

Creating a More Natural Conversation

Abstract

Chapter 4 of Library Technology Reports (vol. 49, no. 8), “Streamlining Information Services Using Chatbots,” covers additional AIML tags. The goal is to make a chatbot more lifelike, allowing more intelligent responses to users and enriching conversations.

The previous chapters have provided the tools to create simple questions and answers using AIML. Let’s now look at some ways to create a more complex and natural conversation flow.

Singleton Tags and Chatbot Properties

As you’ve seen in the examples from the preceding chapters, most AIML tags appear in pairs. The tags are delimited by the less than < and greater than > characters, and in each tag pair, the closing tag is exactly the same as the opening tag except for the addition of a slash / character after the first <. We have already encountered one exception to this pattern in the <star/> tag, which is used to repeat the value of an asterisk wildcard in a template response.

Additional examples of singleton tags are used to assign various attributes to the bot when creating its properties, and appear in the bot.aiml file. An example would be <bot name="age"/>. The values (or attributes) accessed by this file are set using the Properties table in the Pandorabots interface, where they are given specific content for later reference. Creating values for these properties allows you to begin to build the “personality” of your chatbot.

More Complex Matching

<that>

In creating a more natural question-and-answer flow, you may wish to have the bot recall and refer to the last response the user has given. Using the <that> tag allows this to happen. This tag retains the last template response as the “subject” of the next pattern, allowing you to create a two-step question-and-answer pattern. You can think of it as <that> being used to modify the value of the subsequent pattern, allowing the pattern to match only if the value of <that> was the bot’s last response.

For example, the user may ask about a particular item or service, let’s say Storytime. Since the availability and location of this service differs depending on the branch library in question, the bot will ask where the user would like to attend Storytime. When the user replies with a branch name, the bot can provide the appropriate information about Storytime at that branch.

```
<category>
<pattern>Where is Storytime
  </pattern>
<template>At which branch do you
  wish to attend Storytime?
  </template>
</category>
<category>
<pattern>Main Library</pattern>
<that>At which branch do you wish
  to attend Storytime?</that>

<template>Storytime at Main
  Library is held in the
```

```

    Children's Room.</template>
</category>
<category>
<pattern>West Branch</pattern>
<that>At which branch do you wish
to attend Storytime?</that>
<template>Storytime at West Branch
Library is held in the alcove
by the bay window.</template>
<category>

```

Here the user will receive the correct information about the branch where he or she wishes to attend the program, but the above patterns “Main Library” and “West Branch” will generate the assigned response only if they follow the question “At which branch do you wish to attend Storytime?” So a user who simply enters the branch name to find other information about the branch won’t get the “Storytime” response.

<set> and <get>

The <set> tag is used to save an input value for later use or reference. The <get> tag is used to retrieve the saved value. This can also be useful in preparing or collecting information for later processing (as for passing to another database).

Let’s look at the <set> tag first. In its simplest form, a <set> can appear alone when used to assign a value or characteristic to the bot itself. This is similar to the bot properties settings. An example would be gender. The following singleton tag sets the bot’s gender to male.

```
<set_male/>
```

However, the <set> tag is more frequently used to assign a value to a specific variable or term. For example:

- <set_it>horse</set_it>—assigns the value “horse” to the variable it.
- <set_location>West Branch</set_location>—assigns the value “West Branch” to the variable location.
- <set_name>Emma</set_name>—assigns “Emma” as the name of the client.
- <set_xxx>X</set_xxx>—creates a variable xxx and assigns the value “X” to it.

Additionally, <set name="x"></set> can be used to extend AIML by creating custom tags for foreign language pronouns or predicates. In this way, an infinite variety of variables and values can be incorporated.

Let’s now turn to the <get> tag. This tag can also appear alone rather than in a pair, though it usually

<bot_name/>	bot name
<bot_location/>	bot location
<bot_gender/>	bot gender
<bot_birthday/>	bot birthday
<bot_master/>	bot botmaster name
<bot_birthplace/>	bot birthplace
<bot_boyfriend/>	the bot’s boyfriend?
<bot_favoritefood/>	bot favorite food
<bot_favoritemovie/>	bot favorite movie
<bot_favoriteband/>	bot favorite band
<bot_favoritebook/>	bot favorite book
<bot_favoritecolor/>	bot favorite color
<bot_favoritesong/>	bot favorite song
<for_fun/>	what the bot does for fun
<bot_friends/>	the bot’s friends
<bot_girlfriend/>	the bot’s girlfriend?
<bot_wear/>	what does the bot wear
<bot_sign/>	bot’s astrological sign
<bot_looklike/>	what does the bot look like
<bot_music/>	bot’s music preference
<bot_talkabout/>	bot’s favorite subjects for discussion
<getsize/>	bot memory size
<getversion/>	program version

Table 4.1
Read only tags.

appears with a matching <set> tag. On those occasions where <get> appears without a <set> tag, it is accessing one of a set of predetermined property values set outside of the AIML code or process. These values are referred to as “Read Only.” Table 4.1 lists Read Only <get> tags.

It is more usual, however, for the <get> tag to refer to a value assigned by a <set> tag in the same or another AIML file. These values usually are associated with a particular user and can only appear in the <template> of a category. Here are some of the standard <get> tags of this kind:

```

<get_name/> user's name
<get_topic/> a previously assigned
topic of conversation
<get_it/> the value assigned to "it"
<get_location/> the user's
geographic location

```

Now let’s look at an example of a <set> <get> combination. Here the user is asked for a branch name; this name is then used when answering the

user's question about hours.

```
<category>
<pattern>my branch is *</pattern>
<template>
<set name="branch"><star/></set>
  I'll remember your branch is
  <star/>.
</template>
</category>

<category>
<pattern>What are your hours
  </pattern>
<template>
Hours for the <get name="branch"/>:
  Monday through Friday 9 A.M. to
  5 P.M.
</template>
</category>
```

<topic>

The <topic> tag expands upon the functionality of the <set> <get> pair by collecting together a group of categories to which a single topic value can be assigned. <topic> tags appear in pairs surrounding a set of categories that all refer to that topic. If a topic value is matched in a user input, those categories associated with it are searched first for pertinent pattern matches. Thus, the use of <topic> tags allows different responses to a single pattern depending on the topic in which it appears. This provides a more natural flow of conversation as well as allowing more specificity in template responses. Here is an example. The following conversation offers differing responses to the same questions depending on the user's original input (or lack thereof):

```
<category>
<pattern>I like to *</pattern>
<template>Yes, <set name="topic">
  <star index="1" />
</set> is fun. Let's talk about
  it?
</template>
</category>

<topic name="read">
<category>
<pattern>where do you like to
  <get_topic/> </pattern>
<template>I like to read curled up
  on my couch.</template>
</category>
<category>
<pattern>what do you need to
```

```
<get_topic/> </pattern>
<template>Just my glasses and a
  good book.</template>
</category>
</topic>

<topic name="sing">
<category>
<template>I'm happy to sing
  anywhere. But, for the sake of
  others, I try to limit myself
  to the shower and my car.
  </template>
</category>
<category>
<pattern>what do you need
  </pattern>
<template>I either turn the stereo
  up to help drown out my sour
  notes. Or I make sure anyone
  else in the house has earplugs.
  </template>
</category>
</topic>

<category>
<pattern>where do you like to
  <get_topic/> </pattern>
<template>Where do I like to go to
  do.. What?</template>
</category>
<category>
<pattern>what do you need
  <get_topic/> </pattern>
<template>I really cannot answer
  that question without more
  info. What do you like to do?
  </template>
</category>
```

The above example is very simple, but the value is easily seen if you have branches with differing hours of service, material types with different loan periods, etc.

Above you can also see an example of the use of the <get_topic/> tag. You can use this tag to incorporate the assigned value of <topic> into your template responses.

The <topic> tag can also be set inside a template to direct further conversation to the stated topic, as in this example:

```
<category>
<pattern>I want to go to *
  </pattern>
<template>Can I provide you with
  any additional information
  about that branch? <topic
```

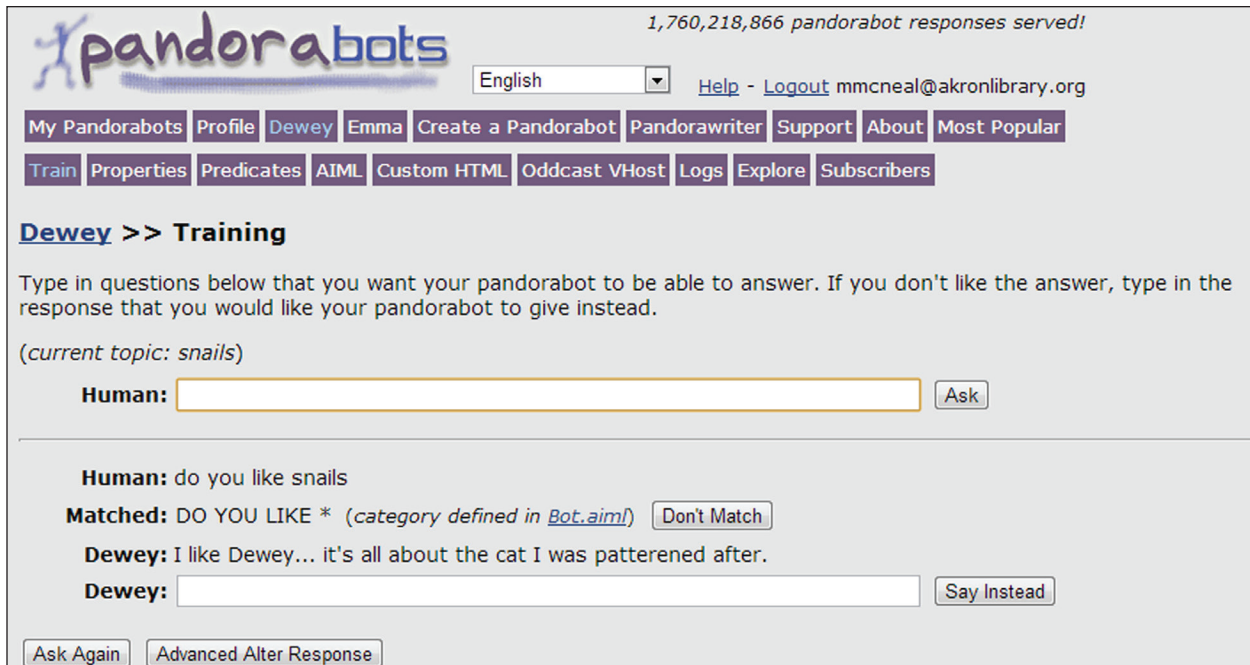


Figure 4.1
Pandorabots Training screen.

```
name="West Branch"></template>
</category>
```

Here, subsequent conversations will revolve around West Branch until the topic is reset.

<person>

The <person> tag provides a pronoun-swapping function that allows the bot to make sensible responses to a variety of statements. The <person> tag is a singleton shorthand for the sequence <person/><star/></person>. The most common use of the tag is to operate on a <star/> variable string to replace included pronouns. In the following example the user states:

The branch is too far from me.

The bot responds:

I understand the branch is too far from you.
What could we do to help?

```
<category>
<pattern>the branch * </pattern>
<topic> I understand the branch
  <person/>. What could we do to
  help?
</category>
```

Similar issues could be addressed with the same code:

User: The branch is too cold for me.

Bot: I understand the branch is too cold for you. What could we do to help?

User: The branch is confusing to me.

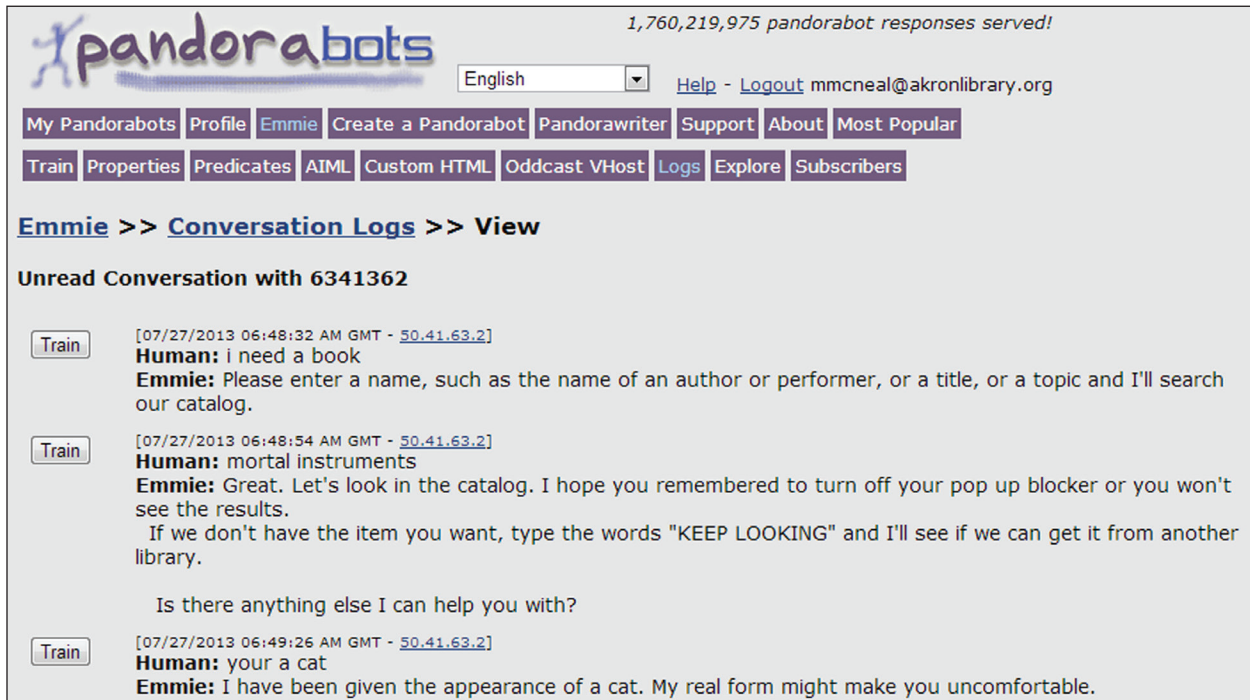
Bot: I understand the branch is confusing to you. What could we do to help?

<think>

Use of the <think> </think> tag pair allows the inclusion of scripting that you don't wish to be displayed (or read) to the user. It can be used to retain a topic for multiple questions or information that is being used to narrow the scope of a particular question.

However, possibly the most important use of the <think> tag in basic-level chatbots is to pass queries to other databases or information resources. This process requires a combination of JavaScript coding in the HTML frame within which the bot's data input field appears and AIML coding within the specific category.

Let's look at a fairly simple example. First, here's the AIML coding to pass a search from the bot to the Ohio Web Library (which hosts a number of magazine and journal indices). The <think> tags are used to hide the process of passing the search and searcharg values to the JavaScript code in the HTML frame. Basically, what happens here is that if a user asks for an article on a topic, the AIML code responds with the text about passing the search to Ohio Web Library while saving the value of the wildcard * as <star/>



1,760,219,975 pandorobot responses served!

English Help - Logout mmcneal@akronlibrary.org

My PandoraBots Profile Emmie Create a PandoraBot Pandorawriter Support About Most Popular

Train Properties Predicates AIML Custom HTML Oddcast VHost Logs Explore Subscribers

[Emmie](#) >> [Conversation Logs](#) >> [View](#)

Unread Conversation with 6341362

Train [07/27/2013 06:48:32 AM GMT - 50.41.63.2]
Human: i need a book
Emmie: Please enter a name, such as the name of an author or performer, or a title, or a topic and I'll search our catalog.

Train [07/27/2013 06:48:54 AM GMT - 50.41.63.2]
Human: mortal instruments
Emmie: Great. Let's look in the catalog. I hope you remembered to turn off your pop up blocker or you won't see the results.
 If we don't have the item you want, type the words "KEEP LOOKING" and I'll see if we can get it from another library.
 Is there anything else I can help you with?

Train [07/27/2013 06:49:26 AM GMT - 50.41.63.2]
Human: your a cat
Emmie: I have been given the appearance of a cat. My real form might make you uncomfortable.

Figure 4.2
Pandorabots Log interface with links to the Training interface.

and assigning the value `ohweblib` to `search`. We have used the term `search` to identify the location where the search is to be conducted and, in some cases, the type of search to be completed (keyword, author, title, audiovisual materials only, etc.)

```
<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>
```

Here's the HTML code that follows the query input form and receives the values from the category above. When an input is processed, the template contained within the HTML file is processed along with that included in the AIML file. This template checks to see if a search value has been identified in the AIML

code. If such a value exists, the template contains JavaScript that opens a new window containing the results of a search done on the appropriate website, using the value of `searcharg` as the search term. The specific search conditions are followed by `<set name="search">nosearch</set>` to return the bot to a "nonsearching" mode for future queries.

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

More complete instructions for creating code to pass searches to other resources are contained in chapter 5.

The Ultimate Default Category

Obviously there will always be some questions for which you have not programmed a response. For these questions, it is useful to create a simple category to respond to users with either a request for additional

information or an explanation that the information they are seeking isn't yet part of the bot's knowledge base. Below are examples of both types of "ultimate default categories."

```
<category>
<pattern>*</pattern>
<template>
I need more information to help
  you. Are you looking for
  something to check out or do
  you have a question about this
  topic?</template>
</category>
<category>
<pattern>*</pattern>
<template>
<random>
<li>I'm sorry. I don't recognize
  your question. Please ask
  one of our human staff for
  assistance.</li>
<li>I'm sorry. I don't have a
  response to that. Check back in
  a few days and I may.</li>
<li>I don't have an answer for
  that right now. Tell me again
  what you're looking for. </li>
</random>
</template>
</category>
```

Adding Knowledge

There are a number of ways to add information to the AIML coded bot.

The simplest is provided by the Training mode of the Pandorabots interface. Here you are presented with a question form similar to what you have on your bot's own interface. However, when you pose a question here, you are provided not only with the answer identified by the current version of your AIML, but also with the category your query matched and the specific AIML file that contains that category. From here you can simply provide a new response, which the Pandorabots interface will include in the updates. aiml file. Another option is to edit the existing AIML. Figure 4.1 displays the Pandorabots Training screen.

In addition, Pandorabots offers an easy way to review recent bot conversations in its Log interface. Here you can view entire conversations, noting where a specific question could be answered more completely or correctly. This interface also offers a quick link to the Training interface beside each question (see figure 4.2).

Finally, when adding an entirely new theme or subject to your bot's knowledge base, you will probably want to start fresh with a new AIML file. Creating separate files for diverse subjects and functions will allow you to keep track of your code and easily identify which files need updating.