Level: Advanced
Works with: Notes/Domino
Updated: 01-Aug-2003

by Robert Perron, Steve Nikopoulos, and Kevin Smith

This article is the second in a series of two. In part one of this series, you learned the basics of using the Domino Objects from a Java application locally and remotely. This month you will learn about SSL encryption, servlets, connection pooling, single sign-on, session timeouts, and recycling. The article also includes a section on troubleshooting. This article assumes that you are familiar with the Domino Java API and have read the first article.

SSL encryption

The previous article in this series discussed running a Java application locally or remotely. Remote calls require HTTP and DIIOP access. You can encrypt transmissions over the DIIOP port using SSL (Secure Sockets Layer). See the previous article for instructions on how to set up DIIOP. The client code signals the desire to encrypt by specifying a new second parameter in the createSession call. This parameter is a String array whose first element has -ORBEnableSSLSecurity as its value, for example:

```
String args[] = new String[1];
args[0] = "-ORBEnableSSLSecurity";
Session s = NotesFactory.createSession("myhost.east.acme.com:63148", args,
"Jane Smith/East/Acme", "topS3cr3t");
```

You still use a non-SSL port (63148 in the above example) to get the IOR. The actual service requests take place over the DIIOP SSL port, which is 63149 by default.

Before running the code, you must set up the server and client with a common trusted root certificate from a certificate authority. This process is best covered as a series of steps.

Step 1
Create a key ring. Open the Server Certificate Admin (certsrv.nsf) database on a Domino server and use its forms to create and populate a key ring. See *Administering the Domino System, Volume 2* or the Domino Administrator Help for detailed information. For testing purposes, you can use the CertAdminCreateKeyringWithSelfCert form to create a key ring with a self-certified certificate.

Step 2
Move the keyring to the server. The keyring consists of a keyring file (KYR file) and stash file (STH file). These files are generated on the computer from which you're accessing the Server Certificate Admin database. Move or copy the two keyring files to the computer containing the Domino server. Place them in the server's data directory. For example, if you create a keyring with a self-certified certificate using default names and copy the files to a computer with a server whose data files are installed at C:\Lotus\Domino\Data, the server files would

be:

```
C:\Lotus\Domino\Data\selfcert.kyr
C:\Lotus\Domino\Data\selfcert.sth.
```

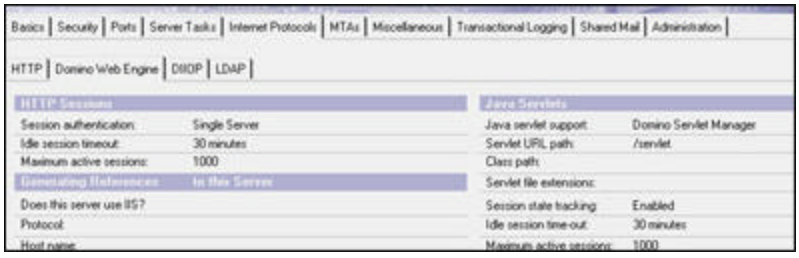
Step 3
Copy TrustedCerts.class to the client and put it in the classpath. Once the keyring files are on the server, starting or restarting the DIIOP task generates a file named TrustedCerts.class in the Domino data directory. Distribute this file to any computer from which you are going to access the server using CORBA with SSL, and put the directory containing the file in the classpath. For example, if you copy the file to C:\Lotus\TrustedCerts.class on a client, set the classpath as follows:

```
set classpath := %classpath%;c:\lotus
```

Step 4
Enable the server for SSL. In the Server document in the server's Domino Directory, go to the Ports tab, then the Internet Ports tab. Under SSL settings, specify the SSL key file name (for example, selfcert.kyr). Go to the DIIOP tab. Ensure that the SSL port number is correct—it defaults to 63149. Enable the SSL port. Set Name & password and Anonymous authentication as desired.

Servlets

The Domino server HTTP task supports servlets by loading a servlet engine and the JVM. In the Server document of the Domino Directory, go to the Internet Protocols tab, the Domino Web Engine tab, and the Java Servlets tab for setup. The Domino Designer Help document "Running servlets in Domino" provides detailed instructions for the general case. Below is a sample of the top part of the Server document:



By default, Domino looks for servlet executable code in its data directory under domino\servlet. For example, if you have an executable servlet in a file called MyServlet.class, you might store it on the server as:

```
c:\lotus\domino\data\domino\servlet\MyServlet.class
```

From a browser (by default), run the servlet by specifying /servlet and the name of the class file in the URL. For example:

```
http://myhost.east.acme.com/servlet/MyServlet
```

If you are accessing Domino Objects, you have two additional concerns. First, for remote access, the Notes.ini file for the server must specify an NCSO archive for the variable JavaUserClasses. This variable is the classpath for the JVM loaded by the HTTP task. A typical specification would be:

```
JavaUserClasses=c:\lotus\domino\data\domino\java\NCSO.jar
```

Second, code that accesses the Domino Objects locally must use NotesThread. Because other means are not possible in a servlet, call NotesThread.sinitThread before using any Domino Objects and NotesThread.stermThread after. The following code demonstrates a simple servlet accessing the Domino Objects locally:


```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class MyServlet extends HttpServlet
{
	public void doGet(HttpServletRequest request,
	HttpServletResponse response) throws IOException
	{
		try
	{
		NotesThread.sinitThread();
		Session s1 = NotesFactory.createSession();
		String name = s1.getUserName();
		response.setContentType("text/html");
		ServletOutputStream out = response.getOutputStream();
		out.println("<HTML><B>Information from servlet:</B><BR>");
		out.println("<BR>Name = " + name);
	}
	catch (Exception e)
	{
		e.printStackTrace();
	}
	finally
	{
		NotesThread.stermThread();
	}
	}
}
```

Local access of the Java classes only requires HTTP. The HTTP task must be running so that a browser can access the server to call the servlet. The DIIOP task is not used. For remote access, the DIIOP task must also be running. The coding is simpler because NotesThread is not used. More than likely, you will not be using the remote classes to access Domino from the same machine. The local classes are preferred. However, you may want to use the remote classes from another servlet manager (WebSphere, for example) running on a different machine. Here is a template:

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class MyServlet extends HttpServlet
{
	public void doGet(HttpServletRequest request,
	HttpServletResponse response) throws IOException
	{
	try
	{
		Session s1 = NotesFactory.createSession("myserver.east.acme.com");
		String name = s1.getUserName();
		s1.recycle();
		response.setContentType("text/html");
		ServletOutputStream out = response.getOutputStream();
```

```
            out.println("<HTML><B>Information from servlet:</B><BR>");
            out.println("<BR>Name = " + name);
        }
        catch (Exception e)
            {
                e.printStackTrace();
            }
        }
}
```

Connection pooling

Connection pooling allows multiple sessions to be created with the same ORB (Object Reference Broker), which reduces network resources because all sessions are sharing one TCP/IP connection. Connection pooling is particularly useful in servlets and server-to-server applications. Use one of the NotesFactory createORB methods to generate the ORB. Then use createSession methods that have an ORB parameter to create sessions.

The danger here is overloading the network connection. Operations on a session that take a long time block operations on other sessions that are sharing the ORB. Do not create more sessions than the connection can handle, and recycle a session when you are done with it.

The following code is a simple example that uses one ORB for up to 10 Domino sessions. The servlet creates a new ORB the first time the servlet runs and then after every tenth time. The example uses SSO as explained in the next section.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class ConnPool3 extends HttpServlet
{
    Session s = null;
    Static org.omg.CORBA.ORB orb = null;
    Static int count = 0;
    Static Object sem = new Object;

    public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException
    {
        try
            {
        synchronized (sem)
    {
        if ((orb == null) || ((count++ % 10) == 0))
            orb = NotesFactory.createORB();
                }
        s = NotesFactory.createSession("myhost1.east.acme.com:63148", orb, request);
        String name = s.getUserName();
        s.recycle();
        response.setContentType("text/html");
        ServletOutputStream out = response.getOutputStream();
        out.println("<HTML><B>Information from servlet:</B><BR>");
        out.println("<BR>Name = " + name);
        out.println("<BR>Count = " + count);
            }
```

```
                catch (Exception e)
                    {
                        e.printStackTrace();
                        }
                    }
                }
```

SSO (single sign-on)

Single sign-on allows access to multiple Domino and WebSphere servers through one sign-on on one of the servers. The servers you are accessing must be set up for SSO as explained in *Administering the Domino System, Volume 2* or the Domino Administrator help. You might also want to check out the Redbook Domino and WebSphere Together Second Edition.

The following signatures create an SSO session following prior authentication by a Domino or WebSphere server.

- Session s = createSession(host, String) bases access on a String token previously obtained from Session.getSessionToken, the LtpaToken cookie used by WebSphere or the HTTP cookie list in a servlet.
- Session s = createSession(host, Credentials) bases access on an org.omg.SecurityLevel2.Credentials object. This method works in a WebSphere environment where the Credentials object is created using loginHelper.
- Session s = createSession(host, null) bases access on the current Credentials object in the WebSphere environment. This method works from an EJB (Enterprise JavaBeans) application in WebSphere.
- Session s = createSession(host, HttpServletRequest) bases access on authentication by the Domino Web server.

The following code demonstrates the basics. It accesses a server using a name and password, gets an SSO token, then accesses another server in the same SSO domain using the token.

```
import lotus.domino.*;
public class sso1 implements Runnable
{
    public static void main(String argv[])
    {
        sso1 t = new sso1();
        Thread nt = new Thread((Runnable)t);
        nt.start();
    }
        public void run()
        {
        String token = null;
        try
            {
                String host1 = "myhost1.east.acme.com:63148";
                String name = "Jane Smith/East/Acme";
                String pw = "topS3cr3t";
                Session s = NotesFactory.createSession(host1, name, pw);
                token = s.getSessionToken();
                s.recycle();

            }
                catch(Exception e)
                    {
                    e.printStackTrace();
                    }
        try
            {
                String host2 = "myhost2.east.acme.com:63148";
```

```
                Session s = NotesFactory.createSession(host2, token);
                System.out.println(s.getUserName());

        }
                catch(Exception e)
                {
                        e.printStackTrace();
                        }
                }
        }
```

The next one is a servlet that gets the SSO token from the HTTP cookie list.

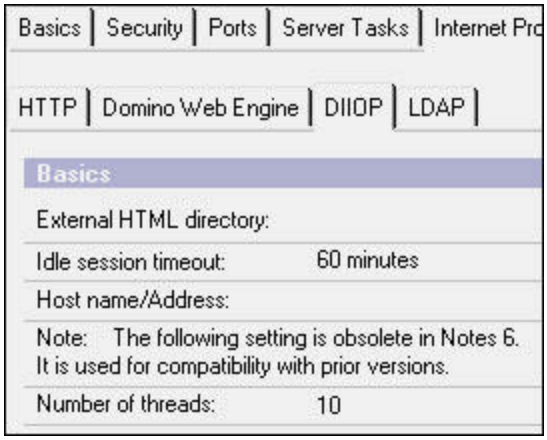
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class sso13 extends HttpServlet
{
        public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
        {
                Cookie[] cookies = null;
                String sessionToken = null;
                try
                        {
                        cookies = request.getCookies();
                        }
                                catch (Exception e)
                                {
                                        e.printStackTrace();
                                }
                                        if (cookies != null)
                                {
                                        for (int i = 0; i < cookies.length; i++)
                                {
                                        if (cookies[i].getName().equals("LtpaToken"))
                                {
                                        sessionToken = cookies[i].getValue();
                                }
                        }
                }
                        if (sessionToken != null)
                        {
                                try
                                {
                                NotesThread.sinitThread();
                                Session session = NotesFactory.createSession(null, sessionToken);
                                response.setContentType("text/plain");
                                ServletOutputStream out = response.getOutputStream();
                                out.println("UserName: " + session.getUserName());
                                }
                                        catch (NotesException e)
                                {
                                        e.printStackTrace();
```

```
                }
                finally
                    {
                        NotesThread.stermThread();
                    }
                }
            }
        }
```

Session timeouts

A server has a maximum idle time for remote sessions. If a session exceeds the idle session timeout, the server terminates the session and releases its resources. In the Server document, go to the Internet Protocols tab, the DIIOP tab, and the Idle session timeout field. The default timeout is 60 minutes.



In the client code, use Session.isValid() to determine if a remote session is available. If a remote session has timed out, isValid() returns false. The following example creates a remote session, then accesses the session again later. Conditional code ensures that the session is still available and, if not, creates a session.

```
import lotus.domino.*;
public class isvalidTest implements Runnable
{
    String host=null, user="", pwd="";
    Session s = null;
    public static void main(String argv[])
    {
        if(argv.length<1)
        {
            System.out.println("Need to supply Domino server name");
            return;
            }
                isvalidTest t = new isvalidTest(argv);
                Thread nt = new Thread((Runnable)t);
                nt.start();
                }
                    public isvalidTest(String argv[])
                {
                    host = argv[0];
                     if(argv.length >= 2) user = argv[1];
                     if(argv.length >= 3) pwd = argv[2];
                }
                     public void run()
```

```
                    {
                        try
                        {
                            s = NotesFactory.createSession(host, user, pwd);
                            String p = s.getPlatform();
                            System.out.println("Platform = " + p);
                        }
                            catch(Exception e)
{
    e.printStackTrace();
    }

    // Later the same day

    try
        {
            if (!s.isValid())
            s = NotesFactory.createSession(host, user, pwd);
            String n = s.getUserName();
            System.out.println("Username = " + n);
            }
                catch(Exception e)
            {
            e.printStackTrace();
            }
        }
}
```

The isValid() method has the effect of activating the session, which resets the timeout clock.

Recycling

Java has no knowledge of the heavyweight back-end Domino Objects, only the lightweight Java objects representing them. For local classes, the Domino Objects use memory in the process running the Java program and no garbage collection occurs without recycling until the program exits. Memory use can be problematic for:

- *A program that runs for a long time*
  Servlets, especially, fall in this category.
- *A program that iterates over Domino Objects*
  Loops that process documents fall in this category.

All Domino Objects have a recycle method. The recycle method destroys the current object and any children, and releases their memory. Recycle is strongly recommended where memory use can be problematic. For example, you might place the following statement (where session is a Session object) at the end of the doGet code in a servlet. This recycles all Domino Objects used after every invocation of the servlet.

```
session.recycle();
```

You might place the following statement at the end of a loop (in any application) that processes documents (where doc is a Document object). This recycles the Document object and any child objects (such as Item) on every iteration.

```
doc.recycle();
```

Recycling is not as critical for remote Domino Objects. There memory use is in the DIIOP process and a thread performs garbage collection. If the TCP/IP connection is lost, the entire session is automatically recycled.

Recycle also has the following signature:

recycle(java.util.Vector objects)

where the vector contains Domino Objects. This signature effectively batches the recycle requests and improves efficiency for remote calls.

When you recycle, observe the following guidelines:

- Recycle an object only if it is no longer needed.
- For local Domino Objects, recycle an object in the same thread in which it is created.
- For local Domino Objects in Session, call recycle() only after all other threads exit.
- Where NotesThread.sinitThread() and NotesThread.stermThread() are used, call recycle() before stermThread.
- If more than one object represents the same Domino element, recycling one recycles all.
- Results are undefined if you attempt to use a recycled object. There may be no error.

Troubleshooting

The following sections describe troubleshooting solutions for common issues.

Classpath and path

Make sure the code can access the necessary classes. The classpath and path must be set as follows assuming that Notes or Domino software is installed at c:\lotus\domino:

Environment	Classpath	Path
Local application or servlet	c:\lotus\domino\Notes.jar	c:\lotus\domino
Remote application or servlet	c:\lotus\domino\data\domino\java\NCSO.jar	Not applicable

PATH is important for the local classes because Notes.jar needs to use the Notes/Domino binaries and Notes.ini.

If you are accessing Domino remotely from a computer that does not have Domino software, the NCSO archive must be copied to that computer. The classpath should reflect the location of the archive on the computer.

For SSL, the classpath must specify the directory containing TrustedCerts.class.

Remote computer

To use the Domino Objects remotely, the remote computer must be running, the Domino server must be running, and the client computer must be using the correct host name or IP address.

In Windows, if you have access to the server computer, you can get information such as the host name, DNS servers, and IP address with the following DOS command:

> ipconfig/all

On the client computer, you must be able to access the server computer. Try the ping command:

> ping *hostname*

or

> ping *ipaddress*

If you cannot ping the remote computer, something is wrong in the network. It could be the physical network connection, the network setup, or the names being entered. Resolve these problems before proceeding.

If you can ping the remote computer, try the telnet command to see if the remote computer is listening on the correct ports. The default ports are 80 for HTTP and 63148 for DIIOP.

```
> telnet host port
```

If you can ping the computer but cannot telnet the port, one of the following has occurred:

- The Domino Server is not running.
- The server is running, but the HTTP or DIIOP task is not running.

If you are accessing the IOR through the Web server port, you should be able to reach the Domino Server from a browser with the following URL:

```
http://hostname
```

You should be able to see the IOR file with the following URL:

```
http://hostname:port/diiop_ior.txt
```

If you have access to the Domino Server, you can examine the DIIOP task with the following console command:

```
> tell diiop show config
```

Here is some typical output:

```
> tell diiop show config
Dump of Domino IIOP (DIIOP) Configuration Settings

Full Server Name:  CN=Buffalo/O=Zoo
Common Server Name:  Buffalo/Zoo
Refresh Interval:  3 minutes

Host Full Name:  Buffalo.myCompany.com
Host Short Name:  Buffalo
Host Address:  9.99.99.999
Public Host Name/Address:  9.99.99.999

TCP Port:  63148  Enabled
SSL Port:  63149  Enabled
Initial Net Timeout: 120 seconds
Session Timeout:  60 minutes
Client Session Timeout:  62 minutes

IOR File:  d:\dom60x\data\domino\html\diiop_ior.txt
SSL Key File:  d:\dom60x\data\mykeyfile.kyr
Java Key File:  d:\dom60x\data\domino\java\TrustedCerts.class

Allow Ambiguous Names:  False
Web Name Authentic:  False
User Lookup View:  ($Users)
Allow Database Browsing:  True
TCP Name/Password Allowed:  True
TCP Anonymous Allowed:  True
SSL Name/Password Allowed:  True
SSL Anonymous Allowed:  True
Multi-Server Session Authentication:  Enabled
Multi-Server Session Configuration:  mySSOConfig
```


Internet Sites: Disabled

Single Server Cookies: Disabled

Error messages
Here are some error messages and possible explanations.

HTTP JVM java.lang.ClassNotFoundException and HTTP JVM java.lang.NoClassDefFoundError
These messages appear on the server console. They indicate that the client is trying to use the remote classes, but that JavaUserClasses is not specified in Notes.ini on the server.

NotesException: Could not get IOR from Domino Server: java.net.ConnectException: Connection refused
This error messaging indicates one of the following:

- Server is not running.
- HTTP task is not running and code gets IOR over HTTP port.
- DIIOP task is not running and code gets IOR over DIIOP port.
- IP port is specified, but not correctly.

NotesException: Could not get IOR from Domino Server: java.net.ConnectException: Operation timed out
This error message indicates a network configuration error.

NotesException: Could not get IOR from Domino Server: java.net.UnknownHostException
This error message indicates one of the following:

- The computer name or IP address is not specified correctly.
- The computer is not running.

NotesException: Could not open Notes session: org.omg.CORBA.COMM_FAILURE: java.net.ConnectException: Connection refused
This error message indicates that the DIIOP task is not running on the server.

NotesException: Invalid user name/password
This error message indicates that the code is trying to access the Domino Directory with an unknown name or invalid password.

NotesException: Server access denied
This error message indicates that Anonymous access has been tried, but not allowed.

NotesException: User* username *is not a server
This error message indicates that a session is trying to run through the Domino Directory on a client.

NotesException: Wrong Password
This error message indicates that the code is supplying the wrong password when trying to get access through the Notes ID. If your code does not send a password, access is initially Anonymous. If and when authentication is required (for example, you attempt to open a database), the Lotus Notes box comes up asking for a password. The box persists until the user gives the correct password or clicks Cancel.

Reference to createSession is ambigous
This error message comes out during compilation. It indicates that null is used as a createSession parameter. You must use "(String)null" in most cases.

UnsatisfiedLinkError: NCreateSession
This error message indicates that NotesThread is not used for local session. Put in NotesThread.sinitThread and stermThread, or otherwise use NotesThread.

NotesFactory signatures
The NotesFactory createSession methods can create either local or remote session objects. Local session calls can be made using either a Notes ID or the Domino Directory. A remote session uses the Domino

Directory.

For the NotesFactory signatures that follow:

hostname = hostname | hostname:port | ipaddress | ipaddress:port

- A local session using Notes ID requires Notes.jar and NotesThread. The following method signature is used to create a local session using a Notes ID.
  createSession()
  createSession((String)null)
  createSession("")
  createSession((String)null, (String)null, password)
  createSession("", (String)null, password)
  createSessionWithFullAccess() // useful on server only
  createSessionWithFullAccess(password) // useful on server only

- A local session using Domino Directory (server only) also requires Notes.jar and NotesThread. The following method signature is used to create a local session using a Domino Directory.
  createSession((String)null, "", "") // Anonymous
  createSession("", "", "") // Anonymous
  createSession((String)null, username, password)
  createSession("", username, password)

- A remote session using Domino Directory requires NCSO.jar or equivalent. In addition, the server must be accessible through DIIOP port or through the HTTP port if a client uses it to get IOR. The following method signature is used to create a remote session using a Domino Directory.
  createSession(hostname) // Anonymous
  createSession(hostname, username, password)
  createSession(hostname, -ORBEnableSSLSecurity, username, password)
  createSessionWithIOR(ior) // Anonymous
  createSessionWithIOR(ior, username, password)
  createSessionWithIOR(ior, -ORBEnableSSLSecurity, username, password)
  getIOR(hostname)

Notes.ini variables
The following Notes.ini variables for the Domino server affect the DIIOP task.

Variable	Description
DIIOPConfigUpdateInterval	Specifies time in minutes that DIIOP should check to refresh its configuration data from the Domino Directory. Defaults to 3 minutes.
DIIOPDNSLookup	Determines whether or not DIIOP should do a DNS name lookup for every client that connects and uses DIIOP services: 1 - Enable DNS lookups 2 - (default) Disable DNS lookups This information is visible when using the server console command show tasks.
DIIOPIORHost	Specifies the host name or IP address through which the server is known over DIIOP. Defaults to a name based on the "Fully qualified Internet host name" field under the Basics tab in the Server document. The preferred method of setting this value is through the Host name/Address field under the DIIOP and Internet Protocols tabs in the Server document. In R5 this variable was called DIIOP_IOR_HOST which is still valid

	for backwards compatibility.
DIIOPIgnorePortLimits	(Linux only) Determines whether or not to ignore port limits so the default DIIOP ports 63148 and 63149 can be used. Some Linux systems set port limits that do not allow use of the usual defaults for DIIOP: 1 - Ignore limits; use ports 63148 and 63149 2 - Honor limits; use ports 60148 and 60149 In R5 this variable was called DIIOP_IGNORE_PORT_LIMITS which is still valid for backwards compatibility.
DIIOPLogLevel	Specifies the level of information that DIIOP reports to the server console and log: 0 - Show errors and warnings only 1 - Also show informational messages 2 - Also show session initialization and termination messages 3 - Also show session statistics 4 - Also show transaction messages This value can be set on the server console with the tell diiop log= *n* command.
DIIOPCookieCheckAddress	For server-based cookies used with applets downloaded by the Domino HTTP server, this variable determines whether or not the IP address of the client accessing the cookie must match the IP address of the client to which the cookie was issued: 0 - (default) Does not have to match 1 - Must match The client IP addresses will not match in most cases. The cookie is issued using HTTP which is typically routed through a proxy server. The user of the cookie is an applet which does not go through a proxy server.
DIIOPCookieTimeout	For server-based cookies used with applets downloaded by the Domino HTTP server, this variable specifies the number of minutes each cookie is valid. An expired cookie cannot be used to obtain a session with the DIIOP task. Defaults to 10. Minimum is 1.

Conclusion

This article series provided you with a comprehensive look at accessing Domino Objects with Java. In part two of this series, we covered SSL encryption, servlets, connection pooling, single sign-on, session timeouts, and recycling. We also included a section on troubleshooting. Together with the first article that covered local and remote calls and access control, you should now be confident when coding your Java applications.

ABOUT THE AUTHORS

Robert Perron is a documentation architect with Lotus in Westford, Massachusetts. He has developed documentation for Lotus Notes and Domino since the early 1990's with a primary concentration on programmability. He developed the documentation for the LotusScript and Java Notes classes and coauthored the book *60 Minute Guide to LotusScript 3 - Programming for Notes 4* . He has authored several *LDD Today* articles. Last year he authored "A Comprehensive Tour of Programming Enhancements in Notes/Domino 6" for *The View* .

Steve Nikopoulos is a Senior Software Engineer with the Domino Server Programmability team. His most recent work includes the remoted Domino Objects, Domino/Websphere integration, and single sign-on. In his spare time, he enjoys bike riding with his family.

Kevin Smith is an Advisory Software Engineer in the IBM Messaging & Collaboration Development organization in Westford, currently responsible for programmability features of the Domino server. He joined Lotus in 1990 as a technical support

engineer for Lotus 1-2-3/M, the famous PC spreadsheet ported to the famous IBM mainframe, one of the earliest IBM-Lotus engineering ventures.

