Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

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

by Robert Perron, Steve Nikopoulos, and Kevin Smith

Getting at the Domino Objects from Java can be a trying experience if you set off without some of the basic knowledge gathered in this article. Not only must the code be precise, but so must client and server environments. This article first covers the basics of local and remote access then addresses access control. The article concentrates on Java applications. A follow-up article next month discusses SSL encryption, servlets, connection pooling, single sign-on, firewalls, timeouts, and recycling, and includes a section on troubleshooting. This article assumes that you are familiar with the Domino Java API.

Overview

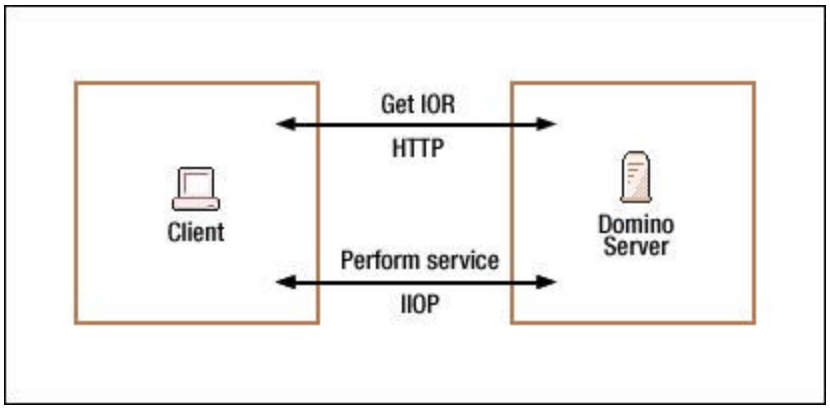
Java access to the Domino Objects is through the high-level package lotus.domino. The interfaces in this package are implemented in one of two other packages depending on the run-time environment:

- lotus.domino.local to service the call from Notes/Domino software on the same computer
- lotus.domino.cso to service the call from a Domino server accessed through a remote connection

For local access, the Java program runs on a computer with a Notes client or Domino server installed. The local classes are built with JNI (Java Native Interface) to access Notes/Domino binaries in the same process as the JVM (Java Virtual Machine).

For remote access the Java program requests the service from a Domino server using CORBA (Common Object Request Broker Architecture). The remote classes use CORBA to access the server over a TCP/IP network. Remote access is in two parts:

- The client obtains the server's initial object as an IOR (Interoperable Object Reference) using the HTTP protocol.
- The client obtains further objects over an IIOP connection.



The NotesFactory class in lotus.domino provides createSession and other methods for initiating access to the Domino Objects in Java applications and servlets. The particular signature determines whether the access is local or remote.

To compile a Java program that uses the lotus.domino package, the classpath must include Notes.jar (local) or NCSO.jar (remote). For example:

```
set classpath=%classpath%;c:\lotus\domino\Notes.jar
```

or

```
set classpath=%classpath%;c:\lotus\domino\data\domino\java\NCSO.jar
```

Notes.jar can be found in the program directory of any Notes/Domino installation. NCSO.jar can be found in the domino\java directory under the data directory in Domino Designer or the Domino server.

Local calls

A createSession signature with no parameters, a null first parameter, or an empty string for the first parameter makes local calls. The following are equivalent:

```
Session s = NotesFactory.createSession()
Session s = NotesFactory.createSession((String)null)
Session s = NotesFactory.createSession("")
```

Cast null to a String to avoid overload conflicts.

To execute local calls from applications and servlets, the path must include the Notes/Domino program directory, and the classpath must include Notes.jar, which is in the Notes/Domino program directory. For example:

```
set path := %path%;c:\lotus\domino
set classpath := %classpath%;c:\lotus\domino\Notes.jar
```

Notes.jar contains the lotus.domino and lotus.domino.local packages.

The local calls require that the NotesThread class manage threads. NotesThread extends java.lang.Thread to include special initialization and termination code for Domino. You have three options:

- Execute threads through inheritance
- Execute threads through the Runnable interface
- Execute threads through the static methods

Execute threads through inheritance

To execute threads through inheritance, extend NotesThread instead of Thread and include a runNotes method instead of run. Start a NotesThread thread the same way as any other thread with the start method. This technique is easy to use and less error prone than the static methods (discussed later).

```
import lotus.domino.*;
public class myClass extends NotesThread
{
    public static void main(String argv[])
        {
            myClass t = new myClass();
            t.start();
                }
                    public void runNotes() // entry point for Notes thread
                {
                    try
                        {
                            Session s = NotesFactory.createSession();
                            // Operational code goes here
                        }
                        catch (Exception e)
                    {
                        e.printStackTrace();
                }
            }
        }
```

Execute threads through the Runnable interface

To execute threads through the Runnable interface, implement Runnable and include a run method as you would for any class using threads. This technique can be used when you cannot extend NotesThread because you're extending another class.

```
import lotus.domino.*;
public class myClass implements Runnable
{
    public static void main(String argv[])
        {
            myClass t = new myClass();
            NotesThread nt = new NotesThread((Runnable)t);
            nt.start();
            }
                public void run() // entry point for thread
            {
                try
                    {
                        Session s = NotesFactory.createSession();
                        // Operational code goes here
                    }
                    catch (Exception e)
                {
                    e.printStackTrace();
            }
        }
    }
```

Execute threads through the static methods

To execute threads through the static methods, call sinitThread() to initialize a thread and stermThread() to terminate the thread. Call stermThread() exactly one time for each call to sinitThread(); putting stermThread in a

"finally" block is recommended. The static methods can be used when inheritance is not possible or when event-based threads need fine control.

```
import lotus.domino.*;
public class myClass
{
    public static void main(String argv[])
    {
        try
            {
                NotesThread.sinitThread(); // start thread
                Session s = NotesFactory.createSession();
                // Operational code goes here
            }
                catch(Exception e)
            {
                e.printStackTrace();
            }
                finally
            {
                NotesThread.stermThread(); // must terminate every thread
            }
        }
    }
```

Each thread of an application making local calls must initialize a NotesThread object. This includes AWT threads that access the Domino Objects. Listener threads must use the static methods because they cannot inherit from NotesThread.

An application that makes both local and remote calls can determine dynamically when to use the static methods sinitThread and stermThread. Remote calls can be made while a local thread is running; however, do not use an object obtained through one session in a call to the other session.

You should avoid multithreading unless you have good reason to use it, such as proceeding while file input/output and Web requests are processing. Observe the following guidelines:

- Within a session, Domino Objects are shared, synchronized, and recycled across threads. Using a different session on each thread loses these capabilities; you must explicitly manage synchronization and recycle on a per-thread basis.
- Do not use DbDirectory across threads.
- Accessing an existing document on multiple threads is permissible, but accessing it on just one thread simplifies memory management. Restricting access to one thread allows you to recycle without checking the other threads. Creating documents across threads is always safe, and these objects can be recycled without reference to the other threads.
- Profile documents are cached on a per-thread basis. In the event of an update contention, the last thread updating takes precedence.
- Take care not to delete a document needed for navigation by a view or by a collection on another thread.
- When child objects are used on threads other than the parent, keep the parent thread alive until all child threads terminate. This is particularly important when using Domino Objects in AWT event handlers.

Remote calls

A createSession signature whose first parameter is a non-empty string makes remote calls. The first parameter identifies the computer containing the Domino server. For example:

```
Session s = NotesFactory.createSession("myhost.east.acme.com")
```

or


```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148")
```

The second example specifies the port number which removes the need for the Domino Web server to be running on myhost.east.acme.com. See the "Getting the IOR" section later in this article for more detail.

To execute remote calls from an application or servlet, the client computer must contain NCSO.jar in the classpath. NCSO.jar contains the lotus.domino package along with the lotus.domino.cso package, the lotus.domino.corba package, and the ORB classes, which contain implementation code for the remote classes. For installed Domino Designer and Domino server software, NCSO.jar is in the domino\java subdirectory under the Domino data directory. For computers without installed Domino software, you must copy the archive from a computer that contains the software.

The classpath must include the archive, for example:

```
set classpath := %classpath%;c:\lotus\domino\data\domino\java\NCSO.jar
```

Coding

Coding is straight-forward. NotesThread is not used for remote calls. You simply make the createSession call with the host name and (optionally) the port number.

Without threads, the template is as follows:

```
import lotus.domino.*;
public class myClass
{
    public static void main(String argv[])
    {
        try
            {
                String host = "myhost.east.acme.com:63148";
                Session s = NotesFactory.createSession(host);
                // Operational code goes here
            }
                catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

The following template uses threads:

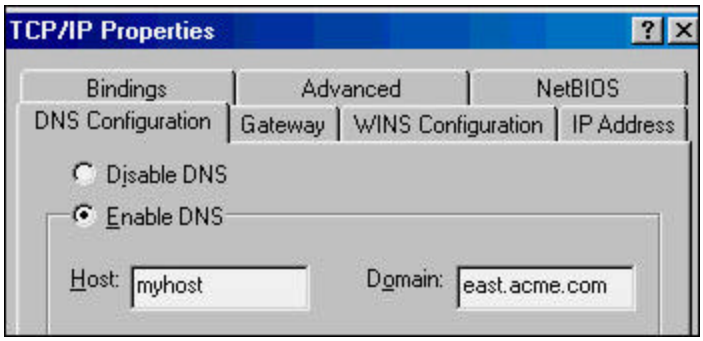
```
import lotus.domino.*;
public class myClass implements Runnable
{
    public static void main(String argv[])
    {
        myClass t = new myClass();
        Thread nt = new Thread((Runnable)t);
        nt.start();
    }
        public void run()
    {
        try
            {
                String host = "myhost.east.acme.com:63148";
                Session s = NotesFactory.createSession(host);
```



```
                // Operational code goes here
            }
                catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

Administration

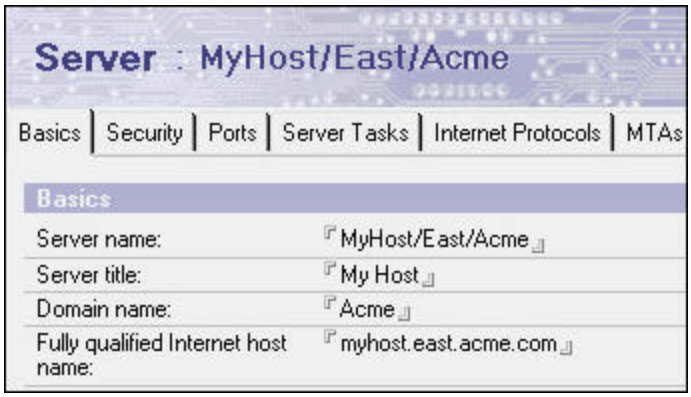
The extra part of remote access is setting up the Domino server and making the TCP/IP connection. The computer containing the server must be accessible over TCP/IP. In the network settings for the computer, check the DNS configuration under the TCP/IP properties for the host and domain names. You must be able to ping the server computer from the client computers using the Internet name. For example, if a Domino server has myhost as host and east.acme.com as the domain:



You should get a valid response from the client computer with:

```
> ping myhost.east.acme.com
```

The host and domain must appear in the Server document in the server's Domino Directory (names.nsf) in the "Fully qualified Internet host name" field under the Basics tab. Typically, the host name is set up when the server is installed, for example:



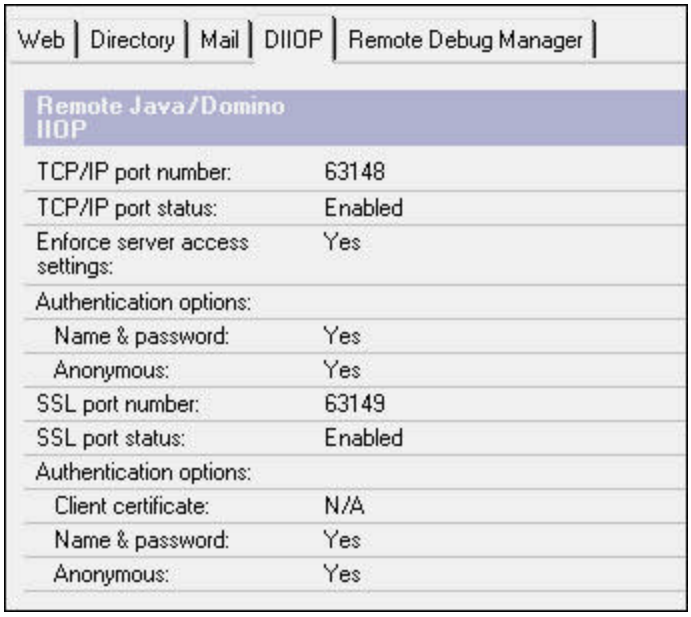
As shown in the preceding coding examples, you use the Internet name as the host (first) parameter to createSession. You can also use the IP address of the server computer. For example, if the IP address for myhost.east.acme.com is 9.95.73.30, either of the following calls is valid:

```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148")
```

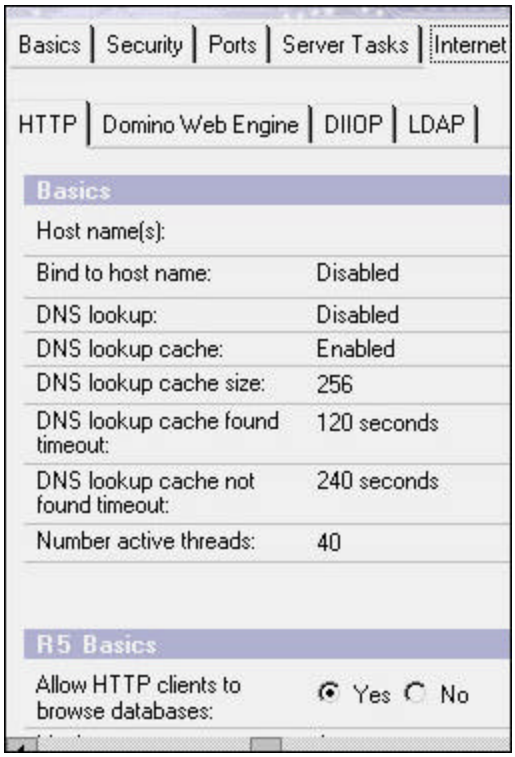
or

```
Session s = NotesFactory.createSession("9.95.73.30:63148")
```

The DIIOP (Domino IIOP) task on the server must be running. The HTTP task may also be needed depending on how you get the IOR. Check the Server document in the Domino Directory. Go to the Ports tab, then to the Internet Ports tab. Look under the Web tab (for HTTP) and find the DIIOP tab. These sections have fields for specifying the port number and enabling/disabling the port. Typically, the port numbers are 80 for HTTP and 63148 for DIIOP. The following screen shows the DIIOP tab:



To access a database without knowing its file name (that is, to use DbDirectory.getFirstDatabase), you must allow browsing over the network. Go to the Internet Protocols tab, HTTP tab, and R5 Basics tab. Then set the "Allow HTTP clients to browse databases" field to Yes.



To start the HTTP and DIIOP tasks with the server, make sure they are in the list of tasks for the ServerTasks variable in the Notes.ini file, which should be the case if the Server document is properly configured. The Notes.ini file should contain a line similar to the following:

```
ServerTasks=Update,Replica,Router,AMgr,AdminP,CalConn,Sched,DIIOP,HTTP,LDAP
```

From a running server, you can load the tasks at the console by entering the following commands:

```
> load http
> load diiop
```

You can stop the tasks using tell commands at the console:

```
> tell http quit
> tell diiop quit
```

You can refresh the DIIOP task:

```
> tell diiop refresh
```

You can restart the HTTP task:

```
> tell http restart
```

Getting the IOR

On Domino servers, the IOR is a file named diiop_ior.txt in the domino\html subdirectory under the Domino data directory. The IOR is a string encoding of an object that contains identifying information for CORBA access to the server. A client decodes the string IOR and uses it to establish the remote session.

By default, a remote client requests the server IOR through the Web server port (which normally services HTTP requests), then makes the session request through the DIIOP port. You can perform the requests separately.

For example:

```
String ior = NotesFactory.getIOR("myhost.east.acme.com"); // Get IOR using Web server port
Session s = NotesFactory.createSessionWithIOR(ior); // Create session using DIIOP port
```

is equivalent to:

```
Session s = NotesFactory.createSession("myhost.east.acme.com");
```

In the NotesFactory calls, you can specify the host port for getting the IOR by appending a colon and the port number to the host name or IP address. You can use this mechanism to service the HTTP request for the IOR through the DIIOP port if, for example, the Web server is not running, for instance:

```
String ior = NotesFactory.getIOR("myhost.east.acme.com:63148"); // Get IOR using DIIOP port
Session s = NotesFactory.createSessionWithIOR(ior); // Create session using DIIOP port
```

However, the two-step coding sequence is not necessary. You can simply say:

```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148");
```

You cannot use the DIIOP port to get text files other than diiop_ior.txt.

If you get the IOR through the Web server port, Anonymous access must be allowed. In the Server document in the Domino Directory, go to the Ports tab, then the Internet Ports tab, then the Web tab. Ensure that the Anonymous field under Authentication options is set to Yes.

Web (HTTP/HTTPS)	
TCP/IP port number:	80
TCP/IP port status:	Enabled
Enforce server access settings:	No
Authentication options:	
Name & password:	Yes
Anonymous:	Yes
SSL port number:	443
SSL port status:	Enabled
Authentication options:	
Client certificate:	No
Name & password:	Yes
Anonymous:	Yes

The ability to specify the DIIOP port to get the IOR is new with Notes/Domino 6. You can now use remote calls without having to allow Anonymous access on the Web server or without running the Web server at all.

You can also get the IOR by other means and use createSessionWithIOR. For example, you can copy the file diiop_ior.txt from the server computer to the client computer. The following code works if the client computer contains a valid diiop_ior.txt file for the server you want to use:

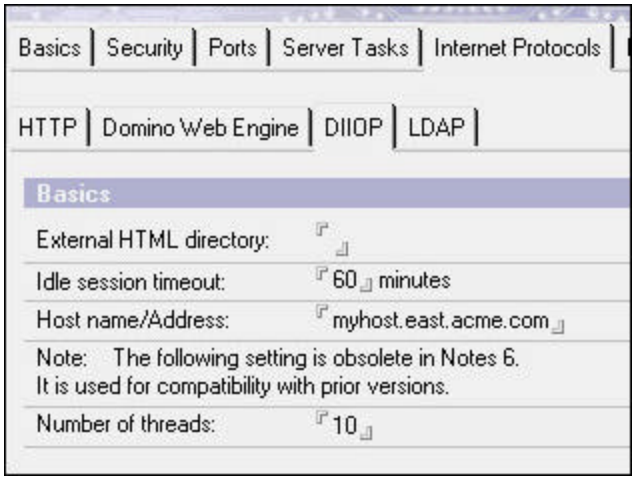
```
import lotus.domino.*;
import java.io.*;
public class platformior
{
    public static void main(String argv[])
```

```
{
    try
        {
            FileInputStream fin = new FileInputStream(
            "c:\\Lotus\\NotesR6\\diiop_ior.txt");
            InputStreamReader fisr = new InputStreamReader(fin);
            BufferedReader br = new BufferedReader(fisr);
            String ior = br.readLine();
            fin.close();
            Session s = NotesFactory.createSessionWithIOR(ior);
            //Operational code goes here
        }
            catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Be aware that the IOR settings can become stale. Any of the following changes on the server obsoletes a diiop_ior.txt file on a client:

- Changing a DIIOP port number
- Enabling or disabling a DIIOP port
- Changing the TCP/IP address

You can eliminate the last bullet by specifying the server host name rather than the server's TCP/IP address. In the Server document, go to the Internet Protocols tab, then the DIIOP tab. Specify the Internet host name for the server in the Host name/Address field.



This forces diiop_ior.txt to use the host name of the server rather than the IP address. You can also force use of the host name with the Notes.ini variable DIIOPIORHost.

Access control

The level of access a user is granted to a Domino server or to Notes/Domino software on a client depends on how you code createSession and how the server or client is set up. Access control is through either:

- *A name and the associated Internet password in a Domino Directory*
  This works for local or remote access. For local access, the computer running the code must be configured as a Domino server.
- *The current Notes ID (as specified by the KeyFilename variable in Notes.ini)*
  This works only for local access. The computer running the code can contain either a Notes client or a

Domino server.

Access through a Domino Directory
For access through a Domino Directory, the code determines whether the user accesses the server as Anonymous or as a user listed in the Domino Directory. A NotesFactory call that specifies only the host, or specifies empty strings for a user name and password, accesses the server as Anonymous, for example:

```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148");
```

or

```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148", "", "");
```

To access the server as a Domino user, specify the user name and Internet password as parameters two and three. The name and password must match a Person document in the Domino Directory for the server, for example:

```
Session s = NotesFactory.createSession("myhost.east.acme.com:63148", "Jane Smith/East/Acme",
"topS3cr3t");
```

For local access to a Domino server installed on the computer running the code, specify the host as an empty string, for example:

```
Session s = NotesFactory.createSession("", "Jane Smith/East/Acme", "topS3cr3t")
```

For Anonymous access, specify empty strings for the name and password:

```
Session s = NotesFactory.createSession("", "", "")
```

For local access, the server does not have to be running.

Settings in the Server document in the Domino Directory regulate Anonymous and named access. In the Server document, go to the Ports tab, the Internet Ports tab, and the DIIOP tab. The following table specifies the settings:

Client code	Authentication options
createSession(host, "", "")	Anonymous must be Yes to access server
createSession(host, name, password)	Name & password must be Yes to access server

For Name & password authentication, the name must be the user name in a Person document in the Domino Directory for the server, and the password must be the Internet password in the same Person document. You can further restrict access by setting in the Server document in the Domino Directory, under the Ports and Internet Ports tab, the Enforce server access settings field to Yes for the DIIOP port. Then specify the desired access in the Server Access tab under the Security tab.

The Server document contains the following fields that can be used to control security for Internet sessions created by the remote or local Java classes. An Internet session has access equivalent to a Web user and occurs when createSession takes a user name and password or an SSO token. The following table lists the fields on the Security tabs.

Field	Comment
Security settings	
Access server	You must be in this list to access the server via a remote or local Internet session.
Not access server	If you are in this list, you cannot get a remote or local

	Internet session.
Create databases & templates	Permission to create new databases and templates via back-end methods.
Create new replicas	Permission to create replica databases via back-end methods.
Create master templates	Permission to create master templates via back-end methods.
Programmability Restrictions	
Run unrestricted methods and operations	Users of the back-end classes that appear in this list can run privileged Java methods. This setting does not apply to native Java operations, such as writing to the file system, which when performed in a remote Java program, execute locally. (In other words, this setting only affects access on the Domino server and does not prevent a client that is using the remote classes from accessing files on its own file system.)
Sign agents to run on behalf of someone else	Agents with run as Web user run with the effective user name of the Internet session login user.
Sign agents to run on behalf of the invoker of the agent	Agents with run as Web user run with the effective user name of the Internet session login user.
(TCP/IP) Anonymous	DIIOP authentication does not require name and password.
(SSL) Name and Password	DIIOP SSL authentication requires name and password.
(SSL) Anonymous	DIIOP SSL authentication does not require name and password.
Internet Access	
Internet Authentication	Applies to the Internet session login user name. See field help or documentation for additional explanation.

The following table lists fields on the DIIOP tab:

Field	Comment
(TCP/IP) Name and Password	DIIOP authentication requires name and password.
(TCP/IP) Anonymous	DIIOP authentication does not require name and password.
(TCP/IP) Enforce server access settings	DIIOP uses Server Access settings under Security tab.
(SSL) Name and Password	DIIOP SSL authentication requires name and password.
(SSL) Anonymous	DIIOP SSL authentication does not require name and password.

The following Notes.ini settings affect the security of an Internet session obtained through the remote or local Java classes.

Notes.ini setting	Description
NoAmbiguousWebNames	NoAmbiguousWebNames=1 causes authentication failure if the DIIOP login user name has multiple matching names in the $Users view.
WebNameAuthentic	WebNameAuthentic=1 causes the DIIOP login name to be treated as the user's distinguished name.
NABWebLookupView	NABWebLookupView="xxx" causes the "xxx" view in the

	Domino Directory to be used to search for the DIIOP login name.

Note also that the database ACL "Maximum Internet Name and Password access" setting affects the security of an Internet session obtained through the remote or local Java classes because the DIIOP login user is restricted to this level of access.

A server refreshes its cache of security options approximately every half hour. The console command "tell diiop refresh" forces an immediate refresh.

Access through a Notes ID
For access through the current Notes ID, specify no parameters:

```
Session s = NotesFactory.createSession()
```

In this case, a dialog box requests the password for the Notes ID when authentication (for example, access to a database) is required. The Notes ID chosen is that specified by the KeyFileName variable in the first Notes.ini file on the search path. The user must enter the correct password or click Cancel to continue.

Instead of specifying no parameters, you can specify the second parameter as null (cast to a String) and the third as the password for the Notes ID:

```
Session s = NotesFactory.createSession((String)null, (String)null, "tops3cr3t")
```

On servers, the above methods restrict access according to Readers fields. For full access, use the following methods:

```
Session s = NotesFactory.createSessionWithFullAccess()
```

or

```
Session s = NotesFactory.createSessionWithFullAccess("tops3cr3t")
```

Conclusion

You now know the basics of using the Domino Objects from a Java application locally and remotely. In the next article, you will learn about SSL encryption, servlets, connection pooling, single sign-on, firewalls, timeouts, recycling, and troubleshooting.

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.

