Level: Intermediate
Works with: Lotus Instant Messaging
Updated: 20-Oct-2003

by
Gray
Norton

As you may have noticed from reading past issues of *LDD Today,* Lotus Instant Messaging (Sametime) bots are a hot topic of late. Two previous articles, Building your own Sametime bots, Part 1 and Building your own Sametime bots, Part 2, provide a good introduction to bots, discussing what they are and why you may want to develop your own. These articles also describe how you can use the Lotus Instant Messaging toolkits to build bots, and they provide sample code to get you started.

This article series introduces a new perspective on Lotus Instant Messaging (LIM) bots, placing them in the context of a broader class of applications known as *interactive agents.* It also introduces an alternative approach to developing bots using ActiveBuddy's BuddyScript platform. We start with a quick introduction to BuddyScript and then move on to a discussion of interactive agents and where LIM bots fit in the big picture. We conclude Part 1 of this series with a high-level overview of the BuddyScript SDK platform and discuss why it's worth considering if you're interested in developing interactive agents of any type, including LIM bots. (In Part 2, we'll walk you through the process of actually developing a simple interactive agent and deploying it as an LIM bot.)

This article assumes that you're familiar with basic instant messaging concepts. No specific development experience is required to understand this article or to use the BuddyScript SDK, but some background in software or Web development is, of course, helpful.

About BuddyScript

Before we get started, a quick overview of ActiveBuddy and its BuddyScript technology will help set the stage. ActiveBuddy was founded in 2000 on the premise that text messaging—already a critical person-to-person communications tool—is also an excellent medium for enabling person-to-computer interactions. Recognizing that the text messaging medium presents software developers with a unique set of possibilities (not to mention a unique set of challenges), the company set out to create a platform tailored specifically for developing and deploying software over text messaging networks. ActiveBuddy called the platform BuddyScript and coined the term *interactive agent* to describe the class of software applications it enabled.

While the BuddyScript platform was in development, it was used by ActiveBuddy's professional services team to develop interactive agents for customers. For the most part, these interactive agents were deployed on public instant messaging networks, where they were used primarily for marketing purposes. At the same time, ActiveBuddy was also developing an interactive agent of its own to demonstrate the capabilities of the BuddyScript platform.

This interactive agent (called SmarterChild) offered users fast, friendly conversational access to a smorgasbord of information, services, and entertainment. This included news, weather forecasts, sports scores, movie reviews and showtimes, stock quotes, reference sources, tools and utilities, and games. Although intended as a technology demo, SmarterChild quickly took on a life of its own—in the course of a year, spreading purely by word of mouth, SmarterChild talked to over seven million unique screen names on the AIM and MSN networks. Recently re-launched as a subscription service (with a free 30-day trial), SmarterChild remains available on AIM and MSN today.

While SmarterChild was busy engaging consumers on the public IM networks, ActiveBuddy was working to release the development and deployment tools that comprise the BuddyScript platform. BuddyScript Server, the cornerstone of the platform, and the free BuddyScript Software Development Kit (SDK) became available in mid-2002. Since then, over 10,000 individuals (representing enterprises, consultancies and integrators, online agencies, and independent software vendors) have become registered BuddyScript.com developers. Dozens of their interactive agents are now in various stages of development and deployment.

What are interactive agents?

Now that the introductions are out of the way, let's take a moment to define exactly what interactive agents are, and how they relate to LIM bots. Simply put, interactive agents are software applications that interact with users in ordinary conversational language, utilizing any form of two-way text messaging.

This definition highlights a few key points. For example, an interactive agent is...well, *interactive.* This may seem obvious, but it's an important distinction because many messaging-based bots are not interactive or are only minimally so. Some bots exist purely to send outbound alerts or notifications, and some accept incoming user queries, but ignore the content, always responding in the same way. While such simple bots offer some utility, an interactive agent offers considerably more, leveraging the two-way capabilities of text messaging systems. An interactive agent is also conversational, employing natural language rather than relying upon a narrowly defined set of commands. What does support for conversational language entail? A well designed interactive agent allows for variation in the phrasing of user requests, accommodates abrupt subject changes, and uses conversational context to infer meaning where possible, freeing the user from needless repetition. An interactive agent can also take the lead in a conversation, prompting the user for additional information, offering a suggestion, or guiding the user through a multi-step process.

As a class of applications, interactive agents are not tied to any one messaging network or protocol. Given the ubiquity and the minimal resource requirements of text messaging, an interactive agent may be deployed virtually anywhere, providing an accessible user interface on practically any combination of network (wired, wireless, Internet, intranet, proprietary); client (IM, chat, email, Web browser); and device (computer, PDA, text pager, phone).

So, where do LIM bots fit in this picture? An interactive agent is an LIM bot only if it's deployed in a Lotus Instant Messaging (Sametime) environment, of course. Conversely, an LIM bot is an interactive agent only if it is interactive and offers some level of support for conversational language. We feel that most, if not all LIM bots should be designed to meet this definition—in our experience, the added utility that comes from interactivity and the enhanced usability that comes from conversational language support make for a much better bot.

Why should you consider developing an interactive agent? We'll answer that question in two different ways. First, we'll discuss some types of applications for which interactive agents are particularly well-suited. Then we'll discuss some of the unique advantages of interactive agents as compared to other types of software.

Interactive agent applications

The article Building your own Sametime bots, Part 1 includes a good list of specific bot applications. Rather than add to that list, we'll examine some high-level categories in which ActiveBuddy and its customers have developed numerous interactive agents, both employee-facing and customer-facing. This is by no means an exhaustive list, but it should give you a good idea of how interactive agents can be used to solve real-world business problems.

Categories within the enterprise include:

- *Employee self-service*
  Companies are always looking for ways to increase operational efficiency and to reduce costs, particularly when the economy is slow. Employee self-service software is designed to make employees more self-sufficient, reducing the resources a company must allocate to functions, such as human resources and IT support. Whether deployed over LIM or in a Web browser, interactive agents are ideally suited for self-service applications, providing fast, friendly assistance and requiring little or no training to use.
- *Productivity tools*
  Used especially in companies where text messaging is already pervasive, interactive agents are a great way to offer employees instant access to a wide array of useful information and services. Rather than opening a book, launching an application, or navigating through a Web or intranet site, an employee can simply fire off a question to an interactive agent that's always present in his or her contact list. Common applications include company-specific reference materials, such as the corporate directory and policy guide; general reference materials, such as dictionaries and encyclopedias; collaboration tools, such as calendars and polls; and utilities, such as conversion and translation.
- *Extending/enhancing existing applications and processes*
  Because text messaging operates in real time on virtually any type of wired or wireless device, interactive agents offer a simple, but extremely powerful way to extend the functionality and the “reach” of existing software applications. Many types of applications can benefit from integration with an interactive agent, but the most obvious candidates are automation, collaboration, or process management applications that require timely user reactions to external events because an interactive agent can react to an application event by proactively initiating a session with a user. In similar fashion, an interactive agent can be developed from scratch to streamline a business process previously executed without the aid of software tools.

Customer-facing categories include:

- *Marketing*
  Interactive agents are a great way of engaging consumers and encouraging them to interact with a brand. A marketing-oriented interactive agent can live on a Web site or on a messaging network and may be designed to inform, entertain, or project a distinctive personality—or all of the above. ActiveBuddy's corporate and agency customers have found their marketing-oriented interactive agents to be extremely effective, often substantially more so than other forms of online marketing.
- *Customer service*
  Just as interactive agents are used for employee self-service within the enterprise, they can be used to offer various forms of customer service, greatly reducing the amount of traffic routed to human support representatives. A customer service interactive agent may be scripted to answer frequently asked questions, or it may provide a conversational front-end to a knowledge base or help desk solution. Because it operates in a text messaging environment, an interactive agent can seamlessly escalate customers to live human agents as needed.
- *Information services*
  A company that deals in information (for example, news, data, or business intelligence) may find that interactive agents provide an excellent vehicle for delivering its product to customers. Users can request information on an as-needed basis in ordinary conversational language. Alternatively, information can be pushed to users when appropriate. Using an interactive agent for information delivery provides the same benefits as using an interactive agent to extend an application: The interactive agent can interact with users in real time across a range of devices and networks.

Benefits of interactive agents

Regardless of application, all interactive agents share a number of unique benefits. Interactive agents owe most of their advantages to two key properties: their support for conversational language and the fact that they operate within text messaging environments.

Because they employ conversational language, interactive agents require little or no user training. Users don’t need to memorize commands or become adept at navigating a structured user interface because they converse with an interactive agent the same way they would converse with another person, using natural language. Interactive agents are also efficient and accommodating. Like a human conversational partner, a well-designed interactive agent is sensitive to conversational context, while retaining the ability to handle abrupt subject changes. By drawing inferences from the content of previous messages, an interactive agent can dramatically reduce the effort required of the user. Despite its context awareness, however, an interactive agent is essentially "modeless." At any given time, regardless of context, a user can typically access any of the agent's services with a single request.

In addition, interactive agents lend themselves to rapid improvement and refinement. From a developer's perspective, interactive agents inherently provide a wealth of feedback on what is working and what isn't. Unlike a traditional desktop application (which provides no direct feedback) or a Web application (whose logs often give little insight into user intent), an interactive agent logs human-readable session transcripts, providing a comparatively transparent view of users' interactions with the application. Questions that weren't answered (or that were answered incorrectly) can be isolated automatically, allowing developers to address problems and to make enhancements quickly.

Because they operate in text messaging environments, interactive agents are convenient for users to access from virtually anywhere. Interactive agents live on the same text messaging networks that users depend on for person-to-person communications. Users often access these networks from desktop computers both at home and at work and increasingly from various types of mobile devices as well. By making data and applications available over the same networks, interactive agents can greatly extend the reach of a company's information assets and business processes. Interactive agents also enable advanced real-time collaboration and workflow management. Because they can take advantage of advanced messaging features like presence and awareness, interactive agents excel at coordinating processes involving multiple people or automated processes requiring time-sensitive human participation. And interactive agents are easy to deploy. Leveraging existing networks and client software, interactive agents typically require no download or installation on the part of the user.

Last but not least, it's worth noting that people just seem to *enjoy* using interactive agents. Whether this affinity derives from interactive agents' accessibility, their natural mode of interaction, or some combination thereof, it's a phenomenon we've observed consistently over the years.

Building an LIM interactive agent

Let's suppose that you want to develop an interactive agent to solve a problem for your business or for one of your clients. You may now ask why you should consider developing your interactive agent on the BuddyScript platform. Because you're reading *LDD Today,* the odds are that you're a Lotus developer and that you're primarily interested in deploying your interactive agent over Lotus Instant Messaging. And if you've read either of the "Building your own Sametime bots" articles, you already know that the LIM toolkits provide you with the low-level access you need to deploy bots in an LIM environment.

BuddyScript benefits

BuddyScript lets you take a high-level approach to interactive agent development. BuddyScript Server handles virtually all of the low-level "plumbing" and gives you a host of specialized tools for building effective interactive agents. This means you can devote your efforts to designing and implementing your specific application. Benefits of BuddyScript's high-level approach include:

- *Approachability*
  The BuddyScript language, although offering a rich set of powerful features, is syntactically simple and specialized for developing interactive agents. In fact, a basic script written in BuddyScript has more in common with a movie script than it does with a program written in Java or C. Consequently, a subject matter expert with little or no software experience can quickly become a productive member of an interactive agent development team.
- *Rapid development*
  Because you don't have to worry about low-level details, you write a lot less code and therefore, spend a lot less time coding and debugging.
- *Maintainability*
  Writing less code means less code to maintain, which makes it easier to fix bugs and to implement changes down the road.
- *Better user experience*
  Let's face it—implementing the conversational language support that makes interactive agents so much more compelling and effective than "mere bots" is no trivial feat. And although we haven't discussed it much in this article, presenting information within the constraints of a messaging environment (pure text, strict

message length limits, and so on) can also be a significant challenge. The BuddyScript platform has been designed specifically to solve these problems. Without BuddyScript, you need to do a lot more work to get the same end result (or more likely, settle for a lesser result).

Another reason to consider the BuddyScript platform is if you think you may want your interactive agent to run in one or more additional (non-Lotus) messaging environments, either now or down the road. BuddyScript takes a "write once, deploy everywhere" approach. Your interactive agent code for LIM will run with no modification in virtually any text messaging environment, including public instant messaging networks (AIM, MSN, Yahoo!, ICQ), other private IM networks (based on Jabber, SIP, and so on), Web chat systems, email, and a host of wireless messaging options (mobile IM, SMS/MMS, RIM, WAP).

Finally, the BuddyScript platform has proven itself to be scalable and reliable. SmarterChild and other BuddyScript-based interactive agents on public IM networks have routinely handled hundreds of thousands of users and millions of messages per day, staying up without interruption for months at a time.

Downloading the BuddyScript SDK

Now it's time to leave the abstract discussion behind and to start getting our hands dirty. First, we'll see how easy it is to get an interactive agent up and running on LIM. Then in Part 2 of this article series, we'll conclude by taking a look at a simple interactive agent that demonstrates a number of key BuddyScript features.

You may want to follow along at home as we walk through these examples. If so, now would be a good time to download and install the free BuddyScript SDK. The SDK includes the BuddyScript IDE, BuddyScript Server Developer Edition, documentation, and sample code. BuddyScript Server DE is full-featured, but limited with respect to usage. Specifically, each interactive agent can talk to just five named users (the first five screen names who contact it).

To get the SDK:

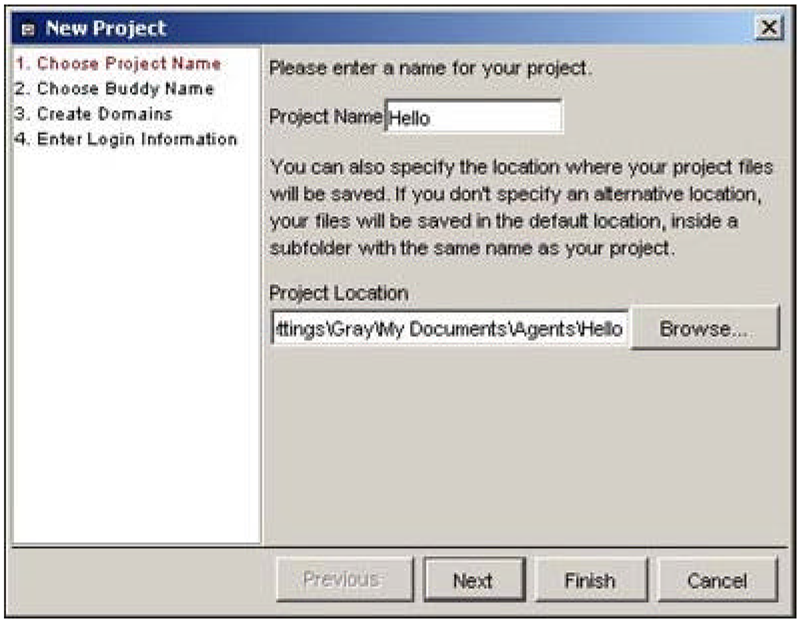
1. Visit the BuddyScript home page.
2. Sign up for a free account.
3. Go to the Downloads section of the site and get the BuddyScript SDK. You should download and install the latest build.

You can also download the sample files for this article from the Sandbox.

Setting up the BuddyScript SDK

To illustrate the benefits of BuddyScript's high-level approach to interactive agent development, we start from scratch and aim to have an interactive agent running on LIM and the Web in just five minutes. Launch the BuddyScript IDE from the BuddyScript SDK program group in the Windows Start menu.

When the BuddyScript IDE launches for the first time, the Tutorial project is automatically opened by default. Because you want to create your own project instead, close the Tutorial project by choosing Project - Close Project in the main IDE window, and clicking OK when prompted to close open project files. Now that you have a clean slate, open the New Project wizard by choosing Project - New Project. In the wizard's first screen, specify a name for our project. Call the project "Hello."



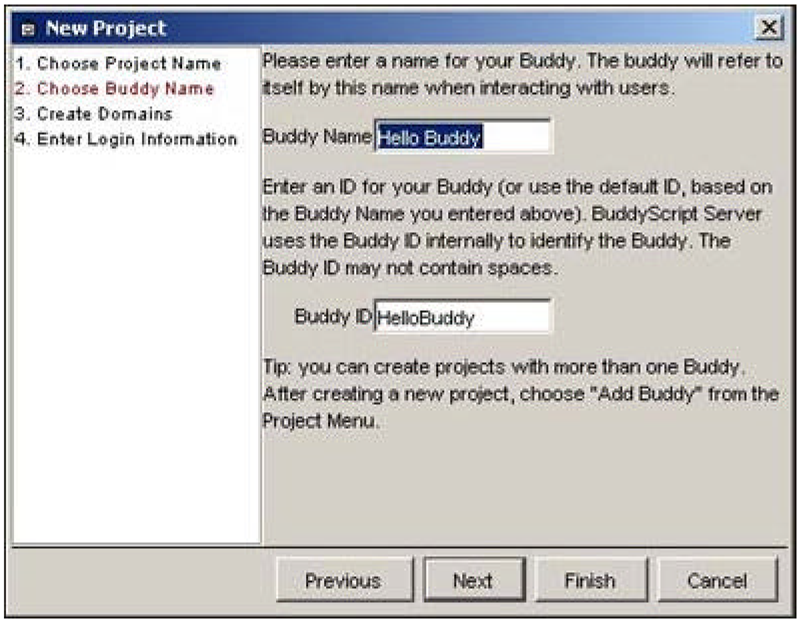
Please enter a name for your project.

Project Name: Hello

You can also specify the location where your project files will be saved. If you don't specify an alternative location, your files will be saved in the default location, inside a subfolder with the same name as your project.

Project Location: ttings\Gray\My Documents\Agents\Hello

Next, you have the option of specifying the Buddy Name (the name by which the interactive agent refers to itself) and the Buddy ID (which BuddyScript Server uses as an internal identifier). Accept the default values, which are based on your project name:



Please enter a name for your Buddy. The buddy will refer to itself by this name when interacting with users.

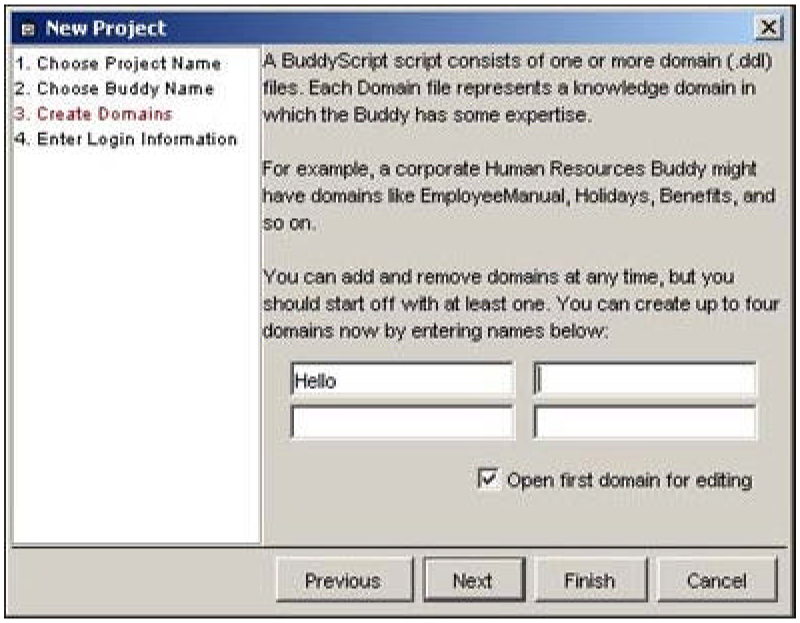
Buddy Name: Hello Buddy

Enter an ID for your Buddy (or use the default ID, based on the Buddy Name you entered above). BuddyScript Server uses the Buddy ID internally to identify the Buddy. The Buddy ID may not contain spaces.

Buddy ID: HelloBuddy

Tip: you can create projects with more than one Buddy. After creating a new project, choose "Add Buddy" from the Project Menu.

Now you are given the opportunity to create one or more domains (areas in which the interactive agent will have knowledge or functionality). Again, accept the default setting, which is to create a single domain with the name of the project. (You can always create additional domains later as needed.)



A BuddyScript script consists of one or more domain (.ddl) files. Each Domain file represents a knowledge domain in which the Buddy has some expertise.

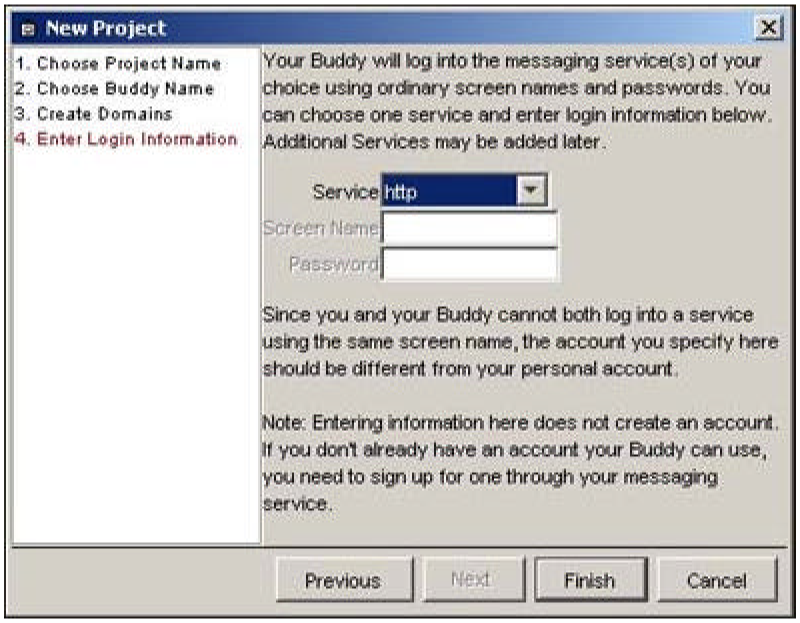
For example, a corporate Human Resources Buddy might have domains like EmployeeManual, Holidays, Benefits, and so on.

You can add and remove domains at any time, but you should start off with at least one. You can create up to four domains now by entering names below:

Hello

[✓] Open first domain for editing

Finally, you are prompted to choose a service on which to run the interactive agent. An agent can run on multiple services, but the New Project wizard only lets you specify the first one. Further, in the interest of simplicity, it only lists services requiring minimal configuration—namely, the public IM networks and the HTTP service, which makes the interactive agent accessible via a Web browser (among other forms of HTTP access). Because LIM/Sametime is not an option within the wizard, you have to specify it later. For now, to demonstrate the ability to run on multiple services, choose HTTP, then click Finish to close the New Project wizard.



Your Buddy will log into the messaging service(s) of your choice using ordinary screen names and passwords. You can choose one service and enter login information below. Additional Services may be added later.

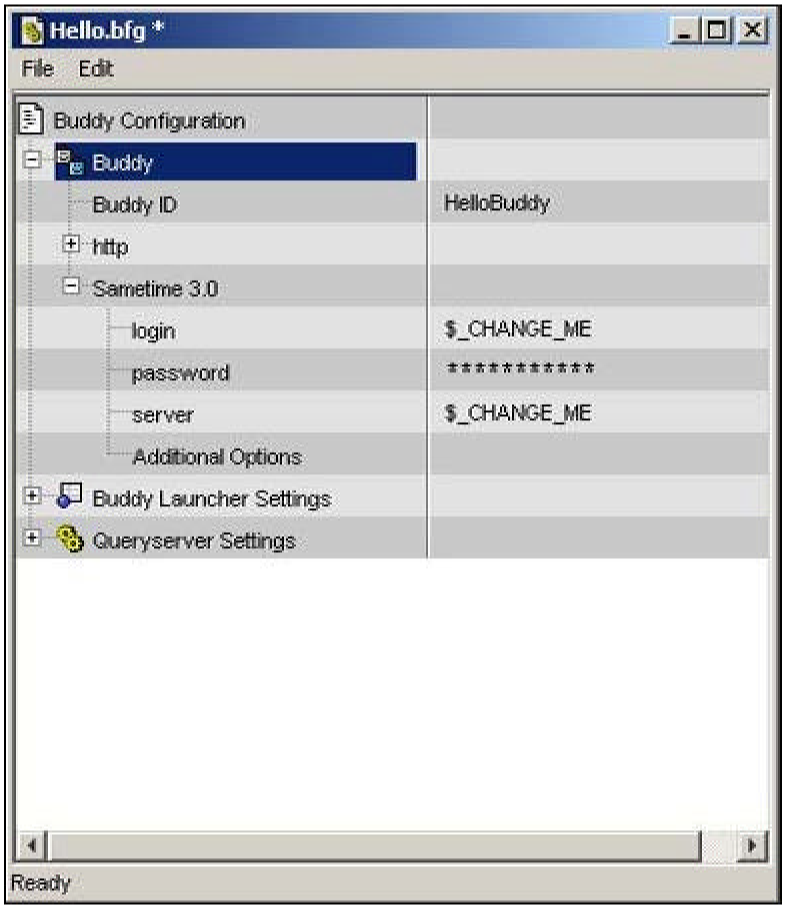
Service: http
Screen Name:
Password:

Since you and your Buddy cannot both log into a service using the same screen name, the account you specify here should be different from your personal account.

Note: Entering information here does not create an account. If you don't already have an account your Buddy can use, you need to sign up for one through your messaging service.

After you complete the wizard, the BuddyScript IDE creates your new project and opens the Hello domain for editing. Before you get cracking on the script, however, let's add the Sametime service to the project. To do so, open the configuration editor by double-clicking the Buddy Configuration icon in the main IDE window, which appears in the top left corner of the screen.

In the editor, add a service by right-clicking the Buddy element and choosing the desired component from the context menu that appears. In this case, add Sametime 3.0, though you may instead add Sametime 2.0 (or sometime in the future, a later version of LIM) if that's what you're running. After you add the component, specify the address for the Sametime 3.0 server and the name and password with which you want the interactive agent to log in. (Note that you need to create a login account for your interactive agent before trying this at home, using the same process you would use to create an account for any new LIM user.) To save your changes, choose File - Save in the configuration editor window, then close the editor:



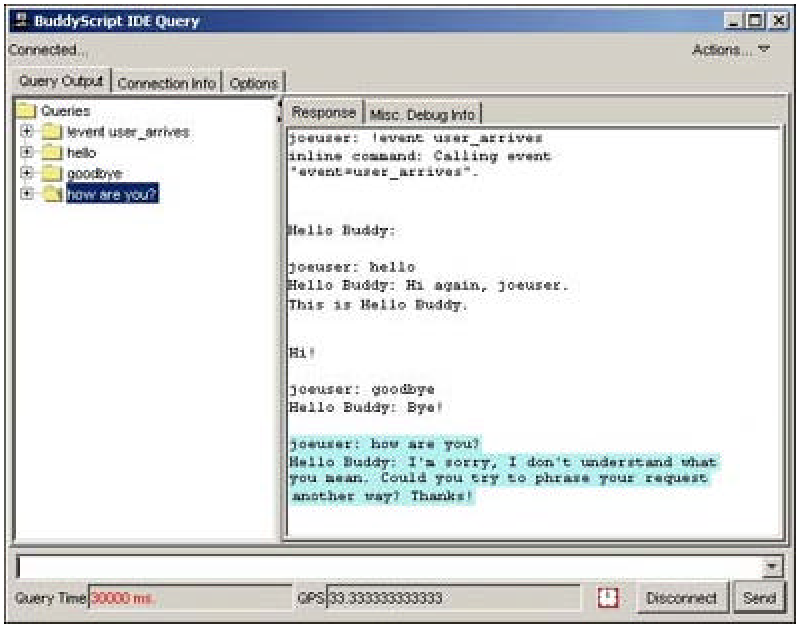
Buddy Configuration	
Buddy	
Buddy ID	HelloBuddy
http	
Sametime 3.0	
login	$_CHANGE_ME
password	************
server	$_CHANGE_ME
Additional Options	
Buddy Launcher Settings	
Queryserver Settings	

You're now ready to start scripting. In the editor window for the Hello domain, type the following, and then save your work by choosing File - Save in the editor window:

```
+ hello
   - Hi!

+ goodbye
   - Bye!
```

With these few lines, you have just created a complete (albeit not very interesting) interactive agent. As you may already have inferred, the lines beginning with plus symbols (called pattern definitions) represent things the user might say; the lines beginning with minus symbols (called outputs) represent the interactive agent's responses. A block of code consisting of one or more pattern definitions and an indented set of instructions (which typically, but not always, includes outputs) is called a *routine.*

You can test your script here in the BuddyScript IDE by choosing Project - Test in the main IDE window or by pressing F5. Doing so starts BuddyScript Server, compiles our script, and lets you talk to your interactive agent in the Query window, which runs across the bottom of the screen. Because your script is so simple, the interactive agent isn't much of a conversationalist—it will of course reply to "hello" and "goodbye," but will say it doesn't understand if you type anything else.



```
joeuser: !event user_arrives
inline command: Calling event
"event=user_arrives".


Hello Buddy:

joeuser: hello
Hello Buddy: Hi again, joeuser.
This is Hello Buddy.


Hi!

joeuser: goodbye
Hello Buddy: Bye!

joeuser: how are you?
Hello Buddy: I'm sorry, I don't understand what
you mean. Could you try to phrase your request
another way? Thanks!
```

Before you launch your interactive agent, let's extend the script just a bit to give you a better idea of how BuddyScript's natural language pattern matching works. Among the things to notice:

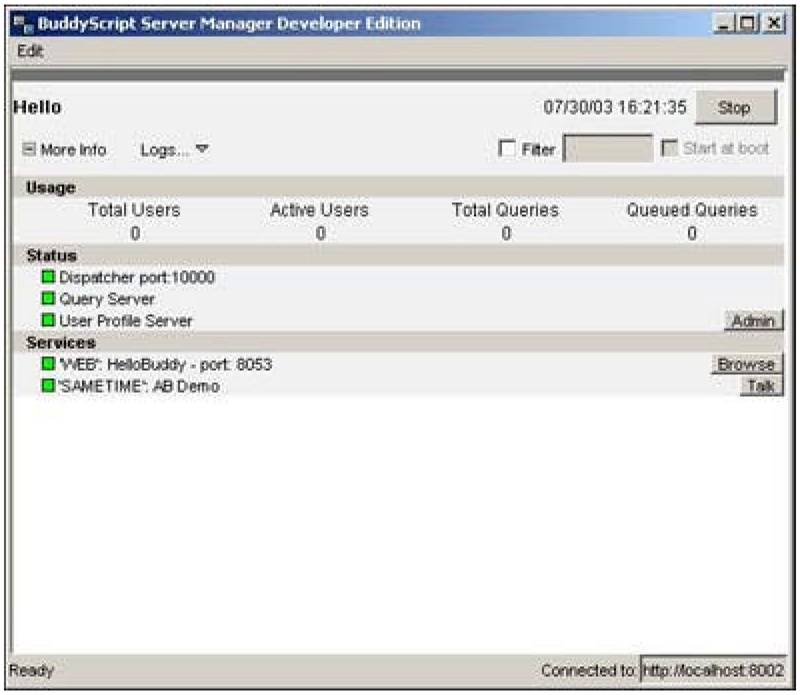
- Alternations, using the (|) syntax, and optional elements, using the [] syntax, have been introduced.
- Subpatterns have been defined, allowing snippets of pattern code to be defined and referenced within a routine.
- Each routine now has multiple pattern definition lines, representing substantially different ways in which a user may express the same basic meaning. (You could, of course, accomplish the same objective using alternations on a single pattern definition line, but doing so would be unwieldy.)
- Each routine now has multiple outputs as well. When multiple outputs are defined in a single block like this, the interactive agent chooses one of the outputs at random each time the routine executes.

Collectively, these changes make our interactive agent more flexible with respect to recognizing user inputs and more interesting with respect to its responses.

```
+ (hello|hi|hey|howdy|yo)
+ good (morning|afternoon|evening)
  - Hello!
  - Hi!
  - Hey, SYS.User.ScreenName!

subpattern You
+ (you|ya|u)

subpattern See
+ (see|c)

subpattern Later
+ (later|l8r)

+ (goodbye|[good] bye)
+ =See =You [=Later]
+ =Later
  - Bye!
  - See you!
  - Later, SYS.User.ScreenName!
```

OK, enough scripting for now—we'll get a much deeper look at the BuddyScript language in Part 2 of this article series when we step through the code for our sample project. Let's go ahead and get this interactive agent up and running on LIM/Sametime and on the Web.

Launching an interactive agent is as simple as choosing Project - Start in the main IDE window or pressing F6. Doing so opens the BuddyScript Server Manager, which provides an interface for stopping, starting, and monitoring interactive agents. Shortly after the BuddyScript Server Manager opens, you see a number of status indicators, representing the various components of BuddyScript Server and the services on which the interactive agent is being launched. If all goes smoothly, these indicators will all be green within a few moments.



You can now talk to your interactive agent both in the LIM/Sametime client and in a Web browser. In the client, add the interactive agent's screen name to your contact list and talk to it as we would talk to a friend or colleague. To get to the browser-based interface, click the Browse button in the BuddyScript Server manager and follow the link for the Web UI in the page that opens.

Now we're ready to begin!

This article introduced you to interactive agents and to the BuddyScript development tool. We've explained how to download and use the BuddyScript SDK and demonstrated a simple interactive agent. In Part 2, we dive deeper into the BuddyScript SDK development environment, stepping through the process of creating an LIM interactive agent. Stay tuned!

ABOUT THE AUTHOR
Gray Norton is a Senior Product Manager for ActiveBuddy, where he oversees the company's BuddyScript product line. He has been in the software indistry since the early 1990's and has played key product management roles for several companies in the interactive media space, including Electrifier, Motion Factory, MetaCreations and Ray Dream. Gray is a graduate of Stanford University's Symbolic Systems Program.