Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Sametime
Updated: 02-Jan-2003

by John W. Rooney
and Dick McCarrick

One of the more exciting new technologies coming out of IBM/Lotus these days is Sametime bots. These are programs that take advantage of Sametime's awareness and real-time collaboration features to provide an interactive, text-based interface to a back-end data source such as a database or Web page. Bots perform specific tasks such as locating information you request, reminding you when an event is approaching, and finding mutually available meeting times in users' calendars. You can design bots that you chat with online and ask questions as though they were real live people!

This article is the first of two discussing Sametime bots. In this article, we introduce what Sametime bots are and how they work. We then describe the basic process of creating a Sametime bot, using available Sametime toolkits. We conclude by examining in detail a simple bot that you can modify to create your own bot. (You can download the code for this bot from the **Sandbox**.)

We assume you're an experienced Sametime and Java programmer and are familiar with the developer toolkits that come with Sametime.

What on earth is a Sametime bot?

We'll be the first to admit that the word *bot*—derived from robot—may remind some people of old horror movies with huge mechanical monsters wreaking havoc on poor defenseless humans. In the world of software, however, bot has a far more benign meaning. In clinical terms, bots can be defined simply as software tools that perform specific actions based on user commands. For example, a bot might find and retrieve information you request, or otherwise act as an intermediary between you and the data you seek. They can interact with you in real-time or work behind the scenes.

When you apply bot technology to Sametime, things quickly get interesting. For example, a Sametime bot can use awareness technology to keep track of an entire Sametime community—who is currently logged in, what their status is, and so on. Plus, the bot can access data stores such as Notes and DB2 databases, retrieving member information such as job titles and areas of expertise. This immediately raises some very intriguing possibilities for solving problems that involve finding the best person available to answer a specific question, among many other applications.

Further, a Sametime bot appears as an online person in your Buddy list. In other words, when a bot is running, you'll see it online as though it were a fellow member of the Sametime community. This lets you design bots that users can actually interact with—ask natural language questions, issue commands, and so forth. To a Sametime user, a well-designed bot may resemble an ever-present expert who can answer questions and execute commands with superhuman speed. Bots can even have built-in "personality" to enhance the user experience (though, of course, simple bots serving a single purpose will likely be limited in this regard).

Another important point is that a Sametime bot is aware of when you and others are online. This allows the bot to

initiate a chat with you when a special event occurs. This lets you design applications both simple (for example a timed reminder) or more complex (for instance, initiating a Sametime meeting when three people you need to meet with are all logged in simultaneously). With a little imagination, you can create a bot to collect certain information for you when you're offline and send you this information when you log in—or pass it along to some other application of your choice, such as a Notes database. This ability to have a Sametime bot notify you could appeal to workflow developers, who may prefer the immediacy of instant messaging (rather than email) to notify people that a document is awaiting their approval.

Put these features all together, and you can design some surprisingly sophisticated interactive applications. For example, at IBM we have a number of Sametime bots available to employees, including:

- Calendar, which searches the calendars of specified people to find a mutually available meeting time
- Dictionary, which looks up requested words in an online dictionary
- Help, which allows users to search across multiple online help systems
- RemindMe, which notifies users when a time/calendar event occurs (for example an upcoming meeting or vacation)
- SkillTap, which finds and connects users with others possessing one or more requested skills
- Stockquote, which retrieves price data from any online stock market ticker (not that we spend a lot of time doing this during work hours, of course!)
- WhatIs, which defines IBM acronyms (and is therefore very popular)

Even within this small sampling of Sametime bots, it's easy to see how one or more could be adapted to your needs. For example, the Help bot could be very useful to a corporate support desk. The bot looks for frequently asked questions, performing natural language parsing to find keywords. It then searches multiple online help databases for these keywords, looking for questions that most closely match the one asked by the user. Finally, the bot sends a Sametime message to the user, listing the questions it found, along with links to appropriate answers. A similar Sametime bot might help you reduce support costs at your site.

Other potential uses of Sametime bots include (but are not limited to):

- Automatically responding to messages from users (including sending customized messages to specific users)
- Server-generated alerts and other general announcements
- Offline message delivery
- Capturing and storing online conversations

It's important to note that Sametime bots can be used wherever Sametime is available. For example, you can enhance a QuickPlace by creating a Sametime bot that serves as an ever-present "team member" whom you can contact for quick information. (QuickPlace, of course, also comes with its own Placebot technology.)

Our goal in this section is to provide you with a basic understanding of what Sametime bots are, how they work, and some of the things you can do with them. If we succeeded, you'll now want to learn more about how you can build a Sametime bot of your own.

Design and deployment considerations

In this section, we offer some suggestions to think about when you design, program, and deploy your Sametime bots.

Designing the bot

When designing a Sametime bot, you should consider the special nature of the way users interact with it. As we mentioned earlier, a bot can have the "look and feel" of chatting online with an actual person, another member of the Sametime community. Therefore, the bot interface can play a crucial role in how successful it is. This is especially true for bots designed to help less technically sophisticated users find information.

So when planning your Sametime bot, think about the keywords and verbs it will recognize and accept. Ideally, these can be incorporated into a natural language interface that feels conversational to the user. Also bear in mind how the bot will respond to users. Messages should read like complete sentences, rather than arcane alphanumeric strings that need to be deciphered. A carefully crafted, "natural" interface may add time and complexity to developing the bot, but it's worth the effort to make your users feel comfortable.

Programming the bot

You use the Sametime developer toolkits to create bots. Sametime toolkits fall into two categories: client and server. Client toolkits primarily interact directly with the user; server toolkits run on the server and provide background services. You can use any of these toolkits to create Sametime bots:

- (Client) Java, C++, and COM
- (Server) Community Server

For more information on these and other Sametime toolkits, see the *LDD Today* article "**A tour of Sametime toolkits**." You can also consult the **Sametime toolkit documentation**, available for download from the Documentation Library.

Deploying the bot
Sametime bots typically run as stand-alone applications, although they can also run as servlets or even Domino agents. A bot can run on either the client or the server. The client is probably more common, although a bot running on the server is more accessible to multiple administrators (and therefore, may be more available for sending out general announcements and messages to the entire Sametime community).

To the Sametime server, a bot is usually considered "just another person." This means the server expects the bot to provide a valid user name and password when it logs in. The server will then authenticate these credentials against its person directory, either a Domino Directory or an LDAP directory. You can provide user name and password information as parameters when you launch the bot, for example *botname user name password.* This means, of course, the user name and password combination you enter must exist in the LDAP or Domino Directory used by the Sametime server. (Often, users will simply enter their own user names and passwords when launching a bot. The bot then runs under the authority of the person who launched it.)

Security can be a bit more complicated for Sametime bots that launch automatically in response to the arrival of a specified date/time or other triggering event. In this case, the bot must supply the Sametime server with a user name and password without human intervention. This information can be stored in the code or a file that the bot can access to pass this information to the server. In this situation, you need to ensure the credential data is inaccessible to unauthorized users.

The "Eightball" sample bot

Now let's turn our attention to our sample Sametime bot, which you can download from the **Sandbox**. We've dubbed this the "Eightball" bot because it responds to user questions with a random message. Granted, this bot as currently written has little application in a real Sametime environment (unless you have way too much time on your hands). However, Eightball is very useful for demonstrating how you program a bot to:

- Log in to the Sametime server
- Listen for incoming user messages
- Respond to user messages (in this case with a random response, although in a real production bot you'd build in logic for analyzing the user's questions and for selecting an appropriate response or action)

We start this section with a brief overview of the Sametime Java Toolkit we used to create this bot. Then we examine the bot in detail, describing what each block of code is doing. (At this point the discussion gets rather technical, so a good understanding of Java programming is essential.)

Sametime Java Toolkit
There are a few things you need to know about the Sametime Java Toolkit, which we used to create this sample Sametime bot, prior to diving into the code. In this sample, we use three important toolkit areas:

- Sametime objects
- Sametime services and components
- Listeners

We briefly describe each of these in the following sections. For complete details on the Sametime Java Toolkit, refer to the **Sametime Java Toolkit Developer's Guide**.

Sametime objects
The Sametime Java Toolkit exposes an object-oriented API for developers. This type of API should be very familiar to Java programmers. Many services provided by the Sametime server and the toolkit require you to create a certain object to use the service. For instance, the STSession object holds the set of components associated with the Sametime session. Each component relates to functions (such as instant messaging or awareness) provided by the Sametime server. You must create an instance of the STSession object to load component functions into your application.

Sametime services and components
The Sametime Java Toolkit includes components for accessing individual services provided by the Sametime server. These services include:

- Community Service, which lets you log in and out from the Sametime server and perform functions such as setting your status

- Instant Messaging Service, which allows users to send messages (text or binary)
- Lookup Service, which lets you query the server for information about users or groups and returns an associated STObject (STUser or STGroup)
- Places Service, which allows you to define a virtual place (such as an n-way chat) and invite users to participate

In our sample, the only service we use is Community Service.

There are several ways to load the required components into your application. The easiest is to use either the loadAllComponents or loadSemanticComponents method of the STSession. You can also load components individually by using the loadComponents method and passing a list of components you want to access.

After you load the appropriate components, you need to get a reference to their respective APIs to use their services. You can do this with the STSession.getCompAPI() method. For example, if you want to access the InstantMessagingService API use the following code:

```
s = (InstantMessagingService)session.getCompApi(InstantMessagingService.COMP_NAME);
```

where *s* is the name of the variable holding the reference to the service API and *session* is the name of the variable holding the reference to the current STSession object.

The STSession does not distinguish between components, so you need to cast the returned reference to the right type (in this case InstantMessagingService). You can use this form to get references to any of the Sametime toolkit component APIs.

Listeners
The API and services of the Sametime Java Toolkit are asynchronous. This means you need to implement listeners to receive events from the server. Therefore, when you call a function on the server (such as login), there may be a delay before the server responds. As to be expected in the world of instant messaging, you will receive events (such as ones associated with new messages or status changes for specific users) from the server at random times. Using listeners, the server notifies you of events you are interested in and provides the ability to perform actions in response to these events.

As mentioned previously, many services require helper objects to use their functions. Some services (such as the InstantMessagingService) return the helper object directly in an event. For example, although you can use the createIM() method to create an IM (instant messaging) object directly, the imReceived() event handler returns an IM object when the event is triggered. We demonstrate the latter technique in our sample Eightball bot.

Eightball bot code
This section examines the code for Eightball, our demo Sametime bot. You can download this bot from the **Sandbox**.

In the first section of code, we import the various packages from the toolkit that we need to provide the functions of our bot. We also import java.util.* for the random response generation:

```
/* Sametime Bot Sample

import com.lotus.sametime.awareness.*;
import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.constants.*;
import com.lotus.sametime.core.types.*;
import com.lotus.sametime.im.*;
import java.util.*;
```

In our core EightBallBot class, we need to identify the listeners that we will implement. In this case, we register listeners for login events and events related to instant messaging:

```
public class EightBallBot implements Runnable, LoginListener, ImServiceListener, ImListener  {

    CommunityService        commService;
    Thread                    engine;
    InstantMessagingService   imService;
```



```
STSession                    stsession;

//responses holds the array of random answers to user inquiries
String [] responses = {     "Yes",
                            "No",
                            "Maybe",
                            "Doesn't look good",
                             "Try again",
                            "Answer hazy"};
```

Typically Sametime applications have only one STSession object. But because you can have multiple sessions, each STSession has a name that is held in a static session table. In any application, there can be only one session of a given name, so we need to wrap the code that creates the session in a try/catch statement:

```
public EightBallBot(String serverName, String userId, String password) {

    try {
        stsession = new STSession("EightBallSession");
    } catch (DuplicateObjectException e) {
        e.printStackTrace();
        return;
    }
```

After the session is created, we can load the necessary components. In this case, we simply load all semantic (non-UI) components and start the session. Because the CommunityService API is used to log in to the Sametime server, we first need to get a reference to that API using the method described previously. After we have that reference, we add a listener to receive events associated with login and use the loginByPassword method to attempt to log in to the server:

```
    stsession.loadSemanticComponents();
    stsession.start();

    commService = (CommunityService)stsession.getCompApi(CommunityService.COMP_NAME);
    commService.addLoginListener(this);
    commService.loginByPassword(serverName, userId, password);

}
```

When you implement a listener interface such as a LoginListener, there are certain events that you will be required to handle (in this case the loggedIn() and loggedOut() events). A good IDE automatically adds stubs for these required events. If we successfully log in to the server, we receive a loggedIn() event. In this handler, we can request a reference to the InstantMessagingService and register our application to receive chat messages. We use the registerImType() method to indicate the type of messages we want to receive. (This is a very important step.) By default the InstantMessagingService rejects any messages that do not have a registered IM type. We then add the listener imService.addIMServiceListener(this) to receive any incoming instant messages:

```
public void loggedIn(LoginEvent e) {

    imService = (InstantMessagingService)stsession.getCompApi(InstantMessagingService.COMP_NAME);
    imService.registerImType(ImTypes.IM_TYPE_CHAT);
    imService.addImServiceListener(this);

}

public void loggedOut(LoginEvent e) {}
```

The imServiceListener fires the imReceived() event and passes an IM object when a user initiates a conversation with our bot. This event is triggered as soon as a user double-clicks on the bot ID in his Buddy list. In many of our other bots within IBM, we use this event (and the properties and methods of the associated IM object) to immediately respond back to the user. This response may include:

- A welcome message
- Help text
- Personalized information specific to the user

For example, in a bot designed to provide the current trading prices for stock in a user's portfolio, we can use this event to immediately return the information to the user.

To receive follow-on messages from the user, we need to add an IMListener using e.getIM().addIMListener(this). In our example:

- *e* is returned by the event and references the IM session we have open with the user who contacted us
- *getIM()* is the method we use to get the IM object associated with this event
- *addIMListener()* is the method on the IM object we use to listen for additional messages in this session

Here's the code:

```
public void imReceived(ImEvent e) {
    e.getIm().addImListener(this);
}
```

The following methods are required when you implement the IMListener interface:

```
public void dataReceived(ImEvent e) {}

public void imClosed(ImEvent e) {}

public void imOpened(ImEvent e) {}

public void openImFailed(ImEvent e) {}
```

In our sample, we focus on events triggered when we receive text from a user. Note, however, that Sametime can also be used to send binary data. Where you have control over the client being used by the Sametime user community (or you are working with messaging between custom applications designed to handle it) using the IM channel to send and receive binary data can be quite powerful.

As we mentioned, our bot is meant only to respond to user inquiries with a random message. We are not particularly interested in the content of the user's message, but in most bots you need to extract content from the message and use it to select an appropriate response or to perform some other function (such as a database query). When a user sends Eightball a text message, the textReceived() event is triggered. You can use the getText() method of the IMEvent to parse the content of the message. For instance, suppose your bot provides users an interface to a database containing parts information. You can use the getText() method to determine the part number being requested by the user, to parse the part number from the message, and to call the appropriate functions to look up the part information in the database. In our code, we simply generate a random response to the user and use the sendText() method to send the message:

```
public void textReceived(ImEvent e) {

    Random rnums = new Random();
    int item = rnums.nextInt(responses.length);

    e.getIm().sendText(true, responses[item]);

}

public void serviceAvailable(AwarenessServiceEvent e) {}

public void serviceUnavailable(AwarenessServiceEvent e) {}

public void start() {
    if (engine == null) {
        engine = new Thread(this, "EightBallThread");
        engine.start();
    }
}

public void run() {
    Thread myThread = Thread.currentThread();
    while (engine == myThread) {
        try {
```



```
                    Thread.sleep(1000);
                } catch (InterruptedException e) {}
            }
        }

        public static void main(String[] args) {
            EightBallBot ebb = new EightBallBot("<server name>", "<user ID>", "<password>");
            ebb.start();
        }

    }
```

As we mentioned, Eightball isn't the most sophisticated of Sametime bots. But it does clearly demonstrate several important tasks—logging in, listening for user messages, and responding to them—used in virtually all interactive bots. We'll discuss far more complex, ready-to-use Sametime bots in part two of this article.

Sametime bots: your ever-alert friends

To conclude this introductory article on Sametime bots, we'd like to leave you with the following important points:

- *Bots are the first step to moving instant messaging beyond chat.*
  The value in tapping into a Sametime community to get information is obvious. (The authors did so frequently while writing this article.) But the person you need may not be online at the moment or may not know the answer, or perhaps you just feel guilty bothering that person so often. A well-designed Sametime bot, on the other hand, is always available, knows the answer to your question, and doesn't become impatient no matter how often you bother it!
- *Bots provide simple convenient interfaces to data.*
  Not everyone can (or wants to) wade through all available corporate data every time a question arises. A Sametime bot can do this for you, quickly and with focused results, in a way that feels like you've asked a co-worker to do the research.
- *Bots can enable two-way interactions.*
  You can initiate communication with a Sametime bot when you need it to provide information or to perform a task. Conversely, the bot can contact *you* when certain triggering events occur. In this regard, the bot serves as your virtual assistant, performing jobs and monitoring situations you may be too busy to do yourself.
- *Bots can bring real-time activity to workflow and business process.*
  As we mentioned earlier, Sametime bots can provide a faster, real-time alternative to email in workflow processes where time is especially critical.

In this article, we introduced you to Sametime bots. We explained what they are, what they do, and how you can use them. We gave you some background and suggestions to consider when you design and deploy your own Sametime bots. And we provided a simple demo bot, describing what the code is doing and how you can adapt it to create a real bot at your own site.

We hope you found this article helpful. In a future article, we'll explain in greater detail examples of using Sametime bots, along with code for real, working bots we use here at IBM. Please stay tuned!

ABOUT THE AUTHOR
John W. Rooney is a Manager of Internet Technology Development at IBM. Rooney (as his friends call him) and his Webahead team have developed numerous applications used widely within IBM, including several that rely heavily on Sametime bot technology.

