Configuring the Sametime business card system, Part 2

Reach more data with custom blackboxes

by Mikkel Heisterberg

Mikkel Heisterberg
Senior Solution Architect
IntraVision ApS

Mikkel is a senior solution architect at IntraVision ApS (www.intravision.dk) in Denmark, a Lotus Domino consulting business specializing in calendaring and scheduling tools and correspondence management. Mikkel has been working with Lotus Notes/Domino since version 3 and often develops with Java, including plug-ins for Sametime 7.5 or Notes 8. He holds a bachelor's degree in biochemistry and a Master of Science in Internet Technologies from the University of Copenhagen, Denmark. You can contact Mikkel via e-mail at mh@intravision.dk or read his Web log at http://lekkimworld.com.

Getting business card data from the Domino Directory or a Lightweight Directory Access Protocol (LDAP) directory is great if the information you want is there — but what if it isn't? A developer could write a Notes agent that populates the Person documents in the Domino Directory with the card information needed, but agents work on a scheduled basis, require maintenance, and need monitoring. Or, what if there's no way to access the data from Notes? Instead of moving data, Java developers can easily customize and implement their own custom blackbox to extend the information available to the Sametime business card system.

Part 1 of this article gave a thorough description of the Sametime business card system, including its components and how to configure the system to return the Notes or LDAP data that you need. Sametime administrators can do that work without a developer's help. Part 2 shows Java developers how to write a custom Java blackbox for retrieving business card data. Customizing a blackbox might sound complex but really isn't — all you need is Java programming skills. I show you just what to do as you follow along with a sample case that involves getting card data from a sample database, which is available as a download at www.eVIEW.com.[1] You also get a JAR package with reusable code that makes it easy to create your own custom blackboxes. With this code, all you have to do is create a single method to fetch a lot of data. The JAR package even makes retrieving images in the correct format simple. I also share how to troubleshoot and debug blackboxes using the HTTP interface to the UserInfoServlet.

To learn how to configure the Sametime business card system, refer to Part 1 of this series. In this article, I assume that you know how to program

[1] You can download the demonstration phonebook and blackboxes databases (phonebook.nsf and theview_blackboxes.jar) at www.eVIEW.com: From the home page, click Publications → Knowledgebase → Browse by issue → May/June 2008 → Configuring the Sametime business card system, Part 2: Reach more data with custom blackboxes.

in Java, work in Eclipse, and use JAR files. If the file UserInfoConfig.xml and the phrase "implementing an interface" don't ring bells, I suggest you go back and read Part 1 again or brush up on Java, respectively.

The example cases

For this article, pretend you're working as a developer/administrator for a company called Ino Chemicals Inc. The business has a working Sametime community that it configured to run against a Domino Directory and hence the business card information by default comes from the Domino Directory on the Sametime server.

Ino Chemicals does not maintain user information, such as phone number, location, and so on, in the Domino Directory. For that purpose, there is a custom Notes application, a phonebook database. (You might want to download it now at www.eVIEW.com so you can follow along.) Let's say that you've decided that you would like to return from this database the data for all supported fields in the Sametime business card (e.g., location, title, and phone number) as well as a photo of the user.

While there are certainly options available to Ino Chemicals besides writing and deploying a custom blackbox implementation, I chose not to use them. My reasoning was as follows.

The simplest solution is to simply update the Domino Directory with the business card information either manually or using a scheduled Notes agent. This way, Ino Chemicals could continue using the default Sametime setup (the UserInfoNotesBB blackbox). While a viable solution, it would mean duplicating and maintaining business card data in multiple places, which the company doesn't want to do.

Another option is to modify the design of the phonebook application to contain a $Users view similar to the one from the Domino Directory, and to add the database to Directory Assistance on the Sametime server. While it's easy, I don't recommend this approach. While a full explanation is outside the scope of this article, the main reason is that adding a corporate phonebook to Directory Assistance would probably cause all users to be in two directories so the system would return the users twice in mail lookups, which isn't desirable.

A better option is to use the UserInfoNotesCustomBB blackbox implementation and reconfigure it to use our custom Notes phonebook database, view, and fields because it is simple and solves our problem. **Figure 1** shows the UserInfoConfig.xml file for implementing this case, and **Figure 2** shows the outcome, a business card displayed in Sametime.

However, let's alter the case a little so that Ino Chemicals is planning to replace the Notes phonebook

```
<?xml version ="1.0" encoding="UTF-8" ?>
<UserInformation>
   <Resources>
      <Storage type="NOTES">
         <CommonField  CommonFieldName="MailAddress"/>
         <Details>
            <Detail Id="MailAddress"  FieldName="InternetAddress" Type="text/plain"/>
         </Details>
      </Storage>
      <Storage type= "NOTES_CUSTOM_DB ">
         <StorageDetails DbName= "phonebook.nsf " View= "lookupByUsername " />
         <Details>
            <Detail Id= "Title " FieldName= "Title " Type= "text/plain "/>
            <Detail Id= "Telephone "  FieldName= "Telephone " Type=
```

Continues on next page

Figure 1 The UserInfoConfig.xml file using the UserInfoNotesCustomBB blackbox


```
"text/plain "/>
            <Detail Id= "Company " FieldName= "CompanyName " Type= "text/plain "  />
          <Detail Id="Name" FieldName="FullName" Type="text/plain"/>
          <Detail Id="Photo" FieldName="Photo" Type="image/jpeg"/>
          </Details>
       </Storage>
    </Resources>
    <ParamsSets>
       <Set SetId="0" params="MailAddress,Name,Title,Telephone,Photo,Company"/>
       <Set SetId="1" params="MailAddress,Name,Title,Telephone,Photo,Company"/>
    </ParamsSets>
    <BlackBoxConfiguration>
       <BlackBox type="NOTES"
          name="com.ibm.sametime.userinfo.userinfobb.UserInfoNotesBB"
          MaxInstances="4" />
       <BlackBox type= "NOTES_CUSTOM_DB "
          name= "com.ibm.sametime.userinfo.userinfobb.UserInfoNotesCustomBB "
          MaxInstances= "4 " />
    </BlackBoxConfiguration>
</UserInformation>
```

Figure 1 (continued)

application in a few months with an SAP ERP HCM system, a custom human resources system, or even files on disk. The best solution when the source of the data might change is to write your own blackbox implementation. It requires Java coding so it's a little more work, but it also means that users can obtain business card data from *any* system that is accessible using Java.

Next, I show you the components that you need for the custom solution.

The basic components for writing a custom blackbox

A blackbox for supplying Sametime with data is simply a Java class that implements an interface called UserInfoBlackBoxAPI. This interface is in the userinfo.jar, which IBM Lotus and the Sametime server supply. The interface has three methods: the two typical lifecycle methods init() and terminate(), and the business method processRequest(), which is

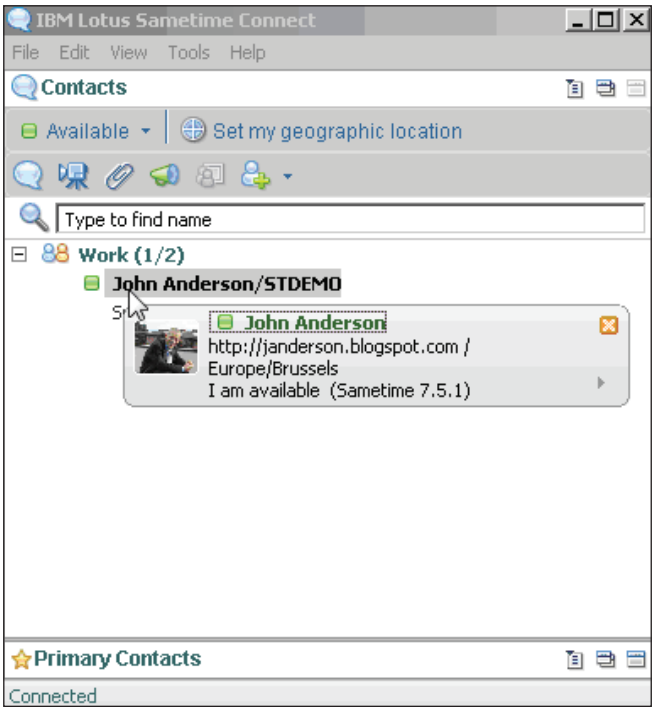

Figure 2 The output from the UserInfoNotesCustomBB blackbox

called upon in a blackbox to return data. **Figure 3** lists these methods. Because a blackbox only needs to fetch the information, the job of writing your own blackboxes is much simpler — the UserInfoServlet class handles the whole process of returning the result to the client, resource pooling, and so on.

> ***Tip!***
>
> Remember to add a default, no argument constructor for the custom blackbox class. If the constructor is not present, the UserInfoServlet class cannot instantiate your custom class and your blackbox won't work.

All the classes we need are in the com.ibm.sametime.userinfo.userInfobbapi package, which you can find in the userinfo.jar file on the Sametime server. This package contains the five classes shown in **Figure 4**, as well the UserInfoServlet class. Recall from the previous article that UserInfoServlet is the actual Sametime servlet (a servlet is a Java class that implements an interface to work with a Web server) that uses blackboxes, implementing the UserInfoBlackBoxAPI interface.

As you can see in **Figure 3**, the processRequest method takes an argument of type RequestContext and must return an object of type Response. The argument of type RequestContext is our handle to the request that UserInfoServlet is processing. We use it to make sure UserInfoServlet can associate the response with the request by specifying the request ID in the Reponse object sent back to the UserInfoServlet.

Method signature	Description
public abstract void init() throws UserInfoException;	Lifecycle method called when the blackbox is initialized. Called once per instance.
public abstract Response processRequest(RequestContext requestcontext) throws UserInfoException;	Called when the blackbox should return user data for a user.
public abstract void terminate();	Lifecycle method called when the blackbox is terminated. Called once per instance with one caveat (see the "Blackbox tips" sidebar later in the article).

Figure 3 Methods from the UserInfoBlackBoxAPI interface that you must apply to your blackbox implementation

Class name	Description
UserInfoBlackBoxAPI	The interface a blackbox must implement.
Response	The object type returned to the UserInfoServlet. This object holds a number of DetailInfo objects – one for each piece of information (e.g., a phone number you would like to return).
DetailItem	The wrapper for each piece of information returned in the Response object (see Response).
RequestContext	The provider of information about the request, such as the user ID of the user being processed and the ID of the request so that the UserInfoServlet can tell the difference between different invocations of the blackbox.
UserInfoException	The signal to the UserInfoServlet that something went wrong in the blackbox.

Figure 4 Classes of the com.ibm.sametime.userinfo.userInfobbapi package

Figure 5 shows the methods of the RequestContext interface.

The response is another matter. The returned Response object is a wrapper object that holds the DetailItem objects, which actually hold the data being returned. The simple Java class DetailItem has methods that set the different pieces of information that the system can return. When we create a DetailInfo object in our blackbox code, we then add it to the Response object using the setRetrievedDetail(String, DetailItem) method. I list the methods of the DetailItem class in **Figure 6**.

A simple custom blackbox for troubleshooting

The simplest blackbox we can write is a blackbox that uses only the API but doesn't return any interesting data — that is, it returns hard-coded data. While this sort of blackbox might seem trivial, it's actually very useful for troubleshooting scenarios where you need to troubleshoot the blackbox configuration in UserInfoConfig.xml rather than the actual blackbox code.

Method signature	Description
public abstract String getUid();	Gets the user ID of the person whose business card data an end user is seeking. Based on the configuration in the UserInfoConfig.xml file, the ID used may vary, though it's normally the e-mail address of the user.
public abstract String[] getReqDetails();	Returns a java.lang.String array of the details being requested. These correspond to the <Detail> elements in the UserInfoConfig.xml for the blackbox.
public abstract boolean isInRequestedDetails(String s);	Helps you to find out if the blackbox should return a particular piece of information.
public abstract long getRequestID();	Returns the ID of the request, which should be provided in the constructor of the Response object. The UserInfoServlet uses this ID to associate the response to a request.

Figure 5 Methods of the RequestContext interface

Method signature	Description
public void setId(String s)	Sets the ID of the object. I never use this method, because I rely on the setName(String) method instead.
public void setName(String s)	Sets the name of the detail.
public void setType(String s)	Sets the MIME type of the data, such as text/plain or image/jpeg.
public void setBinaryValue(byte[] bytes)	Sets the data as an array of bytes and is used for returning binary data.
public void setTextValue(String s)	Sets the data as a String value. It may be used for returning simple text data but also to return base64-encoded binary data by also setting the type accordingly.

Figure 6 The methods of the DetailItem class

Figure 7 shows the code of the example HardcodedBlackbox blackbox. Let's examine it for a moment.

Lines 1 – 7 define the package and import the necessary classes. **Line 9** defines the blackbox class and implements the UserInfoBlackBoxAPI interface, which defines the necessary methods we need to implement. **Line 11** defines the default constructor, which is a constructor that doesn't take any arguments. **Lines 14 – 17** define the init() life-cycle method. Recall that the UserInfoServlet calls this init() method in our blackbox when it constructs the blackboxes. In this implementation I simply print a statement to the server console.

```
 1. package com.eview.bb;
 2.
 3. import com.ibm.sametime.userinfo.userInfobbapi.DetailItem;
 4. import com.ibm.sametime.userinfo.userInfobbapi.RequestContext;
 5. import com.ibm.sametime.userinfo.userInfobbapi.Response;
 6. import com.ibm.sametime.userinfo.userInfobbapi.UserInfoBlackBoxAPI;
 7. import com.ibm.sametime.userinfo.userInfobbapi.UserInfoException;
 8.
 9. public class HardcodedBlackbox implements UserInfoBlackBoxAPI {
10.
11.   public HardcodedBlackbox() {
12.   }
13.
14.   public void init() throws UserInfoException {
15.     // do nothing
16.     System.out.println("Initalizing HardcodedBlackbox...");
17.   }
18.
19.   public Response processRequest(RequestContext req) throws UserInfoException {
20.     if (req == null) {
21.       throw new UserInfoException("Invalid request - null value");
22.     }
23.     System.out.println("HardcodedBlackbox serving data for user: " +
              req.getUid());
24.
25.     // create response object
26.     Response response = new Response(req.getRequestID());
27.     response.setUserFound(true);
28.
29.     // add data
30.     DetailItem item = createNewItem("Company", "Some company (" +
              req.getUid() + ")");
31.     response.setRetrievedDetail(item.getId(), item);
```

Continues on next page

Figure 7 Source code of the HardcodedBlackbox blackbox class


```
32.     item = createNewItem("Department", "Some department(" + req.getUid() + ")");
33.     response.setRetrievedDetail(item.getId(), item);
34.     item = createNewItem("Title", "Some title(" + req.getUid() + ")");
35.     response.setRetrievedDetail(item.getId(), item);
36.     item = createNewItem("Telephone", "555-5555(" + req.getUid() + ")");
37.     response.setRetrievedDetail(item.getId(), item);
38.
39.     // return
40.     return response;
41.   }
42.
43.   private DetailItem createNewItem(String detailName, String value) {
44.     DetailItem respItem = new DetailItem();
45.     respItem.setName(detailName);
46.     respItem.setId(detailName);
47.     respItem.setType("text/plain");
48.     respItem.setTextValue(value);
49.     return respItem;
50.   }
51.
52.   public void terminate() {
53.     System.out.println("Terminating HardcodedBlackbox...");
54.   }
55. }
```

Figure 7 (continued)

> ***Note!***
>
> All blackbox implementations require a default, zero-argument constructor.

Lines 19 – 41 are the most important part of the blackbox. After checking the validity of the argument in **lines 20 – 22**, **line 23** shows a useful method of the RequestContext interface — namely, the getUid() method, which finds the user ID of the person being inquired about. **Lines 25 – 27** construct our Response object, passing the request ID (RequestContext.getRequestId()) in the constructor. **Lines 29 – 37** adds information for the Company, Department, Title, and Telephone items before returning the Response object to the caller in **line 40**.

Lines 43 – 50 define a utility method for wrapping a string value in a DetailItem object. Notice how we set the MIME type to text/plain, indicating that the value is a simple string value. **Lines 52 – 54** are the terminate() life-cycle method that prints a statement to the server console when the blackbox is terminates.

I discuss the deployment of this and other custom blackboxes in the section "Configuring Sametime to use a custom blackbox." Now, let's look beyond hard-coding return data to more extensible blackbox code.

Getting dynamic data into a blackbox

When using a custom blackbox, you really shouldn't have to know anything about the wrapping of the individual parts of response. The framework should handle that for you; unfortunately that isn't the case at the moment. An abstract class can ease the developer's job if it removes the need to deal with Sametime's blackbox API, UserInfoBlackBoxAPI. So, I wrote an abstract class (AbstractBlackbox) and included its source code in the com.eview.bb package at THE VIEW Web site. Any developer can extend this class — the base class even correctly handles images, including the encoding. To understand more about how it works, I recommend that you download and read through the fully commented source code.

Using the AbstractBlackbox base class is simple. **Figure 8** shows a sample implementation called "PhonebookBlackbox" that extends AbstractBlackbox. The download files also include PhonebookBlackbox. Compare this sample with the hard-coded response blackbox in **Figure 7** and notice that the only piece of code that changes between these blackbox implementations is the code that obtains the business card information for the user. All information packaging is exactly the same. The AbstractBlackbox class has one method for the developer to override, which simply specifies where to obtain the business card information. AbstractBlackbox handles all packaging and returning data to the servlet for you.

Let's look closer at the sample implementation, PhonebookBlackbox, in **Figure 8**. **Lines 3 – 15** import the different classes that the PhonebookBlackbox class requires. When the UserInfoServlet sends a request to the PhonebookBlackbox blackbox, it actually interacts with the methods in the AbstractBlackbox class. That class calls the loadValues() method on **line 19** with information about the user and an array with the names of the information pieces the servlet wants. **Lines 24 – 30** initialize a thread to Domino and open a database and a view. In **line 33**, the custom blackbox utilizes the user ID to look up the user's record in the phonebook. **Lines 34 – 51** verify that the blackbox found a document and put information in a java.util.Map using appropriate keys. Finally, this code returns the map.

Java code returning an image to the UserInfoServlet must format the image as base64-encoded bytes.[2] If all you need to do is extract an image from a richtext field in Notes, you can use the

[2] Base64 is an encoding scheme for converting byte data into ASCII characters in such a way as to prevent e-mail or other transit systems from modifying the data.

```
 1.  package com.eview.bb;
 2.
 3.  import java.util.Collections;
 4.  import java.util.HashMap;
 5.  import java.util.Map;
 6.

 7.  import com.ibm.sametime.userinfo.userinfobb.EncodedImage;
 8.  import com.ibm.sametime.userinfo.userinfobb.ImageExtractor;
 9.

10.  import lotus.domino.Database;
```

Continues on next page

Figure 8 The custom PhonebookBlackbox Java class extends the AbstractBlackbox class


```
11.  import lotus.domino.Document;
12.  import lotus.domino.NotesFactory;
13.  import lotus.domino.NotesThread;
14.  import lotus.domino.Session;
15.  import lotus.domino.View;
16.

17.  public class PhonebookBlackbox extends AbstractBlackbox {
18.

19.    protected Map loadValues(String user_id, String[] details) throws
         Exception {
20.      try {
21.        // declarations
22.        Map m = new HashMap();
23.
24.        // initialize Notes environment and create session
25.        NotesThread.sinitThread();
26.        Session session = NotesFactory.createSession();
27.
28.        // get database and view
29.        Database db = session.getDatabase(null, "phonebook.nsf");
30.        View view = db.getView("lookupPersonByEmail");
31.
32.        // lookup user
33.        Document doc = view.getDocumentByKey(user_id, true);
34.        if (null != doc) {
35.          m.put("MailAddress", user_id);
36.          m.put("Name", doc.getItemValueString("PersonFirstname") +
37.               " " + doc.getItemValueString("PersonLastname"));
38.          m.put("Title", doc.getItemValueString("PersonTitle") +
39.               " (" + doc.getItemValueString("PersonDepartment") + ")");
40.          m.put("Telephone", doc.getItemValueString("PersonPhone"));
41.          m.put("Company", doc.getItemValueString("CompanyName"));
42.
43.          // get image
44.          EncodedImage img = this.encodeImage(session, doc, "Picture");
45.          m.put("Photo", img);
46.
47.          // return map
48.          return m;
49.        } else {
50.          return Collections.EMPTY_MAP;
```

Continues on next page

Figure 8 (continued)

```
51.          }
52.        } catch (Throwable t) {
53.          throw new Exception(t);
54.        } finally {
55.          NotesThread.stermThread();
56.        }
57.      }
58.
}
```

Figure 8 (continued)

approach on **lines 43 – 45**, which uses the encodeImage() method of the AbstractBlackbox super class that I provide in the download files. This method uses some classes that IBM provides; it takes a session, document, and field name and does all the heavy lifting of encoding an image for you. On the other hand, if you need to return an image as a byte array, you need to use the AbtstractBlackbox. encodeImage(byte[]) method that takes a byte array instead. It's easy to accomplish either way.

As you can see from the example in **Figure 8**, the AbstractBlackbox class makes it easy for any Java developer to write custom blackboxes. Through inheritance, your code gets a lot of functionality for free — you just have to implement the method that obtains information to match your requirements. If you need to return an image, the AbtractBlackbox class makes that easy too, because it provides methods for extracting picture data from Notes documents.

Getting business card data from Lotus Connections into a blackbox

Imagine Ino Chemicals has already deployed Lotus Connections. All of the business card data that Sametime needs is already in the Profiles component of the Lotus Connections database. In this case, getting the business card data from Profiles would make sense. Since Lotus Connections stores the data for the Profiles component in a relational database, it's easy to retrieve from Java using Java Database Connectivity (JDBC). Let's look at an example implementation called the ConnectionsStandaloneBlackbox. You have to do three things: Create the Java source code for the blackbox, configure the notes.ini file on the Sametime server to recognize the blackbox JAR file, and configure Sametime's UserInfoConfig.xml file. **Figure 9** lists the Java source code for the

```
1. package com.bleedyellow.sametime.bb;

2. import java.io.FileInputStream;
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
```

Continues on next page

Figure 9 The Java source code for the custom ConnectionsStandaloneBlackbox example class, which gets business card data from a Lotus Connections profiles database in DB2


```
 7. import java.util.Properties;

 8. import com.bleedyellow.sametime.bb.util.Base64;
 9. import com.ibm.sametime.userinfo.userInfobbapi.DetailItem;
10. import com.ibm.sametime.userinfo.userInfobbapi.RequestContext;
11. import com.ibm.sametime.userinfo.userInfobbapi.Response;
12. import com.ibm.sametime.userinfo.userInfobbapi.UserInfoBlackBoxAPI;
13. import com.ibm.sametime.userinfo.userInfobbapi.UserInfoException;

14. public class ConnectionsStandaloneBlackbox implements UserInfoBlackBoxAPI {
15.    // constants
16.    private static final String SQL = "SELECT EMPL.PROF_UID UID,
       PROF_DISPLAY_NAME DSP_NAME, PROF_MAIL_LOWER EMAIL, " +
17.      "PROF_MOBILE MOBILE, PROF_PAGER PAGER, PROF_TITLE TITLE,
          PROF_TIMEZONE TIMEZONE, PROF_TELEPHONE_NUMBER PHONE, " +
18.      "PROF_BLOG_URL BLOG_URL, EMPL.PROF_DEPARTMENT_NUMBER DEPT_NO,
           DEPT.PROF_DEPARTMENT_TITLE DEPT_NAME, " +
19.      "EMPL.PROF_ORGANIZATION_IDENTIFIER ORG_IDENT,
          ORG.PROF_ORGANIZATION_TITLE ORG_NAME,
          EMPL.PROF_ISO_COUNTRY_CODE CTRY_ISOCODE, " +
20.      "CTRY.PROF_COUNTRY_DESC CTRY_NAME, P.PROF_IMAGE PHOTO_BYTES,
          P.PROF_FILE_TYPE PHOTO_MIMETYPE FROM EMPINST.EMPLOYEE EMPL LEFT " +
21.      "OUTER JOIN EMPINST.PHOTO P ON EMPL.PROF_UID=P.PROF_UID LEFT OUTER
          JOIN EMPINST.DEPARTMENT DEPT ON " +
22.      "EMPL.PROF_DEPARTMENT_NUMBER=DEPT.PROF_DEPARTMENT_CODE LEFT
          OUTER JOIN EMPINST.COUNTRY CTRY ON " +
23.      "EMPL.PROF_ISO_COUNTRY_CODE=CTRY.PROF_ISO_COUNTRY_CODE LEFT
          OUTER JOIN EMPINST.ORGANIZATION ORG ON " +
24.      "EMPL.PROF_ORGANIZATION_IDENTIFIER=ORG.PROF_ORGANIZATION_CODE
          WHERE EMPL.PROF_MAIL_LOWER=?";
25.
26.    // declarations
27.    private boolean valid = false;
28.    private String url = null;
29.    private String username = null;
30.    private String password = null;
31.
32.    public void init() throws UserInfoException {
33.       try {
34.          // load class
35.          Class.forName("com.ibm.db2.jcc.DB2Driver");
36.
37.          // read properties
38.          FileInputStream fin = new FileInputStream("LCUserInfoConfig.properties");
```

Continues on next page

Figure 9 (continued)

```
39.       Properties props = new Properties();
40.       props.load(fin);
41.
42.       // read properties
43.       this.url = props.getProperty("url");
44.       this.username = props.getProperty("username");
45.       this.password = props.getProperty("password");
46.
47.       // close file and mark valid
48.       fin.close();
49.       this.valid = true;
50.
51.     } catch (Exception e) {
52.       e.printStackTrace();
53.       throw new UserInfoException("Unable to initialize");
54.     }
55.   }
56.
57.   public Response processRequest(RequestContext ctx) throws UserInfoException {
58.     // declarations
59.     Connection conn = null;
60.     PreparedStatement ps = null;
61.     ResultSet rs = null;
62.
63.     // create empty response
64.     Response response = new Response(ctx.getRequestID());
65.
66.     // only process request if the blackbox is correctly configured
67.     if (!this.valid) {
68.       return response;
69.     }
70.
71.     // get user id
72.     String userid = ctx.getUid();
73.
74.     try {
75.       // perform database lookup
76.
77.       conn = DriverManager.getConnection(this.url, this.username, this.password);
78.       ps = conn.prepareStatement(SQL);
79.       ps.setString(1, userid);
80.       rs = ps.executeQuery();
81.       if (rs.next()) {
82.         String display_name = rs.getString("dsp_name");
```

Continues on next page

Figure 9 (continued)


```
83.           String blog_url = rs.getString("blog_url");
84.           String title = rs.getString("title");
85.           String timezone = rs.getString("timezone");
86.           byte[] photo_bytes = rs.getBytes("photo_bytes");
87.           String mimetype = rs.getString("photo_mimetype");
88.
89.           // set data in response
90.           response.setRetrievedDetail("MailAddress",
               this.createTextDetailItem("MailAddress", userid));
91.           response.setRetrievedDetail("Name", this.createTextDetailItem("Name",
               display_name));
92.           response.setRetrievedDetail("Title", this.createTextDetailItem("Title", title));
93.           response.setRetrievedDetail("Telephone",
               this.createTextDetailItem("Telephone", timezone));
94.           response.setRetrievedDetail("Company",
               this.createTextDetailItem("Company", blog_url));
95.           response.setRetrievedDetail("Photo", this.createImageDetailItem("Photo",
               mimetype, photo_bytes));
96.
97.         }
98.       } catch (Exception e) {
99.         e.printStackTrace();
100.
101.        // return empty response
102.        return response;
103.
104.      } finally {
105.        // release resources
106.        if (null != rs) {
107.          try {
108.            rs.close();
109.          } catch (Exception e) {}
110.        }
111.        if (null != ps) {
112.          try {
113.            ps.close();
114.          } catch (Exception e) {}
115.        }
116.        if (null != conn) {
117.          try {
118.            conn.close();
119.          } catch (Exception e) {}
120.        }
121.      }
```

Continues on next page

Figure 9 (continued)

```
122.
123.      // return
124.      return response;
125.    }

126.    public void terminate() {
127.
128.    }
129.
130.    private DetailItem createTextDetailItem(String name, String value) {
131.      DetailItem i = new DetailItem();
132.      i.setName(name);
133.      i.setType("text/plain");
134.      i.setTextValue(this.nullSafeString(value));
135.      return i;
136.    }
137.
138.    private DetailItem createImageDetailItem(String name, String mimetype, byte[]
          bytes) {
139.      DetailItem i = new DetailItem();
140.      i.setName(name);
141.      if (null != mimetype && mimetype.equalsIgnoreCase("image/pjpeg")) {
142.        i.setType("image/jpeg");
143.      } else {
144.        i.setType("image/gif");
145.      }
146.      String  base64 = Base64.encodeBytes(bytes, Base64.DONT_BREAK_LINES);
147.      i.setTextValue(base64);
148.      return i;
149.    }
150.
151.    private String nullSafeString(String s) {
152.      return (null == s) ? "" : s;
153.    }
154. }
```

Continues on next page

Figure 9 (continued)

ConnectionsStandaloneBlackbox class, found in the com.bleedyellow.sametime.bb package, for a blackbox that reads business card data from DB2. **Figure 10** shows the corresponding UserInfoConfig.xml file.

Let's examine the code. **Lines 1 – 13** include the usual Java package statement and import statements. **Line 14** begins the ConnectionsStandaloneBlackbox class as an implementation of the UserInfoBlackBoxAPI interface. **Lines 16 – 24** hold a constant of the Structured Query Language (SQL)[3]

[3] SQL is an ANSI- and ISO-standard interactive programming language for querying and modifying data and for managing databases. Many database vendors support SQL with proprietary extensions to the standard language.


```
<?xml version ="1.0" encoding="UTF-8" ?>
<UserInformation>
   <Resources>
     <Storage type="NOTES">
      <CommonField CommonFieldName="MailAddress"/>
       <Details>
         <Detail Id="MailAddress"  FieldName="InternetAddress" Type="text/plain"/>
       </Details>
     </Storage>
     <Storage type= "LOTUS_CONNECTIONS ">
       <StorageDetails
        DbName= "dummy.nsf " View= "dummy " />

       <Details>
          <Detail Id= "Photo " FieldName= "dummy " Type= "image/jpeg "/>
          <Detail Id= "Location " FieldName= "dummy " Type= "text/plain "/>
          <Detail Id= "Title " FieldName= "dummy " Type= "text/plain "/>
         <Detail Id= "Telephone "  FieldName= "dummy " Type= "text/plain "/>
          <Detail Id= "Company " FieldName= "dummy " Type= "text/plain "/>
          <Detail Id= "Name "  FieldName= "dummy " Type= "text/plain "/>
       </Details>
     </Storage>
   </Resources>
   <ParamsSets>
     <Set SetId="0" params="MailAddress,Name,Title,Location,Telephone,Photo,Company"/>
     <Set SetId="1" params="MailAddress,Name,Title,Location,Telephone,Photo,Company"/>
   </ParamsSets>
   <BlackBoxConfiguration>
     <BlackBox type="NOTES" name="com.ibm.sametime.userinfo.userinfobb.UserInfoNotesBB"
MaxInstances="1"/>
     <BlackBox type= "LOTUS_CONNECTIONS " name= "com.bleedyellow.sametime.bb.
ConnectionsPhotoBlackbox " MaxInstances= "1 "/>
   </BlackBoxConfiguration>
 </UserInformation>
```

Figure 10 UserInfoConfig.xml file customized to use the Lotus Connections blackbox

string that accesses the card data from DB2. Explaining SQL is outside the scope of this article, but you should know that Lotus Connections supports SQL, so we can use it to get a user's Profile data from Connections based on the e-mail address of that user. **Lines 26 – 30** hold variable declarations.

Lines 32 – 55 define the init() method for initializing the blackbox and configuring the class so it is ready to service requests from the UserInfoServlet for business card data. Due to limitations in the Sametime blackbox API, this code cannot get initialization information from UserInfoConfig.xml. Instead, **line 38** uses a custom Lotus Connections blackbox properties file called LCUserInfoConfig.properties to read, parse, and grab the necessary data (URL, user name, and password) for the blackbox. (I discuss how to create this properties file later in this article.) After the code

reads the properties file, **line 49** sets the valid flag to true, indicating that the system has retrieved the properties to access Lotus Connections.

Line 57 starts the processRequest() method, which begins by checking whether the blackbox was correctly initialized (**lines 66 – 69**). **Line 72** retrieves the e-mail address for looking up the business card data before **line 77** creates a java.sql.Connection object to the DB2 database using the standard JDBC approach, the DriverManager class. With this connection, **lines 78 – 87** perform the SQL query against the DB2 database, and **lines 90 – 95** populate the Response object. **Line 124** returns the populated Response object to the caller.

Notice that the code on **line 93** returns the time zone of the user as part of the attribute for phone number (Telephone). This is one of the useful actions you can perform with a custom blackbox, because a standard Sametime client displays the data from Telephone and Company attributes in a hover (i.e., mouse-over) dialog box.

Lines 130 – 136 define a utility method for creating a text DetailItem instance, and **lines 138 – 149** define a utility method for creating an image DetailItem instance. **Lines 151 – 153** define a utility method for safely returning a null String as an empty String because the methods that create DetailItems can't accept a null value.

After writing and compiling your blackbox class, save it as a JAR file. Now that you understand how this custom blackbox works, it's time to configure Sametime to use it.

Figure 10 highlights in bold the changes you need to make to Sametime's UserInfoConfig.xml file for it to use the custom ConnectionsStandaloneBlackbox blackbox. The changes to UserInfoConfig.xml are straightforward. Notice the <StorageDetails> element. You might recall that you used it when configuring LDAP blackboxes in Part 1 of this series. This element must be present to fulfill requirements in IBM's Sametime API code when a custom blackbox has additional configuration properties (unlike the HardcodedBlackbox blackbox). To accommodate the element in cases where the code is required but not used, simply specify dummy values, as you see in the bolded <Storage> block of code in **Figure 10**.

Configuring Sametime to use a custom blackbox

There are three basic steps to configure Sametime to use a custom Blackbox class:

1. Put the custom blackbox class on the Java class path of the Sametime server.
2. Configure the Sametime server's UserInfoConfig.xml file to use the custom blackbox.
3. (Optional) Configure any additional configuration files that the blackbox needs — e.g., to locate and query the data source in the Lotus Connections example.

Please note that step 3 isn't optional for the ConnectionsStandaloneBlackbox described earlier because this blackbox needs the additional configuration information from the properties file.

Let's go through each step in turn.

Configuring the Sametime server's Java class path

Place your custom blackbox's JAR file on the Sametime server's Java class path using the JavaUserClassesExt notes.ini variable. The following example demonstrates how to do this.

Let's say that you packaged a custom blackbox in a JAR file called my_blackbox.jar and copied it to C:\custom_blackbox on your Sametime server. To add my_blackbox.jar to the class path on the Sametime server, open the notes.ini file from the Sametime server in a text editor, such as Notepad, and go to the JavaUserClassesExt INI variable definition. You should see a bunch of lines starting with variable names like LSTJava0, LSTJava1, and so on, as in **Figure 11**. Each of these lines identifies the path to a JAR file, thereby bringing it onto the Sametime server class path.

Figure 12 shows the same notes.ini file after I added the my_blackbox.jar file to the Sametime server class path. The bold text shows a variable called BB1 in the JavaUserClass extension (JavaUserClassExt=) and a line identifying the path to this custom JAR file (BB1=).

```
JavaUserClassesExt=LSTJava0,LSTJava1,LSTJava2,LSTJava3,LSTJava4,LSTJava5,
 LSTJava6,LSTJava7,LSTJava8,LSTJava9,LSTJava10,LSTJava11,LSTJava12,LSTJava13,
 LSTJava14,LSTJava16,LSTJava17
JavaUserClasses=
 LSTJava0=D:\Domino\data\domino\html\sametime\stmeetingroomclient\STMRCRes751\
 properties
LSTJava1=D:\Domino\java
LSTJava2=D:\Domino\stconversion\stconvservlet.jar
LSTJava3=D:\Domino\stconversion\export.jar
LSTJava4=D:\Domino\stconversion
LSTJava5=stcore.jar
LSTJava6=stmtgmanagement.jar
LSTJava7=STNotesCalendar.jar
LSTJava8=mail.jar
LSTJava9=activation.jar
LSTJava10=UserInfo.jar
LSTJava11=telephony_ext\TelephonyService.jar
LSTJava12=TelephonyActivity.jar
LSTJava13=STPolicy.jar
LSTJava14=D:\Domino\data
LSTJava16=D:\Domino\data\domino\html\sametime\stmeetingroomclient\STMRCRes75\
 properties
LSTJava17=TelephonyService.jar
```

Figure 11 Class paths to JAR files configured in the notes.ini file of a standard Sametime server

```
JavaUserClassesExt=LSTJava0,LSTJava1,LSTJava2,LSTJava3,LSTJava4,LSTJava5,
 LSTJava6, LSTJava7,LSTJava8,LSTJava9,LSTJava10,LSTJava11,LSTJava12,LSTJava13,
 LSTJava14,LSTJava16,LSTJava17,BB1
JavaUserClasses=LSTJava0=D:\Domino\data\domino\html\sametime\stmeetingroomclient\
 STMRCRes751\properties
LSTJava1=D:\Domino\java
... (lines omitted for brevity)
LSTJava17=TelephonyService.jar
BB1=c:\custom_blackbox\my_blackbox.jar
```

Figure 12 The bold lines add my_blackbox.jar to the class path of the Sametime server through notes.ini.

After editing and saving the notes.ini file, don't forget to restart the Domino server that hosts the Sametime server.

Configuring the UserInfoConfig.xml file

Next, configure the UserInfoConfig.xml file to make sure that the Sametime server uses your custom blackbox. Configuring the UserInfoConfig.xml file means adding appropriate tags, as I described in Part 1 of this series. For example, if the JAR file I added to the Sametime server in the step above contained the HardcodedBlackbox blackbox class described earlier, I would edit the UserInfoConfig.xml file to look like **Figure 13**, save and close the file, and then restart the HTTP task on the Sametime server. At this point, Sametime users should see business card data from the custom blackbox.

Set up any additional configuration files (optional)

Recall the configuration mechanism whereby the custom Lotus Connections blackbox knows the location of the data source that the user needs to query (**line 38** of **Figure 9**). Because the UserInfoConfig.xml can't hold the configuration data that you need, you must use a custom properties file.

Create a simple properties file (the example blackbox calls it LCUserInfoConfig.properties) and place it in the Domino program directory of the server hosting Sametime. In this .properties file, using standard Java property file syntax, simply specify the name of the blackbox that you are configuring, the URL to the DB2 database being accessed, and the user name and password Sametime needs to make the connection automatically. For example, **Figure 14** shows how to

```
<?xml version ="1.0" encoding="UTF-8" ?>
<UserInformation>
  <Resources>
    <Storage type="NOTES">
      <Details>
        <Detail Id="MailAddress" FieldName="InternetAddress" Type="text/plain"/>
      </Details>
    </Storage>
  </Resources>
  <ParamsSets>
    <Set SetId="0" params="MailAddress,Name,Title,Location,Telephone,Photo,Company"/>
    <Set SetId="1" params="MailAddress,Name,Title,Location,Telephone,Photo,Company"/>
  </ParamsSets>
  <BlackBoxConfiguration>
    <BlackBox type="NOTES" name="com.ibm.sametime.userinfo.userinfobb.UserInfoNotesBB"
      MaxInstances="4" />
    <BlackBox type= "CUSTOM " name= "com.eview.bb.HardcodedBlackbox "
      MaxInstances= "4 " />
  </BlackBoxConfiguration>
</UserInformation>
```

Figure 13 Adding the HardcodedBlackbox blackbox to UserInfoConfig.xml

configure a connection to the database called ”peopledb” on a DB2 server with the host name db2.example.com.

Troubleshooting and debugging

When writing custom blackbox implementations for your Sametime environment or simply configuring the Sametime server for the blackbox system, you need to troubleshoot if the correct data doesn't show up in the business card to all or particular users. Luckily, it's quite easy to debug blackboxes. The UserInfoServlet servlet has an HTTP interface that returns data in an XML format, so if you call UserInfoServlet from a standard Web browser like Mozilla Firefox, you can easily view and debug the returned business-card data as XML in the browser.

```
## Lotus Connections blackbox configuration
url=jdbc:db2://db2.example.com:50000/peopledb
username=db2admin
password=password
```

Figure 14 The LCUserInfoConfig.properties file for configuring the DB2 connection for the Lotus Connections blackbox

Blackbox tips

The Sametime blackbox API includes two lifecycle methods that are appropriately called init() (for initialize) and terminate(). When UserInfoServlet receives its first request for a business card, it creates an instance pool for each type of blackbox that has been configured in UserInfoConfig.xml. It then initializes the pools with the number of blackbox instances that the MaxInstances attribute of the <Blackbox> tag indicates, as in this example:

```
<BlackBox type="NOTES" name="com.ibm.sametime.userinfo.userinfobb.UserInfoNotesBB"
MaxInstances="4"/>
```

When UserInfoServlet constructs the blackbox objects, it calls the init() method on each blackbox instance. Your custom blackbox JAR file must implement the init() method to initialize any special resources, such as database connections, that the blackbox needs to do its work. For example, refer to the init() method in **Figure 9** to see how I used it to read configuration information before starting to service clients with business card data.

When the Domino HTTP task terminates, it unloads UserInfoServlet, which in turn unloads the configured blackboxes. Before unloading the blackboxes, UserInfoServlet calls the terminate() method for each blackbox. At this time, when using Sametime release 7.5.1 CF1/8.0, a code bug causes UserInfoServlet to call the terminate() method for only half of the blackboxes in any pool. Although this issue has been reported to IBM Lotus, you need to be aware that as of the current release, you cannot rely on the terminate() method working consistently. I suggest ensuring that your code closes any acquired resources correctly and properly cleans up blackbox instances upon blackbox termination.

Using the HTTP interface

In my opinion, using the HTTP interface to call UserInfoServlet is the easiest way to troubleshoot the business card system, regardless of which blackbox you use. You don't need a password to log in as a particular user, and you can debug the business card system without interrupting Sametime users on the same server.

To troubleshoot from a browser, you construct URL arguments and call UserInfoServlet with instructions about what to return. For example, **Figure 15** shows an example set of XML code that resulted from a call to UserInfoServlet that I constructed as a URL in a Web browser. The servlet can take up to four of the following arguments in a URL:

- **operation** — valid values are 1, 2, and 3
- **userid** — the user name
- **setid** — valid values are the configured blackbox sets, usually 0 and 1
- **field??** — where ?? is an incremental number, starting at 1

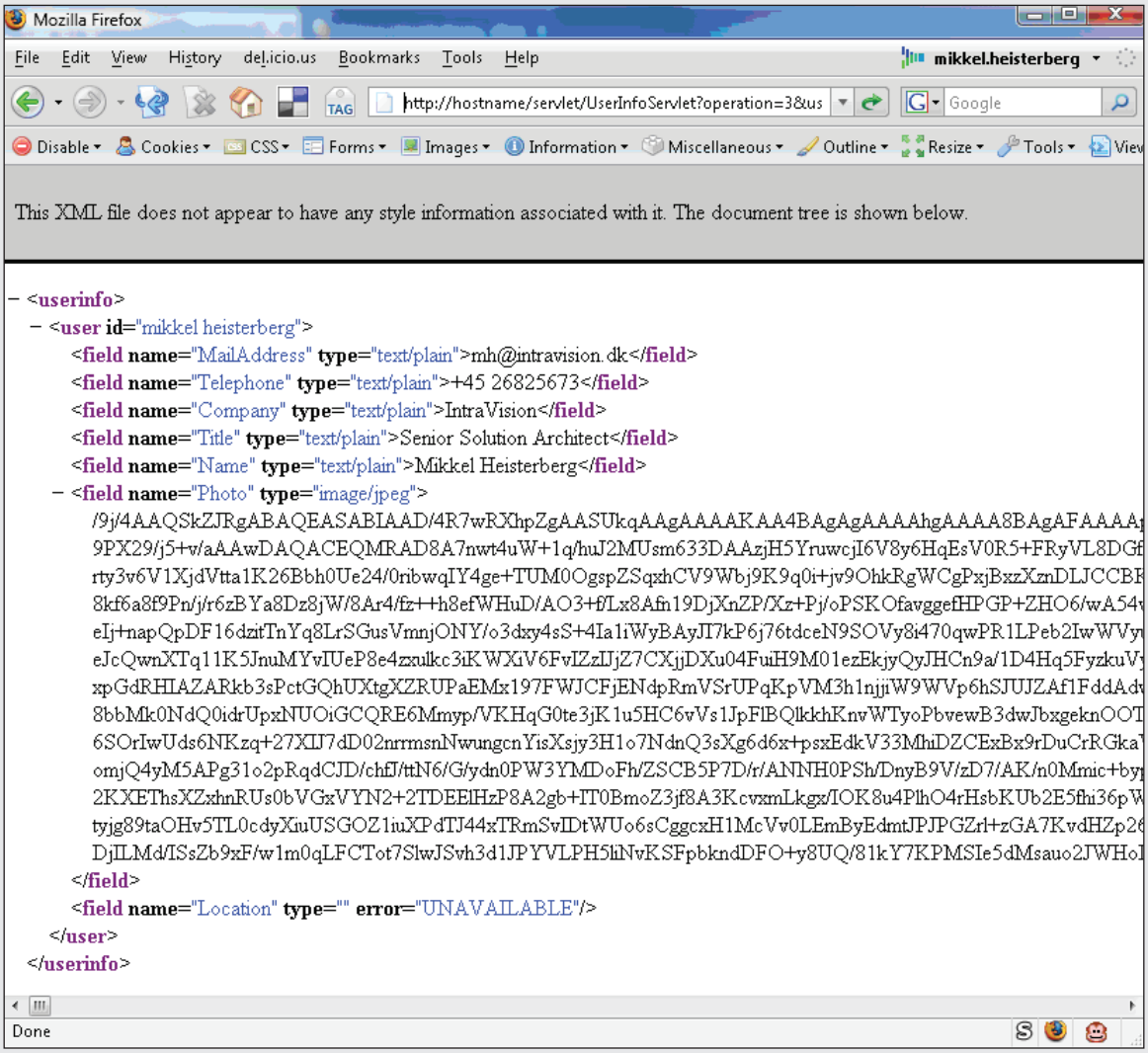

Figure 15 Calling UserInfoServlet from a Web browser

The operation parameter is the main parameter that controls servlet's actions and which other parameters are required. Here is a description of each value for this parameter:

- **operation=1** retrieves values for specific fields based on the field name that the Sametime client uses. You can request information from more than one field at a time. The valid names of the fields are the ones from UserInfoConfig.xml and hence the field names shown when you use the operation=2 parameter.
- **operation=2** gives you a list of the fields configured for the required field set. If you leave out the optional <set id> parameter, it implies a value of 0.
- **operation=3** is the workhorse of the servlet and will return the data that corresponds to the Sametime client's requests. This operation returns, as the others do, an XML document containing the requested user's business card. The photo data is base64 encoded and the type attribute indicates whether it is a JPG or a GIF image.

The table in **Figure 16** shows the URL syntax for each parameter value, an example URL, and the resulting XML code.

Parameter values	URL syntax/example	Resulting XML code
operation=1	**Syntax:** http://hostname/servlet/UserInfoServlet?operation=1&field1=<fieldname>&userid=<username> **Example:** http://hostname/servlet/UserInfoServlet?operation=1&field1=Name&field2=Title&userid=jdoe	<?xml version="1.0" encoding="UTF-8"?> <userinfo> <user id ="jdoe"> <field name="Name" type="text/plain">John Doe</field> <field name="Title" type="text/plain">Marketing Manager</field> </user> </userinfo>
operation=2	**Syntax:** http://hostname/servlet/UserInfoServlet?operation=2&setid=<set id> **Example (left out optional <set id> parameter):** http://hostname/servlet/UserInfoServlet?operation=2&setid=0 **Example (included optional <set id> parameter):** http://hostname/servlet/UserInfoServlet?operation=2&setid=1	<?xml version="1.0" encoding="UTF-8"?> <userinfo> <field name="MailAddress" type="text/plain" /> <field name="Telephone" type="text/plain" /> <field name="Company" type="text/plain" /> <field name="Title" type="text/plain" /> <field name="Name" type="text/plain" /> <field name="Photo" type="image/jpeg" /> <field name="Location" type="text/plain" /> </userinfo> <?xml version="1.0" encoding="UTF-8"?> <userinfo> <field name="Photo" type="image/jpeg" /> <field name="Name" type="text/plain" /> <field name="Title" type="text/plain" /> <field name="MailAddress" type="text/plain" /> <field name="Telephone" type="text/plain" /> </userinfo>

Continues on next page

Figure 16 The syntax of the different requests that you can make to the UserInfoServlet

operation=3	**Syntax:** http://hostname/servlet/UserInfoServlet?operation=3&userid=<username> **Example:** http://hostname/servlet/UserInfoServlet?operation=3&userid=jdoe	<?xml version="1.0" encoding="UTF-8"?> <userinfo> <user id="John Doe/Acme"> <field name="MailAddress" type="text/plain">jdoe@example.com</field> <field name="Telephone" type="text/plain">555-9921</field> <field name="Company" type="text/plain">Acme Corp.</field> <field name="Title" type="" error="UNAVAILABLE"/> <field name="Name" type="text/plain">John Doe</field> <field name="Photo" type="image/gif">R0lGODlhoQCjAOf==</field> <field name="Location" type="" error="UNAVAILABLE"/> </user> </userinfo>

Figure 16 (continued)

For issues related to the Domino HTTP configuration

The UserInfoServlet runs as part of the Domino HTTP stack, so it's also possible that a lack of business card data is due to an Internet site being incorrectly configured in Domino. If you are unable to get business card data, try using a network sniffer or packet tracer to see the requests from the Sametime client.

Conclusion

You've seen how easy it is to customize a Java blackbox so that the Sametime client pulls data for business cards on an as-needed basis from sources beyond the Domino or LDAP directories, such as Lotus Connections or other phonebook applications. This series of articles clarifies how the Sametime business card system can retrieve its data from anywhere. You're now ready to develop your own blackbox for your Sametime 7.5/8.0 environment.

Resources

My blog:
http://lekkimworld.com

A guide to setting up business card photos on a Sametime 7.5 server:
www.bingham.co.za/?p=41

For more information on the Blackbox API, see "Chapter 5: User Information SPI" in stddatkdevguide.pdf (supplied with the Sametime 7.5 SDK)

