CHAPTER 2

Creating Applications That Talk

In 1939, Bell Labs demonstrated a talking machine named "Voder" at the New York World's Fair. The machine was not well received because the voice was robotic and unnatural sounding. Since then, many advances have been made in the area of speech synthesis—specifically in the last five years. Also known as text-to-speech, speech synthesis is one of two key technologies in the area of speech applications.

The second technology is speech recognition. For decades, science fiction movies have featured talking computers capable of accepting oral directions from their users. What once existed only in the thoughts of writers and filmmakers may soon become part of everyday life. In the last few years many advances have been made in the area of speech recognition by researchers such as the Speech Technology Group at Microsoft Research.

Speech-based applications have been slowly entering the marketplace. Many banks allow customers to access their account data through automated telephone systems, also known as Interactive Voice Response (IVR) systems. Yahoo and AOL have set up systems that read e-mail to their users. The National Weather Service (NOAA) has an application that reads the weather.

Speech processing is an important technology for enhanced computing because it provides a natural and intuitive interface for the user. People communicate with one another through conversation, so it is comfortable and efficient to use the same method for communication with computers.

Recently Microsoft released Speech Server as part of an effort to make speech more mainstream. Microsoft Speech Server (MSS) has three main components:

1. Speech Application SDK (SASDK)
2. Speech Engine Services (SES)
3. Telephony Application Services (TAS)

All three components are bundled into both the Standard Edition and the Enterprise Edition. The primary difference between the two depends on how many concurrent users your application must support.

Speech Engine Services (SES) and Telephony Application Services (TAS) are components that run on the Speech Server. The Speech Server is responsible for interfacing with the Web server and the telephony hardware. Web-based applications can be accessed from traditional Web browsers, telephones, mobile phones, pocket PC's, and smart phones.

This chapter will focus primarily on specific components of the SASDK, since this is the component most applicable to developers. The installation files for the SASDK are available as a free download from the Microsoft Speech Web site at http://www.microsoft.com/speech/. Chapters 3 and 4 will expand on the use of the SASDK and will introduce two fictional companies and the speech-based applications they developed.

The Microsoft Speech Application SDK (SASDK)

The Microsoft Speech Application SDK (SASDK), version 1.0 enables developers to create two basic types of applications: telephony (voice-only) and multimodal (text, voice, and visual). This is not the first speech-based SDK Microsoft has developed. However, it is fundamentally different from the earlier ones because it is the first to comply with an emerging standard known as Speech Application Language Tags, or SALT (refer to the "SALT Forum" profile box). Run from within the Visual Studio.NET environment, the SASDK is used to create Web-based applications only.

Speech-based applications offer more than just touch-tone access to account information or call center telephone routing. Speech-based applications offer the user a natural interface to a vast amount of information. Interactions with the user involve both the recognition of speech and the reciting of static and dynamic text. Current applications can be enhanced by offering the user a choice to utilize either traditional input methods or a speech-based one.

Development time is significantly reduced with the use of a familiar interface inside Visual Studio.NET. Streamlined wizards allow developers to quickly build grammars and prompts. In addition, applications devel-

oped for telephony access can utilize the same code base as those accessed with a Web browser.

The SASDK makes it easy for developers to utilize speech technology. Graphical interfaces and drag-and-drop capabilities mask all the complexities behind the curtain. All the .NET developer needs to know about speech recognition is how to interpret the resulting confidence score.

SALT Forum

Founded in 2001 by Cisco, Comverse, Intel, Microsoft, Philips, and ScanSoft, the SALT Forum now has over seventy contributors and adopters. Their collaboration has resulted in the creation and refinement of the Speech Application Language Tags (SALT) 1.0 specification. SALT is a lightweight extension of other markup languages such as HTML and XML. The specification standardizes the way devices such as laptops, personal digital assistants (PDA's), phones, and Tablet PC's access information using speech. It enables multimodal (text, voice, and visual) and telephony (voice-only) access to applications.

The forum operates a Web site at www.saltforum.org which provides information and allows individuals to subscribe to a SALT Forum newsletter. The site also provides a developer forum that contains a download of the latest specification along with tutorials and sample code. Membership is open to all, and interested companies can download a SALT Adopters agreement.

SALT is based on a small set of XML elements. The main top-level outputs/inputs are as follows:

- <prompt…>—A prompt is a message that the system sends to the user to ask for input. This tag is used to specify the content of audio output and can point to prerecorded audio files. It contains the subqueue object used to specify one or more prompt objects.
- <listen…>—Input element used for speech recognition or audio recording. It contains the grammar element used to specify the different things a user can say. The record element is used to configure the recording process.
- <dtmf…>—Short for Dual Tone Multi-Frequency tones. Element used in telephony applications. It is similar to the listen element in that it specifies possible inputs. Like the <listen> element, its main elements are grammar and bind.
- <smex…>—Short for Simple Messaging Extension. Asynchronous element used to communicate with the device. Can be used to receive XML messages from the device through the received property.

(*continued*)

SALT supports different browsers by allowing for two modes of operation, object and declarative. The object mode exposes the full interface for each SALT element but is only supported by browsers with event and scripting capabilities. The declarative mode provides a limited interface and is supported by browsers such as those found on portable devices.

NOTE: VoiceXML, 2.x is simple markup language introduced by the World Wide Web (W3C) Consortium (http://www.w3.org). Like SALT, it is used to create dialogs with a user using computer-generated speech. They are both based on W3C standards.

The VoiceXML specification was created before SALT and was designed to support telephony applications. SALT was designed to run on a wide variety of devices, including PDA's, smartphones, and Tablet PC's.

SALT has a low-level API, and VoiceXML has a high-level API. This allows SALT a finer-level control over the interface with the user.

VoiceXML does not natively support multimodal applications and is used primarily for limited IVR applications. Because of this, Microsoft Speech Server does not support VoiceXML. But everything that can be accomplished with VoiceXML can be accomplished with SALT.

Telephony Applications

The Microsoft Speech Application SDK enables developers to create telephony applications, in which data can be accessed over a phone. Prior to the Speech Application SDK, one option for creating voice-only applications was the Telephony API (TAPI), version 3.0, that shipped with Windows 2000. This COM-based API allowed developers to build interactive voice systems. The TAPI allowed developers to create telephony applications that communicated over a Public Switched Telephone Network (PSTN) or over existing networks and the Internet. It was responsible for handling the communication between telephone and computer.

Telephony application development would further incorporate the use of the SAPI (Speech Application Programming Interface), version 5.1, to provide speech recognition and speech synthesis services. This API is COM based and designed primarily for desktop applications. Like TAPI, it does not offer the same tools and controls available with the new .NET version. Most important, the SAPI is not SALT compliant and therefore does not utilize a common platform.

Telephony applications built with the SASDK are accessed by clients using telephones, mobile phones, or smartphones. They require a third-party Telephony Interface Manager (TIM) to interpret signals sent from the telephone to the telephony card. The TIM then communicates with Telephony Application Services (TAS), the Speech Server component responsible for handling incoming telephony calls (see Figure 2.1). Depending on which version of Speech Server is used, TAS can handle up to ninety-six telephony ports per node, with the ability to add an unlimited number of additional nodes.

Telephony applications can be either voice-only, DTMF (Dual Tone Multi-frequency) only, or a mixture of the two. DTMF applications involve the user pressing keys on the telephone keypad. This is useful when the user is required to enter sensitive numerical sequences such as passwords or account numbers. In some cases, speaking these types of numerical sequences might entail a security violation, because someone might overhear the user.

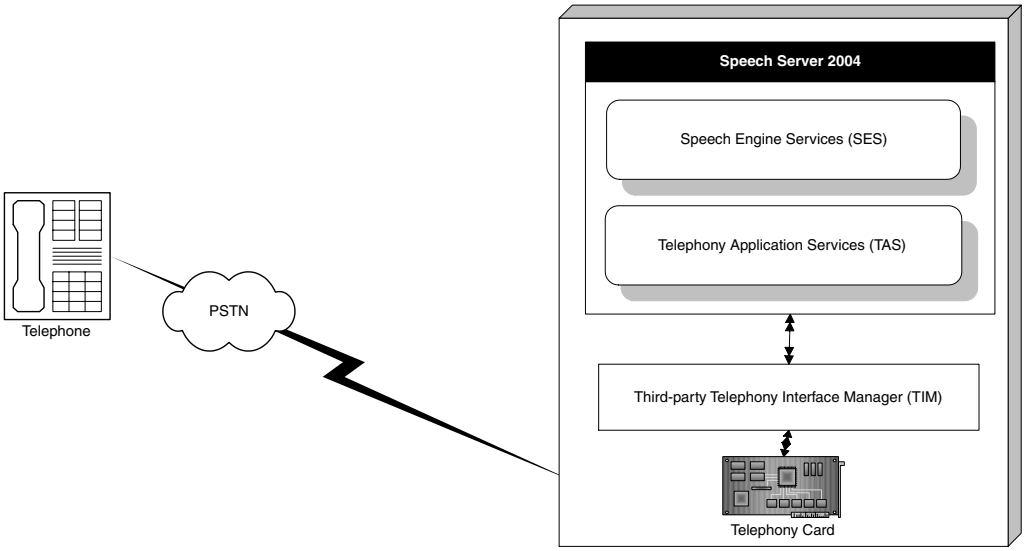


Figure 2.1 The main components involved when telephony applications are received. The user's telephone communicates directly with the server's telephony card across the public telephone network. The Third-party Telephony Interface Manager (TIM) then communicates with Telephony Application Services (TAS), a key component of Speech Server 2004.

Call centers typically use telephony applications to route calls to appropriate areas or to automate some basic function. For instance, a telephony application can be used to reset passwords or request certain information. By automating tasks handled by telephone support employees, telephony applications can offer significant cost savings.

Telephony applications can also be useful when the user needs to iterate through a large list of information. The user hears a shortened version of the item text and can navigate through the list by speaking certain commands. For example, if the telephony application is used to recite e-mail, the user can listen as the e-mail subjects of all unread e-mails are recited. A user who wants to hear the text of a specific e-mail can speak a command such as "Read e-mail." The user can then navigate through the list by speaking commands such as "Next" or "Previous."

Multimodal Applications

Multimodal applications allow the user to choose the appropriate input method, whether speech or traditional Web controls. The application can be used by a larger customer base because it allows the user to choose. Since not all customers will have access to microphones, the multimodal application is the perfect way to offer speech functionality without forcing the user into a corner.

Multimodal applications are accessed via Microsoft Internet Explorer (IE) on the user's PC or with IE for the Pocket PC (see Figure 2.2). Both versions of IE require the installation of a speech add-in. Users indicate that they wish to utilize speech by triggering an event, such as clicking an icon or button.

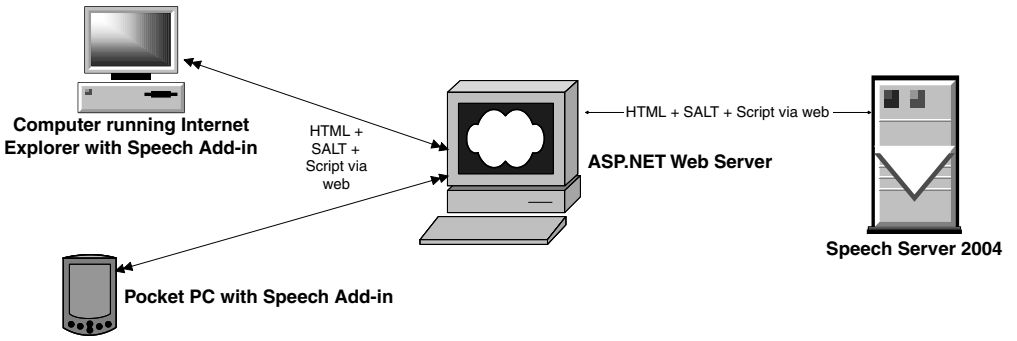


Figure 2.2 The high-level process by which multimodal applications communicate with Speech Server. The ASP.NET application is accessed either by a computer running Internet Explorer (IE) with the speech add-in or by Pocket IE with the speech add-in.

The speech add-in for IE, necessary for interpreting SALT, is provided with the SASDK. It should be installed on any computer or Pocket PC device accessing the speech application. In addition to providing SALT recognition, the add-in displays an audio meter that visually indicates the volume level of the audio input.

Business Benefits of Speech

Self-service applications are on the rise in just about every industry. Customers rarely tolerate anything less than 24/7 access to their information. Speech makes it easier and sometimes faster for customers to get to that information. This can be accomplished using either telephony or multimodal applications.

Telephony applications are great for automating call center functionality. They generally result in increased customer satisfaction and lower costs. If designed properly, a telephony application can reduce the number of telephone transfers or menu layers that a customer must navigate. Telephony applications can also reduce the hold time for customers because they will not have to wait on the availability of a limited number of customer service agents. Best of all, you can be sure that the customer is always treated courteously. You do not have to worry about the customer reaching a rude or untrained customer service agent.

Multimodal applications offer the better of two worlds. They increase customer satisfaction by allowing customers to select the input method that works best for them. For applications that execute on small devices like a PDA or a smartphone, being able to swap between speech and GUI access can speed access time.

Both multimodal and telephony applications can automate certain processes. For instance, an information services department can use a speech-based application to reset passwords, access e-mail, or customer contact information.

Hands-free access is not only useful for busy remote workers, but can be vital to handicapped individuals. Best of all, mobile workers are not limited to using only stylus pens when entering data.

The benefits of speech processing are numerous, especially for businesses. Hardware and software limitations that once hindered the technology are being lowered or removed everyday. Products like Microsoft Speech Server will increase the number of developers with the skills to

build these types of applications. As companies search for innovative ways to increase customer satisfaction and decrease costs, speech-based applications should become more and more common.

How the Speech Engine Works

If you have ever utilized a dictation program to create documents, you know that the program requires a training process to accurately interpret your speech. The training process usually consists of several sessions where the user reads prompts slowly and clearly. The speech engine matches the sounds recorded to the words in the prompt script. Unfortunately, this process usually discourages the use of speech-based products.

The speech engine that is part of the Speech Application SDK works differently. It utilizes what are called grammar rules to interpret what the user is saying. Rather than requiring the engine to be open to any spoken text, the speech application is restricted to certain phrases defined in multiple grammar files. Each grammar file is associated with a particular user interaction.

While this enables the application to accurately interpret the user's speech without a training process, it does require more work by the application developer. The developer must try to anticipate every spoken phrase the user may utter. This is not as much of an issue for the sample application in this chapter. Trained employees and not the general public will use the telephony application. Therefore, it is easier to anticipate and restrict what phrases are appropriate.

The Speech Application SDK allows for the use of dynamic grammars. This is necessary when the content is not known ahead of time, as when the user is reading values from the database. This process will be even more relevant in Chapter 4, where an application used to register students will be examined.

In general, applications built with the Speech Application SDK are least like "real AI," in which the system can act and react like a human would. Even though they offer the user a natural interface, the inputs and outputs must be defined in advance. Nevertheless, they are still an important step toward better enabling mobile workers. These applications offer an enhanced method for accessing data that would not be available using traditional applications.

Installing the SASDK

To run the samples provided in the next two chapters, you will need to install the SASDK. You will first need to download the SDK from the Microsoft Speech Web site at http://www.microsoft.com/speech/. Installation will require completion of the following steps:

1. Ensure that you have a minimum of 702 MB of free hard disk space. This will be needed for the contents of the Setup directory.
2. Double-click **SASDK_V1_Full.exe** and allow it to extract files to C:\SpeechSDK.
3. Double-click the **SASDK_Extract.exe** file to extract additional installation files to C:\SpeechSDK.
4. Go to a command prompt by clicking **Start** and **Run**, type **CMD**, and then click **OK**.
5. Browse to the C:\SpeechSDK directory and type '**SASDK_ExtractAll.cmd C:\SpeechSDK\Setup**' This step will create the setup directory and extract all the necessary installation files. This step may take several minutes to execute.
6. Execute **Setup.exe** from the newly created C:\SpeechSDK\Setup directory and click **Install the Speech Application SDK.**
7. From the Installation Guide dialog box click, **Detect and Install Prerequisites**. The Prerequisites dialog (see Figure 2.3) will list all the necessary components along with either a checkmark indicating that the component is found or a red x to indicate that the component was not found. Click the red x icon to begin updating your computer.
8. Once all the prerequisites have been installed, click **Install the Speech Application SDK** and follow the wizard steps for a complete install until the SDK installation is finished. The installation will take several minutes to complete.
9. Click the **Check for the latest Service Release** link. Requiring an Internet connection, this step will direct the user to the area where all speech-related downloads are located.
10. Once all the steps have been completed, click **Exit** to leave the installation wizard.

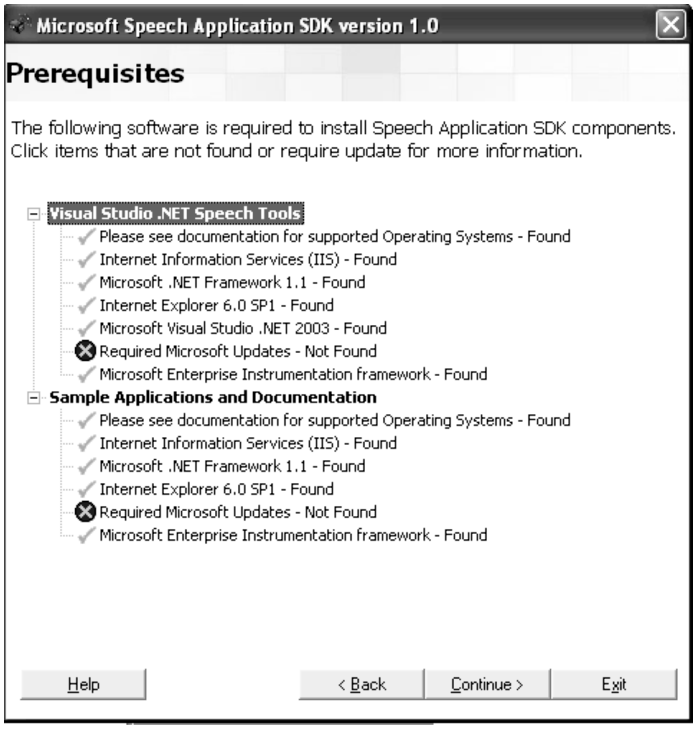


Figure 2.3 Screenshot of the Prerequisites dialog featured in the SASDK installation wizard. This dialog not only allows users to determine which components are missing from their computer, but provides a link to the installation for the missing component. In this example the wizard indicates that one of the required updates was not found.

Creating a Speech Application

The SASDK provides a template for creating new speech applications with Visual Studio .NET. It also provides visual editors for building the prompts (words spoken to the user) and grammars (words spoken by the user). This section will examine the basics of creating a speech application with the SASDK.

To utilize the template provided with the SASDK, open Visual Studio.NET and execute the following steps:

1. Click **File**, **New**, and **Project**. From the **New Project** dialog box, select the desired Project Type and click the **Speech Web Tem-**

plate icon in the Templates window. This template was created when you installed the SASDK. Change the value in the location dropdown box to the desired project name and click **OK**.

2. You can either accept the setting defaults and click **Finish**, or select **Application Settings** and **Application Resources** to specify custom settings.
3. The default application mode is voice-only, so if you want to create a multimodal application, you can change the mode from the Application Settings tab.
4. The Application Resources tab allows you to specify that a default grammar library file will be created and the name it will be called. From here you can also indicate that a new prompt project will be created and specify what the name for it will be.
5. Click Finish at any time to build the new project.

If you choose to build a voice-only application, the project will include a Web page named **Default.aspx**. This page contains two speech controls, AnswerCall and SemanticMap. These are basic controls used in every voice-only application. Their specific functions will be covered in the section titled "Using Speech Controls." The default project will also include a folder named **Grammars** that contains two grammar files, Library.grxml and SpeechWebApplication1.grxml. For voice-only applications the prompt project and Grammars folder are included by default.

If you choose to build a multimodal application, the Default.aspx page is included, but it will contain no controls. There will be a Grammars folder, but no prompt project will be created.

By default, the Manifest.xml file is included for both project types. It is an XML-based file that contains references to the resources used by the project. References include grammar files and prompt projects. Speech Server will preload and cache these resources to help improve performance.

The Prompt Editor

Microsoft recommends that you prerecord static prompts because voice recordings are more natural than the result of the text to speech engine. The prompt editor (see Figure 2.4) is a tool that allows you to specify potential prompts and record wave files associated with each prompt.

The utterance "Welcome to my speech application" represents a single prompt. For voice-only applications, you need to make sure you include a

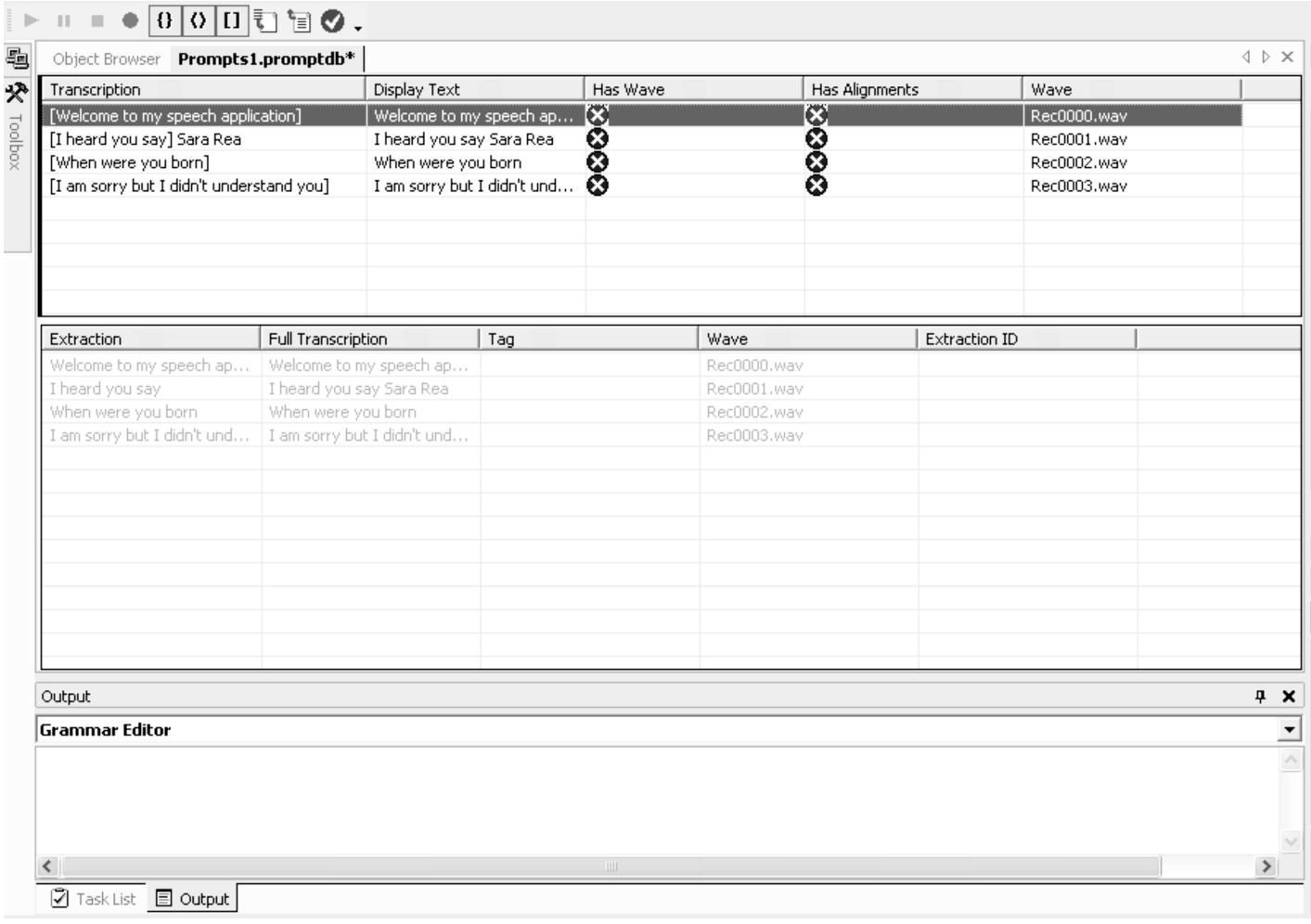


Figure 2.4 Screenshot of the prompt editor in the prompt database project. The prompt editor is used to record the wave files associated with each prompt. The screenshot includes four different prompts.

wide range of prompts. Since the user relies on these prompts to understand how the application works, they need to be clear and meaningful.

The Prompt Database

An application built with the Speech SDK wizard adds a prompt database project by default. If you choose to add another prompt database, it can be done by using the **File** menu and selecting **Add Project** and **New Project** (see Figure 2.5). The new project will be based on the Prompt Project template. Once the project is added, a new Prompt Database can be added by right-clicking the prompt project and then selecting **Add** and **Add New Item**. The Prompt Database item opens up a data grid style screen that allows you to specify all the potential prompts.

The prompt database contains all the prerecorded utterances used to communicate with the user. An application can reference more than one prompt database. One reason for doing this is ease of maintenance. Prompts that change often can be placed in a separate prompt database. By restricting the size of the prompt database, the amount of time needed to recompile is minimized.

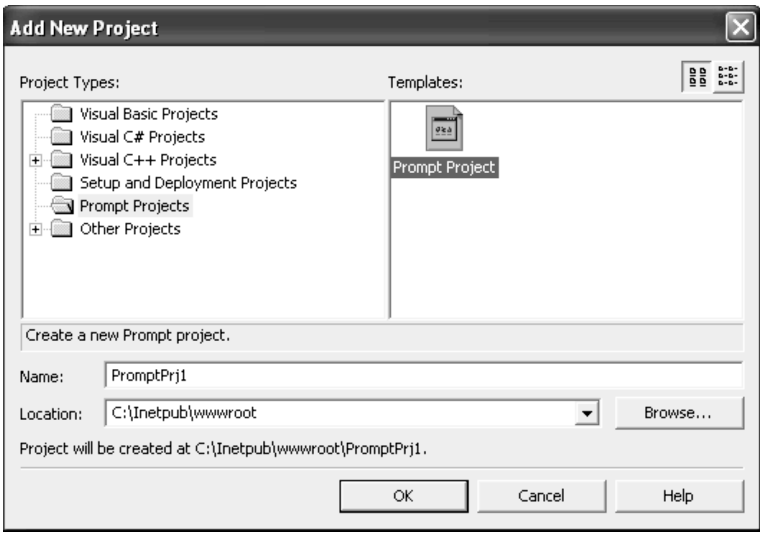


Figure 2.5 Screenshot of the dialog used to add a new prompt project to your speech application. This dialog is accessed by clicking Add project from the File menu and then clicking New Project.

If you followed the instructions in the last section to create a new speech project, you can now open the default prompt database by double-clicking the prompts file from **Solution Explorer**.

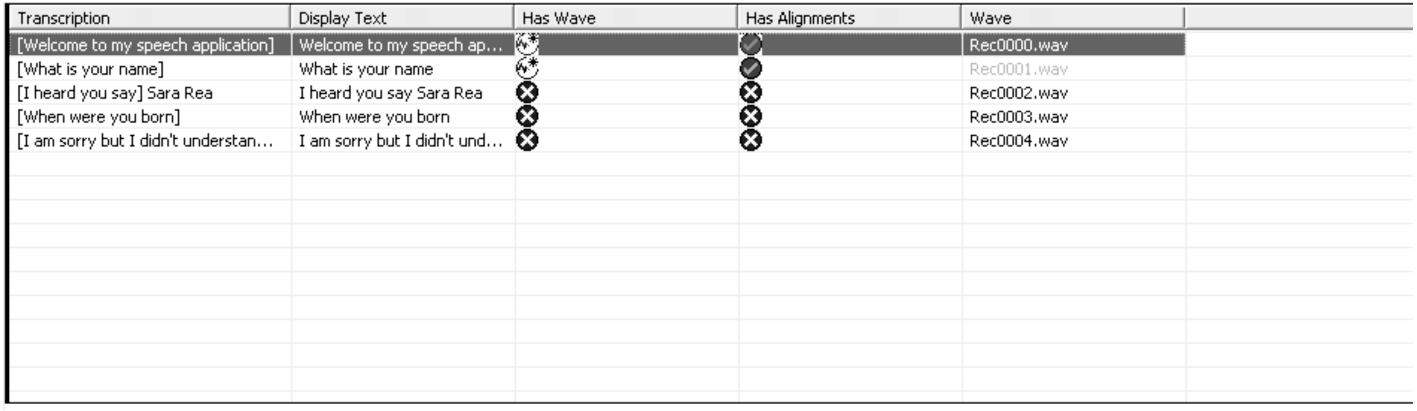
Transcriptions and Extractions

Figure 2.6 is a screenshot of the recording pane in the prompt project database. There are two grids in a prompt project. The top one contains transcriptions, and the bottom one extractions. Transcriptions are the individual pieces of speech that relate to a single utterance. Extractions combine transcription elements to form phrases. Extractions are formed when you place square brackets around the transcription elements.

Sometimes a prompt can involve one or more transcription elements, such as "I heard you say Sara Rea." In this case, the two elements are "I heard you say" and "Sara Rea." In some cases employee names may also be prerecorded in the prompt database. This adds an additional burden, because every time a new employee is added to the database, someone needs to record the employee's name. However, by doing this, we prevent the speech engine from utilizing text-to-speech (TTS) to render the prompt. This is preferred because using recordings results in a more natural-sounding prompt.

Prompts are controlled from prompt functions. These functions programmatically indicate what phrases are spoken to the user. When the speech engine is passed a phrase from the function, it first searches the prompt database to see if any prerecorded utterances are present. It searches the entire database for matches and will string together as many transcription elements as necessary to retrieve the entire phrase.

Because the speech engine parses transcription elements together to form phrases, you can break phrases up to prevent redundancy. For instance, the phrase "Sorry, I am having trouble hearing you. If you need help, say help" may be spoken when an application encounters silence. The phrase "Sorry, I am having trouble understanding you. If you need help, say help" is used whenever the speech engine does not recognize the user's response. Therefore, the subphrase "If you need help, say help" can be recorded as a separate phrase in the prompt database. This means that the subphrase will only have to be recorded once. In addition, the size of the prompt database is minimized.



Transcription	Display Text	Has Wave	Has Alignments	Wave
[Welcome to my speech application]	Welcome to my speech ap...	✓	✓	Rec0000.wav
[What is your name]	What is your name	✓	✓	Rec0001.wav
[I heard you say] Sara Rea	I heard you say Sara Rea	✗	✗	Rec0002.wav
[When were you born]	When were you born	✗	✗	Rec0003.wav
[I am sorry but I didn't understan...	I am sorry but I didn't und...	✗	✗	Rec0004.wav

Figure 2.6 Contents of the Recording pane in the prompt database project. Transcriptions are the individual pieces of speech that can be prerecorded. No utterances have been recorded for prompts with a red X in the Has Wave column.

The Recording Tool

The Recording Tool can be accessed by clicking the red circle icon above the Transcription pane or by clicking **Prompt** and then **Record All.** The text from the transcription item selected is displayed in the Display Text textbox (see Figure 2.7). After clicking **Record**, the person making the recording should speak clearly into the microphone. Click **Stop** as soon as the entire phrase is spoken. Try to select a recording location where background noise is minimized.

In some cases, you may want to utilize professional voice talent to make recordings. There are third-party vendors, such as ScanSoft (see the "ScanSoft" profile box), that can provide professional voice talent and assistance with recordings. Wave files created in a recording studio can be associated with a specific transcription element by clicking **Import** and browsing to the file's location.

If the speech engine is unable to find a match in any of the prompt databases, it utilizes TTS. The result is a machine-like voice that may go against the natural interface you are trying to create. Speech Server comes bundled with ScanSoft's Speechify TTS engine (see the "ScanSoft" profile box), but at present the results from a text-to-speech engine are not as natural-sounding as a recorded human voice. On the other side, it will not always be possible or manageable to prerecord all utterances. You will have to weigh these options when designing your speech application.

ScanSoft, Inc.

In 2003, ScanSoft, Inc. (www.scansoft.com) merged with SpeechWorks to become one of the largest suppliers of speech-related applications and services. SpeechWorks, one of the original founders of the SALT Forum, offered ScanSoft expertise in the areas of speech recognition, text-to-speech (TTS), and speaker verification.

ScanSoft, a publicly traded company (NASDAQ:SSFT), was already a large supplier of popular products, such as Dragon's NaturallySpeaking, which allows users to dictate into any Windows-based application up to 160 words per minute. The merger made the company a dominant force in the area of speech technology.

ScanSoft offers a broad range of products, but also offers such services as assistance in deployment and configuration of speech solutions. In addition, it provides voice talent for companies that wish to have their prompts professionally recorded.

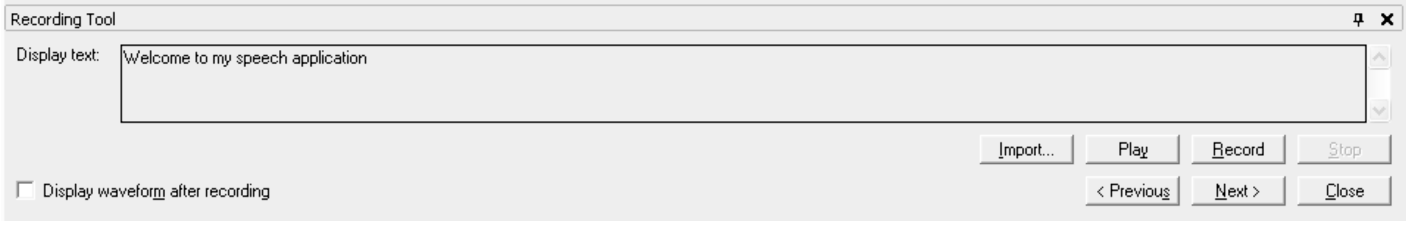


Figure 2.7 The Recording tool allows you to directly record each prompt associated with a transcription. Prompts can also be recorded by professional voice talent in a studio, made into wave files, and imported.

ScanSoft has deployed speech-based solutions to a broad range of industries, including financial services, government, health care, and retail.

Microsoft first licensed the Speechify text-to-speech engine with its Microsoft Speech API (SAPI) 5.0 product in 2001. This was done because of the high quality of the Speechify text-to-speech voice.

In 2002, SpeechWorks and Microsoft formed a strategic alliance which ensured that the Speechify text-to-speech engine would be included with the Microsoft Speech Server product. The Speechify TTS engine is SALT-based and is included out-of-the-box with Speech Engine Services.

Including the Speechify product was important not only because it provides a natural-sounding voice, but also because it supports multiple languages and performs well. Utilizing ScanSoft's TTS engine, Microsoft was able to focus more on perfecting the speech-recognition engine.

At the time of the alliance, SpeechWorks agreed to add support for its OpenSpeech Recognizer (OSR) product. The OSR product is a high-performance recognition engine that can be purchased separately from ScanSoft and then included with the Enterprise Edition of Speech Server.

The recording of prompts is a major consideration when designing a speech-enabled application. If professional talent is used, you will want to try to minimize the need for multiple recording sessions. If the application requires the utilization of text-to-speech for most prompts, you may want to consider purchasing a third-party TTS add-in.

The Grammar Editor

Grammar, the reverse of prompts, represents what the user says to the application. This is a key element of voice-only applications because they rely completely on accurate understanding of the user's commands. The grammar editor builds Extensible Markup Language (XML) files that are used by the speech-recognition engine to understand the user's speech. What is nice about the grammar editor is that you drag-and-drop controls to build the XML instead of having to type it in directly. This helps to reduce the time spent building grammars.

A grammar is stored in the form of an XML file with a grxml extension. Each of its Question/Answer (QA) controls, representing an interac-

tion with the user, is associated with one or more grammars. A single grammar file will contain one or more rules that the application uses to interpret the user's response.

TIP: In order to increase the speed of the application, grammars can be compiled into .cfg files using the grammar compiler. This is typically done after the application is deployed, because that is when you will obtain the most benefit. Using compiled grammar files instead of text files reduces the size of files and thus the amount of time required to download them. Most important, it allows the speech engine to load files into memory faster.

Grammars are compiled with a command-line utility named SrGSGc.exe. It is installed by default in Program Files\Microsoft Speech Application SDK 1.0\SDKTools\bin. The tool can be accessed by clicking Start|Programs|Microsoft Speech Application SDK 1.0|Debugging Tools|Microsoft Speech Application SDK Command Prompt. From the command prompt, type SrGSGc.exe followed by the name of the output file (.cfg) and then the name of the input file (.grxml).

Clicking **Add New Item** from the **Project** menu accomplishes adding a grammar file. From there, select the category **Grammar File** and name the file accordingly. Existing grammars can be viewed by expanding the Grammar folder within Solution Explorer. By default, two grammar files are added when you create a voice-only or multimodal application. The first file, named library.grxml, contains common grammar rules you may need to utilize. For instance, it includes a rule for collecting yes/no responses (see Figure 2.8). It also includes rules for handling numbers, dates, and even credit card information. Rules embedded within the library grammar file can be referenced in other grammar files through the RuleRef control.

The second grammar file is named the same as the project file by default. This is where you will place the grammar rules associated with your application. Although you could store all the rules in a single file, you may want to consider adding subfolders within the main Grammars folder. You can then create multiple grammar files to group similar types of grammar rules. This helps to organize code and makes referencing grammar rules easier.

Grammar rules are built by dragging elements onto the page. Controls are available in the **Grammar** tab of the toolbox. Figure 2.9 is a screenshot of these grammar controls. Most rules will consist of one or all of the following:

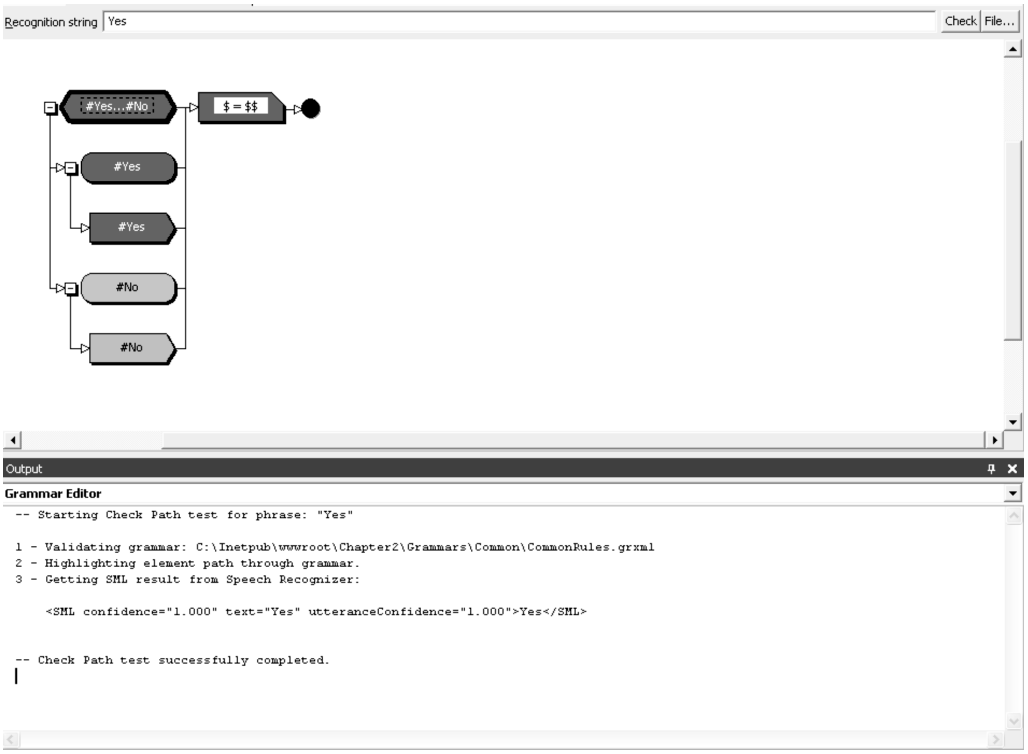


Figure 2.8 Screenshot displaying the yes/no rule inside the grammar editor. This is one of several rules included by default with the Library.grxml file.

Figure 2.9 Screenshot of the Grammar tab, available in the toolbox when creating a new grammar. The elements you will use most often are the List, Phrase, RuleRef, and Script Tag elements.

- Phrase—represents the actual phrase spoken by the user.
- List—contains multiple phrase elements that all relate to the same thing. For instance, a yes response could be spoken as "yeah," "ok," or "yes please." A list control allows you to indicate that all these responses are the same as yes.
- RuleRef—used to reference other rules through the URI property. This is useful when you have multiple grammar files and want to reuse the logic in existing rules.
- Group—used to group related elements. It can contain any element, such as a List, Phrase, or RuleRef.
- Wildcard—used to specify which words in a phrase can be ignored.
- Halt—used to stop the recognition path.
- Skip—used to indicate that a recognition path is optional.
- Script Tag—used to get semantic information from the grammar.

The grammar editor (see Figure 2.8) contains a textbox called **Recognition String**. When dealing with complex rules, it can be used to test the rule without actually running the application. This is very useful when you are building the initial grammar set. To use this feature, just enter text that you would expect the user to say and click **Check**. The output window will display the Semantic Markup Language (SML), which is the XML generated by the speech engine and sent to the application. If the text was recognized, you will see "Check Path test successfully complete" at the bottom of the output window.

TIP: Do not use quotation marks when entering text in the Recognition String textbox. Doing so will cause the speech engine not to recognize the text.

The Script tag element is used to value a semantic item with the user's response. The properties for a script tag include an ellipsis that brings you to the Semantic Script Editor. This editor helps you to create an assignment so that the correct SML result is returned. You can also switch to the Script tab and edit the script directly. Figure 2.10 is a screenshot of the Semantic Script Editor.

When building grammars you will probably not anticipate all the responses on an initial pass. Therefore, grammars require fine-tuning to make the application as efficient and accurate as possible. This process is eased since grammar files are not compiled and instead are available as XML reference files. For this reason, you would not want to compile

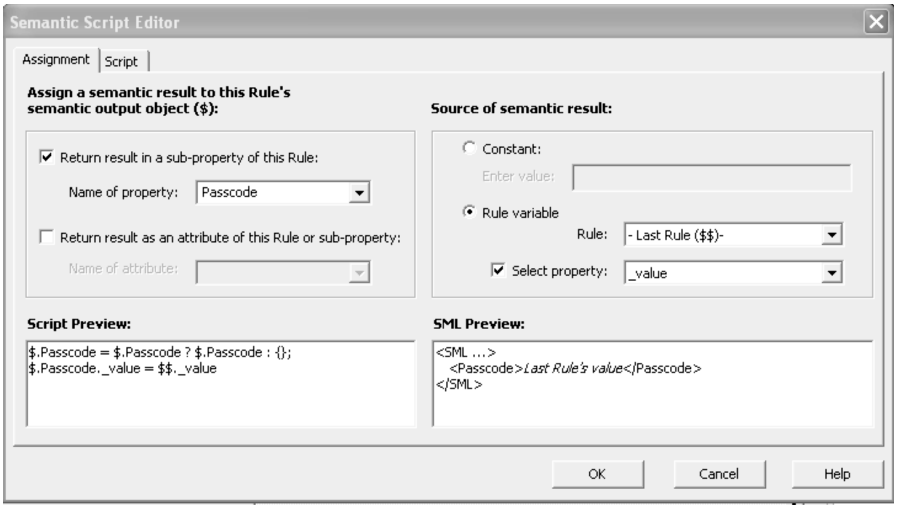


Figure 2.10 Screenshot of the Semantic Script Editor that is available when you use a Script Tag element. The Script Tag is used whenever you need to value a semantic item with the user's response.

grammar files until after the application has been thoroughly tested and is ready to deploy.

Using Speech Controls

A voice-only application has no visible interface. It runs on IIS as a Web page and is accessed with a telephone. When developing and debugging the application, it is executed within the Web browser, and the Speech Debugging Console is used to provide the developer with information about the application dialog. The user will never see the page created, so it is not important what is placed on it visually. Therefore, the only elements on the page will be speech controls, and they will be seen only by the developer.

The Speech Application SDK includes several speech controls that are visible from the Speech tab in the Toolbox. These controls will be dragged onto the startup form as the application is built. Figure 2.11 is a screenshot of the speech controls available in the speech tab of the toolbox. Speech controls are the basic units for computer-to-human interac-

Figure 2.11 Screenshot of all the speech controls available in the speech tab of the toolbox. The QA control is the most basic unit and is utilized in every interaction with the user. SmexMessage, AnswerCall, TransferCall, MakeCall, RecordSound, and DisconnectCall are only applicable for telephony applications.

tion, and the SASDK contains two varieties of controls: dialog and application speech controls.

Dialog Speech Controls

Table 2.1 is a listing of the dialog speech controls used for controlling the conversational flow with the user. A QA control, the most commonly used control, represents a single interaction with the user in the form of a prompt and a response.

Table 2.1 Dialog Speech Controls are used for controlling the conversational flow with the user.

Control Name	Description
Semantic Map	Collection of SemanticItem controls where a SemanticItem control represents a single piece of information collected from the user, such as a last name.
QA	Question/Answer control. This represents one interaction with the user in the form of a question and then a response.
Command	Often used to navigate the application with unprompted commands such as Help or Main Menu.
SpeechControlSettings	Specify common settings for a group of controls.
SmexMessage	Sends and receives messages from a computer-supported telephony application (CSTA) that complies with European Computer Manufacturers Association (ECMA) standards.
AnswerCall	Answer calls from a telephony device. Used for inbound telephony applications.
TransferCall	Transfers a call.
MakeCall	Initiates a new call. Used for outbound telephony applications.
DisconnectCall	Ends a call
CompareValidator	Compares what the user says with some value
CustomValidator	Validates data with client-side script
RecordSound	Records what the user says and copies it to the Web server so it can be played back later.
Listen	Represents the listen element from the SALT specification. Considered a basic speech control.
Prompt	Represents the prompt element from the SALT specification. Considered a basic speech control.

Speech Application Controls

Speech Application Controls are extensions of the basic speech controls. They are used to anticipate common user interaction scenarios. Refer to Table 2.2 for a listing of the application controls included with the SASDK. For instance, the Date control is a speech application control that expands on the basic QA control. It is used to retrieve a date and allows for a wide range of input possibilities. Application controls can reduce development time because much of the user interaction is built directly into them.

Table 2.2 Speech Application Controls available in the Speech tab of the toolbox. These controls can reduce development time by building in typical user interactions.

Control Name	Description
ListSelector	Databound control that presents the user with a list of items and asks user to select one.
DataTableNavigator	Databound control that the user navigates with commands such as Next, Previous, and Read.
AlphaDigit	Collects an alphanumeric string.
CreditCardDate	Collects a credit card expiration date (month and year); does not ensure that it is a future date.
CreditCardNumber	Collects a credit card number and type. Although it does not validate the number, it ensures that the number matches the format for the particular type of credit card.
Currency	Collects an amount in U.S. dollars that falls within a specified range.
Date	Used to collect either a complete date or one broken out into month, day, and year.
NaturalNumber	Collects a natural number that falls within a specified range.
Phone	Collects a U.S. phone number where area code is three numeric digits, number is seven numeric digits, and extension is zero to five numeric digits.
SocialSecurityNumber	Collects a U.S. Social Security number.
YesNo	Collects a yes or no answer.
ZipCode	Collects a U.S. zip code where the zip code is five numeric digits and the extension is four numeric digits.

Creating Custom Controls

If no control does everything you need, you have the option of creating a custom control. Custom controls allow you to expand on the functionality already available with the built-in speech controls. Utilizing the concept of inheritance, custom controls are created using the ApplicationControl class and the IDtmf interface. The developer will create a project file that is compiled into a separate DLL for each custom control.

The Samples solution file, installed with the SASDK, includes a project titled ColorChooserControl. The ColorChooserControl project by itself is installed by default in the C:\Program Files\Microsoft Speech Application SDK 1.0\Applications\Samples\ColorChooserControl directory. This project can serve as a template for any custom control you wish to create. The Color Chooser control is a complex control that consists of child QA controls used to prompt the user for a color and then confirm their selection. The grammar and prompts associated with the control are built directly in. This particular control supports voice-only mode.

The ColorChooserControl is a custom control used to control the dialog flow with the user. It demonstrates what considerations must be made when building these types of controls. It is an excellent starting point for anyone wanting to create custom controls.

Debugging and Tuning a Speech Application

The SASDK provides several tools that can be used to debug and fine-tune your application. These tools allow developers to simulate the user's environment, which is important not only when developing the application, but during testing and deployment. The SASDK also includes logging and reporting tools that can be used to evaluate the impact and effectiveness of the application.

Speech Debugging Console

The Speech Debugging Console is a tool essential for building voice-only applications. It allows you to simulate the user's experience while displaying important information. Figure 2.12 shows a screenshot of the tool.

The **Options** menu allows you to toggle on and off the following: Break on Listen, Break on DTMF, Play prompts, Edit SML, and Show/Hide other windows. When building your application you will typi-

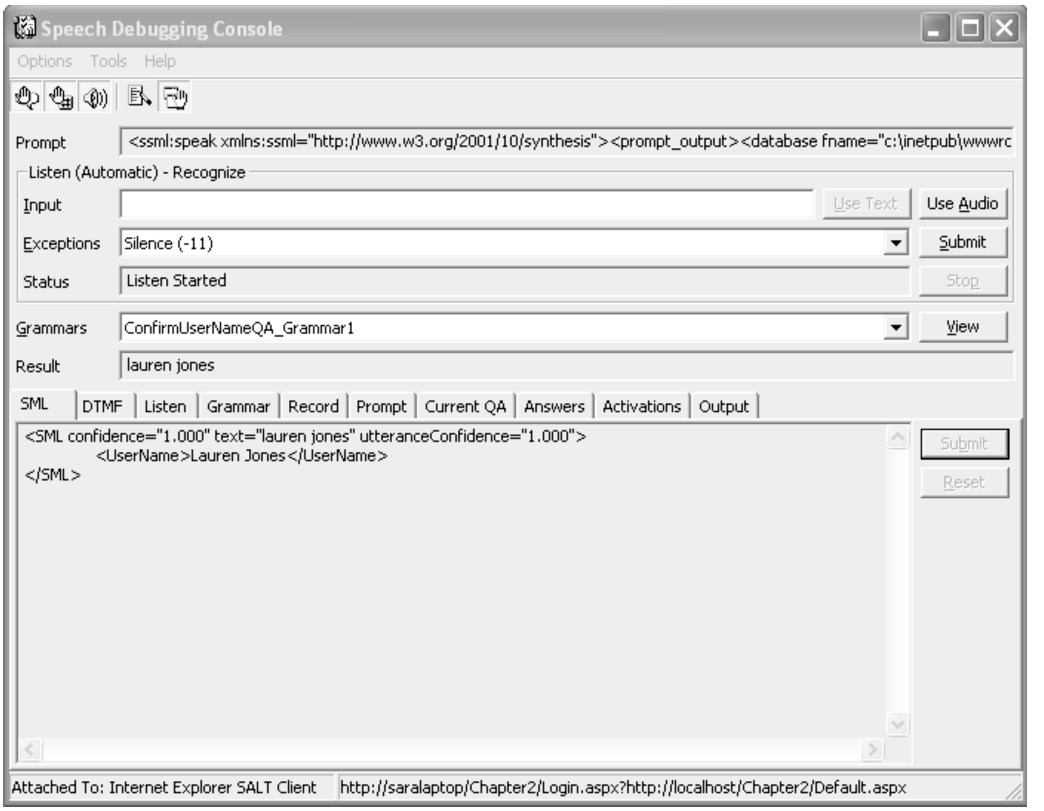


Figure 2.12 Screenshot of the Speech Debugging console after it has processed a QA control. The SML tab contains the SML created by the speech recognition engine. Since text was used instead of real speech in this situation, a confidence score of 1.0, or 100 percent, was assigned.

cally have all these options turned on. Break on Listen and Break on DTMF are important because they allow you time to inspect results within the tabs without being interrupted. Otherwise, your delays would be interpreted as silences and certain events might be triggered.

The Speech Debugging console offers the developer useful debugging information. The **Output** tab is shown by default and will contain a stream of messages returned from the Web server. These messages are critical if you experience an error or unexpected result. You can trace through the output to understand the application's dialog flow.

For each control that is activated, an entry is made inside the **Activations** tab. Whenever the RunSpeech engine tries to activate a control it

places a node inside this tab. From here you can expand the nodes and determine the state of the semantic items associated with each activation step.

The **SML** tab shows the SML for the last semantic item processed. This can be useful when you are trying to determine why the speech engine did not recognize the grammar correctly. It will also show you the confidence level the item was recognized at. For the example in Figure 2.12, the confidence score was 1.000, or 100 percent. The SML tab also allows the developer to edit the SML output and therefore can be used to simulate different outcomes.

Telephony Application Simulator (TASim)

Available with the SASDK, TASim allows you to simulate the client experience for telephony applications. Where Speech Debugging Console is used to design and debug your application, TASim is used for testing and deploying it. You access TASim from the **Debugging Tools** submenu beneath the Microsoft Speech Application SDK 1.0 programs menu item. Go to **File** and **Open URL** to specify the http path to your application. Figure 2.13 is a screenshot of the Telephony Application Simulator.

When debugging telephony applications, TASim is the only way to enter DTMF input or numerical digits. The DTMF tab is not available when using the Internet Explorer Add-in client. In order to execute within TASim, a Web page must contain an AnswerCall control to initiate

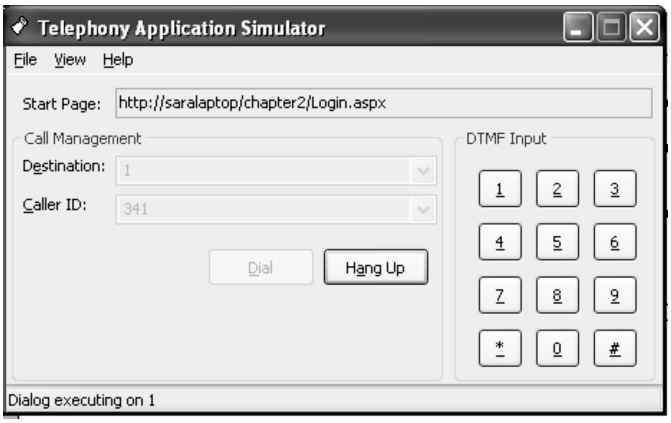


Figure 2.13 Screenshot of the Telephony Application Simulator used to simulate the client experience.

the call. With TASim, it is not necessary to install Speech Server in order to build, debug, and test telephony applications.

Analysis and Reporting

MSS provides two primary means for analysis and reporting:

1. **Call Viewer**—Allows the developer to analyze the results of one or more calls. This tool is generally used by developers to identify problem areas, such as the grammar or the confidence threshold.
2. **Speech Application Reports**—Built on Microsoft SQL Server Reporting Services, they are used to analyze data for multiple calls. Used by both developers and IT decision-makers, they include a few predesigned reports that anticipate common analysis needs.

Each call is logged to the Windows Event Trace Log and stored in a log file with an .etl (event log tracing) extension. This file is then imported into a SQL database using a prebuilt SQL Server Data Transformation Services (DTS) package. Both the Call Viewer application and the Speech Application Reports access the SQL database to analyze imported data.

Several utilities provided with the SASDK allow the developer to extract data from .etl files. In addition, the developer can install the speech application log analysis tools, available by default in the C:\SpeechSDK\Setup\Redistributable Installers\Microsoft Log Analysis Tools for Speech Application directory. This directory should have been created if you followed the instructions in the section titled "Installing the SASDK." After executing **Setup.exe**, you should be able to access the Call Viewer application by browsing to **Microsoft Speech Application SDK 1.0** and **Log Analysis Tools**.

Setting Up a Telephony Server

The Telephony Server provides the interface between the Web server and the telephone network. This enables phone access to SALT applications. For speech applications that run on a Web browser (multimodal), an Internet Explorer (IE) plug-in is available to interpret SALT tags. For telephony applications, the server itself acts as the SALT interpreter.

The Telephony Server can be a standard server machine running Windows 2003. It will include Telephony Application Services (TAS), third-party interface software, or the Telephony Interface Manager (TIM), and a telephony board. It will communicate directly with the Private Branch Exchange (PBX) or the Public Switched Telephone Network (PSTN). The telephony board provides the physical connection to the network. TAS is the piece that interprets SALT tags. And finally, the TIM is what connects the telephony board to TAS.

There are many considerations when setting up a telephony server, such as how many ports to use and whether to use analog or digital connections. You may want to consider working with a Microsoft Speech partner to set up your telephony server. This should reduce the amount of time it takes to deploy your telephony application. A list of current Microsoft partners can be found on the Microsoft Speech Server Web site at http://www.microsoft.com/speech.

Summary

- In the spring of 2004, Microsoft released Speech Server as part of an initiative to make speech applications mainstream. The Speech Application SDK (SASDK), version 1.0, is the component that allows Visual Studio.NET developers to create two types of applications—telephony and multimodal. The SASDK complies with an emerging standard, Speech Application Language Tags or SALT. SALT is a lightweight extension of other markup languages such as HTML and XML that standardizes the way devices use speech.
- Telephony or voice-only applications are accessed by a standard telephone, mobile phone, or smartphone. They can accept input in the form of spoken text or numerical digits pressed on the keypad. Telephony applications have typically been used to make call centers more efficient. The built-in controls offered in the SASDK give them the potential to offer much more.
- Multimodal applications are accessed by either a desktop PC or a pocket PC device. They allow users to select the input mechanism they prefer, whether traditional Web controls or spoken text. A speech add-in installed with Internet Explorer (IE) allows the client to access speech applications.
- There are many opportunities for voice-only applications in today's society. These applications will lower the barriers between human and com-

puter interaction. They will also provide cost-efficient alternatives to traditional information-retrieval methods. The problems that once plagued these applications are being removed, and development has been eased by tools like the Microsoft Speech SDK.

- The speech engine recognizes what the user says by applying predefined grammar rules. Although this requires more effort by the developer, it results in more accurate recognition.
- Once the SASDK is installed, a new speech application is created using the template type for a Speech Web application. Depending on the application type selected (multimodal or telephony), certain files are included by default.
- The prompt editor is a tool for managing an applications prompt database. This database contains prerecorded phrases used throughout the application to prompt the user for input. If phrases are not recorded in the prompt database, the text-to-speech (TTS) engine will speak the phrase. Unfortunately the TTS engine is not very natural sounding, so it is usually best to prerecord prompts.
- As prompts represent what is spoken to the user, grammars represent what the user says. An application consists of several grammar files that each specify a range of possible responses. This increases the efficiency and accuracy of the speech engine, since it knows what to expect.
- A series of speech controls represent the content of a single page. Since the voice-only application has no visible interface, it relies on these controls to direct the user flow. The question/answer (QA) control is the main control used; it represents a single interaction with the user.
- The Telephony Application Simulator (TASim) and the Speech Debugging Console are both used when developing and debugging a voice-only application. The TASim is used to simulate the client experience for telephony applications, and the Speech Debugging Console allows developers to view specifics of the dialog flow.
- Call Viewer and Speech Application Reports are two tools provided with Microsoft Speech Server that allow you to analyze and report on call event data. The developer does not need to install Speech Server to access these tools. They can be accessed from the Redistributable Installers directory that is part of the SASDK installation files.
- Once a voice-only application is developed and tested, it is deployed on a telephony server. The Microsoft Speech Server is the piece that allows phones to access voice-only applications. The telephony server is integrated with Windows and works to interpret the SALT tags for the application.