Sending Email Alerts from IBM Tivoli Monitoring Situations for Various Operating System Platforms

March 1, 2011

IBM Advanced Technical Skills

Mike Bonett
Executive I/T Specialist
Special Notices

This document reflects the IBM Advanced Technical Skills organization experiences integrating IBM Tivoli Monitoring with various email programs on several operating system platforms. It was produced and reviewed by the members of the IBM Advanced Technical Skills organization. This document is presented “As-Is” and IBM does not assume responsibility for the statements expressed herein. It reflects the opinions of the IBM Advanced Technical Skills organization. These opinions are based on hands on experiences with the products, and the examples provided may not be applicable to all environments.

The examples provided are also provided “as-is” and may not be applicable in their existing form to all environments. Use of these examples is at the discretion of the reader; no warranties are assumed, implied, or intended.

If you have questions about the contents of this document, please direct them to Mike Bonett, IBM Advanced Technical Skills (bonett@us.ibm.com).

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries: CICS, DB2, IBM, MQSeries, Parallel Sysplex, System z, WebSphere, z/OS, zSeries. A full list of U.S. trademarks owned by IBM may be found at http://www.ibm.com/legal/copytrade.shtml.

Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

LINUX is a registered trademark of Linus Torvalds.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

ITIL® is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

Other company, product and service names may be trademarks or service marks of others.

Acknowledgements

Many thanks to Art Eisenhour, IBM Advanced Technical Skills, for reviewing this paper and providing input.
Introduction......................................................................................................................................... 4
Situation email alert integration flow – generic architecture.......................................................... 4
Email function options.................................................................................................................... 5
UNIX/Linux platform...................................................................................................................... 6
Windows platform......................................................................................................................... 6
z/OS platform............................................................................................................................... 7
Custom programming.................................................................................................................... 7
JavaMail API................................................................................................................................. 8
Example scenarios......................................................................................................................... 9
Situation customization................................................................................................................... 9
Recipient lookup........................................................................................................................... 11
Situation command execution on TEMS running on Windows, using blat................................. 11
Situation command execution on TEMS running on Linux, using sendmail............................... 13
Situation command execution on TEMS running on z/OS, using SMTPNOTE............................ 14
Situation execution on TEMS on any Java supported platform, using JavaMail....................... 17
Custom program example: REXX on z/OS.................................................................................. 21
Additional considerations............................................................................................................. 23
Summary........................................................................................................................................ 23
Introduction

IBM Tivoli Monitoring (ITM) provides the ability to invoke an action based on the value of a monitored metric, or a combination of metrics. This function is called “reflex automation”, and can be defined using an ITM situation. When a situation is defined, one or more thresholds are set, using the desired metrics from the agent the situation is associated with. If the thresholds are reached the situation becomes true; when this occurs, one action that can be taken is to execute a command.

Sending alerts via email is desired in many environments. This provides direct notification to a human (or to an automated email processing function) of an event for which assistance may be required to resolve. ITM does not have an built in email alerting capability. However, since email commands are available – via native functions, from third parties, or custom programs – on many operating system platforms, integrating platform email functions with ITM situations is very doable and satisfies the email alerting requirement. This also allows the email function and associated standards to be kept separate and distinct from ITM, which allows changes and upgrades to the email functions to have much less of an impact on ITM.

This paper will describe the basics of integrating ITM with email functions to send emails alerts from situations. It will describe the basic architecture and flow, highlight the options available on the major operating system platforms, and provide base examples of how this integration can be implemented. It is meant to provide enough information to get started with this function, and provides pointers to further detailed information that can be used to provide a great level of implementation, security, and standards.

There is another level of automation within ITM called workflows. Workflows allow multiple situations from multiple agents to be correlated and actions taken based on that correlation. While this paper will not cover workflows, an email alert can be sent from a workflow by defining it as a command, in a similar manner as the commands in the follow sections will be defined.

Situation email alert integration flow – generic architecture

The following picture provides an overview of the components that must be integrated and customized to enable situation email alerts:
1. A situation is defined in ITM. The situation is associated with an agent, describes the metric(s) and values that will cause the situation to become true, and can be set to display as a color when true on the Tivoli Enterprise Portal. The situation runs on the Tivoli Enterprise Monitoring Server (TEMS) that is monitoring the agent.

2. Each situation can have a command string associated with it. The command string can be executed at either the platform the agent is running on, or the platform of the TEMS the agent is connected to. For this case, to better centralize operations, the command runs on the TEMS. The command string is passed to the operating system on the TEMS platform and executed as a command line. For z/OS, it is executed as a console command. Values can be passed from the situation attributes as part of the command string.

While the situation can directly invoke the actual platform email command, a better practice is to have the situation invoke a command procedure suitable for the platform (e.g. a UNIX/Linux shell script, Windows command file, z/OS System REXX procedure, or Java program). This allows more logic to be applied to the request (e.g. lookup the target email address based on the situation contents), allows the command procedure to be shared by many different situations, and keeps email contacts and standards changes from impacting situation definitions – only the command procedure has to be changed, and not every situation that invokes the script.

3. The command procedure invokes the platform email function, to send the message. Regardless of the command or custom program, there are specific characteristics use to send the email:
   a. Simple Mail Transport Protocol (SMTP) is used. SMTP is an internet standard for email transmission across IP networks. The protocol is published and anyone can write their own email interface code. RFCs 821 and 2821 describe the protocol details and are readily available on the internet.
   b. A connection to a SMTP server. For security reasons most SMTP servers require authentication before accepting and forwarding mail, to prevent them from being used for spamming purposes. Many also require a secure connection using Secure Sockets Layer (SSL) and/or Transport Level Security (TLS), so that the authentication and email information are encrypted. Whichever email function is used must be customized to properly connect and authenticate to the SMTP server; this is likely where most of the troubleshooting efforts will occur.

4. The SMTP server receives the message from the email function and forwards it to the intended destination, such as a desktop users inbox or a mobile device.

**Email function options**

There are many options available for the email function, both free and commercial software packages. This paper uses examples with options that are provided as part of the operating system platform, or are readily available as open source software. The JavaMail API will also be used, as this is a cross-platform option that can be used on any platform that supports Java.

Regardless of which email function is used, the mail information created by the command string invoked by the situation must include at least the following:
• The target email address to which the alert will be sent
• The source email address that the receiver will see it is from. It can be an address other than the one used to authenticate to the SMTP server.
• The body of the email. In some cases this can be passed to the client from a file or dataset; the command procedure invoked by the situation alert might have to build or update a file that can then be passed to the email client.

Before trying to call the mail function from a situation, it is important verify that it works by manually invoking it from the platform command line. Configuring the email function to properly connect to the SMTP server requires careful planning and testing, based on the various connectivity, security, and authentication options being used by the server. Manually validating the connection will save time before implementing a situation mail alert.

**UNIX/Linux platform**

The **sendmail** command is provided as part of Unix/Linux distributions and is the simplest method to use for sending mail from a UNIX/Linux platform. The details of configuring sendmail and all of its parameters can be found in the online help, as well as the documentation for the specific distributions. Many sendmail configuration examples are available on the internet.

The basic syntax of the sendmail command is

```
sendmail -t <recipient>  < <filename>
```

Where `<recipient>` is(are) the target email address(es) to receive the message, and `<filename>` is a file containing the message contents. sendmail takes the message information (both headers and message body text) from standard input, so this is normally sent via standard input to the command. The command procedure written around the sendmail command would build the message text and then either echo the text to the command, or write it to a temporary file and redirect it via standard input to the command.

**Windows platform**

Windows does not have a built-in email function, but there are commercial and open source packages available for use. For this white paper **blat** will be used. Blat is an open source, public domain program that operates in a similar manner to sendmail. Details on downloading, installing, and using blat can be found at [http://www.blat.net](http://www.blat.net).

Once installed and configured to access a SMTP server, the syntax of the blat command is

```
blat <filename> -to <recipients>
```

Where `<filename>` contains the message text, and `<recipient>` is(are) the target email address(es) to receive the message. The command procedure written around the blat command would build a
temporary file containing the desired message contents, and then invoke blat, using the file name as one of the parameters.

**z/OS platform**

For z/OS, IBM Communications Server provides the **SMTPNOTE REXX** command procedure for sending mail. SMTPNOTE must be customized for the appropriate z/OS SMTP function – either the SMTPD server or CSSSMTP client - which may be running on the same image or a different image, which it connects to via JES2 NJE.

SMTPNOTE runs under TSO, in batch (via the TSO command processor) or can be called by another REXX procedure, including System REXX. Once configured, the command line syntax is:

```
SMTPNOTE TO(<recipient>) SUBJECT(<text>) DATASET(<dataset name>) BATCH
```

where `<recipient>` is(are) the target email address(es) to receive the message, `<subject text>` is the text of the subject, and `<dataset name>` is the name of a file (sequential or partitioned dataset member) containing the message contents.

SMTPNOTE can be executed from a situation alert in several ways. For example:

- The command can invoke a z/OS started task that runs TSO in batch to run SMTPNOTE.
- The command can invoke a z/OS started task that submits a batch job to the internal reader, to run SMTPNOTE.
- The command can execute a command procedure on a non-z/OS platform, that builds the required JCL to run SMTPNOTE, and uses ftp to submit the job for execution.

For UNIX Systems Services (USS) on z/OS, sendmail, as described above for UNIX/Linux platforms, is also available as an option. The command procedure that uses sendmail would run as a USS process; it is also possible to have REXX programs that run in z/OS under TSO, batch, or NetView call USS to execute sendmail.

**Custom programming**

The SMTP protocol essentially consists of a set of text commands sent to a SMTP server via TCP/IP. Any programming language with a TCP/IP socket interface can be used to write a custom function to send email for this purpose. Some languages provide prepackaged routines to ease reading and writing these commands to sockets by providing mail functions (Java is one example, which will be described in the next section). On z/OS, in addition to the programming languages, the commands can also be sent to a local SMTP via a batch job and a JES2 internal reader.

Here is an example of the SMTP command strings to send mail that would be sent via TCP/IP (or included in a z/OS batch job). For TCP/IP each line ends with a carriage return/linefeed sequence.
HELO smtp_server_name
AUTH LOGIN ...
MAIL FROM:<itmmonitor@mysite.com>
RCPT TO:<sysadmin@remotesite.com>
DATA

To: sysadmin@remotesite.com
Reply-To: itmmonitor@mysite.com
Subject: Situation Alert
X-Mailer: my custom program
Situation Alert 0001 has been raised.
Please contact the duty manager if more information is needed.
.
QUIT

- HELO identifies the target SMTP server.
- AUTH LOGIN sends (if needed) the authentication information to the SMTP server. There are various authentication mechanisms that are beyond the scope of this document; the format of the data sent will depend on the mechanisms that are being used.
- MAIL FROM identifies the mail sender.
- RCPT TO indicates the recipient(s).
- DATA contains the message headers (From, To, CC, etc.) and the message text. The information is terminated by a period on its own line.
- QUIT ends the connection.

After each command (and its associated data) is sent, the SMTP server replies with a status to indicate the success or failure of the command. A custom program can use the status to determine if any corrective actions need to be taken, and/or to notify of an error in the process.

Writing such a function is not difficult; however, the greater effort is in maintaining the code and ensuring that it is keep functioning as system changes occur. This might be a greater effort for programming languages as compared to command procedures.

JavaMail API

Java provides the JavaMail API for sending and receiving email. The API runs on any platform that supports Java, and provides classes and functions to send and receive mail; SMTP, IMAP, and POP3 protocols are supported. The API package is included in the Java Enterprise Edition (EE) platform; for Java Standard Edition (SE) platform it is optional, and can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index-135046.html.

The Java Activation Framework (JAF) is required to use JavaMail; this is part of Java SE 6. For earlier Java versions it can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index-135046.html.

To enable the API, the following classes must be added to the Java CLASSPATH:
• mail.jar (from JavaMail)
• activation.jar (from JAF)

The main components of the API are:
• A Session object, which is created with a set of attributes such as the SMTP server host and IP port, user ID and password to sign on with, authentication method, if SSL and/or TLS are to be used, etc.).
• A Message object, which contains the message headers and message text information.
• A Transport object, which defines the protocol to be used for the session.

The main benefit of using the JavaMail API is that the same Java program can be used on multiple platforms, making it easier to maintain and to migrate – for example, if the platform on which the situation command runs changes.

Example scenarios

From the above information, it is clear that there is great flexibility generating email alerts from situations. This section covers a few examples that show how to apply some of the options, and the initial customization required for implementation.

In each case the examples define a situation to invoke the command procedure. When the situation becomes true, several situation attributes are passed to command procedure, which uses one of them to perform a lookup to find the recipient email address. The command procedure then builds the message header and text, and invokes the email function (command custom program) to send the email.

The “lookup” component is used to show one method of making the command procedure more flexible. This type of lookup (be it to a file or database) allows information (in this case the recipient) to be changed without having to modify the situation or the command procedure.

Situation customization

To use these mail alert options, a situation has to be customized to invoke them. The details of defining a situation is beyond the scope of this paper, but is well documented in the IBM Tivoli Monitoring Users Guide and the Tivoli Enterprise Portal online help. A situation can also be defined using the tacmd command line interface, which is documented in the IBM Tivoli Monitoring Command Reference.

The key situation definition information can be found, in the TEP situation editor, on the Action tab of the situation:
The following options are used:

- **Action Selection: System Command**
- **Only take action on first item**
- **Execute the Action at the Managing System (TEMS)**
- **Don’t take action twice in a row (wait until the situation goes false then true again)**

The System Command text box contains the command string to be sent for execution at the TEMS. Variable information from the situation can be included by selecting the **Attribute Substitution** button and selecting the attributes containing the desired information. These attributes appear in the System Command box, in the format

\[
&\{\text{AttributeGroup.Attribute Name}\}
\]

For initial situation command verification the **echo** command is useful to see the attribute contents. For example, if a situation is created to report whenever an agent on a particular system goes offline, the following string in the System Command text box will, for a non-z/OS TEMS, write the situation timestamp, agent status, system name, and agent product code values to a file on the TEMS system (for a z/OS TEMS the values will appear in the SYSLOG):

\[
echo \&\{\text{ManagedSystem.Timestamp}\} \&\{\text{ManagedSystem.Status}\} \&\{\text{ManagedSystem.Name}\} \&\{\text{ManagedSystem.Product}\} >>\text{server3_offline.log}
\]

Here are a few lines from the resulting file, when agents on the monitored system went offline:

```
1110213161905000 *OFFLINE server3:KUL UL
1110213162148000 *OFFLINE server3:SY SY
1110222213235001 *OFFLINE server3:UA UM
1110217031215000 *OFFLINE server3:LZ LZ
1110222213235000 *OFFLINE server3ASPSdp:UAGENT00 UA
```

Seeing the values that are passed will aid in having the command procedure extract the desired information to include in the email text.
Recipient lookup

For easier change maintenance, the email address(es) of the recipient(s) should not be coded in the situation action. Instead, provide a method for the command procedure to find it based on one or more of the situation values passed to it. The lookup function can be a database query, LDAP query, a simple file lookup based on a key value, etc. For example, if the product code is the attribute value to be used, and a file lookup will be performed, each line of the lookup file will have two tokens separated by one or more spaces – the product code and the email address that is to be notified:

```
NT  ntadmin@somewhere.com
LZ  linuxadmin@somewhere.com
UX  unixadmin@somewhere.com
MQ  mqadmin@somewhere.com
```

The lookup function would then use the product code value to scan the lookup file and select the corresponding email address.

Situation command execution on TEMS running on Windows, using blat

In this example, a disk capacity alert will send an email when the disk on a particular system exceeds a defined threshold. The following Windows command procedure is invoked from the situation, and performs the following actions:
1. Changes to a directory to be used to store a file containing the mail message contents.
2. Writes attribute values passed from the situation (Timestamp, Logical Disk, Megabytes Left, and System) to a file to create the message text (the Timestamp value is also used as part of the file name).
3.Obtains the correct email recipient by performing a lookup on the email address file (emaillookup.txt in this case). When it finds the record where the product code matches the product code from the situation, it saves the email address into a variable.
4. Invokes blat to send the file to the recipient.
@echo off
REM Windows command file: send "disk capacity problem" email alert using blat
REM Change to designated directory
REM
d:
REM
REM Write the situation values to the mail text file
REM
echo %1 %2 %3 %4 >>sendemailalert.log
echo %1 > sit_%1.log
echo %4 >> sit_%1.log
echo %2 >> sit_%1.log
echo %3 MB left >> sit_%1.log
REM
REM Lookup the recipient email address based on a situation attribute
REM
setlocal enableextensions
For /F "tokens=1,2" %%B in (emaillookup.txt) DO (REM @echo %%B %%C
   if %%B==%4 @echo "correct email is" %%C
   if %%B==%4 SET tmpmail=%%C )
@echo email is %tmpmail%
REM
REM Invoke blat to send mail message
REM
d:\blat\blat sit_%1.log -to %tmpmail% -subject "%4 Disk %2 Capacity Problem"
REM
REM Delete the mail text file
REM
del sit_%1.log

The situation System Command definition contains the following (all as one line):

d:\ats\senddiskalert.cmd &{NT_Logical_Disk.Timestamp} &{NT_Logical_Disk.Disk_Name} &{NT_Logical_Disk.Free_Megabytes} &{NT_Logical_Disk.Server_Name}

When the situation triggers, the results are seen in the alerts raised in the TEP and the inbox of the target recipient:
Situation command execution on TEMS running on Linux, using sendmail

In this example, agents on a system connected to a Linux on System z TEMS send email alerts for a file system capacity situation. When the situation becomes true, the following Linux shell script procedure is invoked from the ITM situation. It performs the same steps as described above for the Window TEMS command procedure:

```bash
#!/bin/sh
# bash shell script - send filesystem capacity email alert using sendmail
#
# Change to the designated directory
# cd /ats
#
# Write the situation values to the mail text file
# echo "Subject: " $4 $2 "Filesystem problem" >sit_$1.log
echo " " >> sit_$1.log
echo $1 >> sit_$1.log
echo $4 >> sit_$1.log
echo $2 "has less than " $3 "pct free space" >> sit_$1.log
#
# Lookup the recipient email address based on a situation value
# maildest=`(grep $4 emaillookup.txt| cut -d" " -f2)`
echo $maildest
#
# Invoke sendmail to send the mail message
# /usr/sbin/sendmail -t $maildest -Froot@hasl104.mysystem.com < sit_$1.log
#
# Delete the mail text file
# rm -f sit_$1.log
```

The situation System Command definition contains the following (all as one line):

/ats/sendfsalert.sh &{KLZ_Disk.Timestamp} &{KLZ_Disk.Mount_Point} &{KLZ_Disk.Disk_Free_Percent} &{KLZ_Disk.System_Name}

When the situation triggers, the results are seen in the alerts raised in the TEP and the inbox of the target recipient:
Situation command execution on TEMS running on z/OS, using SMPTNOTE

In this example, agents on a system connected to a z/OS TEMS will send email alerts for selected situations. The following System REXX command procedure is invoked from the ITM situation and performs the following steps:

1. Obtains the correct email recipient by performing a lookup on a created email address dataset based using a situation attribute (Product Code).
2. Write the situation attribute values (Timestamp, Status, Agent Name, Product Code) to a temporary dataset to create the message text (the Timestamp value is used as part of the file name).
3. Invokes SMPTNOTE to send the file to the recipient.
/* SYSTEM REXX EXEC - send agent email alert using sendmail */

arg sit1 sit2 sit3 sit4 .

/*
Lookup recipient email address based on situation value
*/
"ALLOC DA('SMTPMAIL.SIT.LOOKUP') FI(LOOKUP) SHR"
"EXECIO * DISKR LOOKUP (FINIS STEM emaillist.)"
"FREE FI(LOOKUP)"
do i= 1 to emaillist.0
  if word(emaillist.i,1)=sit4
    then do
      recipient=word(emaillist.i,2)
      leave
    end
end
say "RECIPIENT" recipient

/*
Write the situation values to the mail text file
*/
subject=sit2 sit3
msg.0=3
msg.1=sit1
msg.2=sit3
msg.3=sit2
cctime=time('L')
parse var cctime "." LLQ
tempdsn='TEMP.MSGTXT.T'||LLQ
"ALLOC DA('tempdsn') FI(MSGOUT) NEW "||,
"SPACE(1,0) TRACKS LRECL(80) RECFM(F B)"
"EXECIO * DISKW MSGOUT (FINIS STEM msg.)"
"FREE FI(MSGOUT)"

/*
Invoke SMPTNOTE to send the mail message
*/
parmstring="TO("recipient")"||,
  "SUBJECT('"subject"') "||,
  "DATASET('"tempdsn"') "||,
  "BATCH"
call SMPTNOTE parmstring
"DELETE" tempdsn
exit

To execute this as a System REXX command, the command procedure, along with the SMPTNOTE command procedure, was copied to a System REXX library (the default is SYS1.SAXREXEC). A small modification was made to SMPTNOTE because it creates a temporary dataset for its staging work; the modification made the dataset name unique, otherwise multiple invocations of SMPTNOTE at the same time would cause conflict on the dataset and cause all but one of the requests to fail.
The following command string is defined in the situation System Command definition:

```
REXXCB SENDML2 &{ManagedSystem.Timestamp} &{ManagedSystem.Status}
&{ManagedSystem.Name} &{ManagedSystem.Product}
```

Where REXXCB is the System REXX command prefix for the target system, and SENDML2 is the name of the command procedure.

When an agent (in this case the OMEGAMON XE for Mainframe Network agent and subagents) goes offline, the situation invokes the SENDML2 command procedure, causing the following messages to appear on the SYSLOG:

<table>
<thead>
<tr>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:05:20</td>
<td>EZA5460I 03/06/11 21:05:20 BSMTTP Helo Domain: HSLCBNJE I've never heard of you!</td>
</tr>
<tr>
<td>21:05:20</td>
<td>EZA5474I 03/06/11 21:05:20 Received Note 00000019 via BSMTTP From</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:STCRACF@HSLCBNJE">STCRACF@HSLCBNJE</a> 303 Bytes</td>
</tr>
<tr>
<td>21:05:26</td>
<td>EZA5460I 03/06/11 21:05:26 BSMTTP Helo Domain: HSLCBNJE I've never heard of you!</td>
</tr>
<tr>
<td>21:05:27</td>
<td>EZA5474I 03/06/11 21:05:27 Received Note 00000020 via BSMTTP From</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:STCRACF@HSLCBNJE">STCRACF@HSLCBNJE</a> 299 Bytes</td>
</tr>
<tr>
<td></td>
<td>EZA5461I 03/06/11 21:05:29 Delivered Note 00000019 to <a href="mailto:netadmin@somewhere.com">netadmin@somewhere.com</a></td>
</tr>
<tr>
<td></td>
<td>EZA5461I 03/06/11 21:05:30 Delivered Note 00000020 to <a href="mailto:netadmin@somewhere.com">netadmin@somewhere.com</a></td>
</tr>
<tr>
<td></td>
<td>EZA5461I 03/06/11 21:05:31 Delivered Note 00000021 to <a href="mailto:netadmin@somewhere.com">netadmin@somewhere.com</a></td>
</tr>
</tbody>
</table>

Three messages appear at the target users’ email inbox, one for each agent/subagent:
While System REXX is the most straightforward method, there are several other options to invoke this command procedure:
- the situation submits a started task or batch job that runs the TSO Command Processor in batch, passing the situation attribute values to the started task or batch job.
- The REXX procedure can be compiled into a z/OS load module and invoked directly as a started task or batch job.

In addition, if the TEMS or agent is not on z/OS, but the execution is still desired on z/OS, ftp can be used. A command procedure executing on a non-z/OS platform can build the JCL and use ftp to send the JCL for execution on z/OS.

**Situation execution on TEMS on any Java supported platform, using JavaMail**

Since JavaMail can run on all of these platforms, it can be a more portable option when the situation execution platform varies. In this example the following Java program was used to perform the same action on a Windows TEMS for the disk capacity situation action described in the Windows TEMS example:
public class senddiskalert
{
    public static void main(String[] args)
    {
        //Lookup the recipient email address - use the last 2 characters of the System string
        String prodcode = args[3].substring(args[3].length() - 2);
        String m_to = null;
        try
        {
            InputStream in = new FileInputStream("emaillookup.props");
            Properties emailprops = new Properties();
            emailprops.load(in);
            in.close();
            m_to = emailprops.getProperty(prodcode);
        }
        catch (Exception e)
        {
            System.out.println("Fatal error: " + e.toString());
            System.exit(1);
        }

        //Build the subject and text using the situation attribute values

        //Build the required properties for the smtp mail session object
        String d_email = "Admin@hasle320.mysystem.com";
        String d_uname = "bonett@somewhere.com";
        String d_host = "na.relay.ibm.com";
        int d_port = 25;
        Properties props = new Properties();
        props.put("mail.smtp.user", d_email);
        props.put("mail.smtp.host", d_host);
        props.put("mail.smtp.port", d_port);
        props.put("mail.smtp.starttls.enable", "false");
        props.put("mail.smtp.debug", "true");
        props.put("mail.smtp.auth", "false");
        props.put("mail.smtp.socketFactory.port", d_port);
        props.put("mail.smtp.socketFactory.fallback", "false");

        //Create the session and send the email
        Session session = Session.getInstance(props);
        session.setDebug(true);
        try
        {
            Message msg = new MimeMessage(session);
            msg.setText(m_text);
            msg.setSubject(m_subject);
            msg.setFrom(new InternetAddress(d_email));
            msg.addRecipient(Message.RecipientType.TO, new InternetAddress(m_to));
            Transport transport = session.getTransport("smtp");
            transport.connect();
            transport.sendMessage(msg, msg.getAllRecipients());
            transport.close();
        }
        catch (Exception e)
        {
            System.out.println("Error sending message: " + e.toString());
        }
    }
}
The program does the following:

- Look up recipient email address – this uses a Java properties object, which is convenient for finding specific name-value pairs. In this case the name is the product code, and the value is the recipient email address.
- Build the subject and message text – string variables are created to hold the situation attribute values passed as arguments and any additional text.
- Build the properties for the mail session – a properties object is built to create an instance of a JavaMail session object. The example does not show all possible properties that can be used; it is dependent upon the connectivity requirements and security of the target SMTP server.
- Create the session and send the mail – the session object is created, and it, along with the message subject and text strings created earlier in the program, are used to create a transport object that connects to the SMTP server and sends the mail.

The java program is best run within a command procedure, to allow the locations of the java executable and the required JavaMail jar files to be defined. For example:

```bash
set JAVA_HOME="C:\Program Files\IBM\Java50\jre\bin"
set CLASSPATH=%CLASSPATH%;d:\ats;d:\ats\mail.jar;d:\ats\activation.jar
d:
cd \ats
%JAVA_HOME%\java senddiskalert %1 %2 %3 %4
```

The situation would execute the Windows command procedure (here called senddiskalert) and pass it the appropriate situation values with the following situation System Command definition:

```cmd
\ats\senddiskalert.cmd &{NT_Logical_Disk.Timestamp}
&{NT_Logical_Disk.Disk_Name} &{NT_Logical_Disk.Free_Megabytes}
&{NT_Logical_Disk.Server_Name}
```

With debug set on in the Java program, its execution steps to send the email can be seen:
Recipient: sysadmin@somewhere.com

DEBUG: setDebug: JavaMail version 1.4.4
DEBUG: getProvider() returning javax.mail.Provider[TRANSPORT, smtp.com.sun.mail.smtp.SMTPTransport, Sun Microsystems, Inc]
DEBUG SMTP: useEhlo true, useAuth false
DEBUG SMTP: trying to connect to host "smtpserver.somewhere.com", port 25, isSSL false
DEBUG SMTP: connected to host "smtpserver.somewhere.com", port: 25

EHLO hasle315
250-d03av04.boulder.ibm.com Hello hasle315.mysystem.com [9.82.38.16]
], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE 20000000
250-DSN
250-AUTH DIGEST-MD5 CRAM-MD5
250-STARTTLS
250-DELIVERBY
250-HELP
DEBUG SMTP: Found extension "ENHANCEDSTATUSCODES", arg ""
DEBUG SMTP: Found extension "PIPELINING", arg ""
DEBUG SMTP: Found extension "8BITMIME", arg ""
DEBUG SMTP: Found extension "SIZE", arg "20000000"
DEBUG SMTP: Found extension "DSN", arg ""
DEBUG SMTP: Found extension "AUTH", arg "DIGEST-MD5 CRAM-MD5"
DEBUG SMTP: Found extension "STARTTLS", arg ""
DEBUG SMTP: Found extension "DELIVERBY", arg ""
DEBUG SMTP: Found extension "HELP", arg ""
DEBUG SMTP: use8bit false
MAIL FROM:<Admin@hasle320.mysystem.com>
250 2.1.0 <Admin@hasle320.mysystem.com>... Sender ok
RCPT TO:<sysadmin@somewhere.com>
250 2.1.5 <sysadmin@somewhere.com>... Recipient ok
DEBUG SMTP: Verified Addresses
DEBUG SMTP: sysadmin@somewhere.com
DATA
354 Enter mail, end with "." on a line by itself
From: Admin@hasle320.mysystem.com
To: sysadmin@somewhere.com
Message-ID: <279187620.0.1299600611060.JavaMail.Administrator@hasle315>
Subject: primary:HASLE320:NT Disk D:Capacity Problem
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

1110308112821545
Primary:HASLE320:NT
C:
857MB left
.
250 2.0.0 p28G7luK020342 Message accepted for delivery
QUIT
221 2.0.0 d03av04.boulder.ibm.com closing connection

The mail result:
To run the Java program on a z/OS TEMS, the situation could call a started task or batch job to run USS in batch, invoke the program and pass the parameters to the program.

**Custom program example: REXX on z/OS**

It is not difficult to write a custom program, as the SMTP protocol is well documented and easy to implement any programming language that provides a TCP/IP socket interface. This sample code fragment written in REXX could be used by the “agent offline” situation command procedure described earlier, to replace the call to SMPTNOTE:
/* Send email via REXX - and alternative to SMTPNOTE for agent offline situation */

/* define smtp required information */
smtp_server = 'smtpserver.somewhere.com'
smtp_port = 25
smtp_from = 'itmadmin@somewhere.com'
smtp_address = 'itmadmin@somewhere.com'
smtp_to = recipient
smtp_replyto = 'itmadmin@somewhere.com'
crlf = x2c('0d25')

/* Initialize TCP/IP socket information */
response_string = Socket('initialize', 'SMTPSOCK')
Parse Var str sockrc subtaskid maxdesc tcpipuser
response_string = Socket('Socket', 'af_inet', 'stream', 'tcp')
Parse Var response_string sockrc sockid

/* Be sure to set data to ASCII */
response_string = Socket('SetSockOpt', sockid, 'sol_socket', 'SO_ASCII', 'on')

/* Connect to SMTP server */
server_info = 'AF_INET' smtp_port || smtp_server
response_string = Socket('Connect', sockid, server_info)
response_string = Socket('Recv', sockid, 20000)
Parse Var response_string sockrc data_length smtp_response

msg= 'HELO ' || smtp_server || crlf
response_string = Socket('Send', sockid, msg)
response_string = Socket('Recv', sockid, 20000)

msg= 'MAIL FROM:<' || smtp_from || '>' || crlf
if traceflag then say msg
response_string = Socket('Send', sockid, msg)
response_string = Socket('Recv', sockid, 20000)

msg= 'RCPT TO:<' || smtp_address || '>' || crlf
if traceflag then say msg
response_string = Socket('Send', sockid, msg)
response_string = Socket('Recv', sockid, 20000)

msg= 'DATA' || crlf
if traceflag then say msg
response_string = Socket('Send', sockid, msg)
response_string = Socket('Send', sockid, crlf)

subject_string = sit2 sit3
msg = 'To:' smtp_to || crlf,
|| 'Reply-To:' smtp_replyto || crlf,
|| 'Subject:' subject_string || crlf,
|| 'X-Mailer: REXX Exec on ZOS' || crlf
if traceflag then say msg
response_string = Socket('Send', sockid, msg)
response_string = Socket('Send', sockid, crlf)
response_string = Socket('Send', sockid, crlf)
if traceflag then say msg
response_string = Socket('Send', sockid, crlf)

msg = crlf || '.' || crlf
if traceflag then say msg
response_string = Socket('Send', sockid, msg)
response_string = Socket('Recv', sockid, 20000)
say "Received reply:" response_string

/* End SMTP connection and terminate socket */
msg = 'QUIT' || crlf
response_string = Socket('Send', sockid, msg)
response_string = Socket('Close', sockid)
response_string = Socket('Terminate', subtaskid)
say 'Email sent to ' smtp_to
Exit
Additional considerations

These examples show the basic foundation for defining and sending mail alerts from ITM situations. From this foundation additional functions can be added, and additional considerations will apply:

- Security, of course, must be carefully planned. For example, if user authentication is required to access the SMTP server, where will that information reside? Rather than build it into the situation or the command procedure, it may be more secure to perform a lookup to obtain the security information, in a similar manner that the examples showed a lookup function to determine the destination address.

- It is helpful to not just receive an email when a problem occurs, but one when the problem is resolved. While this cannot be done in a single situation, a workflow policy can be used to:
  - Wait for a situation to be true
  - When it becomes true, send the mail alert
  - Wait for the situation to become false
  - When it becomes false, send a mail alert indicating the problem is resolved

The IBM Support Center Technote “Run One Command When a Situation is True and Another Command When False”, which can be found at http://www.ibm.com/support/docview.wss?uid=swg21407263, describes in detail how to define a workflow policy to address this type of requirement.

- Avoid using the “take action on every item” option for situations that send email. A potential email flood may occur if a situation has many rows in a monitored table causing it to become true. A situation requiring this option may not be the best candidate to use email as a notification option; it should be examined to determine what the “worst case” might be for the potential number of emails generated. Another, more complex solution would be to code the command procedure to analyze the incoming requests, determine if this is a “burst” (e.g. multiple requests with the same timestamp value), and then implement logic to minimize the number of emails actually sent.

Summary

Although IBM Tivoli Monitoring does not have a built-in email client, it is not a difficult task to develop such an interface. The existence of mail sending function on operating system platforms, and the ease of developing a custom client using Java or any programming language, makes defining this interface very possible. There are security and customization considerations that careful planning, testing, and validation can address, so that sending email alerts can be a viable option in many environments. While this paper does not cover every potential software option available that can be used for sending mail, the flow and examples described can be used as a template to get started, across many combinations of email functions and operating system platforms supported by ITM.