which receives requests, and one or more queue server address spaces, which the queue manage sends the request for processing. The queue server address spaces are created and dispatched by the Workload Manager (WLM) based on defined policies, or manually. The combination of the queue manager address space and one or more queue server address spaces are the focus of availability, since application requests require both to be active.

### WebSphere Application Server 3.5

WebSphere 3.5 runs entirely with an IBM HTTP Server address space. It executes all application code within that address space. The focus of availability is the HTTP address space environment (either standalone or scalable, as described above) and the WebSphere functions within that environment.

### WebSphere Application Server 4.0

WebSphere 4.0 runs as a collection of servers. Each server is comprised of one or more server instances. Each server instance is made up of a control region and zero or more server regions. As in

 

the HTTP Server scalable mode, the control regions route work submitted to the server to a server region.

WebSphere 4.0 requires the following servers to be running, to support applications:
- Daemon Server (a single control region)
- Systems Management Server
- Interface Repository Server
- Naming Server

The applications themselves run in the following types of servers:
- J2EE Server, for Java applications (servlets, jsps, EJBs)
- MOFW Server, for CORBA (Component Broker) applications

The focus of availability for WebSphere 4.0 is twofold:
1. The availability of the WebSphere “subsystem” servers (Daemon, Systems Management, Interface Repository, Naming) and their functions.
2. The availability of the appropriate application servers, which require the WebSphere subsystem servers to be available.

## Event Sources

Event sources are where information (including status information) is normally recorded about a product or function. Typically these are provided without the need for an additional product (though product customization may be required to activate the event recording). On OS/390 the primary event sources are product logs, SMF, system messages (recorded in the SYSLOG/OPERLOG), SNA alerts, and TCP/IP traps.

### Product Logs

#### IBM HTTP Server

The IBM HTTP Server creates several different logs:

- Server Access Log
- Fast Response Cache Accelerator (FRCA) Access Log
- Proxy Access Log
- Cache Access Log
- Agent Log
- Referrer log
- Server Error log
- CGI Error Log

 

Of these, the access log is best for determining when the HTTP Server was or was not available to serve pages. Each entry in the access log records a request against the server. The entry is time stamped, and shows:

- What was requested
- Who requested it
- The request method
- The file being requested
- A return code indicating the request success or failure

Using this information, different types of availability can be determined. For example:

- When was the server available or unavailable in terms of serving pages? If the HTTP Server normally receives one or more requests per minute, gaps in record time stamps that are greater than a minute indicate periods when the HTTP Server was down. In addition, a succession of failure requests for all requests to the server can reveal periods when the HTTP Server was up, but unable to service requests for some reason (a configuration error, missing files, etc.).
- When were specific server resources available or unavailable? Since each request contains the name of requested resource (usually a file, but it can also be the invocation of an executable process such as a CGI program, servlet, or EJB), the log shows when specific resources where available or unavailable to all users, or to particular sets of users.

**WebSphere Application Server 3.5**

WebSphere Application Server also creates additional logs:

- native.log (messages produced by WebSphere C code)
- ncf.log (messages sent to system.out and system.err from WebSphere applications)

The content of these logs will depend upon the applications running in WebSphere Application Server using them to write status and error information. Because of this, these two logs “as-is” may not be the best source of availability information for WebSphere Application Server or its applications. Later in this paper monitoring techniques will be discussed which can be used to log availability information to the ncf.log file.

**WebSphere Application Server 4.0**

The error log stream in WebSphere 4.0 is a system logger application. There can be a common log stream for all of WebSphere 4.0, or individual log streams for servers and server instances. The error log stream contains primarily WebSphere error information:

- Assertion failures
- Unrecoverable error failures
- Vital resource failures, such as memory

 

- Operating system exceptions
- Programming defects in WebSphere code

Since it meant primary for diagnosis, it may not be the best source of WebSphere availability status by itself.

WebSphere Java applications can use the provided message logger function. This built upon using JRas support (interfaces and methods), which are extensions to the Ras Toolkit for Java. Using the message logger allows:

- Defining messages inline, or via a separate message properties file.
- Assigning the message type - error, warning, or informational.
- Sending the message to one or more of these destinations: SYSLOG, error log stream, or CTRACE data set.

An example of Java code using the message logger function is provided in the ***WebSphere Application Server for z/OS and OS/390: Assembling J2EE Applications*** manual. This can be used to have the application log a “ready for processing” message when it starts, and a “terminating processing” message when it finishes, which can then be used for availability measurement purposes.

## SMF

### IBM HTTP Server

SMF record type 103 contains configuration and performance statistics for the IBM HTTP Server. The layout and content of this record type is documented in the ***IBM HTTP Server Planning, Installing, and Using*** manual. The statistics in these records are very useful for determining HTTP Server performance and required tuning, but not as well suited for availability measurements.

### WebSphere Application Server 3.5

WebSphere Application Server 3.5 does not record information to SMF.

### WebSphere Application Server 4.0

SMF record type 120 contains interval and activity record subtypes for WebSphere Application Server version 4.0. The layout and content of this record type is documented in the ***WebSphere V4.0 for z/OS and OS/390 Operations and Administration*** manual.

 

The server activity record records activity within an Application Server (a J2EE or MOFW server); a single record is created for each activity that is run inside a server or server instance. The container activity record records activity within a container located in a WebSphere transaction server.

Since activity records are timestamped when the activity occurred, server or server instance availability can be derived if the activity occurs frequently enough. If server transactions are expected at least once a minute, gaps in activity record timestamps that are greater than a minute can indicate server unavailability.

## System Messages

### IBM HTTP Server

For performance reasons the HTTP Server is best run as a started task under OS/390.
Several key messages are issued to the console and SYSLOG/OPERLOG to indicate the availability status of the HTTP Server:

| Message ID | Description |
|---|---|
| **IMW3534I** | The HTTP Server task has begin the initialization process. |
| **IMW3536I** | The HTTP Server task has completed initialization and can begin processi requests |
| **IMW3535E** | The HTTP Server failed to initialize. Errors need to be fixed and the serve restarted before requests can be processed. |
| **IMW3537I** | The HTTP Server has been requested to terminate and restart, usually as a result of an operator request. |
| **IMW3538I** | The HTTP Server restart was successful, and request processing can resur |
| **IMW3539E** | The HTTP Server restart was unsuccessful. Errors need to be fixed and th server restarted before requests can be processed. |
| **IMW3540I** | The HTTP Server has stopped processing work and has begun to shut down, usually as a result of an operator request. |
| **IMW354II** | The HTTP Server has completed shutdown and the address space is terminating. |
| **IMW3542E** | The HTTP Server is ending due to a program check. A dump is taken for problem diagnosis purposes and then it shuts down. |

These messages can be captured in real time (using an automation product) or extracted from the SYSLOG or OPERLOG, and used to determine when the HTTP Server was or was not available. For example:

- Messages IMW3536I and IMW3538I indicate availability or “up” status.
- Messages IMW3535E, IMW3537I, and IMW3540I indicate “unavailability” or “down” status.

Calculating the time periods between “up” and “down” messages provides availability and unavailability measurements, in terms of when the HTTP Server was available to process requests. Note that the HTTP Server can be available even though the WebSphere Application Server is unavailable, so these messages by themselves will not give a complete view of availability for WebSphere applications.

**WebSphere Application Server 3.5**

No messages are logged to the console or SYSLOG that report the state of WebSphere Application Server (Messages do appear in the SYSOUT file of the HTTP Server started task, but these are not visible to the console or SYSLOG/OPERLOG). Any messages from WebSphere Application Server to the console have to be generated by applications. Java applications (servlets, JSPs, or EJBs) can send messages to the console by calling the **exec** method of the Java **runtime** class to send one of the following command to the UNIX System Services (USS) shell.

a. The **logger** command, which will send a message to the console.
b. If using Tivoli NetView for OS/390, the **netvcmd** REXX program (supplied with Tivoli NetView), which sends a request to NetView. NetView receives the requests and issues a message using the NetView WTO function.

The calls allow applications to write status messages to the log (active, terminating, etc.) for use in availability measurements. This method can be used to have a servlet loaded at startup time and write a message that indicates WebSphere is active. The termination routine of that servlet (assuming it is terminated only when WebSphere ends) can be used to write a message indicating that WebSphere is terminating.

The following Java code:

```
Runtime rt = Runtime.getRuntime();
Process pr = null;
try
 {
  pr = rt.exec(“logger -d1 -a MWB005I Message from a Java application”);
```

Will result in the follow message on the console and in SYSLOG:

```
+MWB005I: STCRACF: 33554574: Message from a Java application
```

The ***UpTime*** sample servlet in Appendix A illustrates how this is done using Tivoli NetView for OS/390.

**WebSphere Application Server 4.0**

 

Several key messages are issued by WebSphere 4.0 to the console and SYSLOG, that indicate control and server region address spaces start up and shutdown:

| Message ID | Description |
|---|---|
| **BBOU0007I** | The daemon address space is starting. |
| **BBOU0016I** | The daemon address space initialization is complete. |
| **BBOU008I** | The daemon address spaced ended normally. |
| **BBOU009E** | The daemon address space ended abnormally. |
| **BBOU0001I** | The control region address space is starting. |
| **BBOU0020I** | The control region address space initialization is complete. |
| **BBOU0002I** | The control region address space has ended normally. |
| **BBOU0003E** | The control region address space has ended abnormally. |
| **BBOU0004I** | The server address space is starting. |
| **BBOU0021I** | The server address space initialization is complete. |
| **BBOU0005I** | The server address space has ended normally. |
| **BBOU0006I** | The server address space has ended abnormally. |

These messages can be captured in real time (using an automation product) or extracted from the SYSLOG or OPERLOG, and used to determine when an address space component of WebSphere 4.0 was or was not available. For example:

- Messages BBOU0001I or BBOU0020I indicate availability or “up” status for a control region address space.
- Messages BBOU0002I or BBOU0003E indicate “unavailability” or “down” status for a control region.

Calculating the time periods between “up” and “down” messages provides availability and unavailability measurements, for a single address space. By aggregating this data across all appropriate address spaces, the availability of a WebSphere server (either a subsystem server or an application server) can be determined). Aggregating the server data for the Daemon, Systems Management, Naming, and Interface Repository servers will determine “WebSphere Subsystem” availability. Aggregating the data for a particular application server will determine the “Application Subsystem” availability - which may or may not be the same as the application availability, depending on the application characteristics.

Details on aggregating component measurements to get accurate availability measurements for a group of components can be found in the white paper ***Measuring End-to-End Availability: How To Get Started***, which is available from the author.

 

### SNA Alerts

The IBM HTTP Server and WebSphere Application Server do not send out SNA alerts. Alerts can be generated from applications running in this environment (e.g. CGI programs or servlets) by using the USS interface provided by Tivoli NetView for OS/390 V1R2 or later. An alert can be sent by following these steps:

1. The application calls the **netvcmd** REXX program (supplied with Tivoli NetView) and passes to it the NetView GENALERT command with appropriate parameters.
2. NetView receives the command string and executes it.

### SNMP Traps

#### IBM HTTP Server

A SNMP management information base (MIB) and SNMP subagent is supplied with the IBM HTTP Server. This allows any SNMP-capable management system to retrieve status and performance information about the server. Details on implementing the SNMP function are in the *IBM HTTP Server for OS/390 Planning, Installing, and Using* manual.

Currently the HTTP Server does not issue traps. However, the MIB variables can be queried and, in some cases, set. For example, the MIB variable applOperStatus indicates the operational status of the server. An SNMP manager can query this variable at a regular interval and use the query results to determine the HTTP Server availability status.

#### WebSphere Application Server 3.5 and 4.0

There is no SNMP support in WebSphere. Code to issue SNMP traps will have to be embedded in WebSphere applications.

## Monitoring Techniques

If the event sources do not provide sufficient information for the desired measurements, the following monitoring techniques can be evaluated for use. All require some work to implement, but can provide

 

greater detail on availability. These techniques all provide real time information on product/function status. This information that can be captured and saved for longer term availability reporting.

Each monitoring technique described can be used in the following “generic” process:

1. The monitoring function (standalone or part of a systems management product, running on the same platform as HTTP Server or WebSphere or on a connected platform) begins an observation by invoking a monitoring technique against the HTTP Server or WebSphere.
2. A response is returned from the HTTP Server or WebSphere to the monitoring function.
3. The monitoring function examines the response, or determines if there was a response, and uses that to determine the availability status (the specific function being measured depends on the type of monitoring technique employed).
4. The monitoring function records status in a data repository somewhere. It may choose to record status from all observations, or record only when an observation indicates a status change from the previous observation.

## Heartbeat

A ***heartbeat*** is a signal at a regular interval indicating that the component is still functioning. When the heartbeat is present -- either logged to a file, or sent as an event to receiving software -- the component is available. It indicates that the component is executing -- but it may not show that it is able to service requests. Absence of a heartbeat indicates the component is unavailable.

### IBM HTTP Server

There is no built-in heartbeat function for the IBM HTTP Server. An indirect one can be created from the contents of the server access log. For example, if an automated function requests a URL from the HTTP Server at regular intervals, the access log will record the request whenever the HTTP Server is able to accept them. The access log then has to be processed to analyze when the “heartbeats” did or did not occur, and determine when the server was available. Periods when the automated function that issues the heartbeats was not running would also have to be accounted for.

### WebSphere Application Server 3.5 and 4.0

As with the IBM HTTP Server, there is no built-in heartbeat function for the WebSphere Application Server. An indirect one can be created in several ways:

 

- For application requests from the HTTP Server, using the contents of the server access log, as described above. The major difference is that the automated function requests a URL that is normally processed by WebSphere. The log information for both the request and the request return code determines if the recorded heartbeat was successful -- a heartbeat recorded but with an unacceptable request code indicates that the HTTP Server was running, but WebSphere Application Server was not.
- From a servlet. Servlets can be autoloaded and started when WebSphere starts. This allows a servlet to write heartbeat time stamp records to a file on the server, to one of the WebSphere logs (ncf.log in 3.5, using the message logger in 4.0), or to send them to another platform. As long as the servlet is always running when WebSphere is running, the created heartbeats indicate WebSphere availability.
- For WebSphere 4.0, SMF activity records. Since these are timestamped when the activity occurred, they can be considered “heartbeats”, assuming that they occur frequently enough for an application. When there is no activity there will be gaps in the timestamps, which may indicate unavailability of an application server or container.

A program running on the same OS/390 platform as the HTTP Server or WebSphere can provide a heartbeat function, by checking at a regular interval for the existence of the appropriate address spaces (e.g. Using the system command **D A,L**) and recording the results to a file.

## Ping

A ***ping*** sends an event or query to a target component; the response (or lack thereof) from the target is used to indicate if the component is available. A ping timeout can mean the component is unavailable, or some other component in path between the ping source and the target is unavailable. This situation must be accounted for when using ping type functions for measuring availability.

The TCP/IP **ping** command is the most well known implementation of this type of monitoring. It conveys that a component’s TCP/IP stack is able to communicate and the response time for its “base” communication. It does not report on specific applications or ports. But it is very easy to implement ping-like functions at the port or application level, by knowing what port(s) an application is using and the command strings to which it will respond. Sending a command string to a port, and measuring the time it takes the response to return, is all the ping function has to do.

### IBM HTTP Server

A ping measurement can be implemented for the HTTP Server by sending an HTTP protocol command string to the server port (normally 80 or, for SSL, 443) and measuring the time it takes to receive a response. A programmed function can, at regular intervals, send an HTTP request to the server port, capture the response, calculate the time it took to receive the response, and use the information to determine if the server is available. The function could send the request using TCP/IP socket

 

programming or, using a language with URL constructs built in such as Java, send a URL to the server port. A timeout waiting for a response, or a response time above a defined threshold, indicates that the server is unavailable.

### WebSphere Application Server 3.5

WebSphere 3.5 runs in the same address space as the HTTP Server and receives requests via the same port, so the same technique can be used as described for the HTTP Server. The difference is that the ping request should be a URL that is normally processed by WebSphere, to accurately determine if WebSphere is running. WebSphere provides the SnoopServlet servlet, which is used to verify that it has been installed properly; this servlet can be used as a "ping", by invoking it from an automation program, determining if a response was received, and measuring the length of time to receive the response.

### WebSphere Application Server 4.0

WebSphere 4.0 can receive requests from the HTTP Server or from a Java or CORBA application running on the same or different platform. Based on the method of application invocation, monitoring software can send a ping-like message and use the response to determine if WebSphere (or the resource being pinged) is active.

## Remote Command

A ***remote command*** is sent from a monitoring platform to the target component. The results of the command are sent back to the monitoring platform, which looks at the response to determine the component status. The response indicates that the component is running and able to respond to service requests.

### IBM HTTP Server

Two types of remote commands can be sent to determine HTTP Server status:

1. Console commands to determine if the server process is active. On OS/390 the HTTP Server runs best as a started task. The **D A, <name>** command (where <name> is the name of the HTTP Server started task address space) returns whether or not the server is active. The server also supports modify commands, which are documented in the *HTTP Server for OS/390 Planning, Installing, and Using* manual. A command such as **F <name>,APPL=-d stats** (where <name> is

 

the name of the HTTP Server started task address space) returns server statistics; the response (or absence of one) can be used to determine server availability for measurement purposes.

The console commands can be sent from an automation program or function running on another platform. The **ROUTE** command can be used if the HTTP Server runs in a sysplex. Tivoli NetView for OS/390 **RMTCMD** can be used if NetView connectivity exists between the OS/390 monitoring platform and the HTTP Server image. If the monitoring platform is not OS/390, another technique such as TCP/IP socket programming can be used. In all cases, security for sending commands to the target platform must be in place.

2. HTTP commands to the server port to determine if the server process is active. This is the same as sending "pings" to the server port as described earlier, but more information is requested and (hopefully) returned than with a ping function. The HTTP can be URLs normally called via a browser. An automation function or programming language can send the URL request to the server port at a regular interval, capture the response, and analyze it to get detailed server status information.

   The HTTP Server provides a Server Activity Monitor. Details on its use are documented in the *IBM HTTP Server for OS/390 Planning, Installing, and Using* manual. It provides several URLs that can be called to retrieve various server statistics such as request activity, request response times, and network traffic to/from the server. For example, the default URL to access server activity statistics is **http://<servername>/Usage/Initial**. This URL returns an HTML page containing real time statistics that are updated each time the URL is called.

   Custom URLs can also be written that invoke programs supported by the HTTP Server (such as CGI or GWAPI) to provide information for availability measurements of specific applications or functions within the server.

**WebSphere Application Server 3.5**

There are no OS/390 console commands that interface with WebSphere on OS/390. However, HTTP commands, as described above for the HTTP Server, can also be used with WebSphere. The URLs would invoke WebSphere programs (servlets, JSPs, EJBs, etc.) that return information on WebSphere status. The more detailed the programs, the more information that can be returned about WebSphere.

An example of a command that can be written is the ***UpTime*** Servlet, which is listed is in Appendix A. The servlet performs two functions:

1. When it is first invoked (normally loaded when WebSphere starts), it records the time when it was loaded.

 

2. When it receives a URL request, it returns the time when it was loaded (which is also the time the WebSphere functions became available), and the elapsed time since it was loaded (which is the length of time WebSphere has been running since it was last started).

Calling the UpTime servlet using a URL such as **http://<servername>/servlet/UpTime** at regular intervals provides a running update on WebSphere availability. A failed URL call (no response returned) is a good indication of WebSphere unavailability.

Additional modifications can be made to UpTime to report WebSphere availability in different ways. Simple changes to UpTime can cause it to take actions such as issuing an OS/390 message or alert at startup and shutdown (using the methods described earlier in this white paper), or monitor and report on the status of back end connections from WebSphere to database or transaction systems.

### WebSphere Application Server 4.0

The Systems Management Scripting API provides a command-driven method of configuring and monitoring WebSphere 4.0 The API allows status and control commands to be issued from a REXX program running in USS, to carry out the same functions that can be performed by the WebSphere Systems Management End User Interface. An example of a remote command is REXX program that issues the API **CB390CMD("-action list")** command; the command returns the status of servers and related server instances. The response to this command can be examined to determine if the desired servers or server instances are active.

WLM system commands also report WebSphere Application Server status, specifically the status of application environments. Since an application environment equates to an application server, WLM commands that provide application environment information may contain server status information.

For example, the command **D WLM,APPLENV=<server name>** returns the state of <server name> - available, quiesced, or stopped. An automation product, such as Tivoli NetView for OS/390, can issue this command on a regular interval. The results returned can be recorded as the availability state of the monitored application server.

## User Simulation

***User simulation*** attempts to measure availability as an end user would see it. It interacts with the application, either at a keystroke or function level, as a real end user would, and measures the availability and response time of the application. Implementing user simulation is more complicated that any of the other monitoring techniques, but can provide the most accurate measure of the availability experienced by users of applications running on HTTP Server and WebSphere.

 

For both IBM HTTP Server and WebSphere, implementing user measurements can be done using languages that can easily interact with URLs and the HTTP protocol. For WebSphere 4.0, the languages may also have to directly invoke a WebSphere application, if the application code is accessed by a method other than a URL. Programming a user simulation would include:

- Building the proper HTTP protocol headings and URL
- Invoking form processing by passing the form data a user would normally fill in
- Capturing the HTML response
- Analyzing the response to determine if additional HTTP requests are required
- Measuring the elapsed time from sending a request and receiving a response
- Feeding the appropriate data into a measurement repository (log, relational database, etc.)

Languages such as C, C++, Java, and REXX can be used to implement this function

Another technique, usually as part of a software product or a specialized "script" language, is "screen scraping" or "playback/record". Actual keystrokes using the application user interface are recorded (or, if supported, coded into a "script" file), and then played back under control of the scripting program. The response back to the scripting program is captured and analyzed to determine application availability.
The program or script sees only the HTTP protocol, web browser, or keystrokes; it does not matter what platform the server runs on. Any language or product with support for these functions, and that can connect to an IBM HTTP Server or WebSphere application residing on OS/390, can be used. Once the user simulation is created, automation products can invoke the program or script at regular intervals to capture the data that will be placed in a measurement repository.

## Custom Monitoring Agents

All of the above monitoring techniques can be developed using various programming tools. They are also contained in various software products. These products usually use some sort of agent, running on either the same or different platform as HTTP Server or WebSphere, to invoke one or more of these monitoring techniques and provide information into the data sources to be used for measurements. The products that provide these functions include:

- Event management products, which provide agents to capture events from a variety of components.
- Application management products, which provide APIs to imbed in application code to measure transaction availability and response time.
- Application stress testing products, which provide scripting languages and/or record/playback functions to run against web based applications and produce availability and performance measurements.

Examples of IBM products in these categories are:

 

- **Tivoli Business Systems Manager (TBSM),** which can monitor the HTTP Server and WebSphere components, and associate them with a “line of business” to provide business views levels of monitoring.
- **Tivoli Web Services Manager (TWSM),** which can measure IBM HTTP Server and WebSphere applications for availability and response time.
- **Tivoli Applications Performance Management (TAPM),** which provides an API to user for defining transactions in an application and obtaining availability and response time measurements for that transaction.
- **Tivoli NetView Performance Monitor for IP (NPM/IP),** which can monitor the availability and “ping” response time of any IP port, including those used by HTTP Server.

## Summary

The techniques outlined above are simple actions that provide powerful information. The IBM HTTP Server on OS/390 and WebSphere on OS/390 are becoming critical components for providing end-to-end service for increasingly important business applications. Collecting measurement data from the event sources or monitoring techniques described above, and merging it with existing availability data from the operating system, network, database systems, and transaction systems, will provide a total end-to-end availability picture than can be managed and improved as business requirements dictate.

## Appendix A - UpTime Servlet

The following is example code for a servlet that reports WebSphere start time and elapsed time. When called using the URL http://<servername>/servlet/UpTime, it returns the following:

**UpTime Servlet loaded at: 2001.02.19 23:32**

**Uptime Servlet uptime: 1176111016 milliseconds**

**Uptime Servlet uptime: 326:41:51**

The first line shows the date/time the servlet was loaded; since it was defined to load when WebSphere started, it shows when WebSphere began to execute requests.

The second line shows the elapsed time since the servlet was loaded, in milliseconds.

The third line shows the elapsed time since the servlet was loaded, in HH:MM:SS.

This version uses the Tivoli NetView for OS/390 USS command interface to send messages to the console when it is load and when it is unloaded (shutdown).

```
/*UPTIME SERVLET (Author: Mike Bonett)

This servlet performs the following functions:
  - When loaded, writes a message to the System Log
  - When invoked, returns the amount of time since loaded (if loaded at
    WebSphere Application Server (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
  - This is an example. No warranties intended, implied, or assumed. Use at
    your own risk. Modify to your hearts' content.


*/

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
        {

         //The init method is run once, when the servlet is first loaded.
         //This is where the current date/time is obtained and used as
         //the WAS startup time, and where the startup message is
         //is sent to the SYSLOG via Tivoli NetView for OS/390
         super.init(config);
         df = new SimpleDateFormat("yyyy.MM.dd HH:mm");
         StartupTime=System.currentTimeMillis();
         startdate = new Date(StartupTime);

         Runtime rt = Runtime.getRuntime();
         Process pr=null;
         try
            {
             pr=rt.exec("netvcmd TESTWTO WAS001I WebSphere Application Server
                         Has Started at " + df.format(startdate));
             int rc = pr.waitFor();
            }
         catch(Exception e)
            {
             System.out.println("UpTime init error: " + e.getMessage());
            }
         }

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

         //The doGet method is run for every URL request.
         //The elapsed time since startup is calculated and
         //Returned in a simple HTML page.

         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("UpTime Servlet loaded at: " + df.format(startdate));
         out.println("<P>");
         out.println("Uptime Servlet uptime: " + Dtime + " milliseconds");
         out.println("<P>");
         out.println("Uptime Servlet uptime: " + htime +":"+ mtime + ":"
+stime);
         out.println("</BODY></HTML>");
        }
```

 

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

         //If UpTime is called with the HTTP Post method, invoke doGet
         doGet(req,res);
        }

 public void destroy()
        {

         //This method is executed when the server is destroyed (unloaded).
         //Write out a “shutdown” message to the SYSLOG using
         //Tivoli NetView for OS/390

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

         Runtime rt = Runtime.getRuntime();
         Process pr=null;
         try
            {
             pr=rt.exec("netvcmd TESTWTO WAS999I WebSphere Application Server
                         Has Been Stopped at " + df.format(stopdate));
             int rc = pr.waitFor();
            }
         catch(Exception e)
            {
             System.out.println("UpTime init error: " + e.getMessage());
            }
         }

}
```

## Appendix B. Reference Publications

- HTTP Server for OS/390 - Planning, Installing, and Using (SC31-8690)
- WebSphere Application Server V3.5 Standard Edition Planning, Installing, and Using (GC34-4806)
- WebSphere Application Server V4.0 for z/OS and OS/390 Installation and Customization (GA22-7834)
- WebSphere Application Server V4.0 for z/OS and OS/390 Operations and Administration (SA22-7835)
- WebSphere Application Server V4.0 for z/OS and OS/390 Systems Management Scripting API (SA22-7839)
- WebSphere Application Server V4.0 for z/OS and OS/390 Messages and Diagnosis (GA22-7837)
- Enterprise Java Beans for z/OS and OS/390 (SG24-6283 - redpiece as of 10/1/2001)
- An Introduction to Tivoli NetView for OS/390 V1R2 (SG24-2254)
- Measuring End-to-End Availability: How To Get Started (White Paper available at http://www.ibm.com/support/techdocs)