Chapter 5

Tiers of Functionality

Abstract

Chapter 5 of Library Technology Reports *(vol. 49, no. 8), "Streamlining Information Services Using Chatbots," explores options in functionality, with advice on the coding required and the benefits. Chatbot interfaces take a wide range of forms, from simple text boxes to talking characters that pass questions to other resources.*

Having discussed the nuts and bolts of coding in AIML, it's appropriate to take a step back and think about the way in which the chatbot will integrate into existing library services and resources. The first decision to be made is exactly you want the chatbot to do. The technology behind the chatbot can be expanded from a simple beginning, but it's helpful to think of bots as having at least three tiers of functionality and to decide at the outset to which of these tiers you aspire.

The first functional tier is a bot programmed to respond to a specific and limited group of questions with a predetermined set of responses. Think of this as a virtual FAQ resource. The creator considers the questions most likely to be asked and programs the full responses into the chatbot's code. This is a good place for a library bot to start, and indeed Emma's first iteration was designed to respond to fifteen questions about the Mentor Library and its services.

You are likely to already have a FAQ or "How do I" section on your website or in a library brochure that will provide you with an initial set of questions and responses. Using these questions, create a set of categories and their associated template responses. It's a good idea to think of simple category patterns (like "hours," "fines," or "location") rather than using full questions at this point. For these simple patterns, compose fully formed template responses. Your next step will be to consider all the possible ways you can pose questions about each category pattern. Code these more complete question patterns as efficiently as possible using wildcards, and refer each back to the appropriate basic pattern categories using `<srai>` templates. You may wish to work in separate code files for each basic pattern to help you keep track of your questions and responses. Table 5.1 provides an example of this process.

We talked about the efficiency of `<srai>` tags as a means to shorten initial coding in chapter 3. But you can also think of `<srai>` as a way to limit the number of changes you will need to make later if the responses to your questions were to change. In the example in table 5.1, a change in library hours would necessitate only a single change in one line of code to update the responses to all the associated questions.

This is really the extent of the virtual FAQ tier of chatbot. As interactive and informative as many commercial chatbots are, this is really the core of what they are all about: giving information about a particular institution's products and services in response to inquiries about those products and services.

To expand upon the virtual FAQ tier, one need only access additional prepared resources that are already available on your website (or others). This is as simple as embedding HTML coded hyperlinks into your template responses. For those unfamiliar with HTML, here's a quick example of the process.

Let's say you want to give the location of your library in a template response, but also provide a link to a map. Compose the template response, then select the text you wish to link to another website. Surround this text with `<a>  </a>` tags. Complete the information in the initial tag with the URL of the target

Basic FAQ question	What are the library’s hours?	

First AIML file (basic pattern)	`<?xml version="1.0" encoding="UTF-8">`
	`<aiml version="1.0">`
	
	`<category>`
	`<pattern>LIBHOURS</pattern>`
	`<template>We are open from nine to five, Monday through Friday.`
	`</template>`
	`</category>`
	
	`</aiml>`	

Expanded question set and associated wildcard formats	What are the library’s hours? What are library hours? What are the library hours on Saturday? When is the library open? Are you open on Sunday?	* LIBRARYS HOURS * LIBRARY HOURS * LIBRARY HOURS * * OPEN * OPEN *
	Note that we didn’t use the term hours alone with wildcards at this point in case we end up with questions using the word hours that are not referring to the library’s hours of opening. An example would be “How many hours before I can pick up my hold?”	

Resulting AIML file	`<?xml version="1.0" encoding="UTF-8">`
	`<aiml version="1.0">`
	
	`<category>`
	`<pattern>LIBHOURS</pattern>`
	`<template>We are open from nine to five, Monday through Friday.`
	`</template>`
	`</category>`
	
	`<category>`
	`<pattern>* LIBRARYS HOURS</pattern>`
	`<template><srai> LIBHOURS </srai></template>`
	`</category>`
	
	`<category>`
	`<pattern>* LIBRARY HOURS</pattern>`
	`<template><srai> LIBHOURS </srai></template>`
	`</category>`
	
	`<category>`
	`<pattern>* LIBRARY HOURS *</pattern>`
	`<template><srai> LIBHOURS </srai></template>`
	`</category>`
	
	`<category>`
	`<pattern>* OPEN</pattern>`
	`<template><srai> LIBHOURS </srai></template>`
	`</category>`
	
	`<category>`
	`<pattern>* OPEN *</pattern>`
	`<template><srai> LIBHOURS </srai></template>`
	`</category>`
	
	`</aiml>`	

Table 5.1
Example of the process of coding a basic library question.

website like this: `<a href="URL of website">`. You may also wish to have the target website open in a new window. In this case add to the initial `<a>` tag the following text: `target="_blank"` (see table 5.2).

The same kind of linking can be done with any existing resources from your website to give access to full text of procedural, policy, or instructional documentation that expands upon the information provided in your answer template.

The highest tier of functionality, and one which

Basic FAQ question	Where is the library?

First AIML file	`<?xml version="1.0" encoding=”UTF-8”>`
	`<aiml version="1.0">`
	
	`<category>`
	`<pattern>LIBLOC</pattern>`
	`<template>The library is located at 60 S. Main St., Hometown, Ohio 44312. Our phone number is 330-643-9157. You may also wish to view a map of our location.</template>`
	`</category>`
	
	`</aiml>`

First AIML file with HTML link	`<?xml version="1.0" encoding="UTF-8">`
	`<aiml version="1.0">`
	
	`<category>`
	`<pattern>LIBLOC</pattern>`
	`<template>The library is located at 60 S. Main St., Hometown, Ohio 44312. Our phone number is 330-643-9157. You may also wish to view a`
	`<a href="http://maps.google.com//maps?q=60+s+main+st+hometown+oh+44312" target="_blank"> map of our location. </a></template>`
	`</category>`
	
	`</aiml>`

Table 5.2
Example of adding an HTML link to chatbot code.

commercial bots are not usually interested in achieving, is the ability of a bot to pass queries to other resources. This highlights the difference we addressed earlier in the overall goals of our library bots as compared to those of most commercial enterprises. Our goal is not to simply publicize our services, products, and resources (though we definitely do want to do this). Library chatbots also seek to provide our users access to the answers to their questions from whatever resource is most efficient and best able to provide these answers. We can achieve this goal by passing information queries directly to our library catalogs, to magazine and journal article index databases, to dictionary or encyclopedia websites, and to a plethora of other resources.

This tier of functionality requires a fuller articulation of the coding between the HTML page in which the bot is displayed and the AIML code that informs its interaction with users. To understand this, we'll first need to look at how the bot is placed into a web page.

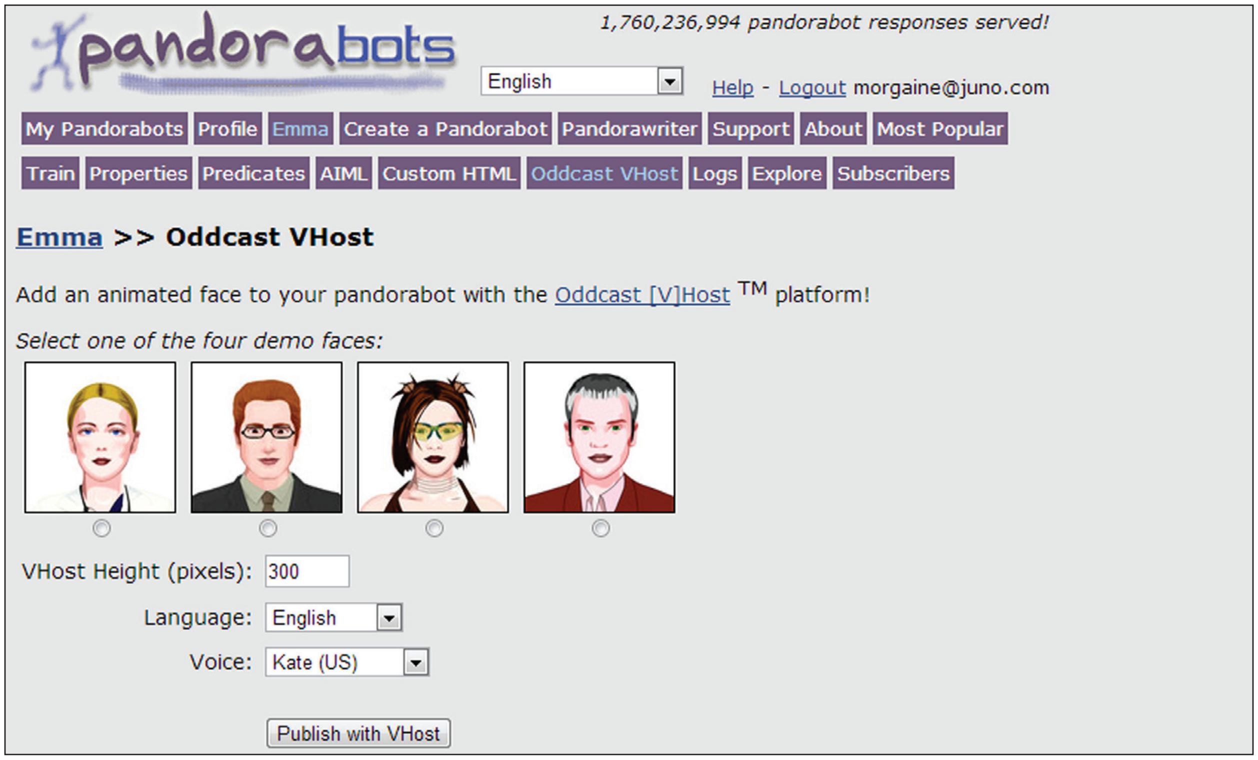
Text-Only Interface

It is perhaps best to start a discussion of embedding your chatbot in a web page by providing a template for creating a basic text-only interface. Such an interface is composed of a form to submit the user's question. In Pandorabots, the form must contain the string !CUSTID! to identify the bot that is posing the question. This string should appear on a line by itself. Your bot's response will be called by the string `!OUTPUT!`. This string need not appear by itself, so you can use spacing and punctuation to identify the response for users.

```
<html>
<body onLoad="document.form.input.
   focus();">
Text Only Chatbot
<form method="POST" name="form">
!CUSTID!
You ask: <input type="TEXT"
   autocomplete="off" name="input"
   size="30">
</form>
Reply: !OUTPUT!
</body>
</html>
```

Talking Avatar

While it is true that the above text-only interface encompasses the functional entirety of the chatbot concept, creating a more visually (and auditorially) interesting (and entertaining) interface is both simple and desirable.


1,760,236,994 pandorabot responses served!
English
Help - Logout morgaine@juno.com
My Pandorabots | Profile | Emma | Create a Pandorabot | Pandorawriter | Support | About | Most Popular
Train | Properties | Predicates | AIML | Custom HTML | Oddcast VHost | Logs | Explore | Subscribers
Publish with VHost

Emma >> Oddcast VHost

Add an animated face to your pandorabot with the Oddcast [V]Host ™ platform!

Select one of the four demo faces:

VHost Height (pixels): 300

Language: English

Voice: Kate (US)

Figure 5.1
Oddcast VHost screen for adding a talking avatar to a Pandorabots chatbot.

The first step to adding a talking avatar to a Pandorabots' chatbot is to select the Oddcast VHost button at the top of the Pandorabots screen. This will bring you to a page similar to that shown in figure 5.1, where you can select one of four demo faces. After making your image selection, assign a height in pixels, select a language and a voice from the drop-down lists, and click Update VHost Settings. Leave the selection Customized HTML Skin selection as Plain Text for the time being.

After you update your settings, the system will assign you a URL where you can view and test the avatar. Click this link to view your results.

Test your interface a bit. Play with the settings for the size, face, and voice of your bot to find a combination you like.

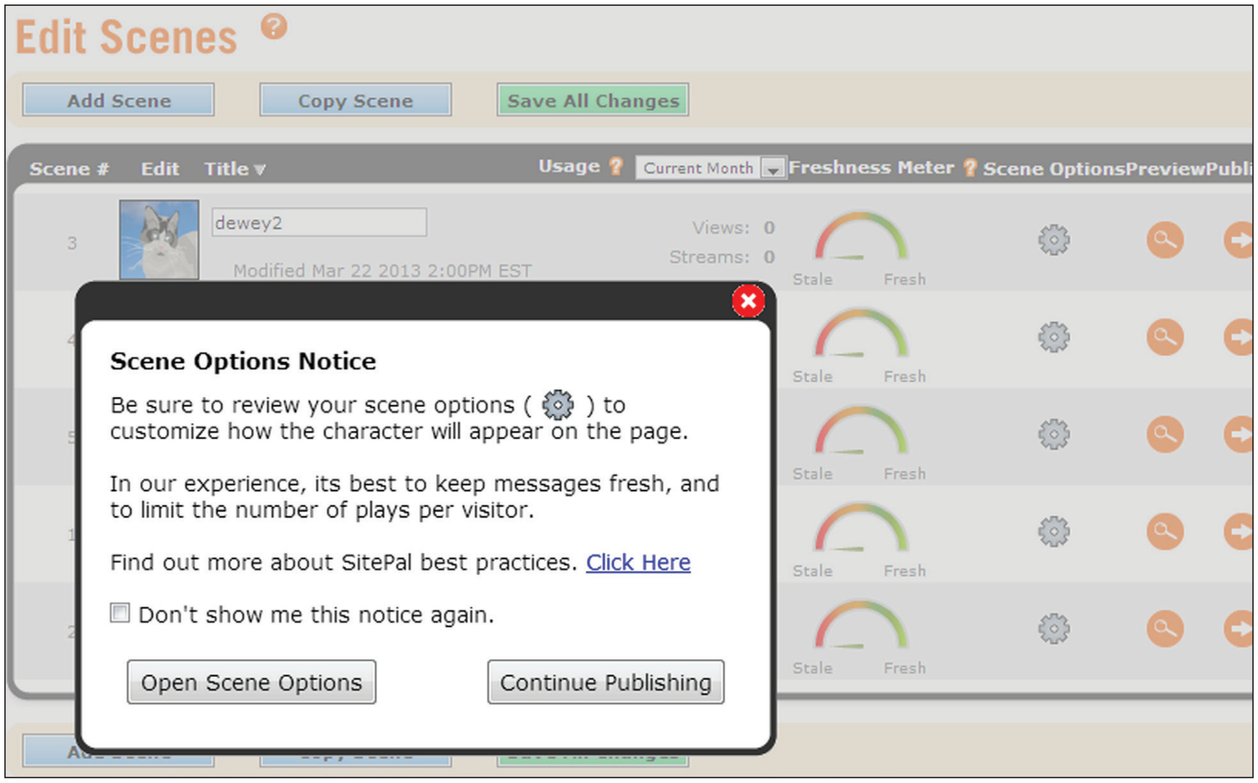
Now let's look at the code that is embedding this bot in your page. If you view the page source for figure 5.1, you'll see that it is basically a frameset page comprised of two separate frames: a "vhost" or avatar frame, and an input frame:

```
<head>
<title>Oddcast Bot:</title>
</head>
<frameset rows="350,*"
   border="0"">
<frame name="vhost" src="/
   pandora/talk-host-
   oddcast?botid=b52b47062e341e19"
   scrolling="no" frameborder="no"
   noresize="noresize"></frame>
<frame name="input" src="/pandora/
   talk-input-oddcast?botid=b52b47
   062e341e19"></frame>
</frameset>
```

You could simply embed the demo frameset in an iframe within the desired web page for testing. But you can also further customize the frames to fine-tune your bot's appearance and performance. Further customization, however, requires subscription to the SitePal avatar service.

Once your subscription is active, you will need to bring three bits of information from SitePal. First, you will need your SitePal account number. You will also need the scene number and the configuration string from your SitePal Editor. To locate these, click the orange button with an arrow under the Publish option on your SitePal account page, then click Continue Publishing (see figure 5.2.)

On the next screen, select the Embed in a Web Page button. On the following screen, select Click Here to Embed, and then look for the code generated at the bottom of the subsequent page (see figure 5.3). This code will contain all the information you will need. Your account number will appear where you see yyyyyy in the code below. The scene number appears



Edit Scenes

Scene Options Notice

Be sure to review your scene options (⚙) to customize how the character will appear on the page.

In our experience, its best to keep messages fresh, and to limit the number of plays per visitor.

Find out more about SitePal best practices. Click Here

☐ Don't show me this notice again.

Figure 5.2
SitePal Editor showing Continue Publishing button. SitePal is a registered trademark of Oddcast Inc. Portions reproduced with permission from Oddcast.

where you see zzzzzzz below. Finally, the alphanumeric sequence in single quotes below represents the choices you've made in publishing your scene.

```
<script language="JavaScript"
   type="text/javascript" src=
   "http://vhss-d.oddcast.com/
   vhost_embed_functions_v2.php
   ?acc=yyyyyy8&js=1"></script>
   <script language="JavaScript"
   type="text/javascript">
   AC_VHost_Embed(yyyyyy,300,
   400,'',1,1, zzzzzzz, 0,1,0,'
   e12345b1c12345bc8d1234ae5bb1
   ef12',9);</script>
```

Armed with these three bits of information, you can begin to customize your frameset.

As with the text-only example, your custom frame will be composed of a frameset with two frame elements: the avatar and the input/output box. To create the frameset, you will first need to locate the botid number that is displayed within Pandorabots. After logging into Pandorabots, click the name of the bot with which you wish to work.

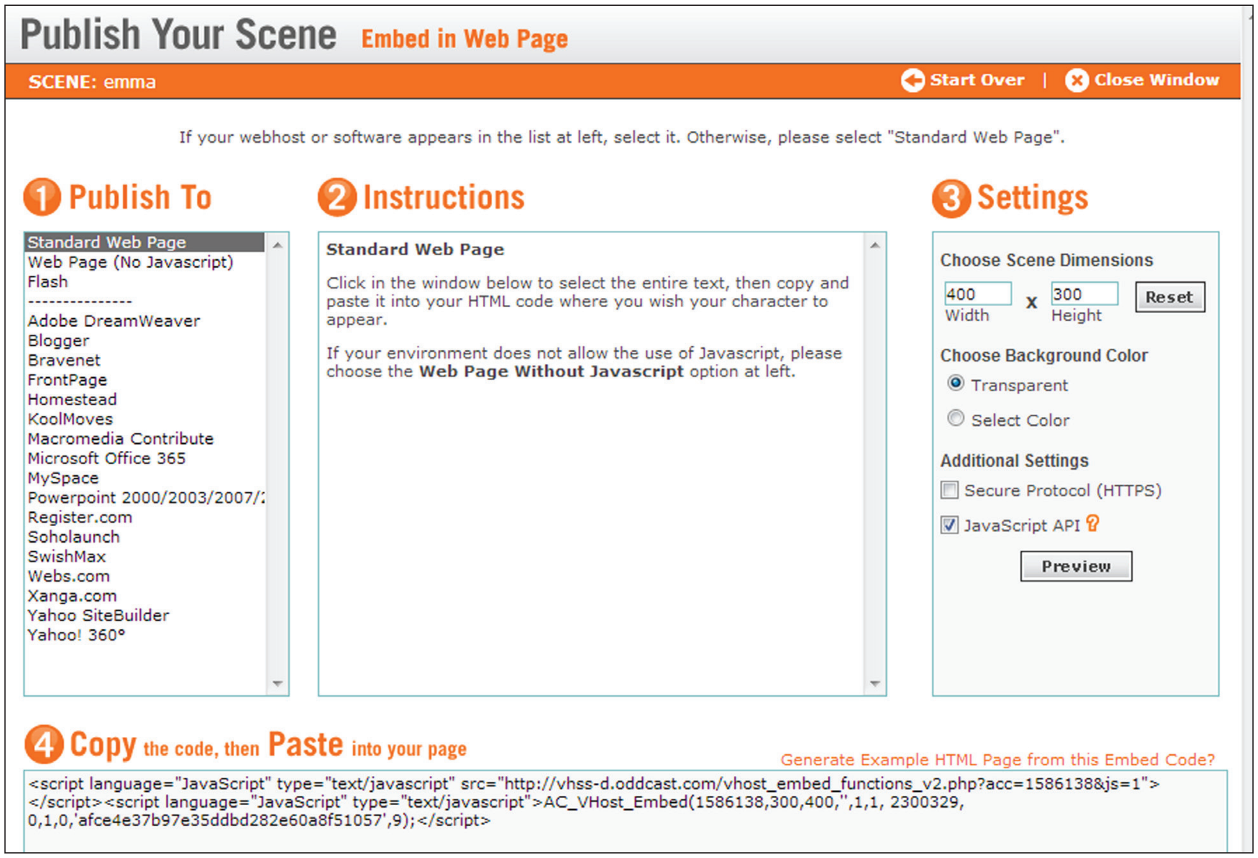
Now look at the URL at which your bot is published. You will see the botid there as the end of the URL string (see figure 5.4). Copy this botid sequence for use in the frameset you will be writing.

Now move to the Pandorabot Custom HTML editor and compose a frameset using the template below. In each frame source statement, replace the botid xxxxxxxxxx with the one you copied from your account.

```
<html>
<head><title>Bot</title>
</head>
<frameset border="0"
   frameborder="0"
   framespacing="0" rows="330,*">
<frame src="/pandora/talk?bot
   id=xxxxxxxxxx&skin=ivhost"
   name="vhost">
<frame src="/pandora/talk?botid=xx
   xxxxxxxx&skin=input&speak=true"
   name="input">
</frameset>
</html>
```

Assign the frameset a meaningful name and save it.

Now we need to compose the two frame source files. Let's start with the ivhost file. Let's have the



Publish Your Scene Embed in Web Page

SCENE: emma

Start Over | Close Window

If your webhost or software appears in the list at left, select it. Otherwise, please select "Standard Web Page".

1 Publish To

Standard Web Page
Web Page (No Javascript)
Flash

Adobe DreamWeaver
Blogger
Bravenet
FrontPage
Homestead
KoolMoves
Macromedia Contribute
Microsoft Office 365
MySpace
Powerpoint 2000/2003/2007/2
Register.com
Soholaunch
SwishMax
Webs.com
Xanga.com
Yahoo SiteBuilder
Yahoo! 360°

2 Instructions

Standard Web Page

Click in the window below to select the entire text, then copy and paste it into your HTML code where you wish your character to appear.

If your environment does not allow the use of Javascript, please choose the **Web Page Without Javascript** option at left.

3 Settings

Choose Scene Dimensions

400 Width x 300 Height Reset

Choose Background Color

[✓ Transparent]

Select Color

Additional Settings

Secure Protocol (HTTPS)

[✓ JavaScript API]

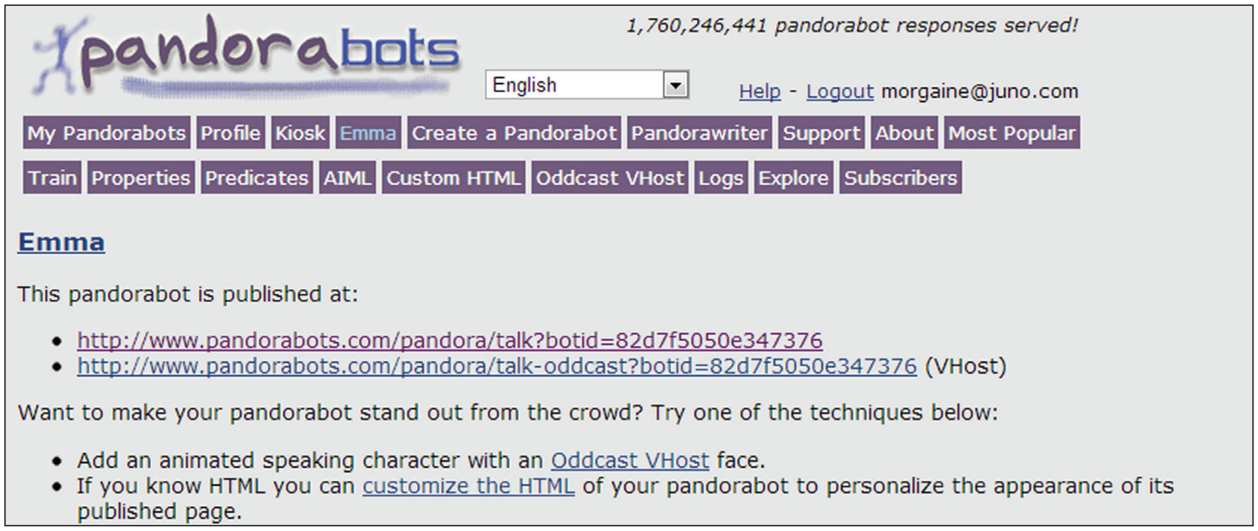
Preview

4 Copy the code, then Paste into your page

Generate Example HTML Page from this Embed Code?

```
<script language="JavaScript" type="text/javascript" src="http://vhss-d.oddcast.com/vhost_embed_functions_v2.php?acc=1586138&js=1">
</script><script language="JavaScript" type="text/javascript">AC_VHost_Embed(1586138,300,400,'',1,1, 2300329,
0,1,0,'afce4e37b97e35ddbd282e60a8f51057',9);</script>
```

Figure 5.3
SitePal page showing code with the information required for further avatar customization. SitePal is a registered trademark of Oddcast Inc. Portions reproduced with permission from Oddcast.



pandorabots

1,760,246,441 pandorabot responses served!

English

Help - Logout morgaine@juno.com

My Pandorabots | Profile | Kiosk | Emma | Create a Pandorabot | Pandorawriter | Support | About | Most Popular

Train | Properties | Predicates | AIML | Custom HTML | Oddcast VHost | Logs | Explore | Subscribers

Emma

This pandorabot is published at:

- http://www.pandorabots.com/pandora/talk?botid=82d7f5050e347376
- http://www.pandorabots.com/pandora/talk-oddcast?botid=82d7f5050e347376 (VHost)

Want to make your pandorabot stand out from the crowd? Try one of the techniques below:

- Add an animated speaking character with an Oddcast VHost face.
- If you know HTML you can customize the HTML of your pandorabot to personalize the appearance of its published page.

Figure 5.4
Pandorabots page showing botid.

chatbot welcome the user and then be ready to respond to questions. Here's the HTML code file. Table 5.3 provides an explanation of the various elements.

```
<html>
<body
   style="background-color:white">
```

`<html>`	This is the standard HTML page beginning.
`<body style="background-color:white"">`	This assigns a background color to the frame.
`<script language="JavaScript" type=""text/javascript" src="http://vhss-d.oddcast.com/vhost_embed_functions_v2.php?acc=yyyyyyy&js=1&followCursor=1"></script>`	This JavaScript identifies the account from which the avatar will be displayed and sets the avatar to "watch" or "follow" the cursor.

`<script>`
`var vhostLoaded=false;`
`var vhostSpeaking=false;`
`function vh_sceneLoaded() {`
`sayText("Welcome to the Library. How can I help you today?",1,1,2);`
`vhostLoaded=true;`
`}`
`function vh_talkEnded() {`
`vhostSpeaking=false;`
`}`
`</script>`
	This script sets some parameters for the initial loading of the JavaScript. In this case, once the scene is loaded, the avatar greets the user and asks how it can help.

`<script language="JavaScript" type="text/javascript">AC_VHost_Embed(yyyyyy,300,400,'',1,1, zzzzzzz, 0,1,0,' e12345b1c12345bc8d1234ae5bb1ef12', 9);</script>`	This script reconfirms the account number and identifies the selected scene and display parameters.

`</body>`
`</html>`
	These tags close the Body and the HTML page.

Table 5.3
Explanation of elements in the code for the ivhost file.

```
<script language="JavaScript"
   type="text/javascript"
   src="http://vhss-d.oddcast.
   com/vhost_embed_functions_v2.p
   hp?acc=yyyyyyy&js=1&followCur
   sor=1"></script>
<script>
var vhostLoaded=false;
var vhostSpeaking=false;
function vh_sceneLoaded() {
sayText("Welcome to the Library. How
   can I help you today?",1,1,2);
vhostLoaded=true;
}
function vh_talkEnded() {
vhostSpeaking=false;
}
</script>

<script language="JavaScript"
   type="text/
   javascript">AC_VHost_
   Embed(yyyyyy,300,400,'',1,1,
   zzzzzzz, 0,1,0,'
   e12345b1c12345bc8d1234
   ae5bb1ef12',9);
   </script>
</body>
</html>
```

Next we need to compose the input frame source file. This frame contains the form in which users will type their questions and where the bot's answers will be displayed. Here's the code for the input frame file. Table 5.4 provides the explanation.

```
<html>
<head>
<script>
<!--
function sf(){document.f.input.
   focus();}
// -->
</script>
</head>
<body lang=en-US onload=sf()
   style="background-color:white">
<script language="JavaScript">
function sayit() {
if (parent.vhost) {
if (parent.vhost.vhostLoaded) {
parent.vhost.vhostSpeaking=true;
parent.vhost.sayText("<template
   context="tts"><response/></
   template>",1,1,2);
} else {
setTimeout("sayit()", 500);
}
}
```

`<html>` `<head>`	This is the standard HTML page beginning.
`<script>` `<!--` `function sf(){document.f.input.focus();}` `// -->` `</script>`	This script creates a function that will direct the cursor to the input box so that a user doesn't need to click in the box before starting to type.
`</head>` `<body lang=en-US onload=sf() style="background-color:white">`	The first tag closes the Head section of the HTML file; the second opens the Body section. Note that the background color of the page is set to white, and that an onload command tells the page to run the `sf()` function, placing the cursor in the input box once the page is loaded.
`<script language="JavaScript">` `function sayit() {` `if (parent.vhost) {` `if (parent.vhost.vhostLoaded) {` `parent.vhost.vhostSpeaking=true;` `parent.vhost.sayText("<template context="tts"><response/></template>",1,1,2);` `} else {` `setTimeout("sayit()", 500);` `}` `}` `}` `sayit();` `</script>`	This is the JavaScript that activates the bot's speech capability.
`<table>` `<tr>` `<td bgColor=#FFFFFF >`	These tags simply set up a table in which the form will be displayed. The last tag assigns the background color for the table cell to be `#FFFFFF` (white).
`<form name=f action="" method=post>`	The `<form>` tag sets up the form so the input will be submitted.
`!CUSTID!`	This is the all important `!CUSTID!` line that directs the inquiry to the correct bot.
`<p>Type your question in the box below, then strike the "Enter" key.</p>`	This is just a paragraph providing instructions for using the form.
`<p>Ask me a question!</font> <input size=40 name=input> </p>`	This paragraph directs the user to type the question and provides an input form element to hold it.
`<p>You Said: &nbsp;<em><template><input/></template></em></p>`	This paragraph redisplays the user's input text after the user hits the Enter key. The input is represented by the `<input/>` tag.
`<p>My reply: &nbsp;<em><b>!OUTPUT!</b></em></p>`	Here's a new paragraph to contain the bot's reply. The actual template response is represented by `!OUTPUT!`. This text is emphasized by the `<em>` (emphasis) and `<b>` (bold) tags.
`</form>` `</td></tr>` `</table>` `</html>`	These tags close the form, table data cell, table row, the table itself, and the HTML form.

Table 5.4
Explanation of elements in the code for the input frame file.

```
}
sayit();
</script>
<table>
<tr>
<td bgColor=#FFFFFF >
<form name=f action=""
   method=post>
```

```
!CUSTID!
<p>Type your question in the box
   below, then strike the "Enter"
   key.</p>
<p>Ask me a question!</font>
   <input size=40 name=input> </p>
<p>You Said:
   &nbsp;<em><template><input/>
   </template></em></p>
<p>My reply:
   &nbsp;<em><b>!OUTPUT!</b>
   </em></p>
</form>
</td></tr>
</table>
</html>
```

Once you've created and saved these three files (test.html, vhost.html, and input.html), set your default view to the new test.html file. You will then need to click the My Pandorabots button at the top of the screen and publish the chatbot to activate the files before testing your bot.

Congratulations! You now have a talking animated chatbot! You can embed this bot in any web page using <iframe> tags. Copy the URL of your test file as the source of your iframe.

```
<iframe src="http://www.
   pandorabots.com/pandora/
   ................">
   </iframe>
```

Passing Queries—Expanding the Bot's Knowledge with Access to Additional Resources

In chapter 4 we discussed increasing your chatbot's knowledge base by connecting it to additional resources to which you can pass user questions. To accomplish this, you need a combination of AIML and HTML coding. We touched on this coding previously, but will now expand upon it and look at customizing the code for different resources.

The AIML code we presented for passing queries from the chatbot to the Ohio Web Library is:

```
<category>
<pattern>ARTICLE ON *</pattern>
<template>
I'm opening a link to the Ohio
   Web Library, which contains a
   variety of magazine and journal
   articles to help you. If you
   don't see the results, please
   turn off your pop-up blocker.
<think>
<set name="searcharg"><star/>
   </set>
<set name="search">ohweblib</set>
</think>
</template>
</category>
```

Paired with the code above we need some code embedded in the HTML input frame. In both code snippets the values of `search` and `searcharg` are key to the correct passing of the query. search is an alphanumeric string identifying the resource to which the query will be passed. `searcharg` is the text string that will be searched at that site.

```
<template>
<condition name="search"
   value="ohweblib">
<script language="JavaScript">var
   myWindow =window.open('http://
   ohioweblibrary.org/?q=<get nam
   e="searcharg"/>&amp;defaultcat
   =All');
</script>
</condition>
<set name="search">nosearch</set>
</template>
```

The remaining key element is the construction of an open URL to which the query can be directed and the placement of that URL in a JavaScript window.open command. In many cases, you can simply look at the URL of a completed search to identify the needed URL.

Let's construct a fresh example, starting with the URL of the target resource. Encyclopaedia Britannica offers a free online version that can be searched at www.britannica.com. Upon completing a simple search (on hubris) at this site, we see the search results URL is http://www.britannica.com/bps/search?query=hubris. If we remove the searched word from the URL, we get http://www.britannica.com/bps/search?query=. This string can be used to create the JavaScript below. The search word hubris is replaced with a <get> tag identifying the name of the variable that holds the text string to be searched (in this case, `searcharg`).

```
<script language="JavaScript">var
   myWindow =window.
   open('http://www.britannica.
   com/bps/search?query=<get
   name="searcharg"/>');
</script>
```

The above script is embedded in the code snippet we used before, replacing the script to search Ohio

Web Library, and the value of the search variable changed to `britannica`.

```
<template>
<condition name="search"
   value="britannica">
<script language="JavaScript">var
   myWindow =window.
   open('http://www.britannica.
   com/bps/search?query=<get
   name="searcharg"/>');
</script>
</condition>
<set name="search">nosearch</set>
</template>
```

This code is appended to the end of the input frame after the form tag is closed. I usually close any open tables or divisions as well and set the text font to the color of the page background to avoid having confusing text appear in the frame. Here's an example of a complete input frame HTML file with several pass-through resources appended. Note that there are several catalog search value options encoded: `catalog` (which searches the entire collection), `catav` (which limits the results to audiovisual items), `catauth` (which performs an author search), `cattitle` (which performs a title search), and `catsub` (which performs a subject search). The type and number of such options will vary depending on the scoping and search parameters available in your catalog. Note also the `<set>` tag near the end of the code: `<set name="search">nosearch</set>`. This clears any search parameters that have been set in a given search process so that subsequent queries start fresh and can access the full gamut of choices.

```
<html>
<head>
<link rel="stylesheet"
   title="default" href="/
   botmaster/common/default.css"
   type="text/css">
<script>
<!--
function sf(){document.f.input.
   focus();}
// -->
</script>
</head>
<body lang=en-US onload=sf()
   style="background-color:white">
<script language="JavaScript">
function sayit() {
if (parent.vhost) {
if (parent.vhost.vhostLoaded) {
parent.vhost.vhostSpeaking=true;
parent.vhost.sayText("<template
   context="tts"><response/>
   </template>",1,1,2);
} else {
setTimeout("sayit()", 500);
}
}
}
sayit();
</script>
<font face="arial" size="-1">
<table>
<tr>
<td bgColor=#FFFFFF >
<form name=f action=""
   method=post>
!CUSTID!
<p>Type your question in the box
   below, then strike the "Enter"
   key.</p>
<p>Ask your question:</font>
   <input size=40 name=input> </p>
<p>You Said:
   &nbsp;<em><template><input/>
   </template></em></p>
<p>My reply:
   &nbsp;<em><b>!OUTPUT!</b>
   </em></p>
</form>
</td></tr>
</table>
<font color="white">
<template>
<condition name="search"
   value="catalog">
<script language="JavaScript">var
   myWindow =window.open('http://
   catalog.akronlibrary.
   org/search/Y?SEARCH=<get
   name="searcharg"/>');
</script>
</condition>
<condition name="search"
   value="catav">
<script language="JavaScript">var
   myWindow =window.open('http://
   catalog.akronlibrary.org/
   search/Y?SEARCH=<get name="sear
   charg"/>&amp;searchscope=1');
</script>
</condition>
<condition name="search"
   value="catkid">
<script language="JavaScript">var
   myWindow =window.open('http://
   catalog.akronlibrary.org/
   search/Y?SEARCH=<get name="sear
   charg"/>&amp;searchscope=2');
</script>
</condition>
<condition name="search"
   value="catauth">
<script language="JavaScript">var
   myWindow =window.open('http://
   catalog.akronlibrary.
   org/search/a?SEARCH=<get
   name="searcharg"/>');
</script>
</condition>
<condition name="search"
   value="cattitle">
<script language="JavaScript"”>var
   myWindow =window.open('http://
   catalog.akronlibrary.
   org/search/t?SEARCH=<get
   name="searcharg"/>');
</script>
</condition>
<condition name="search"
   value="catsub">
<script language="JavaScript">var
   myWindow =window.open('http://
   catalog.akronlibrary.org/
   search/d?SEARCH==<get
   name="searcharg"/>');
</script>
</condition>
<condition name="search"
   value="ohweblib">
<script language="JavaScript">var
   myWindow =window.open('http://
   ohioweblibrary.org/?q=<get nam
   e="searcharg"/>&amp;defaultcat
   =All');
</script>
<condition name="search"
   value="wolfram">
<script language="JavaScript">var
   myWindow =window.
   open('http://www.wolframalpha.
   com/input/?i=<get
   name=”searcharg"/>');
</script>
</condition>
<condition name="search"
   value="dictionary">
<script language="JavaScript">var
   myWindow =window.
   open('http://www.merriam-
   webster.com/dictionary/<get
   name="searcharg"/>');
</script>
</condition>
<condition name="search"
   value="britannica">
<script language="JavaScript">var
   myWindow =window.
   open('http://www.britannica.
   com/bps/search?query=<get
   name="searcharg"/>');
</script>
</condition>
<set name="search">nosearch</set>
</template>
</font>
```