Tellme More

Jonathan Eisenzopf

February 23, 2001

In Part I of our quickstart VoiceXML tutorial, we will learn how to sign-up as a Tellme.com developer and write our first interactive VoiceXML application.

Overview

Adding VoiceXML to your Web site can be an effective way to make your content accessible to many more customers. This is Part I of a comprehensive tutorial on the VoiceXML specification, so get ready to have some fun and be sure to come back for Part II where we will develop a complete dynamic VoiceXML application. In this tutorial, we will create and launch our first live VoiceXML application in a few simple steps. In fact, you don't even have to learn VoiceXML to get started. You can follow the steps in this tutorial to register as a Tellme developer, paste in the example source code, and dial the 800 number to test your new application.

Examples

As we step through each code example, you will also be able to dial into Tellme and demo the code live on the Mother of Perl Tellme extension. Simply dial *1-800-555-tell* , wait for the Tellme menu, and dial *1-73759* . Press *1* to select the examples for tutorial 20. You will hear a greeting and a list of the examples. Press the number that corresponds with the example in the tutorial to demo it. For example, when you see:

Example 1.

You will be able to interact with the source code by dialing the Mother of Perl Tellme extension and pressing 1 on your phone. At the end of the example, you will be taken back to the Mother of Perl menu. Each example is also linked to the XML source code file where you can examine the file in your Web browser, or launch it directly into your favorite XML editor.

In addition to the live examples, you will find the source code for all examples in the tutorial20.zip [tutorial20.zip] file, which you can use for free to create your own VoiceXML applications.

VoiceXML

VoiceXML is an XML grammar developed by the VoiceXML Forum. VoiceXML makes it possible to access Internet content by phone and to develop speech-based telephony applications. The forum was founded by AT&T, IBM, Lucent, and Motorola and is now made up of several hundred corporate members.

Until now telephony solutions required specially trained technicians to install, configure and maintain them. Because the language was developed as an XML format, developers can integrate this new language with their existing tools. This brings telephony integration within reach of many Web developers. In fact, Webreference.com now has its very own VoiceXML channel available at http://webref.com/news [http://webref.com/news] thanks to Weblog 1.71 [http://www.webreference.com/perl/tutorial/19/].

Back to the Basics

While version 1.0 of the specification has been around since March of 2000, VoiceXML has not garnered as much hype and acceptance as WAP. Recently, however, users and developers are realizing that it's time to get back to that basic form of communication. *The telephone!*

That's right. The international telecommunications backbone has a common operating platform that we can fully utilize without any additional work. Almost everyone understands how to use a phone, so I decided to look into what was available for creating applications for the telephone.

Voice Portals

What I found were several companies that provide managed voice portal services to customers who want to develop voice applications but do not want to build and manage their own infrastructure. Another appealing feature is their online developer programs that allow you to test your applications for free before having to commit significant resources. This is especially helpful when many companies are still evaluating the benefit of rolling out a voice portal or application.

Tellme

Currently, Tellme.com is well funded and probably the leader in the voice portal services market. They provide free online developer tools, extensive documentation, libraries of code and tutorials to help you get started.

Once you have registered as a developer, you can immediately start coding your VoiceXML applications and testing them for free via their toll-free Tellme Studio test

platform. After you've tested your application, you can launch it onto its own Tellme extension, which makes it available to anyone in the US via the main Tellme 800 number.

Platform Incompatibilities

One thing to note is that VoiceXML applications developed for Tellme.com may not work on another platform because of differences that presently exist between the VoiceXML implementations, so beware. The primary differences are in the grammar definition languages, which we will discuss in more detail later in the tutorial.

VoiceXML Basics

XML Rules

Since VoiceXML is XML, we must adhere to the basic XML rules. Without going into excruciating detail, you should adhere to the following basic XML guidelines:

- Must encode default entities. Default entities are reserved characters that can't be put into XML directly, but are entered using special codes. These reserved characters are <, >, &, ', and ". They are encoded respectively like so: *<*, *>*, *&*, *'*, and *"*.
- Must have a root element. In this case, the root element is *vxml*.
- Elements must be properly nested. Nesting requires that you close tags in the same order you opened them or else you will get an error. Web browsers are forgiving with HTML, but XML parsers are not.

Hello World

There are 47 elements (or tags) in the VoiceXML format compared to 91 for HTML. VoiceXML is capable of something as simple as delivering content over the phone or as complex as a full-fledged E-Commerce application. No introduction is complete without a Hello World example.

Example 1. [example1.xml]

```
<?xml version="1.0"?>
<vxml version="1.0">
```

```
  <form>
    <block>Hello World!</block>
  </form>
</vxml>
```

The example above will synthesize "Hello World!" and then exit. Go ahead and try it right now by calling 1-800-555-tell and then 1-73759. Then select 1 from the menu options.

Fortunately, not all VoiceXML dialogs have to be synthesized. We could have also played a recording of someone saying "Hello World" by including an *audio* element with a *src* attribute containing the url to the audio file.

```
<?xml version="1.0"?>
<vxml version="1.0">
  <form>
    <block>
           <audio src="hello.wav">Hello World!</audio>
         </block>
  </form>
</vxml>
```

Dialogs

Each VoiceXML document contains one or more dialogs. There are two types of dialogs, *forms* and *menus*. A form collects a user's inputs from fields, just like an HTML form. Menus present a list of choices that the user can select from, similar to a list of links that a user would select from on a Web site. More advanced applications will contain multiple dialogs and possibly even sub-dialogs.

For example, when you dial the number on the back of a credit card, it might ask you to enter your credit card number and your social security number or zip code. This is an example of a form, because the application is gathering information from you via your touch-tone phone. This information is submitted to the application, which then might give you a list of options. That's an example of a menu dialog.

By default on Tellme.com, if you create a VoiceXML document that contains multiple forms, it will execute each form in the order in which they occur. For example, let's add a second form to our Hello World example and see what happens.

Example 2. [example2.xml]

```
<?xml version="1.0"?>
<vxml version="1.0">
```

```
  <form>
    <block>Hello World!</block>
  </form>
  <form>
    <block>My name is mud.</block>
  </form>
</vxml>
```

If you call Tellme, you will hear:

Tellme: Hello world, my name is mud.

Goto

Sometimes you may want to skip from one form to another in a document, but not sequentially. This is possible via the *goto* element. With it, you can skip to another location in the current document, or even in a separate document. As a simple example, we'll add a third form to our Hello World example that skips from the first to the third form within the document.

Example 3. [example3.xml]

```
<?xml version="1.0"?>
<vxml version="1.0">
  <form id="hello">
    <block>Hello World!
           <goto next="#jack"/>
    </block>
  </form>
  <form id="mud">
    <block>My name is mud.</block>
  </form>
  <form id="jack">
    <block>My name is Jack.</block>
  </form>
</vxml>
```

If you demo the example on 800-555-tell, you will hear the following:

Tellme: Hello World, my name is Jack.

Prompts

At some point, you're probably going to want to get input from the user. This is where the *prompt* element comes in. The prompt element synthesizes the text contained in the element and waits for the user to provide input.

```
<prompt>Please enter your
 4 digit pin code.</prompt>
```

You can also play an audio file instead of synthesized text.

```
<prompt><audio src="pin.wav">Please enter your
 4 digit pin code.</audio></prompt>
```

The code above will attempt to play the `pin.wav` file. You're probably wondering why there's text in addition to the audio file . In the case that Tellme cannot retrieve the audio file, it will synthesize the text as a last resort, therefore, it's a good idea to include the text even when you're linking to an audio file.

```
<?xml version="1.0"?>
<vxml version="1.0">
  <form id="hello">
    <block>Hello World!</block>
      <field name="name">
        <prompt>What is your name?</prompt>
                </field>
  </form>
</vxml>
```

The code above is not a demo on the Mother of Perl Tellme extension. That's because the code isn't ready to run yet. If you tried to run it as is, it would prompt for your name and then report that an error had occurred. This is because we have to add additional logic to handle user input and error conditions. We'll cover these issues in the next three sections.

Handling Form Input

VoiceXML includes a highly granular syntax for handling forms, and does require more work when compared to HTML forms. Each VoiceXML field can contain a prompt, code to handle the user inputs and errors, and a grammar. Grammars define the range of input values the user can provide. Once all fields have been filled, the form is submitted to another VoiceXML document or to a script on a Web server via the HTTP protocol.

There are two ways users can provide input to form fields. The first is via the phone keypad (DTMF tones). The second is through voice commands, which will be covered in Part II [http://webref/perl/tutorial/21] of the VoiceXML series. One common task is to ask a user to input their PIN code to identify who they are. Let's simulate this dialog:

Tellme: Please enter your 4 digit pin code.

User: 1234

Tellme: Good Morning Mr. Eisenzopf. How may I help you?

Now let's write the VoiceXML document that implements this interface.

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <prompt>Please enter your 4 digit pin
code.</prompt>
            <filled>
              <submit
next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
      </filled>
    </field>
  </form>
</vxml>
```

The *filled* element is called when the system receives input for the field. The user input for the *pin* field is submitted to the CGI script specified in the *next* attribute of the *submit* element. Writing scripts for VoiceXML input is virtually identical to writing scripts for HTML forms.

The *filled* element can occur either within the *field* element:

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <prompt>Please enter your 4 digit pin
code.</prompt>
            <filled>
              <submit
next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
      </filled>
    </field>
  </form>
</vxml>
```

or within the *form* element:

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <prompt>Please enter your 4 digit pin
```

```
>
     </field>
     <field name="ccnum">
       <prompt>Please enter your credit card
 number.</prompt>
     </field>
     <filled>
               <submit
 next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
     </filled>
   </form>
 </vxml>
```

This allows you to control input handling at both the field and/or the form level. For example, in some cases, you may want to process field values each time they're filled. Other times, you will want to wait until all fields have been filled.

In our examples above, we are using the *submit* element to send the field values to a script specified by the *next* attribute. There are a number of other attributes that you can use to control how and what is sent. By default, Tellme will send the form values via the HTTP protocol to the specified url using the GET method. You can override the default by specifying POST instead, via the *method* attribute:

```
<submit next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"
method="POST"/>
```

When submitting field values via the *filled* element within the *form* scope as opposed to a *field*, you can control what fields get passed with the *namelist* attribute:

```
<submit next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"
        namelist="pin ccnum"/>
```

In addition to the *submit* element, we have a number of other options for acting on field input. First, we could have sent the user to another form:

```
       <filled>
                <goto next="#first_name"/>
       </filled>
```

or included a message:

```
       <filled>
                <prompt>Thank you Mr. Rich. It will take 24 to
               48 hours to funnel money into your off-shore
               accounts.</prompt>
         <submit
 next="http://www.textant.com/cgi-bin/launder.pl"/>
       </filled>
```

Tellme also offers several options for processing input inline instead of submitting it to a script. The simplest alternative is the *result* element. If the value of the *name* attribute matches the field value, it will execute the statements inside.

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <prompt>Please enter your 4 digit pin
code.</prompt>
            <filled>
              <result name="1234">
          <prompt>Good Morning Mr. Eisenzopf. How may I help
you?</prompt>
          <goto next="#secret_laboritory"/>
        </result>
        <result name="2468">
          <prompt>Good Morning Mr. King. How may I help
you?</prompt>
          <goto next="#control_room"/>
        </result>
      </filled>
    </field>
  </form>
</vxml>
```

While the *result* element can match simple strings, it doesn't give us the power of real condition statements like *if* and *else*, but it does work well where you have a pre-defined set of inputs. If the contents will be more dynamic, like a PIN code, we really would need conditional statements.

Handling Events

Invalid Input

Handling forms requires more than simply gathering field input. If the user provides input that doesn't match the grammar, Tellme will look for the *nomatch* element. If it doesn't exist, the application will croak. The most common way to handle invalid input is to tell the user so and ask them for the information again.

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <grammar>
```

```
                    <![CDATA[
                                        Four_digits
                                ]]>
      </grammar>
      <prompt>Please enter your 4 digit pin
code.</prompt>
             <filled>
               <submit
next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
      </filled>
      <nomatch>Invalid pin code.
        <reprompt/>
      </nomatch>
    </field>
  </form>
</vxml>
```

In the example above, we are using one of the Tellme grammars, *Four_digits*, to regulate the input. The grammar contains rules that will cause the document to call the *nomatch* event unless the user enters four numbers via the phone keypad or by voice. The *reprompt* element repeats the last prompt, **Please enter your 4 digit pin code** after telling the user that they've entered an invalid PIN code. Tellme will keep reprompting the user as long as they keep entering invalid input.

Of course, if the user is repeatedly entering invalid input, giving them the same error message and prompt won't help the user any. In fact, it would be downright user unfriendly. This is why the *nomatch* element includes a *count* attribute that allows us to give the user a different warning message each time the *nomatch* event is called.

```
      <nomatch count="1">Invalid pin code.
        <reprompt/>
      </nomatch>
      <nomatch count="2">Please press or say exactly four numbers.
        <reprompt/>
      </nomatch>
      <nomatch count="3">Too many attempts. Please call back
                         another time.
        <exit/>
      </nomatch>
```

Now, the second time the user provides bad input, we give them more specific instructions, "`Too many attempts. lease call back another time.`" The third time the user provides bad input, we exit the program.

No Input

Like handling invalid input, it's also critical to handle no input. Tellme calls the

noinput event when the user does not provide any input after a prompt has been played.

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="hello">
    <field name="pin">
      <grammar>
                    <![CDATA[
                                        Four_digits
                              ]]>
      </grammar>
      <prompt>Please enter your 4 digit pin
code.</prompt>
             <filled>
               <submit
next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
      </filled>
      <noinput>
        <reprompt/>
      </noinput>
      <nomatch>Invalid pin code.
        <reprompt/>
      </nomatch>
    </field>
  </form>
</vxml>
```

Like the *nomatch* element, if you don't handle the *noinput* event, your VoiceXML application will return an error when it's called. Also, like *nomatch*, you can set the *count* attribute so that a different message is played each time the user fails to provide input.

```
      <noinput count="1">No PIN entered.
        <reprompt/>
      </noinput>
      <noinput count="2">You must enter your PIN number to proceed.
        <reprompt/>
      </noinput>
      <noinput count="3">Please press or say exactly four numbers.
      </noinput>
```

If the *noinput* event is called more than 3 times, Tellme will continue to repeat the last *noinput*, "**Please press or say exactly four numbers.**", until the user provides some input.

Tellme.com Studio

Overview

Tellme.com provides the infrastructure and tools necessary to build your very own VoiceXML applications. Once you have registered, you will be able to test your applications via the developer studio toll-free number, *1-800-555-vxml*.

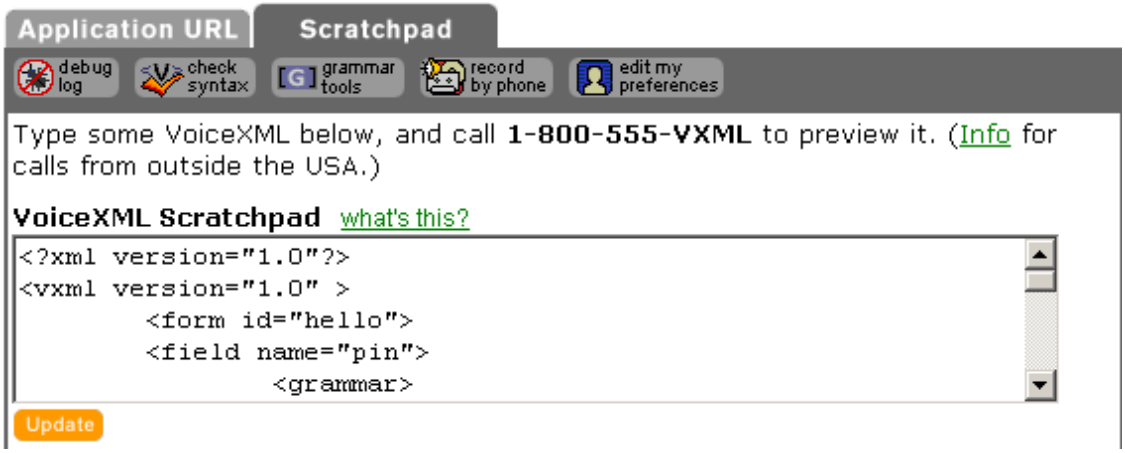
Tellme is capable of synthesizing text and audio files, recognizing spoken and phone keypad (DTMF tones) input, recording spoken input, and transferring a call to another number.

Developer Registration

Registering for a developer account is very straightforward. Complete the registration form [http://studio.tellme.com/apply/] and you're up and running. Best of all, registration is completely free.

My Studio

Ok, let's get acquainted with Tellme Studio. Once you have registered and have logged into studio.tellme.com [http://studio.tellme.com], click on the *My Studio* link located at the top left of the screen. You should see two tabs on the top center portion of the screen, *Application url* and *Scratchpad*.



The first tab allows you to enter a url for your VoiceXML file if it's located on a Web server. In the beginning, you will want to use the Tellme Scratchpad. It's an HTML textarea where you can tweak your VoiceXML document real-time without having to upload it to a server each time you make a modification. This is useful when you're testing a document, as we'll see later.

Within the main screen, you should also see five buttons, *debug log*, *check syntax*,

grammar tools, *record by phone*, and *edit my preferences*. The *debug log* button comes in handy when you're trying to figure out how to fix a programming error. Once you've pasted some VoiceXML data into the scratchpad and checked the syntax (with the check syntax button), try clicking the debug log button and then testing your application. You will see the screen fill with debugging information as you run your tests. You probably won't use the grammar tools initially until you begin writing your own grammars, which we'll touch on later.

My Extensions

Once you've fully tested your VoiceXML application, you can make it available to the world via your own Tellme.com extension, which can be reached from the main tellme number, 1-800-555-tell.

To activate your extension, click on the button labeled *Activate Extensions*. At this point, your application will need to be accessible from a Web server so Tellme knows where to point when someone dials into your extension. Enter the url and check the *Enable my Extension* box and click the update button. Your VoiceXML application will then be available.

Tellme also allows you to record your voice prompts over the phone. Call Tellme Studio, 1-800-555-vxml, and say, "Record By Phone!" Then enter your developer ID and pin and record your prompts. You can then have these prompts saved or emailed to you.

Documentation and Libraries

Links to the Tellme documentation and libraries are located on the left side of the Tellme Studio pages. This information will go a long way in your quest to develop full-blown VoiceXML applications. I would highly recommend downloading and printing the VoiceXML Reference [http://studio.tellme.com/voicexmlref/] and the Grammar Reference [http://studio.tellme.com/grammarref/].

Testing Your Applications

After working with Tellme Studio for about a month, I've found a really nifty way to test things out. As I develop a new application, I work almost exclusively in the Scratchpad to flesh things out. When I add a new prompt or feature, I call the Tellme Studio phone number to test it out. One really neat trick is, when you press ** on the phone after you've entered your developer ID and password, it will loop through the document over and over again. You can make changes on the fly, and listen to the results as it loops over the document. This is a great troubleshooting and testing feature. Often, I find that testing things out before I actually go through the trouble of putting it

on a server saves alot of time, because what you expect it to do and how it actually works tend to be different.

Dynamic VoiceXML

Continuing with our PIN number scenario, it's time to develop the CGI script that provides a response to the user's input. First, let's look at our completed VoiceXML form.

Example 4. [example4.xml]

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <form id="login">
    <field name="pin">
             <grammar>
                      <![CDATA[
          Four_digits
               ]]>
                   </grammar>
                   <prompt>Please enter your 4 digit pin
code.</prompt>
                   <filled>
                     <submit
next="http://www.webreference.com/cgi-bin/perl/20/pin.pl"/>
             </filled>
      <noinput count="1">No PIN entered.
        <reprompt/>
      </noinput>
      <noinput count="2">You must enter your PIN number to proceed.
        <reprompt/>
      </noinput>
      <noinput count="3">Please press or say exactly four numbers.
      </noinput>
             <nomatch count="1">Invalid pin code.
        <reprompt/>
      </nomatch>
      <nomatch count="2">Please press or say exactly four numbers.
        <reprompt/>
      </nomatch>
      <nomatch count="3">Too many attempts.
                         Please call back another time.
        <exit/>
      </nomatch>
          </field>
  </form>
</vxml>
```

As you've probably already guessed, the CGI script will be written in Perl. As it turns out, if you know how to use the *CGI.pm* Perl module, you'll know how to handle VoiceXML form input.

```
#!/usr/bin/perl -w
use strict;
use CGI;
my %users = (
     '1234' => 'Eisenzopf',
     '2468' => 'King'
);
my $q = new CGI;
my $pin = $q->param('pin');
print "Content-Type: text/xml\n\n";
print <<VXML;
<?xml version="1.0"?>
<vxml>
  <form>
    <block>
VXML
print "Good Morning Mr. $users{$pin}. How may I help you?"
   || "$pin is an invalid pin code.";
print <<VXML;
    </block>
  </form>
</vxml>
VXML
```

Basically, we create a new instance of the `CGI.pm` module. We access the value of the *pin* field with the following line of code:

```
my $pin = $q->param('pin');
```

It's important to make sure you return the proper HTTP header. The default header will not work properly. Instead, you must use the following statement as the first output back to Tellme:

```
print "Content-Type: text/xml\n\n";
```

The rest of the script is straightforward Perl. We lookup the user's last name based on their PIN code. If we find a name, we welcome them, otherwise, we return an error. It's a very rudimentary script, but it's our first taste of developing a dynamic VoiceXML application.

Conclusion

In this tutorial, you've learned the basics of creating VoiceXML documents. You've learned how to write a Perl script to handle basic VoiceXML inputs. In the next tutorial, you'll learn how to create a full-fledged dynamic VoiceXML application. If you've ever read a "Choose Your Own Adventure" book or played Zork, you won't want to miss Part II of the VoiceXML series.

Resources

- Tellme.com [http://www.tellme.com]
- VoiceXML Reference [http://studio.tellme.com/voicexmlref/]
- Grammar Reference [http://studio.tellme.com/grammarref/]
- VoiceXML Forum [http://www.voicexml.org]
- VoiceXML Specification [http://www.voicexml.org/specs/VoiceXML-100.pdf]
- BeVocal [http://www.bevocal.com]
- Voxeo [http://www.voxeo.com]
- Voice Genie [http://www.voicegenie.com]
- VoiceXML Central [http://www.voicexmlcentral.com]