Speechweb

User manual

Written By: Paul D. Meyer

Contents

Getting Started

Introduction

Speechweb works by taking voice input from the user, converting it to text, and then using an executable program to take in that text as input and produce some output as text. That output text is then spoken back to the user. Basically, the Speechweb application as a whole is just that executable program wrapped in a convenient speech interface (with some other graphical and web capabilities as well). If you want to get right into learning the Speechweb development process, then you can skip ahead to the next section, "Creating a basic Speechweb application". However, if this is your first time attempting to create a Speechweb application, or if you have a program that you want to test further before you implement it as a full Speechweb application, then it may be useful to start out by wrapping your program in a simpler web interface without speech capabilities.

A speechless web application

Creating a simple web application that accesses a remote program on a web server will involve three files:

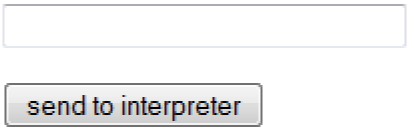
- A webpage (HTML file)
- A CGI script
- An executable program

An example of all three of these files can be downloaded together for convenience at http://speechweb.cs.uwindsor.ca/pages/manual/my_app.zip. Simply unzip the files and copy them onto a web server that has CGI capability. Make sure that the permissions on all of these components are set to 755, including the folder that contains them. This must be done so that the application can be run from over the internet. In order to run the application, simply access the webpage "my_app.html" from an internet browser.

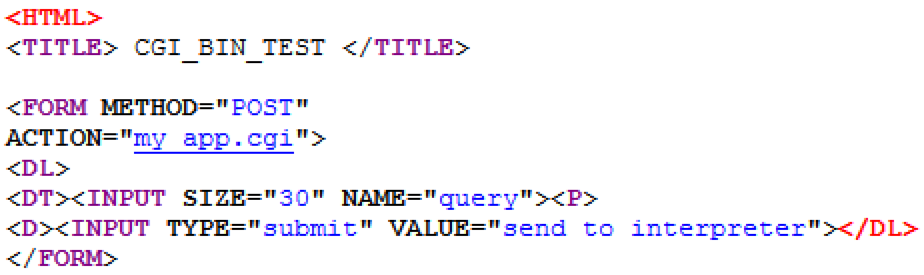
If you are a computer science student at the University of Windsor, you are provided a web server in the "public_html" directory on your user account on the computer science servers. By copying the application files into this directory, you will be able to access the webpage at "http://cs.uwindsor.ca/~UWINID/my_app.html", with "UWINID" replaced by your University of Windsor account username.

The webpage

The webpage “my_app.html” provides a simple form for providing input to the executable program that looks like this:



It works by submitting the text in the textbar to the CGI script, which then runs the executable program with that text as input. After the program supplies output and ends, the CGI file will display this output. By viewing the source code of this webpage, you can see that the entire process takes up only a few lines of code (although this is not including the work of the CGI script):



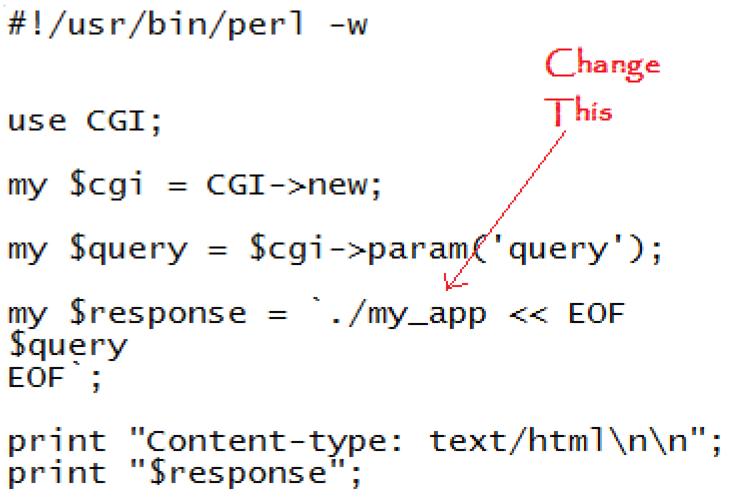
```
<HTML>
<TITLE> CGI_BIN_TEST </TITLE>

<FORM METHOD="POST"
ACTION="my_app.cgi">
<DL>
<DT><INPUT SIZE="30" NAME="query"><P>
<D><INPUT TYPE="submit" VALUE="send to interpreter"></DL>
</FORM>
```

Although simple, the same basic principles are used in full Speechweb applications, except that much more code must be used to handle all the speech input and output and other added functionality of Speechweb.

The example webpage allows interaction with the “my_app” program. You can try submitting inputs such as “hello”, “hi there”, or “help” to test it out.

The CGI script

The example CGI script “my_app.cgi” is currently linked to the “my_app” program (which in turn links the main webpage to that program as well). In order to use a program of your own, the program must be written in a programming language that can be run from a CGI script (C, C++, Java, Miranda, Haskell, etc.). Additionally, the CGI script must be altered slightly:



```
#!/usr/bin/perl -w

use CGI;

my $cgi = CGI->new;

my $query = $cgi->param('query');

my $response = `./my_app << EOF
$query
EOF`;

print "Content-type: text/html\n\n";
print "$response";
```

You must change “my_app”, as indicated, to the name of your executable program. The program must be in the same folder as the CGI script and webpage.

Creating a basic Speechweb application

Introduction

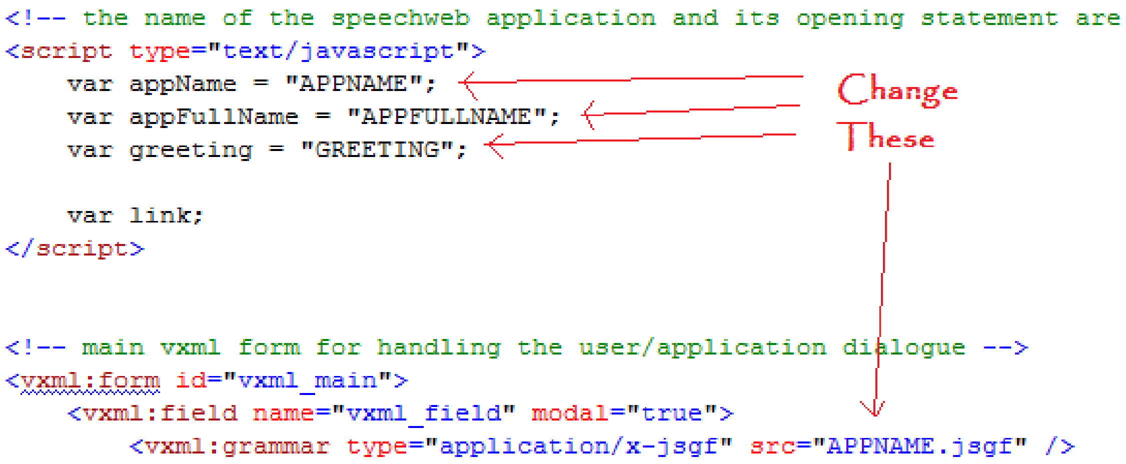
Although there are countless forms a Speechweb application can take, the most basic would be the simple voice-in/voice-out type of application. This kind of Speechweb application follows a simple pattern of repeatedly taking in voice queries and returning voice responses. Due to its simplicity, it is likely the easiest type of application to put together for a first-time Speechweb developer. This application (and any Speechweb application in general) works by using four key elements:

- An X+V XML webpage
- A JSGF grammar file
- A CGI script
- An executable program

The XML page is used by the browser to display the Speechweb application, the JSGF grammar file is used to specify the full range of possible voice input that the application may receive from the user, and the CGI script handles the transfer of input and output between the Speechweb application and the executable program. Thus, creating a Speechweb application in essence means creating these four different files. However, you will find that the entire development process is actually very straightforward and painless, especially since the behind-the-scenes executable program (which is the heart of any given Speechweb application) can be written in any programming language the developer is familiar with (C, C++, Java, Miranda, Haskell, etc.).

The X+V XML webpage

The XML page may appear lengthy and complicated, but very little of the XML page is actually altered for each specific Speechweb application, so preparing your application's XML page is very simple. A template XML page can be found at http://speechweb.cs.uwindsor.ca/pages/manual/template/templatexml.txt. The parts that must be changed are all located near the top of the page so that they are easy to find. The following image points out all of these specific parts:



```
<!-- the name of the speechweb application and its opening statement are
<script type="text/javascript">
    var appName = "APPNAME";
    var appFullName = "APPFULLNAME";
    var greeting = "GREETING";

    var link;
</script>


<!-- main vxml form for handling the user/application dialogue -->
<vxml:form id="vxml_main">
    <vxml:field name="vxml_field" modal="true">
        <vxml:grammar type="application/x-jsgf" src="APPNAME.jsgf" />
```

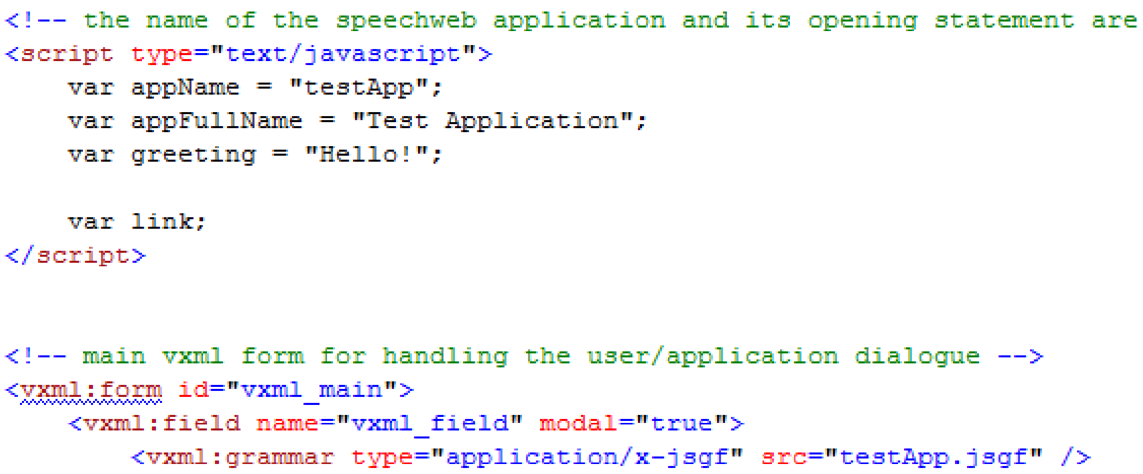
As can be seen above, all of the parts of the XML file that need to be changed for each specific Speechweb application are marked with ALLCAPS placeholders.

The APPNAME placeholder appears twice and should be replaced with the filename you have chosen to use for all of your application's component files. For example, if you were making a Speechweb application and have named the four component files "testApp.xml", "testApp.cgi", "testApp.jsgf", and "testApp", you would change both of the APPNAME placeholders to "testApp". That way, since only the suffixes of the filenames change, the XML file will know how to find all of the other files. Technically, the XML file itself could be called by a different filename since it does not need to be located by any of the other components. However, it is recommended that you keep your filenames consistent regardless.

The FULLAPPNAME placeholder should be replaced by the Speechweb application's full title. This title will appear at the top of the page while the application is being used. For example, the testApp application's full name could be "Test Application", and so would replace the APPFULLNAME placeholder.

The GREETING placeholder should be replaced by the desired opening statement of your Speechweb application. The way this works is that whenever a user runs your application, they will first be greeted by this opening statement before the query/response process begins. The greeting could be utilized to explain what sort of things the user can say, or just to say hello.

The following image shows how the example Test Application's XML file might look after these changes have been made:



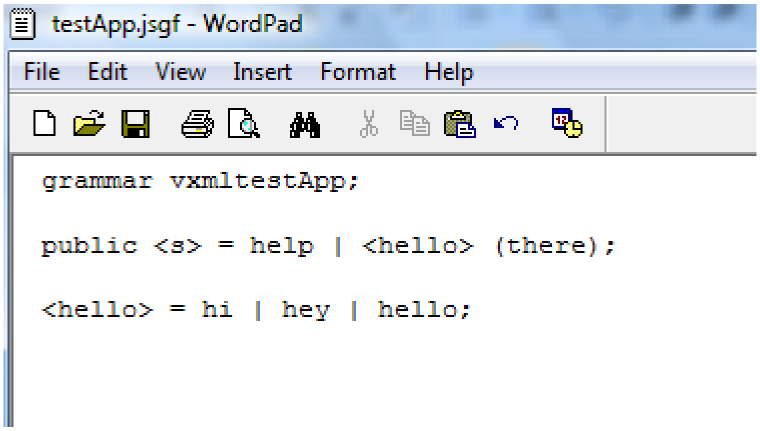
```
<!-- the name of the speechweb application and its opening statement are
<script type="text/javascript">
    var appName = "testApp";
    var appFullName = "Test Application";
    var greeting = "Hello!";

    var link;
</script>


<!-- main vxml form for handling the user/application dialogue -->
<vxml:form id="vxml_main">
    <vxml:field name="vxml_field" modal="true">
        <vxml:grammar type="application/x-jsgf" src="testApp.jsgf" />
```

Additionally, once the file has been changed specifically to suit your Speechweb application, make sure to remember to rename the file "APPNAME.xml", where APPNAME is changed to whatever the other APPNAME placeholders were changed to. In this case, the finished XML file is called "testApp.xml".

The JSGF grammar file

The JSGF file is used to specify the complete spectrum of valid voice input that can be received by the Speechweb application from the user. JSGF stands for Java Speech Grammar Format, and although it is used by Speechweb applications, it is not specific to Speechweb, and so there are many external resources on how to create JSGF grammar files. Because of this, this guide will not go into great detail on how the JSGF grammar is constructed. The following webpage explains the entire grammar format in full detail: http://www.w3.org/TR/jsgf. It may also be helpful in learning how to write JSGF grammars by looking over the grammars of some of the many existing Speechweb applications.

As a very simple example though, the following could be used as a grammar for our Test Application that allows the user to either say hello in a few different ways or ask the application for help:

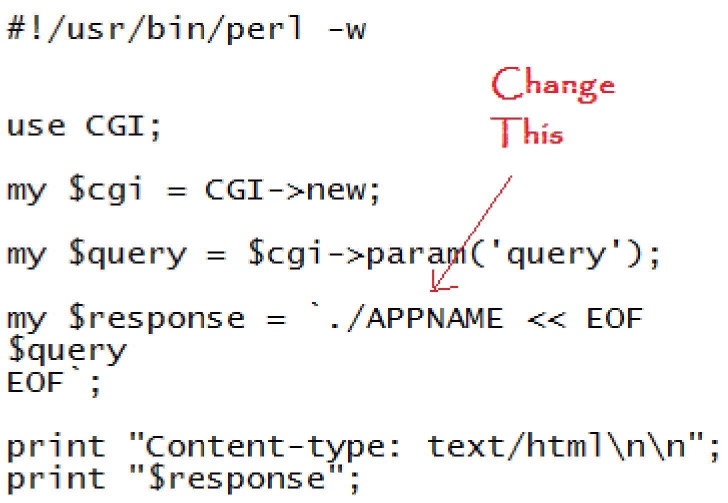


```
grammar vxmltestApp;

public <s> = help | <hello> (there);

<hello> = hi | hey | hello;
```

Notice that the file is called “testApp.jsgf”. This is what will allow the XML page to find the correct grammar to use.

The CGI script

The CGI file contains a simple script which is called on by the XML page. It directs the user's input to the main executable program, and then redirects the output from that program back to the XML page to be spoken by the Speechweb application. Like the XML page, a template for the CGI script can be found at http://speechweb.cs.uwindsor.ca/pages/manual/template/templatecgi.txt, and only the one APPNAME placeholder needs to be changed, as can be seen in the following image:



```
#!/usr/bin/perl -w

use CGI;

my $cgi = CGI->new;

my $query = $cgi->param('query');

my $response = `./APPNAME << EOF
$query
EOF`;

print "Content-type: text/html\n\n";
print "$response";
```

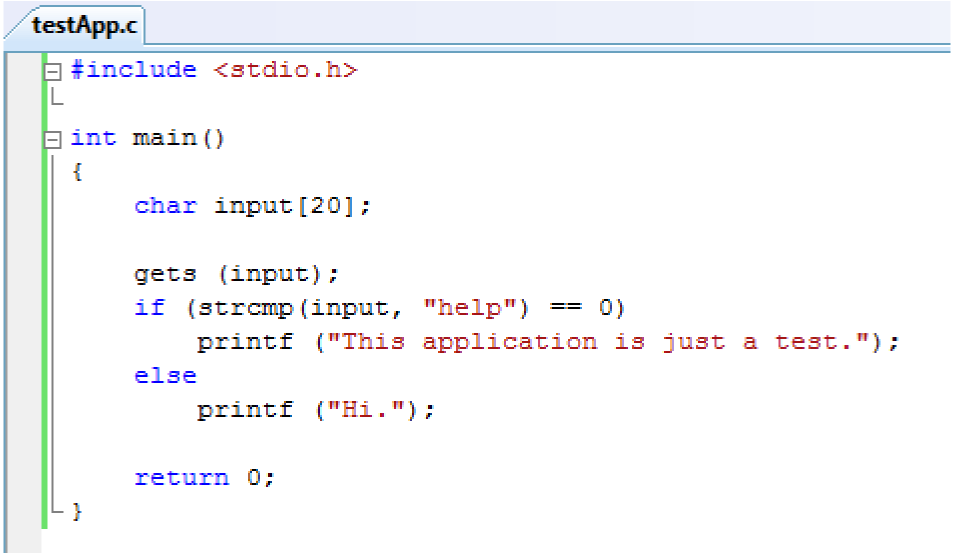
Once that is changed, all that needs to be done is to rename the file "APPNAME.cgi" (and again change the APPNAME placeholder, just as you have been so far).

The executable program

The executable program is the driving force behind any Speechweb application, and so could be considered the most important of the four main components. This program will take in all input and provide all output for your application. This also means that it is up to this program alone to process the input in entirety and produce all of the desired output. Luckily, this program can be written in just about any programming language, so long as it can be executed by the CGI script.

For this very basic voice-in/voice-out Speechweb application, the executable program must conform to a similar text-in/text-out format. The program must take in only text input, and should be able to handle all possible input that can be received based on the defined grammar within the JSGF file. Also, so that the Speechweb application is able to properly speak the output to the user, the output should be text in the format of regular speech. If not, Speechweb will still attempt to speak the output, but it may just sound like nonsense.

For the example Test Application, I have written a small C program that is able to handle any input the user can speak based on the JSGF grammar shown on Page 9. Its code is shown here:



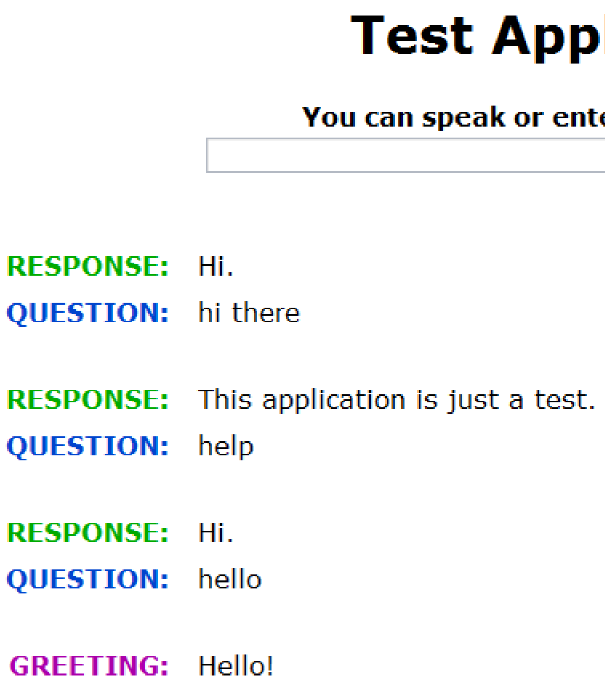
```
#include <stdio.h>

int main()
{
    char input[20];

    gets (input);
    if (strcmp(input, "help") == 0)
        printf ("This application is just a test.");
    else
        printf ("Hi.");

    return 0;
}
```

Once the program is compiled, the executable must be named “testApp”.

Bringing it all together

Once the XML webpage, JSGF grammar file, CGI script, and executable program have been created, they must all be placed in the same folder on a web server that has CGI capability (which, if you are a computer science student at the University of Windsor, would be done by creating a folder in your 'public_html' directory on your user account on the computer science servers). Make sure that the permissions on all of the components of the Speechweb application are set to 755, including the folder that contains them. This must be done so that the application can be run by users over the internet. Once this is done, the Speechweb application is complete, and it can be used by running the XML webpage on the web server from Opera 9.10. This is the only browser so far that can properly run Speechweb applications and it can be downloaded from http://speechweb.cs.uwindsor.ca/Opera_9.10_Eng_Setup.exe. You will also need to activate Opera's Voice capabilities by downloading IBM's speech plugin. This is done by selecting "Tools" > "Preferences" > "Advanced" > "Voice", and then enabling the voice option. This will prompt the download and installation of the speech plugin.

The completed Test Application can be run from http://speechweb.cs.uwindsor.ca/pages/manual/testApp/testApp.xml, but make sure that it is ran from Opera 9.10. Its basic functionality is shown in the following image:



Test Application

You can speak or enter your input here:

RESPONSE: Hi.
QUESTION: hi there

RESPONSE: This application is just a test.
QUESTION: help

RESPONSE: Hi.
QUESTION: hello

GREETING: Hello!

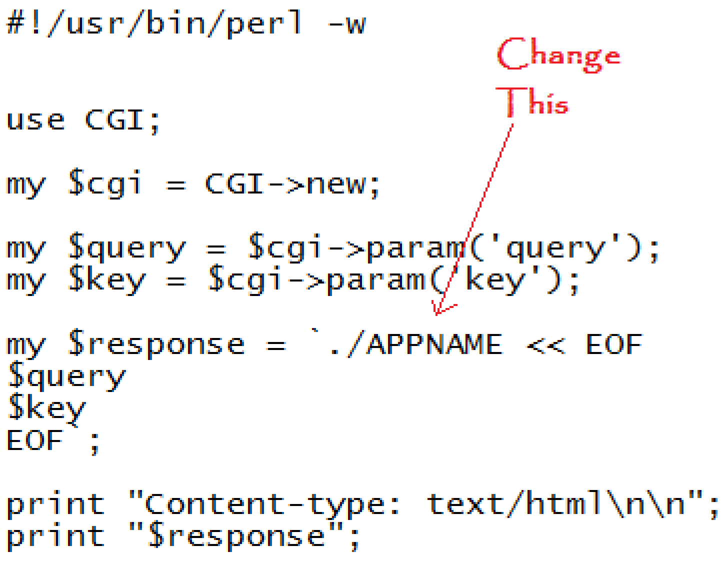
Creating a more advanced Speechweb application

Introduction

Although the simple voice-in/voice-out type Speechweb application can be used for many situations, sometimes you will find that you will want to do more with your application. For instance, creating an application that can remember information from past uses or distinguish between different users is beyond the capabilities of an application created merely in the manner outlined so far in this guide. These two very important techniques will be explained in this section. Firstly, though, a slightly different CGI script must be used in order to keep track of different users. The XML page and JSGF grammar file can be used in exactly the same way as with a voice-in/voice-out Speechweb application.

The CGI script

The CGI script that will now be used is treated in exactly the same manner as it was before, only from a slightly different template, which can be found at http://speechweb.cs.uwindsor.ca/pages/manual/templateadvcgi.txt.



```
#!/usr/bin/perl -w

use CGI;

my $cgi = CGI->new;

my $query = $cgi->param('query');
my $key = $cgi->param('key');

my $response = `./APPNAME << EOF
$query
$key
EOF` ;

print "Content-type: text/html\n\n";
print "$response";
```

This new CGI script contains two additional lines that send an identification key along with each query to the executable program. The XML webpage generates this key for each user and saves it in his/her browser cookies. Each user has a unique key, thereby allowing the program to distinguish which user it is receiving input from. This way, applications that need to save and recall information can do so for each user separately. Without the identification key, the executable program would have no way of recognizing which user it was currently dealing with, and could get even more confused if multiple users were sending queries to it simultaneously.

Note that if you only need to save data, but have no reason for your application to distinguish between different users, then you can continue to use the simplified CGI script outlined on Page 10.

How to save data between queries

With a normal Speechweb application, each query sent in by the user simply results in the executable program being run with the query as the input, having the output collected, and then terminating. Thus, when the next query comes, the program is then executed again without any information transferred over from the previous execution. All stored variables and other data will be gone by the time the response is returned to the user. This means that no real conversation or progression of output can be achieved since no data is being exchanged or saved between executions. There is, however, a simple solution. Files can be created and then accessed repeatedly by every execution of the program. Using this technique, a Speechweb application can actually collect information from the user, and then apply that information later on in the conversation. You can even use files to exchange information between multiple Speechweb applications. This is what makes files ideal for saving any kind of data.

How to distinguish between users

Even if you are using a save file to carry data over between queries, the Speechweb application will be using the same saved data for queries from all users, not just the user who supplied the data in the first place. In some cases, this could lead to unwanted functionality in the application. However, using this new CGI script, the program will now not only take in the input query from the user, but also a unique identifying key which will set each user apart from the others. In order to do this, the program must read in two lines. The first line will contain the input from the user, and the second will contain the unique identifying key. This key is stored in a cookie on each user's browser, and will stay the same for each user indefinitely, even after closing and reopening the application (so long as the cookies are not cleared). Because of this, Speechweb applications utilizing this identifying key can store specific data for multiple users for an unlimited amount of time. This can be done using the file storage technique, but with a different file for each user. The easiest way to do this would be to simply create one file for each user, and name the files using the identifying keys. A good example of this is the Tic-Tac-Toe Speechweb application, which stores game data separately for all users by using the multiple key file technique. This is what allows it to run games for multiple users from the same executable program simultaneously.

Special Output

HTML

When the XML webpage takes the text output from the executable program and displays it to the user, it actually displays it as HTML. Thus, HTML tags can be inserted into the output from the program to increase the graphical capabilities of your Speechweb application. For example, the Artist application uses image tags to display its drawings, and the Tic-Tac-Toe and Tile Puzzle Game applications both use table tags to display their playing fields.

In addition to all the standard HTML tags, the XML webpage has been programmed to recognize a few special tags that can be inserted into the text output in the same way. These special tags will be outlined and explained in the sections that follow.

<asay>

The <asay> tag specifies text that should not be spoken. For example, if the executable program were to return an output of “Look at these numbers<asay>: 1 2 3 4 5</asay>”, the user would see “Look at these numbers: 1 2 3 4 5”, but only “Look at these numbers” would be spoken. The <asay> tag can be useful in displaying things to the user that might not necessarily make sense to be spoken.

<ashow>

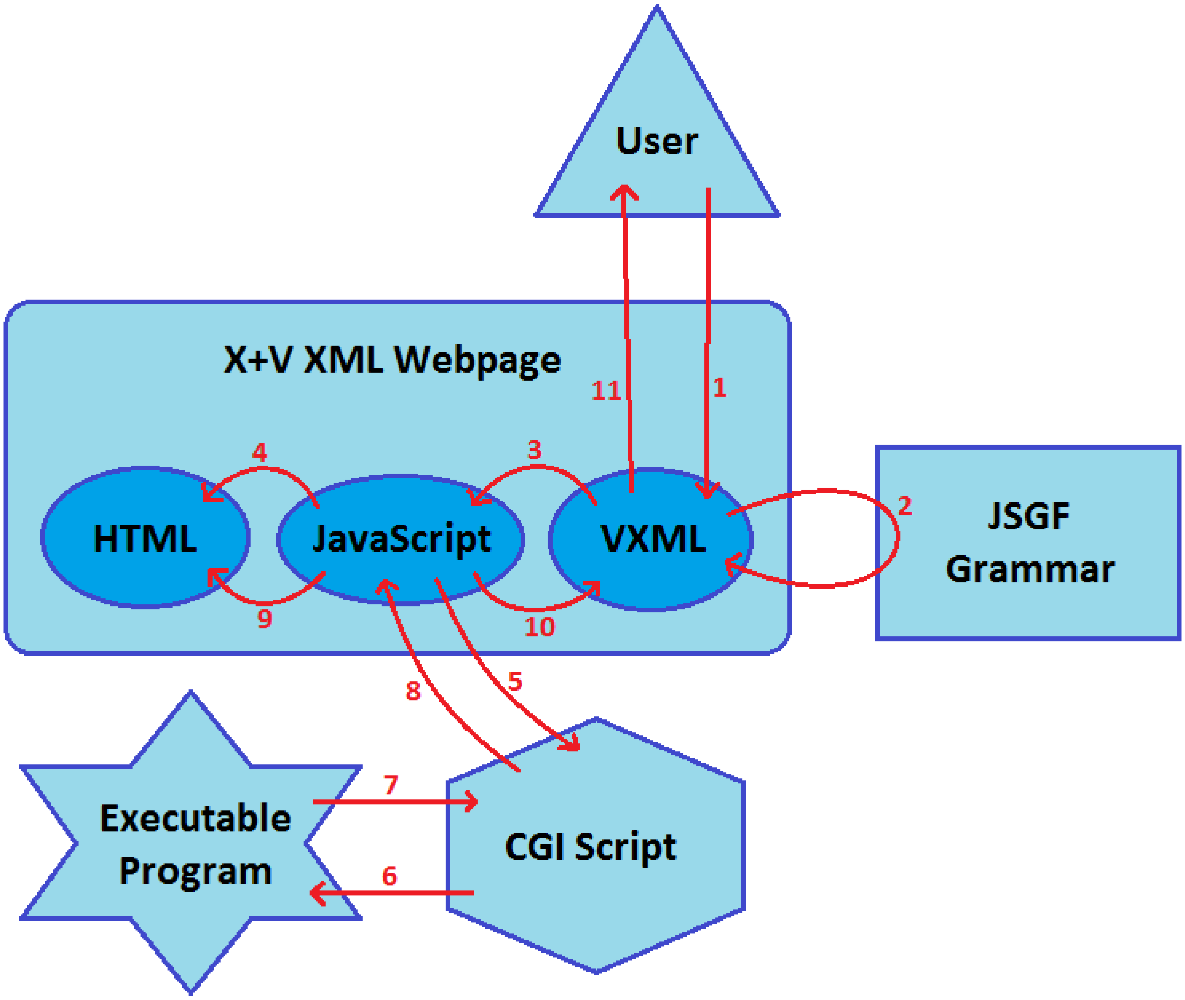
The <ashow> tag is very similar to the <asay> tag, except that it specifies text that is not to be shown. For example, if the executable program were to return an output of “one<ashow> is the number that comes before two”, then the application would speak “one is the number that comes before two” to the user, but the user would only see “one” displayed. The <ashow> tag is useful for allowing the application to say something about what is being displayed, without crowding the response with the full text that is being spoken.

<goto>

The <goto> tag is used to travel to another webpage. Its usage pattern is <goto wait=*time*>*URL*</goto>, with *URL* replaced by the address of the webpage you want to go to, and *time* replaced by the number of milliseconds you want the application to wait before transferring the user to the new page. The <goto> tag is useful for linking Speechweb applications together, and is the tag that is used to navigate the main Speechweb menus at http://speechweb.cs.uwindsor.ca/applications/. All other output outside the <goto> tag will still be handled normally by the Speechweb application, so the ‘wait’ specifier should be used to keep the application from transferring the user to the next page before it has finished speaking the response.

How Speechweb works

The following diagram outlines conceptually the order of events that unfold for each speech command supplied by the user to a typical Speechweb application.



1. The user issues a speech command, which is collected by a VXML form.
2. This speech command is compared to the JSGF grammar file so that it may be converted to text.
3. This text is sent to a JavaScript function that is called by the VXML form.
4. The JavaScript function sends the text to the HTML section, where it is displayed as a question.
5. The JavaScript function also sends the text to the CGI script (and possibly a key along with it).
6. The CGI script runs the executable program with this text as input (and possibly the key as well).
7. The executable program returns text output to the CGI script and then terminates.
8. The CGI script returns this text to the JavaScript function.
9. The JavaScript function sends the text to the HTML section, where it is displayed as a response.
10. The JavaScript function ends and returns the text to the VXML form it was called from.
11. The VXML form converts this text to speech, which it then speaks to the user.