Redbooks Paper

Jason Herne

Systems Management APIs for z/VM

Introduction

This IBM® Redpaper describes the z/VM® Systems Management Application Programming Interface (API), a programming interface to manage many aspects of a z/VM system. The interface is based on the Open Network Computing Remote Procedure Call (ONC/RPC) protocol, a platform-independent and language-neutral protocol for remote procedure execution defined in RFC1831. (This document is available on the Web; see “Referenced Web sites” on page 36.)

ONC/RPC offers programmers the freedom to choose many supported platforms and languages when developing system management applications for z/VM. Operation of the Systems Management API relies on a z/VM virtual machine (VSMSERVE by default) that listens for incoming RPC requests. Authorized requests are either processed by VSMSERVE, or passed to the z/VM directory management system.

The Systems Management API is TCP/IP-based and provides a rich set of functions to handle many aspects of z/VM management, such as:

- Creating and deleting virtual machines
- Defining guest storage
- Changing permission classes and modifying passwords
- Granting guest access to devices
- Creating, modifying, and deleting z/VM virtual networks (VM guest LAN and VSWITCH)

In this paper, you will learn about:

System requirements

The Systems Management API Version 1 was introduced in z/VM Version 4.4.0. This z/VM version and a directory manager are required for the Systems Management API to work properly. The IBM-supplied Directory Maintenance Facility (DirMaint™) product supports the interface; other commercial directory managers may support it as well. Check your directory manager's documentation for more information.

Configuring the Systems Management API

In discussing the topics in this section, we assume that you have some knowledge of z/VM system administration.

DirMaint configuration for the Systems Management API

The Systems Management API requires a directory manager. This section provides a quick-start procedure to help you configure DirMaint for use with the Systems Management API; we assume DirMaint is installed and operational.

Note: For details on general DirMaint installation and configuration, consult the *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6024. See the product documentation if you use another directory manager product.

1. Grant authorization to MAINT by adding the following lines to the AUTHFOR CONTROL file on the DIRMAINT 1DF disk:

```
ALL  MAINT       *                   140A ADGHMOPS
ALL  MAINT       *                   150A ADGHMOPS
```

2. In order for VSMSERVE to correctly operate with DirMaint, execute the **DIRM FOR VSMSERVE D8ONECMD FAIL** command from MAINT.
3. Create the file CONFIGSM DATADVH on the DIRMANT 1DF disk. To allow VSMSERVE unchecked access to all DirMaint commands, add the line:

```
ALLOW_ASUSER_NOPASS_FROM= VSMSERVE *
```

Note: DirMaint must be restarted for the changes to take effect.

Portmap configuration

The port mapper daemon maps RPC programs to TCP/IP ports. As it is RPC-based, the Systems Management API relies on the portmap daemon. Configuring the portmap daemon in z/VM is fully explained in the manual entitled *TCPIP Planning and Customization*, SC24-6019; this chapter serves as a quick reference.

1. In many cases, the portmap daemon does not require configuration; it is configured by default. Simply **XAUTOLOG** the PORTMAP guest to start the service.
2. Verify configuration by executing the **RPCINFO -P** command as MAINT. The following output indicates the portmap daemon is functioning:

```
program vers proto   port
100000     2   udp    111  portmapper
100000     2   tcp    111  portmapper
Ready; T=0.01/0.01 16:22:37
```

Note: If the **RPCINFO** command is not found, link to and access the disk containing TCP/IP commands (TCPIP 592 by default) by using:

```
LINK TCPIP 592 592 RR
ACC 592 Q
```

VSMSERVE configuration

In this section, we summarize the steps to configure the VSMSERVE guest. More information can be found in *Systems Management Application Programming*, SC24-6063.

1. The first step is to allow VSMSERVE to bind to the default port (845). Add the following line to the appropriate section of the PROFILE TCPIP file on the TCPMAINT 198 disk.

   ```
   845  TCP VSMSERVE           ; VSM API Server
   ```

   Note: The TCP/IP stack must be restarted for the change to take effect.

2. Before running VSMSERVE, verify the VSMSERVE AUTHLIST file on the VSMSERVE 191 disk authorizes users to execute Systems Management API procedures. The following entries are included by default; nothing more is needed for simple setups:

   ```
   MAINT    ALL          ALL
   VSMSERVE ALL          ALL
   ```

   The first column defines the authorized user. The second column specifies which virtual machines the user is allowed to affect. The third column defines which procedures the user is allowed to call.

   The default definition grants MAINT and VSMSERVE authority to execute all Systems Management API procedures for any virtual machine.

3. Once VSMSERVE is configured, log the guest on.

4. Test that the Systems Management API is registered with the portmap daemon using the **RPCINFO -P** command (as MAINT). The expected output appears as:

   ```
   PROGRAM VERS PROTO PORT
   100000     2   udp  111  portmapper
   100000     2   tcp  111  portmapper
   300762     1   tcp  845
   Ready; T=0.01/0.01 16:49:30
   ```

5. Note the entry for program number 300762 on port 845. To see if the Systems Management API is functioning, issue the **RCPINFO -T 127.0.0.1 300762** command as MAINT. Expected output looks like:

   ```
   program 300762 version 1 ready and waiting
   ```

Systems Management API theory of operation

Systems Management API relies on the ONC/RPC protocol for remote procedure execution between a management system (the client) and the z/VM host (the server). In order to execute an RPC call, the name of the procedure and procedural arguments must be specified in a format known as eXternal Data Representation (XDR). XDR is common to all ONC/RPC implementations and allows endpoints to exchange platform-independent, language-neutral messages.

The RPC caller encodes and sends an XDR message over the network; it is received and decoded by the server. The server returns an XDR message that may contain a return value and any return parameters.

XDR files

Before using Systems Management API RPC procedures, you must generate stub source files in the desired implementation language (C, C++, or Java for instance) from the XDR definition file from the MAINT 193 disk. These define the data structures and remote procedures that can be called. The generated files are used on the code development system.

Using Remote Tea to generate Java source files

Remote Tea is a Java-based ONC/RPC implementation licensed under GNU Lesser General Public License (LGPL). Remote Tea consists of two components:

- A run-time component to generate and transmit XDR data streams.

  This component must be installed on any system that communicates with the Systems Management API.

- A development component to generate Java stub files.

  Remote Tea includes the **jrpcgen** command to generate Java stub files from an XDR definition. We use this to generate the Java source to access the Systems Management API.

Obtaining and installing Remote Tea

The latest version of Remote Tea can be found at:

http://acplt.plt.rwth-aachen.de/ks/english/remotetea.html

Note: In this example, we use Remote Tea Version 1.0.2.

1. Install Remote Tea as explained on the Web site on both the development workstation (where the Java application is coded) and on the Systems Management API client workstation (where the application runs).
2. To test installation, compile a simple Java program that imports a Remote Tea library class:

```
import org.acplt.oncrpc.*;
public class example1
{
    public static void main(String[] args)
    {
        System.out.println("Remote Tea has been located.");
    }
}
```

   If compilation succeeds, Remote Tea is properly installed. Incorrect installation can result in compile-time error messages such as:

```
package org.acplt.oncrpc does not exist
```

   In this case, the error indicates an imported class cannot be found. Return to the Remote Tea installation instructions and reinstall.

Obtaining the Systems Management API XDR definitions

The XDR definitions are available on the MAINT 193 disk. Copy the XDR definition file (DMSVSMA X) from the MAINT 193 disk to your development workstation.

Generating Java source files from the XDR file

In this section, we provide the steps necessary to generate Java source files from the XDR files to interact with the Systems Management API.

1. To begin, execute the **jrpcgen** command provided with Remote Tea against the DMSVSMA.X file:

   ```
   java -jar jrpcgen.jar -p RPCPackage -nobackup DMSVSMA.X
   ```

   Tip: Replace RPCPackage with the actual package name you wish to use. A package name is a Java construct that allows developers to group a set of related objects. If unsure what package name to use, we suggest examples.smapi.

   If **jrpcgen** is not in the current directory, an error message results:

   ```
   Exception in thread "main" java.util.zip.ZipException: No such file or directory
   ```

   To correct the error, switch to the directory containing **jrpcgen**, or provide the full path to the jrpcgen.jar file (found in the classes subdirectory of the Remote Tea zip file).

2. Because a constant is defined twice in the XDR file, **jrpcgen** fails the first time it is executed with the error message:

   ```
   jrpcgen: error in line 77: const identifier "RS_AUTHERR_CONNECT" already defined
   ```

   To correct the error, remove the second definition of the RS_AUTHERR_CONNECT constant (on line 77 in the DMSVSMA.X file). The offending line is:

   ```
   77:const RS_AUTHERR_CONNECT   =   8;
   ```

   Tip: If you replace this line with a blank line (instead of deleting it), line numbers in the file remain the same. We refer to the line numbers for other changes later.

3. After correcting this condition, **jrpcgen** fails again because preprocessor directives are not supported (as stated on the Remote Tea Web site). The message in this case is:

   ```
   Illegal character "#"
   ```

   To correct the problem, we remove all preprocessor directives (as they are z/VM specific and are required only for client code running on z/VM). The lines (and their respective line numbers) that should be removed from the DMSVSMA.X file are shown in Example 1.

Example 1 Lines to remove from the DMSVSMA.X file

```
154: #ifdef __VM__
155: #ifdef RPC_HDR
156: %/* For VM, the charNA type cannot be opaque.  Instead, we */
157: %/* define the charNA type to be what RPCGEN gives us.     */
158: %/* Then later we define the xdr_charNA() routine also to  */
159: %/* be the same as what RPCGEN would give us, plus the     */
160: %/* translation to and from ascii to ebcdic and visa versa.*/
161: %typedef struct {
162: %    u_int charNA_len;
163: %    char *charNA_val;
164: %} charNA;
165: %bool_t xdr_charNA();
166: #endif /* RPC_HDR */
```

```
167: #else  /* __VM__ */
176: #endif /* __VM__ */
2225: #ifdef __VM__
2226: #ifdef RPC_XDR
2227: %/* For VM, the charNA type cannot be opaque.  Instead,...
2228: %/* define the charNA type to be what RPCGEN gives us...
2229: %/* Now we define the xdr_charNA() routine to be the same...
2230: %/* as what RPCGEN would give us, plus the translation to...
2231: %/* and from ascii to ebcdic and visa versa...
2232: %
2233: %/* The externs are defined in VMRPC TXTLIB */
2234: %extern char asciitoebcdic[];        /* If tables not...
2235: %extern char ebcdictoascii[];        /* If tables not...
2236: %#pragma linkage(CMXLATE, OS)
2237: %extern CMXLATE();                   /* Translation...
2238: %
2239: %extern char rpcAToE[256];           /* Ascii to Ebcdic...
2240: %extern char rpcEToA[256];           /* Ebcdic to Ascii...
2241: %extern int  tablesLoaded;           /* Initially not...
2242: %
2243: %bool_t
2244: %xdr_charNA(xdrs, objp)
2245: %    XDR *xdrs;
2246: %    charNA *objp;
2247: %{
2248: %
2249: %    /* If translate tables are not loaded..load them now */
2250: %    if (tablesLoaded == 0) {
2251: %       loadRPCTranslateTables();
2252: %    }
2253: %
2254: %    /* If we are encoding, encode the data now */
2255: %    if (xdrs->x_op == XDR_ENCODE) {
2256: %       if (tablesLoaded) { /* Use what we loaded */
2257: %          CMXLATE(objp->charNA_val,
2258: %                  rpcEToA,
2259: %                  objp->charNA_len);
2260: %       }
2261: %       else { /* Use the one linked with us */
2262: %          CMXLATE(objp->charNA_val,
2263: %                  ebcdictoascii,
2264: %                  objp->charNA_len);
2265: %       }
2266: %    }
2267: %
2268: %    /* Now do what the rpcgen'ed xdr_charNA does */
2269: %    if (!xdr_bytes(xdrs,
2270: %                   (char **)&objp->charNA_val,
2271: %                   (u_int *)&objp->charNA_len, ~0)) {
2272: %         return (FALSE);
2273: %    }
2274: %
2275: %    /* If we are decoding, decode the data now */
2276: %    if (xdrs->x_op == XDR_DECODE) {
2277: %       if (tablesLoaded) { /* Use what we loaded */
2278: %          CMXLATE(objp->charNA_val,
2279: %                  rpcAToE,
2280: %                  objp->charNA_len);
2281: %       }
2282: %       else { /* Use the one linked with us */
```

```
2283: %          CMXLATE(objp->charNA_val,
2284: %                  asciitoebcdic,
2285: %                  objp->charNA_len);
2286: %       }
2287: %    }
2288: %
2289: %    return (TRUE);
2290: %}
2291: #endif /* RPC_XDR */
2292: #endif /* __VM__ */
```

4. At this point, **jrpcgen** should run without error.

> **Note:** The **jrpcgen** command produces almost 300 Java source files corresponding to the Systems Management API procedures and data structures.

Although **jrpcgen** uses the specified package name, the generated Java source files are not placed in the proper directory structure. To correct this, you can either:

- Create the proper directory structure, then run **jrpcgen** from that directory.
- Move the generated files to the proper location after they are created.

In our example, we used the package name examples.smapi and created the examples and examples/smapi directories for the generated Java source files.

One final problem remains: the code generated by **jrpcgen** is slightly flawed. The Java source does not compile as there is a problem translating the XDR file to Java.

> **Note:** We used Remote Tea Version 1.0.2. A later version may not have this problem.

The reported compilation errors are:

```
AUTHLISTADD_res.java:26: incompatible types
found   : examples.smapi.Return_Code
required: int
        switch ( rc ) {
IMAGEACTIVATE_resfail_buffer.java:26: incompatible types
found   : examples.smapi.Reason_Code
required: int
        switch ( rs ) {
```

The first error is due to an improper use of the Return_Code data type. The intent is to access the value property of the rc variable.

To correct the error, we use a simple Bash shell script suitable for Linux® to find and replace all instances of rc with rc.value:

```
#!/bin/bash
#-------------------------------------------------------------------------------
for n in *.java; do sed 's/switch ( rc ) /switch ( rc.value )/' $n > $n'2'; done
for n in *.java; do mv $n'2' $n; done
#-------------------------------------------------------------------------------
```

> **Note:** Create and execute the script in the same directory as the Java source files.

The second error is also due to an improper variable usage - this time, the rs variable. To correct this error, run a similar script:

```
#!/bin/bash
#-------------------------------------------------------------------------------
for n in *.java; do sed 's/switch ( rs ) /switch ( rs.value )/' $n >$n'2'; done
for n in *.java; do mv $n'2' $n; done
#-------------------------------------------------------------------------------
```

Again, create and execute the script in the same directory as the Java source files.

Once these fixes are applied, the generated Java source files should compile cleanly. The Java classes can be imported into a Java application that uses the Systems Management API.

Note: If you still encounter errors, the code fixes may have been incorrectly applied. Erase the Java source files, regenerate them, and reapply the code fixes.

Basic Systems Management API structure

Parameters accepted by a Systems Management API procedure are object types specific to the procedure. A naming convention associates an object type to its procedure. For the examples presented in this section, we use the Systems Management API procedures:

- QUERY_DIRECTORY_MANAGER_LEVEL_1 (QUERYDIRMGRLEVEL)
- IMAGE_ACTIVATE_1 (IMAGEACTIVATE)
- IMAGE_CREATE_1 (IMAGECREATE)
- IMAGE_NAME_QUERY_1 (IMAGENAMEQUERY)

The name in parenthesis name corresponds to the prefix used in the naming convention. The naming conventions and examples used here are specific to the Java interface generated by **`jrpcgen`**. They are not guaranteed to be consistent across languages.

Args object

Each procedure takes an argument object that contains all argument for the procedure. Different procedures accept a different set of arguments. Therefore, a different argument object exists for each procedure. The naming convention for arguments objects is:

*PROCNAME*_args

The argument object names for the example procedure calls are:

- QUERYDIRMGRLEVEL_args
- IMAGEACTIVATE_args
- IMAGECREATE_args
- IMAGENAMEQUERY_args

To illustrate, we examine the data members of the IMAGEACTIVATE_args object:

- Session_Token SessionToken

  We discuss the session token in “Session token” on page 9

- String TargetIdentifier

  This is a string that represents the name of the guest that we wish to activate.

Before executing the IMAGEACTIVATE procedure, we must assign values to these variables.

Note: A complete listing of all Systems Management API procedures and arguments is available in *Systems Management Application Programming*, SC24-6063.

When a procedure call is made, the appropriate arguments object is passed as a parameter to the function that performs the Systems Management API procedure call.

Session token

The session token is a value passed between the Systems Management API server and client and is used to track the connection. When a client connects to the Systems Management API server and authenticates, a session token is returned. The client must send the session token back to the server on the next procedure call. A missing or invalid token causes the call to immediately fail. In this case, the client must reconnect and authenticate before another call succeeds.

Note: Each procedure returns a session token. Be sure the token is saved and reused for the next procedure call.

Procedure call

Procedure calls exist as methods of the Systems Management API client object (of type dmsvsmaClient). The client contains methods that correspond to the documented Systems Management API procedure calls. Each method accepts a single parameter (an argument object specific to the RPC method). To illustrate, invocation of the RPC methods in our example would be coded as:

```
client.QUERY_DIRECTORY_MANAGER_LEVEL_1( ... );
client.IMAGE_ACTIVATE_1( ... );
client.IMAGE_CREATE_1( ... );
client.IMAGE_NAME_QUERY_1( ... );
```

Result objects

All Systems Management API procedures return a result object. As with argument objects, each procedure returns its own result object. The result object is used to return:

- Information returned from the remote procedure
- A new session token
- A return code indicating success or failure

The naming conventions for result objects are:

- QUERYDIRMGRLEVEL_res
- IMAGEACTIVATE_res
- IMAGECREATE_res
- IMAGENAMEQUERY_res

Each result object contains different data members, but all contain some common data members. They are listed below:

- *PROCNAME*_resok resok

  The resok object contains a valid session token if the procedure call succeeds.

- *PROCNAME*_resfail resfail

  The resok object contains a valid session token if the procedure call fails.

- Return_Code rc

This code corresponds to an error (zero indicates success). If successful, the resfail object is null and the new session token can be found in resok. In the case of failure, the resok object is null, and the new session token can be found in resfail.

Important: Be sure to check the value of the return code before taking any action based on the return of this procedure call. This is particularly true when accessing the new session token.

Within the resok and resfail objects, another numerical code is stored in the Reason_Code rs data member. In some cases, this provides additional information. For example, IMAGE_CREATE_1 sets rs to 8 if the call causes the directory to go offline.

Note: See *Systems Management Application Programming*, SC24-6063, for information on the return code and rs values for each Systems Management API procedure call.

To illustrate this point, in Example 2, we examine a section of code that checks the return code:

Example 2 Return code routine

```
result = client.IMAGE_CREATE_1(args);
int rc;
int rs;
Session_Token sessionToken;
rc = result.rc.value;
if (result.resok != null)
{
   //There was not an error, we use resok.
   sessionToken = result.resok.SessionToken;
   rs = result.resok.rs.value
}
else
{
   //There was an error, use resfail.
   sessionToken = imageCreateResult.resfail.SessionToken;
   rs = result.resfail.rc.value;
}
```

In this example, if the call succeeded, the resok object is not null. Otherwise, we use the resfail object.

Based on the rc and rs values, additional failure information may be available. In Example 3 on page 10, we examine the rc and rs values from IMAGE_CREATE:

Example 3 Error checking of rc and rs values

```
if (rc != 0)
{
   //Handle the error
   if (rc == 100 && rs == 8)
      System.out.println("ESM Authentication error.");
   else if (rc == 120 && rs == 0)
      System.out.println("Bad username and/or password.");
   else if (rc == 128 && rs == 0)
      System.out.println("Password expired.");
   else if (rc == 188)
      System.out.println("ESM Failure.");
```

```
   else System.out.println("login error (unknown): rc=" +
      rc + " rs=" + rs + ".");
   return;
}
//If we get here, there was no error.
```

If rc is 0, the call has succeeded. In this case, no information other than the session token, rc, and rs values are returned.

Note: In this type of routine, always provide an else clause to catch unexpected (and possibly undocumented) error conditions. Error codes are documented in *Systems Management Application Programming*, SC24-6063.

Asynchronous operations

Some Systems Management API procedures execute in what is referred to as *asynchronous mode*. In this case, the procedure returns control to the caller before the call is actually complete. This is usually the case for commands that take some time to complete (such as clearing of a DASD pack on guest deletion).

The special QUERYASYNCOP call handles asynchronous operation. When an operation completes asynchronously, it returns an rc value of 592. The rs value is an operation ID to pass as a parameter to QUERYASYNCOP:

- If the operation is still in progress, the returned rc/rs values are 0/104.
- If the operation has completed, the return values are 0/100.
- Any other values indicate an error has occurred.

Note: In the event of an asynchronous call error, the rc/rs returned values are 0/108. In this case, no additional error information on the exact cause of the failure is available from the Systems Management API.

Asynchronous operation is illustrated in Example 5 on page 16.

Establishing a Systems Management API connection

To initiate a Systems Management API connection, create an instance of the Systems Management API client object (Java type dmsvsmaClient). The constructor accepts connection-related parameters. After the connection is established, authentication must occur. To authenticate, call the LOGIN method of the client object. Before calling LOGIN, create a LOGIN_args object to specify authentication information.

Sample code that creates a client object and authenticates follows:

```
InetAddress serverIP = InetAddress.getByName("192.168.0.100");
int program = 300762;
int version = 1;
int port = 845;
int proto =  OncRpcProtocols.ONCRPC_TCP
dmsvsmaClient client;
client = new dmsvsmaClient(serverIP,program,version,port,proto);
loginArgs = new LOGIN_args();
loginArgs.AuthenticatedUserid = "MAINT";
loginArgs.loginpw = "MAINT";
loginResult = client.LOGIN_1(loginArgs);
```

The code first sets the required values for the program, version, port, and protocol arguments of the client constructor:

- **Program**
  The program argument is an integer data type that specifies the Systems Management API is to be to invoked. The portmap service uses the program number to identify the server program to execute. For z/VM 4.4.0, the Systems Management API program number is 300762 (documented in *Systems Management Application Programming*, SC24-6063 and identified by the number at the end of the DMSVSMA.X file).
- **Version**
  The version argument (of type integer) represents the version of the program to execute. This allows multiple versions of the same program to run on the same server. For z/VM 4.4.0, use Version 1 (as documented in *Systems Management Application Programming*, SC24-6063).
- **Port**
  The port argument specifies the Systems Management API port (845).
- **Protocol**
  The protocol argument specifies the transport layer protocol to use. Use the Remote Tea library constant OncRpcProtocols.ONCRPC_TCP (TCP protocol).

The connection is established when an instance of the dmsvsmaClient object is created.

Note: The sample code neglects proper error handling to save space. Production code should catch exceptions.

```
java.net.UnknownHostException
OncRpcException
java.net.ConnectException
java.io.IOException
```

Java requires all but java.net.ConnectException to be caught.

After the connection is established, we create a LOGIN_args object and set its AuthenticatedUserid and loginpw data members. The userid needs to be a z/VM user authorized to use the Systems Management API interface (as discussed in “VSMSERVE configuration” on page 3). Provide the VM login password for this guest.

We call the LOGIN_1 procedure, passing the LOGIN_args parameter. On return, examine the rc and rs values:

```
if (loginResult.rc.value != 0)
{
   //Handle the error
   int rc = loginResult.rc.value;
   int rs = loginResult.resfail.rs.value;

   if (rc == 100 && rs == 8)
      println("ESM Authentication error.");
   else if (rc == 120 && rs == 0)
      println("Bad username and/or password.");
   else if (rc == 128 && rs == 0)
      println("Password expired.");
   else if (rc == 188 )
      println("ESM Failure.");
   else println("login error (unknown): rc=" + rc + " rs=" + rs);
}
```

If successful, other Systems Management API calls can be made.

Using Systems Management API procedure calls

This section provides some practical applications for using Systems Management API procedure calls in your everyday systems administration.

Using the IMAGE_NAME_QUERY procedure to list all guests

On occasion, you may have a need to know all of the guests defined to the z/VM system. Example 4 provides code to do this using the IMAGE_NAME_QUERY procedure.

Example 4 Example code to list all z/VM users

```
package examples;
import org.acplt.oncrpc.*;
import examples.smapi.*;
import java.net.*;  //for InetAddress
public class QImageNames
{
   private static dmsvsmaClient client; //Client object
   private static Session_Token sessionToken;

   private static int rc; // return code
   private static int rs; // reason code

   public static void main(String[] args)
   {
      //Set connection information
      int program = 300762;
      int version = 1;
      int port = 845;
      int proto =  OncRpcProtocols.ONCRPC_TCP;

      try
      {
         InetAddress serverIP = InetAddress.getByName("192.168.0.100");

         System.out.println("Connecting...");
         client = new dmsvsmaClient(
            serverIP,program,version,port,proto);
         System.out.println("Connected.");

         //Set login arguments
         LOGIN_args loginArgs = new LOGIN_args();
         loginArgs.AuthenticatedUserid = "MAINT";
         loginArgs.loginpw = "MAINT";
         //Making login call
         System.out.println("Logging in...");
         LOGIN_res loginResult = client.LOGIN_1(loginArgs);
         //Find rc/rs and save session token
         rc = loginResult.rc.value;
         if (loginResult.resok != null)
         {
            //There was not an error, we use resok.
            sessionToken = loginResult.resok.SessionToken;
            rs = loginResult.resok.rs.value;
         }
```

```
else
{
   //There was an error use resfail.
   //There is no session token on a failed login.
   rs = loginResult.resfail.rs.value;
}
//Check for errors
if (rc != 0)
{
   //Handle the error
   if (rc == 100 && rs == 8)
      System.out.println("ESM Authentication error.");
   else if (rc == 120 && rs == 0)
      System.out.println("Bad username and/or password.");
   else if (rc == 128 && rs == 0)
      System.out.println("Password expired.");
   else if (rc == 188)
      System.out.println("ESM Failure.");
   else System.out.println("login error (unknown): rc=" +
      rc + " rs=" + rs + ".");
   return;
}
System.out.println("Logged In.");

//Set arguments for IMAGE_NAME_QUERY_1.
IMAGENAMEQUERY_args nameQueryArgs =
   new IMAGENAMEQUERY_args();
nameQueryArgs.SessionToken = sessionToken;

//Make the call
System.out.println("Performing image name query.");
IMAGENAMEQUERY_res nameQueryResult;
nameQueryResult = client.IMAGE_NAME_QUERY_1(nameQueryArgs);

//Find rc/rs and save session token
rc = nameueryResult.rc.value;
if (nameQueryResult.resok != null)
{
   //There was not an error, we use resok.
   sessionToken = nameQueryResult.resok.SessionToken;
   rs = nameQueryResult.resok.rs.value;
}
else
{
   //There was an error, use resfail.
   sessionToken = nameQueryResult.resfail.SessionToken;
   rs = nameQueryResult.resfail.rs.value;
}

if (rc != 0)
{
   //Handle the error
   if (rc == 8 && rs == 0)
      System.out.println("Bad session token.");
   else if (rc == 100 && rs == 8)
      System.out.println("ESM authentication error.");
   else if (rc == 100 && rs == 12)
      System.out.println("Dir. mgr. auth. error.");
   else if (rc == 100 && rs == 16)
      System.out.println("SMAPI server auth. error.");
```

```
            else if (rc == 196 )
               System.out.println("SMAPI Server error.");
            else if (rc == 400 && rs == 4 )
               System.out.println("Image definition not found.");
            else if (rc == 500 && rs == 8 )
               System.out.println("Dir. mgr. not available.");
            else if (rc == 596 )
               System.out.println("Dir. mgr. err. rs=" + rs + ".");
            else System.out.println("Error (unknown): rc=" +
               rc + " rs=" + rs + ".");

            return;
         }

         //Examine and display results.
         sessionToken = nameQueryResult.resok.SessionToken;

         IMAGENAMEQUERY_Return_Buffer nextEntry;
         nextEntry = nameQueryResult.resok.returnbuffer;
         while(nextEntry != null)
         {
            System.out.println(nextEntry.Image_Name);
            nextEntry = nextEntry.next_entry;
         }
         //Logout
         System.out.println("Disconnecting...");
         LOGOUT_args logoutArgs = new LOGOUT_args();
         logoutArgs.SessionToken = sessionToken;
         client.LOGOUT_1(logoutArgs);
         System.out.println("Disconnected.");
      }
      catch(OncRpcException e)
      {
         e.printStackTrace();
      }
      catch(UnknownHostException e)
      {
         e.printStackTrace();
      }
      catch(java.io.IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

In summary, the code in Example 4 achieves the desired results by:

1. Initiating a connection to the Systems Management API server and login as MAINT.
2. Creating the arguments object for the IMAGE_NAME_QUERY using its single parameter (the session token), and invoking the procedure.
3. On return, checking the return values and proceeding only if no error is encountered.

In this example, the return value is returned as a list. The Systems Management API uses a linked-list style data structure named IMAGENAMEQUERY_Return_Buffer containing two data members:

- Image_Name
  This String contains the name of a single guest.

- next_entry
  This points to the next IMAGENAMEQUERY_Return_Buffer element.

  The last element has been reached when the next_entry data member is null:

```
IMAGENAMEQUERY_Return_Buffer nextEntry;
nextEntry = nameQueryResult.resok.returnbuffer;
while(nextEntry != null)
{
  System.out.println(nextEntry.Image_Name);
  nextEntry = nextEntry.next_entry;
}
```

Creating and deleting z/VM guests using a Java application

In the following example, we provide a simple command line application to create and delete z/VM guests. The application uses the IMAGE_CREATE and IMAGE_DELETE Systems Management API procedures. We use a custom exception defined as CreatorException:

```
package examples;
public class CreatorException extends java.lang.Exception
{
    public CreatorException() { }
    public CreatorException(String msg)
    {
        super(msg);
    }
}
```

Prototype files

The application utilizes user directory prototypes, a template directory entry used to define a guest. The name of the prototype file is passed to DirMaint, and the statements define the guest's directory entry. With DirMaint, you can create a prototype with XEDIT on the DIRMAINT 1DF disk. Alternatively, you can use the Systems Management API procedure PROTOTYPE_CREATE. For more information on prototype files, refer to the **ADD** command documented in *Directory Maintenance Facility Command Reference*, SC24-6025.

The code is shown in Example 5.

Example 5 Example code to create and delete z/VM guests

```
package examples;
import org.acplt.oncrpc.*;
import examples.smapi.*;
import java.net.*;  //InetAddress
import java.io.*;  //InputStreamReader, BufferedReader
public class GuestCreator
{
   private dmsvsmaClient client; //Client object
   private Session_Token sessionToken;

   private int rc; // return code
   private int rs; // reason code

   //Connect/Login information
   private String strIPAddress; //IP address for SMAPI server
   private String port; // Port SMAPI server is listening on.
   private String loginName; //Guest name used to authenticate.
   private String loginPassword; //Password for authentication.
```

```
//Create/Delete information
private boolean isCreateJob;  //True if we creating guest
private String guestName;  //Guest to create or delete
private String guestPassword; //Password for created guest.
private String protodirFile; //Protodir File for new guest.



public static void main(String[] args)
{
   GuestCreator gc = new GuestCreator(args);
}

public GuestCreator(String[] args)
{
   try
   {
      //Check for correct argument usage.
      if (args.length == 4)
      {
         //Read args
         strIPAddress = args[0];
         port = args[1];
         loginName = args[2];
         loginPassword = args[3];

         getInput();
         connectAndLogin();
         if(isCreateJob) createGuest();
         else deleteGuest();

         disconnect();
      }
      else
      {
         printUsage();
      }
   }
   catch(OncRpcException e)
   {
      e.printStackTrace();
   }
   catch(UnknownHostException e)
   {
      e.printStackTrace();
   }
   catch(java.io.IOException e)
   {
      e.printStackTrace();
   }
   catch(CreatorException e)
   {
      System.out.println(e.getMessage());

      try
      {
         disconnect();
      }
      catch (Exception e2)
      {
```

```
            //We do nothing on error at this point as we have
            //already seen the cause of this error.  Now we just
            //want to try and close the connection.
         }
      }
   }
   private void connectAndLogin()
      throws OncRpcException,UnknownHostException,
      java.io.IOException, CreatorException
   {
      //Set connection information
      InetAddress serverIP = InetAddress.getByName(strIPAddress);
      int rpcPort = Integer.parseInt(port);
      int program = 300762;
      int version = 1;
      int proto =  OncRpcProtocols.ONCRPC_TCP;

      System.out.println("Connecting...");
      client = new dmsvsmaClient(
         serverIP,program,version,rpcPort,proto);
      System.out.println("Connected.");

      //Creating Login arguments
      LOGIN_args loginArgs = new LOGIN_args();
      loginArgs.AuthenticatedUserid = "MAINT";
      loginArgs.loginpw = "MAINT";
      //Making login call
      System.out.println("Logging in...");
      LOGIN_res loginResult = client.LOGIN_1(loginArgs);
      //Getting rc and rs and session token
      rc = loginResult.rc.value;
      if (loginResult.resok != null)
      {
         //There was not an error, we use resok.
         sessionToken = loginResult.resok.SessionToken;
         rs = loginResult.resok.rs.value;
      }
      else
      {
         //There was an error, use resfail.
         //There is no session token on a failed login.
         rs = loginResult.resfail.rs.value;
      }
      //Check for errors
      if (rc != 0)
      {
         //Handle the error
         if (rc == 100 && rs == 8)
            System.out.println("ESM Authentication error.");
         else if (rc == 120 && rs == 0)
            System.out.println("Bad username and/or password.");
         else if (rc == 128 && rs == 0)
            System.out.println("Password expired.");
         else if (rc == 188)
            System.out.println("ESM Failure.");
         else System.out.println("login error (unknown): rc=" +
            rc + " rs=" + rs + ".");
         throw new CreatorException("Program terminated.");
      }
      System.out.println("Logged In.");
```

```
}

private void disconnect()
   throws OncRpcException,java.io.IOException
{
   //Create args
   LOGOUT_args logoutArgs = new LOGOUT_args();
   logoutArgs.SessionToken = sessionToken;

   System.out.println("Disconnecting...");

   client.LOGOUT_1(logoutArgs);
   System.out.println("Disconnected.");
}

private void createGuest()
   throws OncRpcException,java.io.IOException, CreatorException
{
   //Set arguments for IMAGE_CREATE_1.
   IMAGECREATE_args imageCreateArgs = new IMAGECREATE_args();
   imageCreateArgs.SessionToken = sessionToken;
   imageCreateArgs.Image_Prototype = protodirFile;
   imageCreateArgs.TargetIdentifier = guestName;
   imageCreateArgs.Initial_Password = guestPassword;
   imageCreateArgs.Initial_Account_Number = "";
   //Make the call
   System.out.println("Creating guest " + guestName);
   IMAGECREATE_res resImageCreate;
   resImageCreate = client.IMAGE_CREATE_1(imageCreateArgs);
   //Find rc/rs and save session token
   rc = resImageCreate.rc.value;
   if (resImageCreate.resok != null)
   {
      //There was not an error, we use resok.
      sessionToken = resImageCreate.resok.SessionToken;
      rs = resImageCreate.resok.rs.value;
   }
   else
   {
      //There was an error, use resfail.
      sessionToken = resImageCreate.resfail.SessionToken;
      rs = resImageCreate.resfail.rs.value;
   }
   //Asynchronouss operation code
   if(rc == 592)
   {
      int asyncOperationID = rs;
      //Handle Asynchronous blocking
      asynchronousCall(asyncOperationID);
   }
   else if (rc != 0)
   {
      //Handle the error
      if (rc == 8 && rs == 0)
         System.out.println("Bad session token.");
      else if (rc == 24)
         System.out.println("Syntax error in function param.");
      else if (rc == 100 && rs == 8)
         System.out.println("ESM authentication error.");
      else if (rc == 100 && rs == 12)
```

```
            System.out.println("Dir. mgr. authentication error.");
         else if (rc == 100 && rs == 16)
            System.out.println("SMAPI server authentication error.");
         else if (rc == 196 )
            System.out.println("SMAPI Server error.");
         else if (rc == 400 && rs == 8)
            System.out.println("That guest already exists.");
         else if (rc == 400 && rs == 20)
            System.out.println("That PROTODIR file does not exist.");
         else if (rc == 444 && rs == 0)
            System.out.println("Password policy error.");
         else if (rc == 448 && rs == 0)
            System.out.println("Acct number policy error.");
         else if (rc == 596 )
            System.out.println("Dir. mgr. error.");
         else System.out.println("Create error (unknown): rc=" +
            rc + " rs=" + rs + ".");
         throw new CreatorException("Program terminated.");
      }
      //If we get here, the guest was created.
      System.out.println("Guest created.");
   }

   private void deleteGuest()
      throws OncRpcException,java.io.IOException, CreatorException
   {
      //Set arguments for IMAGE_DELETE_1.
      IMAGEDELETE_args imageDeleteArgs = new IMAGEDELETE_args();
      imageDeleteArgs.SessionToken = sessionToken;
      imageDeleteArgs.TargetIdentifier = guestName;

      Data_Security_Erase_Values dseVals;
      dseVals = new Data_Security_Erase_Values();
      dseVals.value = 0; //Use installation default values
      imageDeleteArgs.Data_Security_Erase = dseVals;
      //Make the call
      System.out.println("Deleting guest " + guestName + ".");
      IMAGEDELETE_res resImageDelete;
      resImageDelete = client.IMAGE_DELETE_1(imageDeleteArgs);
      //Find rc/rs and save session token
      rc = resImageDelete.rc.value;
      if (resImageDelete.resok != null)
      {
         //There was not an error, we use resok.
         sessionToken = resImageDelete.resok.SessionToken;
         rs = resImageDelete.resok.rs.value;
      }
      else
      {
         //There was an error, use resfail.
         sessionToken = resImageDelete.resfail.SessionToken;
         rs = resImageDelete.resfail.rs.value;
      }
      //Asynchronous operation code
      if(rc == 592)
      {
         int asyncOperationID = rs;
         //Handle asynchronous blocking
         asynchronousCall(asyncOperationID);
      }
```

```
   else if (rc != 0)
   {
      //Handle the error
      if (rc == 8 && rs == 0)
         System.out.println("Bad session token.");
      else if (rc == 24)
         System.out.println("Syntax error in function param.");
      else if (rc == 100 && rs == 8)
         System.out.println("ESM authentication error.");
      else if (rc == 100 && rs == 12)
         System.out.println("Dir. mgr. authentication error.");
      else if (rc == 100 && rs == 16)
         System.out.println("SMAPI server authentication error.");
      else if (rc == 196 )
         System.out.println("SMAPI Server error.");
      else if (rc == 400 && rs == 4)
         System.out.println("That guest was not found.");
      else if (rc == 400 && rs == 4)
         System.out.println("That guest is locked.");
      else if (rc == 400 && rs == 4)
         System.out.println("That guest cannotbe deleted.");
      else if (rc == 400 && rs == 20)
         System.out.println("That PROTODIR file does not exist.");
      else if (rc == 500 && rs == 4)
         System.out.println("Dir. mgr. not accepting updates.");
      else if (rc == 500 && rs == 8)
         System.out.println("Dir. mgr. is not available.");
      else if (rc == 596 )
         System.out.println("Dir. mgr. error.");
      else System.out.println("Delete error (unknown): rc=" +
         rc + " rs=" + rs + ".");
      throw new CreatorException("Program terminated.");
   }
   //If we get here, the guest was deleted.
   System.out.println("Guest deleted.");
}

private void asynchronousCall(int asyncOperationID)
   throws OncRpcException, java.io.IOException, CreatorException
{
   System.out.println("Waiting for return of asynchronous call.");
   int timer=0;

   while(true)
   {
      //Set arguments
      QUERYASYNCOP_args queryAsyncOpArgs;
      queryAsyncOpArgs = new QUERYASYNCOP_args();
      queryAsyncOpArgs.OperationID = asyncOperationID;
      queryAsyncOpArgs.SessionToken = sessionToken;

      //make the call
      QUERYASYNCOP_res queryAsyncOperationRes = null;
      queryAsyncOperationRes =
         client.QUERY_ASYNCHRONOUS_OPERATION_1(queryAsyncOpArgs);
      int rc;
      int rs;
      //Find rc/rs and save session token
      rc = queryAsyncOperationRes.rc.value;
      if (queryAsyncOperationRes.resok != null)
```

```
{
   //There was not an error, we use resok.
   sessionToken = queryAsyncOperationRes.resok.SessionToken;
   rs = queryAsyncOperationRes.resok.rs.value;
}
else
{
   //There was an error, use resfail.
   sessionToken = queryAsyncOperationRes.
      resfail.SessionToken;
   rs = queryAsyncOperationRes.resfail.rs.value;
}
//Check results and return only if operation is finished.
//Tell user in the case of an error, and loop if we
//are still waiting.
if (rc == 8)
   System.out.println(
      "Async error: Session Token invalid.");
else if (rc == 100 && rs == 8)
   System.out.println(
      "Async error: ESM Authentication Error.");
else if (rc == 100 && rs == 12)
   System.out.println(
      "Async error: Dir. mrg. authentication error.");
else if (rc == 100 && rs == 16)
   System.out.println(
      "Async error: SMAPI authentication error.");
else if (rc == 196)
   System.out.println(
      "Async error: Server error rs= " + rs + ".");
else if (rc == 500 && rs == 8)
   System.out.println(
      "Async error: Dir. mgr. not available.");
else if (rc == 596)
   System.out.println(
      "Async error: Dir. mgr. error code " + rs + ".");
//if we've made it here, there is no error presumably.
if (rc == 0 && rs == 100)
{
   //Call is complete, we can return.
   System.out.println("Asycnronous call returned OK.");
   return;
}
else if (rc == 0 && rs == 108)
{
   //Call has failed.
   System.out.println("Async. call failed.  No " +
      "futher information is available.");
   throw new CreatorException("Program terminated.");
}
else if (rc == 0 && rs == 104)
{
   //5s * 48 = 4m
   if (timer == 48)
   {
      //Only allow 4 minutes for async call to return.
      System.out.println(
         "Async error: Timed out waiting for "
            + "asynchronous call to return.");
```

```
            throw new CreatorException("Program terminated.");
         }

         timer++;
         try
         {
            //Sleep for 5 seconds and try again
            Thread.sleep(5000);
         }
         catch (java.lang.InterruptedException e)
         {
            e.printStackTrace();
            throw new CreatorException("Program terminated.");
         }
      }
      else
      {
         System.out.println("Asynchronous Call Error: RS=" +
            rs + " RC= " + rc + ".");
         throw new CreatorException("Program terminated.");
      }
   }
}

private void printUsage()
{
   System.out.println("Usage: java examples.GuestCreator "
      + "VM_IP_ADDRESS PORT LOGIN_NAME PASSWORD");
}

private void getInput()
{
   //Get create/delete
   System.out.println("To create a guest, type create");
   System.out.println("To delete a guest, type delete");
   System.out.println("press ctrl+C at any time to quit.");
   System.out.println("");

   String input = "";
   InputStreamReader isr = new InputStreamReader ( System.in );
   BufferedReader br = new BufferedReader ( isr );

   while( !(input.equals("create") || input.equals("delete")) )
   {
      System.out.print("Enter create or delete: ");

      try
      {
         input = br.readLine();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }

   if(input.equals("create"))
   {
      isCreateJob = true;
   }
```

```
   else if(input.equals("delete"))
   {
      isCreateJob = false;
   }

   //Get rest of needed info from user.
   getGuestName();
   if(isCreateJob)
{
      getProtodir();
      getPassword();
}
}

private void getGuestName()
{
   String input = "";
   InputStreamReader isr = new InputStreamReader ( System.in );
   BufferedReader br = new BufferedReader ( isr );

   while(input.length() > 8 || input.length() < 1)
   {
      System.out.print("Enter the guest name to create/delete: " );

      try
      {
         input = br.readLine();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
   guestName = input;
}

private void getProtodir()
{
   String input = "";
   InputStreamReader isr = new InputStreamReader ( System.in );
   BufferedReader br = new BufferedReader ( isr );

   while(input.length() > 8 || input.length() < 1)
   {
      System.out.print("Enter the protodir file to use: " );

      try
      {
         input = br.readLine();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
   protodirFile = input;
}

private void getPassword()
{
```

```
        String input = "";
        InputStreamReader isr = new InputStreamReader ( System.in );
        BufferedReader br = new BufferedReader ( isr );

        while(input.length() > 8 || input.length() < 1)
        {
           System.out.print("Enter the password for the new guest: " );

           try
           {
              input = br.readLine();
           }
           catch (IOException e)
           {
              e.printStackTrace();
           }
           guestPassword = input;
        }
     }
}
```

Systems Management API information for C programmers

In this section, we describe special considerations for C programmers who may want to use Systems Management API from Linux or another UNIX® variant. In this case, a version of glibc that supports RPC and the **rpcgen** command is required. When the **rpcgen** command is executed against the Systems Management API XDR definition file, it produces working C/C++ code.

Note: For Systems Management API, the generated files are:

DMSVSMAX C	the XDR wrapper routines.
DMSVSMAC C	the client stub source file
DMSVSMA H	common server/client include files

An ONC/RPC implementation for Microsoft® Windows® can be found at:

http://www.plt.rwth-aachen.de/ks/english/oncrpc.html

Example 6 illustrates a C implementation of a Systems Management API application similar to the one presented in Example 5 on page 16.

Example 6 C implementation of a Systems Management API application

```
#include <stdlib.h> //malloc(), free()
#include <stdio.h> //printf(), getline()
#include <rpc/rpc.h> //clnttcp_create(), CLIENT,
#include <rpc/clnt.h>
#include <netinet/in.h> //sockaddr_in, in_addr, htons()
#include <arpa/inet.h> //inet_addr()
#include <string.h> //strncpy(), strncmp(), strlen(), strrchr()
#include <unistd.h> //sleep()

#include "dmsvsma.h"
#include "dmsvsma_clnt.c"
#include "dmsvsma_xdr.c"
```

```
//Function declarations
int connectAndLogin();
void disconnect();
int createGuest();
int deleteGuest();
int asynchronousCall(int asyncOperationID);
void printUsage();
void getInput();
void getGuestName();
void getPassword();
void getProtodir();
void stripNewline(char* myStr);

//Globals
struct sockaddr_in* tcpSocket;
CLIENT* client;
char sessionToken[8];

int rc;
int rs;

//Connect/Login information
char strIPAddress[16]; //IP address used to connect to z/VM SMAPI server
char port[6]; // Port SMAPI server is listening on.
char loginName[9]; //Guest name used to authenticate.
char loginPassword[9]; //Password used to authenticate.

//Create/Delete information
int isCreateJob;  //1 if we are creating a guest, 0 if deleting.
char guestName[9];  //Guest to create or delete
char guestPassword[9]; //Password for newly created guest.
char protodirFile[9]; //Protodir File used to create new guest.

//****** main() ******//
int main(int argc, char* argv[])
{
   int returnValue;
   //Check for correct argument usage.
   if (argc == 5)
   {
      //Read args
      strncpy(strIPAddress, argv[1], 16);
      strncpy(port, argv[2], 6);
      strncpy(port, argv[3], 9);
      strncpy(port, argv[4], 9);

      getInput();

      returnValue = connectAndLogin();
      if (returnValue != 0) return(-1);

      if(isCreateJob) createGuest();
      else deleteGuest();

      disconnect();
   }
   else
   {
      printUsage();
```

```
      return(-1);
   }

   return(0);
}

//****** connectAndLogin() ******//
int connectAndLogin()
{
   //Create int for socket number
   int* rcpAny = (int*)malloc(sizeof(int));
   *rcpAny = RPC_ANYSOCK;

   //Create socket for SMAPI connection
   tcpSocket = malloc(sizeof(struct sockaddr_in));
   tcpSocket->sin_family = AF_INET;
   tcpSocket->sin_port = htons(atoi(port));
   tcpSocket->sin_addr.s_addr = inet_addr(strIPAddress);

   //Create Client object
   printf("Connecting...\n");
   client = clnttcp_create(tcpSocket, 300762, 1, rcpAny, 0, 0);

   //Error check the connection.
   if (client == NULL)
   {
      printf("Connection attempt failed.\n");
      return(-1);
   }

   printf("Connected.\n");

   //Creating Login arguments
   LOGIN_args* loginArgs = (LOGIN_args*)malloc(sizeof(LOGIN_args));
   loginArgs->AuthenticatedUserid = "MAINT";
   loginArgs->loginpw = "MAINT";

   //Making login call
   printf("Loggin In...\n");
   LOGIN_res* loginResult;
   loginResult = login_1(loginArgs, client);

   //Free unneeded memory
   free(rcpAny);
   free(loginArgs);

   //Getting rc and rs and session token
   rc = loginResult->rc;
   if(rc == 0)
   {
      //There was not an error, we use resok
      strncpy(sessionToken,
         loginResult->LOGIN_res_u.resok.SessionToken, 8);
      rs = loginResult->LOGIN_res_u.resok.rs;
   }
   else
   {
      //There was an error, use resfail.
        //There is no session token on a failed login.
      rs = loginResult->LOGIN_res_u.resfail.rs;
```

```
   }

   //Check for errors
   if (rc != 0)
   {
      //Handle the error
      if (rc == 100 && rs == 8)
        printf("ESM Authentication error.\n");
      else if (rc == 120 && rs == 0)
        printf("Bad username and/or password.\n");
      else if (rc == 128 && rs == 0)
        printf("Password expired.\n");
      else if (rc == 188)
        printf("ESM Failure.");
      else printf("login error (unknown): rc=%d rs=%d.\n", rc, rs);

      return(-1);
   }
   printf("Logged In.\n");
   return(0);
}

//****** disconnect() ******//
void disconnect()
{
   //Create args
   LOGOUT_args* logoutArgs = (LOGOUT_args*)malloc(sizeof(LOGOUT_args));
   strncpy(logoutArgs->SessionToken, sessionToken, 8);

   printf("Disconnecting...\n");

   //Call logout_1
   logout_1(logoutArgs, client);
   printf("Disconnected.\n");

   //Free unneeded memory
   free(logoutArgs);
   free(tcpSocket);
}

//****** createGuest() ******//
int createGuest()
{
   //Set arguments for IMAGE_CREATE_1.
   IMAGECREATE_args* imageCreateArgs =
      (IMAGECREATE_args*)malloc(sizeof(IMAGECREATE_args));
   imageCreateArgs->Image_Prototype = protodirFile;
   imageCreateArgs->TargetIdentifier = guestName;
   imageCreateArgs->Initial_Password = guestPassword;
   imageCreateArgs->Initial_Account_Number = "";

   //Copy session token
   strncpy(imageCreateArgs->SessionToken , sessionToken, 8);

   //Make the call
   printf("Creating guest %s.\n",guestName);
   IMAGECREATE_res* imageCrateResult;
   imageCrateResult = image_create_1(imageCreateArgs, client);

   //Free unneeded memory
```

```
free(imageCreateArgs);

//Find rc/rs and save session token
rc = imageCrateResult->rc;
if(rc == 0)
{
   //There was not an error, we use resok
   strncpy(sessionToken,
      imageCrateResult->IMAGECREATE_res_u.resok.SessionToken,8);
   rs = imageCrateResult->IMAGECREATE_res_u.resok.rs;
}
else
{
   //There was an error, use resfail.
    strncpy(sessionToken,
      imageCrateResult->IMAGECREATE_res_u.resfail.SessionToken,8);
   rs = imageCrateResult->IMAGECREATE_res_u.resfail.rs;
}

//Asynchronous operation code
if(rc == 592)
{
   int asyncOperationID = rs;

   //Handle asynchronous blocking
   int asyncRes = asynchronousCall(asyncOperationID);
   if (asyncRes != 0) return(-1);
}
else if (rc != 0)
{
   //Handle the error
   if (rc == 8 && rs == 0)
     printf("Bad session token.\n");
   else if (rc == 24)
     printf("Syntax error in function parameter.\n");
   else if (rc == 100 && rs == 8)
     printf("ESM authentication error.\n");
   else if (rc == 100 && rs == 12)
     printf("Dir. mgr. authentication error.\n");
   else if (rc == 100 && rs == 16)
     printf("SMAPI server authentication error.\n");
   else if (rc == 196)
     printf("SMAPI Server error.\n");
   else if (rc == 400 && rs == 8)
     printf("That guest already exists.\n");
   else if (rc == 400 && rs == 20)
     printf("That PROTODIR file does not exist.\n");
   else if (rc == 444 && rs == 0)
     printf("Password policy error.\n");
   else if (rc == 448 && rs == 0)
     printf("Acct number policy error.\n");
   else if (rc == 596)
     printf("Dir. mgr. error.\n");
   else printf("Create error (unknown): rc=%d rs=%d.\n",rc,rs);

   return(-1);
}

//If we get here, the guest was created.
printf("Guest created.\n");
```

```
}

//****** deleteGuest() ******//
int deleteGuest()
{
   //Set arguments for IMAGE_DELETE_1.
   IMAGEDELETE_args* imageDeleteArgs =
      (IMAGEDELETE_args*)malloc(
         sizeof(IMAGEDELETE_args));
   imageDeleteArgs->TargetIdentifier = guestName;

   Data_Security_Erase_Values* dseVals =
      (Data_Security_Erase_Values*)malloc(
         sizeof(Data_Security_Erase_Values));
   *dseVals = 0; //Use installation default values
   imageDeleteArgs->Data_Security_Erase = *dseVals;

   //Copy session token
   strncpy(imageDeleteArgs->SessionToken,sessionToken,8);

   //Make the call
   printf("Deleting guest %s.\n",guestName);
   IMAGEDELETE_res* imageDeleteResult;
   imageDeleteResult = image_delete_1(imageDeleteArgs, client);

   //Free unneeded memory
   free(imageDeleteArgs);
   free(dseVals);

   //Find rc/rs and save session token
   rc = imageDeleteResult->rc;
   if(rc == 0)
   {
      //There was not an error, we use resok
      strncpy(sessionToken,
         imageDeleteResult->IMAGEDELETE_res_u.resok.SessionToken,8);
      rs = imageDeleteResult->IMAGEDELETE_res_u.resok.rs;
   }
   else
   {
      //There was an error, use resfail.
       strncpy(sessionToken,
         imageDeleteResult->IMAGEDELETE_res_u.resfail.SessionToken,8);
      rs = imageDeleteResult->IMAGEDELETE_res_u.resfail.rs;
   }

   //Asynchronous operation code
   if(rc == 592)
   {
      int asyncOperationID = rs;

      //Handle asynchronous blocking
      int asyncRes = asynchronousCall(asyncOperationID);
      if (asyncRes != 0) return(-1);
   }
   else if (rc != 0)
   {
      //Handle the error
      if (rc == 8 && rs == 0)
        printf("Bad session token.\n");
```

```
      else if (rc == 24)
        printf("Syntax error in function parameter.\n");
      else if (rc == 100 && rs == 8)
        printf("ESM authentication error.\n");
      else if (rc == 100 && rs == 12)
        printf("Dir. mgr. authentication error.\n");
      else if (rc == 100 && rs == 16)
        printf("SMAPI server authentication error.\n");
      else if (rc == 196)
        printf("SMAPI Server error.\n");
      else if (rc == 400 && rs == 4)
        printf("That guest was not found.\n");
      else if (rc == 400 && rs == 4)
        printf("That guest is locked.\n");
      else if (rc == 400 && rs == 4)
        printf("That guest cannotbe deleted.\n");
      else if (rc == 400 && rs == 20)
        printf("That PROTODIR file does not exist.\n");
      else if (rc == 500 && rs == 4)
        printf("Dir. mgr. is not accepting updates.\n");
      else if (rc == 500 && rs == 8)
        printf("Dir. mgr. is not available.\n");
      else if (rc == 596)
        printf("Dir. mgr. error.\n");
      else printf("Delete error (unknown): rc=%d rs=%d.\n",rc,rs);

      return(-1);
   }

   //If we get here, the guest was deleted.
   printf("Guest deleted.\n");
}

//****** asynchronousCall() ******//
int asynchronousCall(int asyncOperationID)
{
   printf("Waiting for return of asynchronous call.\n");
   int timer = 0;

   while(TRUE)
   {
      //Set arguments
      QUERYASYNCOP_args* queryAsyncOpArgs =
         (QUERYASYNCOP_args*)malloc(sizeof(QUERYASYNCOP_args));
      queryAsyncOpArgs->OperationID = asyncOperationID;

      //Copy session token
      strncpy(queryAsyncOpArgs->SessionToken,sessionToken,8);

      //Make the call
      QUERYASYNCOP_res* queryAsyncOperationRes;
      queryAsyncOperationRes =
         query_asynchronous_operation_1(queryAsyncOpArgs, client);

      int rc;
      int rs;

      //Free unneeded memory
      free(queryAsyncOpArgs);
```

```
//Find rc/rs and save session token
rc = queryAsyncOperationRes->rc;
if(rc == 0)
{
   //There was not an error, we use resok
   strncpy(sessionToken,
      queryAsyncOperationRes->QUERYASYNCOP_res_u.
         resok.SessionToken,8);
   rs = queryAsyncOperationRes->QUERYASYNCOP_res_u.resok.rs;
}
else
{
   //There was an error, use resfail.
   strncpy(sessionToken,
      queryAsyncOperationRes->QUERYASYNCOP_res_u.
         resfail.SessionToken,8);
   rs = queryAsyncOperationRes->QUERYASYNCOP_res_u.resfail.rs;
}

//Check results and return only if operation is finished.
//Tell user in the case of an error, and loop if we
//are still waiting.
if (rc != 0)
{
   if (rc == 8)
     printf("Async error: Session Token invalid.\n");
   else if (rc == 100 && rs == 8)
     printf("Async error: ESM Authentication Error.\n");
   else if (rc == 100 && rs == 12)
     printf("Async error: Dir. mrg. authentication error.\n");
   else if (rc == 100 && rs == 16)
     printf("Async error: SMAPI authentication error.\n");
   else if (rc == 196)
     printf("Async error: Internal server error rs=%d.\n",rs);
   else if (rc == 500 && rs == 8)
     printf("Async error: Dir. mgr. not available.\n");
   else if (rc == 596)
     printf("Async error: Dir. mgr. error code=%d.\n");
   else printf("Asynchronous Call Error: RS=%d RC= %d.\n");

   return(-1);
}

//If we've made it here, there is no error presumably.
if (rc == 0 && rs == 100)
{
   //Call is complete, we can return.
   printf("Asycnronous call returned OK.\n");
   return(0);
}
else if (rc == 0 && rs == 108)
{
   //Call has failed.
   printf("Async. call failed.  No ");
   printf("futher information is available.\n");
   return(-1);
}
else if (rc == 0 && rs == 104)
{
   //5s * 48 = 4m
```

```
            if (timer == 48)
            {
               //Only allow 4 minutes for async call to return.
               printf("Async error: Timed out waiting for async. ");
               printf("call to return.\n");

               return(-1);
            }

            timer++;

            //Sleep for 5 seconds and try again
            sleep(5000);
         }
         else
         {
            printf("Asynchronous Call Error: RS=%d RC= %d.\n");
            return(-1);
         }

         return(0);
      }
}

//****** printUsage() ******//
void printUsage()
{
   printf("Usage: GuestCreator VM_IP_ADDRESS PORT LOGIN_NAME ");
   printf("PASSWORD\n");
}

//****** getInput() ******//
void getInput()
{
   //Get create/delete
   printf("To create a guest, type create\n");
   printf("To delete a guest, type delete\n");
   printf("press ctrl+C at any time to quit.\n\n");

   char* input;
   int size = 0;

   int loop = TRUE;
   while(loop)
   {
      printf("Enter create or delete: ");
      size = getline(&input, &size, stdin);

      if(strncmp(input, "create", 6) == 0)
      {
         isCreateJob = 1;
         loop = FALSE;
      }
      else if(strncmp(input, "delete",6) == 0)
      {
         isCreateJob = 0;
         loop = FALSE;
      }
   }
```

```
   //Free unneeded memory
   free(input);

   //Get rest of needed info from user.
   getGuestName();
   if(isCreateJob)
   {
      getProtodir();
      getPassword();
   }
}

//****** getGuestName() ******//
void getGuestName()
{
   char* input;
   int size = 0;
   int loop = TRUE;

   while(loop)
   {
      printf("Enter the guest name to create/delete: ");
      size = getline(&input, &size, stdin);

      if(strlen(input) <= 9 && strlen(input) >= 2)
      {
         //Copy the result to the global variable
         //make sure to strip the newline character.
         stripNewline(input);
         strncpy(guestName, input,9);
         loop = FALSE;
      }
   }

   //Free unneeded memory
   free(input);
}

//****** getProtodir() ******//
void getProtodir()
{
   char* input;
   int size = 0;
   int loop = TRUE;

   while(loop)
   {
      printf("Enter the protodir file to use: ");
      size = getline(&input, &size, stdin);

      if(strlen(input) <= 9 && strlen(input) >= 2)
      {
         //Copy the result to the global variable
         //make sure to strip the newline character.
         stripNewline(input);
         strncpy(protodirFile, input,9);
         loop = FALSE;
      }
   }
   free(input);
```

```
}

//****** getPassword() ******//
void getPassword()
{
   char* input;
   int size = 0;
   int loop = TRUE;

   while(loop)
   {
      printf("Enter the password for the new guest: ");
      size = getline(&input, &size, stdin);

      if(strlen(input) <= 9 && strlen(input) >= 2)
      {
         //Copy the result to the global variable
         //make sure to strip the newline character.
         stripNewline(input);
         strncpy(guestPassword, input,9);
         loop = FALSE;
      }
   }
   free(input);
}

//****** stripNewline() ******//
void stripNewline(char* myStr)
{
   char* charLocation;

   //Remove carriage return
   charLocation = strrchr(myStr, '\r');
   if (charLocation != NULL) *charLocation = '\0';

   //Remove line feed
   charLocation = strrchr(myStr, '\n');
   if (charLocation != NULL) *charLocation = '\0';
}
```

The C code is similar to the Java code in Example 5 on page 16. However, we note the following differences:

- Because resok and resfail are C unions, we cannot check them for null to detect an error. Instead, we rely on the value of the return code. In the event of an error, rc is non-zero. Also note that exceptions are not supported in C.
- The C naming scheme does not always follow the conventions used in the Java code. For example, the LOGIN_1 Java method in named login_1 in the C code.
- The C code does not use an object-oriented paradigm. Instead, the CLIENT object is passed as the second argument to all Systems Management API procedure calls.
- A for-loop is used to traverse and copy the session token (implemented as a eight-byte character array). An ISO C restriction does not allow direct value assignment to an array.

Although written in C, the code is easily converted to C++. Most data structures and procedures need not change.

Additional material

This Redpaper refers to additional material that can be downloaded from the Internet.

Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the IBM Redbooks™ Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/REDP3882

Alternatively, you can go to the IBM Redbooks Web Site at:

http://www.ibm.com/redbooks

Select Additional materials and open the directory that corresponds to the Redpaper form number REDP3882.

Using the Web material

The additional material is a zipped file (redp3882.zip) containing the code samples discussed in this Redpaper. The zip file contains:

File name	*Description*
CreatorException.java	The Java exception class discussed in “Creating and deleting z/VM guests using a Java application” on page 16.
GuestCreator.java	The Java application discussed in “Creating and deleting z/VM guests using a Java application” on page 16.
GuestCreator.c	The C application discussed in “Systems Management API information for C programmers” on page 25.

Related publications

These publications are suitable for a more detailed discussion of the topics covered in this Redpaper:

- *CP Command and Utility Reference*, SC24-6008
- *TCP/IP User’s Guide*, SC24-6020
- *TCPIP Planning and Customization*, SC24-6019
- *Systems Management Application Programming*, SC24-6063
- *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6024
- *Directory Maintenance Facility Command Reference*, SC24-6025

Referenced Web sites

These Web sites are relevant for further information:

- The VM on-line documentation Web site:

  http://www.vm.ibm.com/library

- RFC1831

  ftp://ftp.rfc-editor.org/in-notes/rfc1831.txt

- ONC/RPC for Windows NT/95

http://www.plt.rwth-aachen.de/ks/english/oncrpc.html

- Remote Tea ONC/RPC for Java™

  http://www.plt.rwth-aachen.de/ks/english/remotetea.html

- UNIX man page for portmap

  http://www.rt.com/man/portmap.8.html

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Jason Herne is a co-op in the Systems Group at IBM Poughkeepsie. He has a bachelors degree in Computer Science from Clarkson University and plans on extending his studies to obtain a Masters degree. His interests include Linux and open source software, operating system design, and virtualization.

Thanks to the following people for their contributions to this project:

Greg Geiselhart
International Technical Support Organization, Poughkeepsie Center

Robert Brenneman, Bernice Casey, Eli M. Dow, Kyle Smith
IBM Poughkeepsie

Christine Casey, Mark Ritter, Garrett Schanck
IBM Endicott

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those 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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on July 15, 2004.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DirMaint™	ibm.com®	Redbooks™
@server®	IBM®	z/VM®
@server®	Redbooks (logo) ™	

The following terms are trademarks of other companies:

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

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

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

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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