Chapter 1

Introducing PowerScribe® Software Development Kit

Objectives

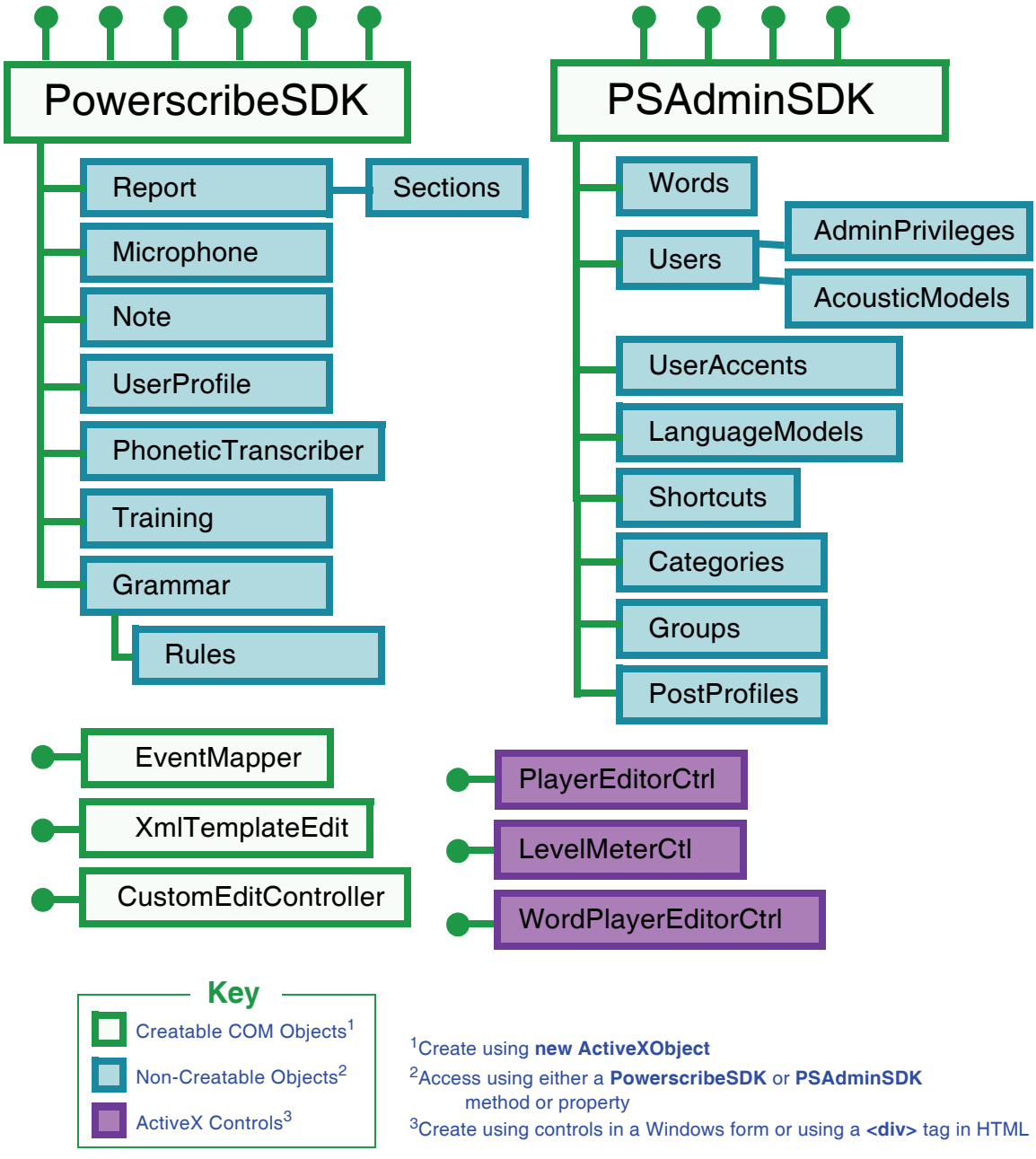
PowerScribe® Software Development Kit (*SDK*) is a set of COM objects and application programmer interfaces (APIs). The *SDK* helps you create a system that records speech to create documents (such as medical reports) and uses speech recognition to transcribe them. The system you create might also (or alternatively) record audio, then manage that audio as users manually transcribe it into text documents and then edit the text. In addition, the *SDK* helps you use speech recognition to retrieve transient realtime data from the speaker, such as part numbers.

This chapter presents an overview of the object model and the types of capabilities that the *SDK* provides:

PowerScribe SDK Object Model

PowerScribe SDK is a toolkit for creating a custom report management system that captures dictated reports and uses speech recognition to transcribe them, then allows end users to correct the text and take other actions.

The object model for the *SDK* (shown below) contains several types of objects. Most of these objects belong to either the **PowerscribeSDK** or the **PSAdminSDK** object group.



Types of Applications You Can Develop

The *PowerScribe SDK* product contains tools to create two types of applications:

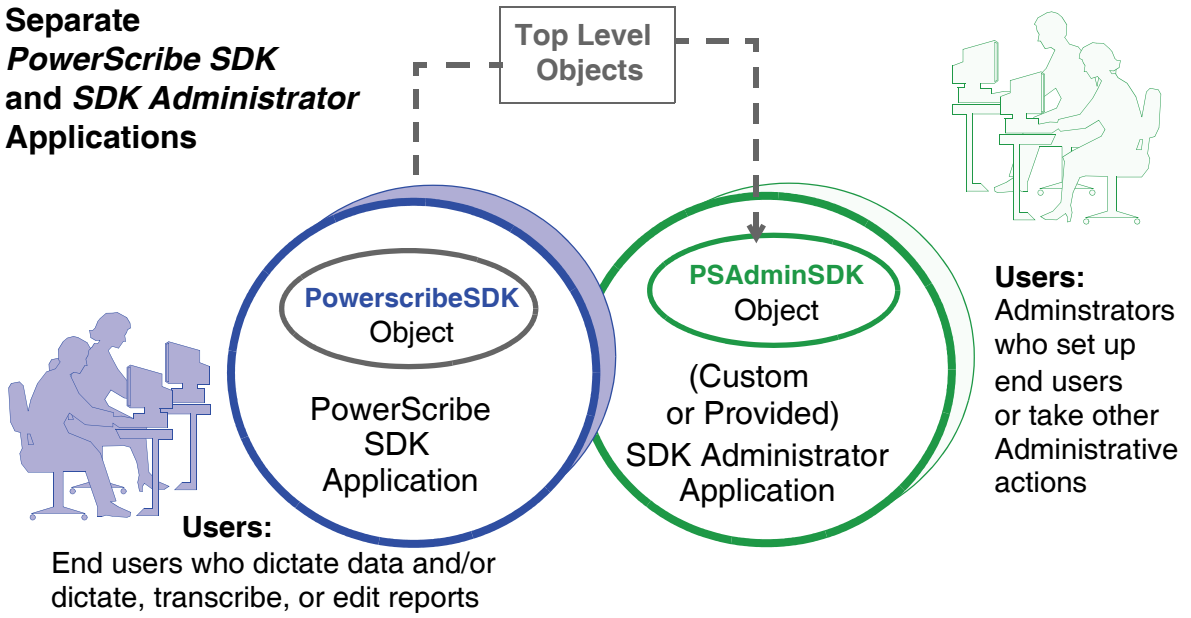
- *PowerScribe SDK* applications that deploy speech recognition (work with **PowerscribeSDK** object group of object model)
- *SDK Administrator* applications—Administer the system (work with **PSAdminSDK** object group in the object model)

You usually write your own *PowerScribe SDK* application(s) and utilize the *SDK Administrator* program provided; however, you can create custom *SDK Administrator* applications as well.

You are required to use the *SDK Administrator* application provided to manage logins, schedule purging of reports, and set system parameters, because you have limited access to these capabilities through the *SDK* APIs.

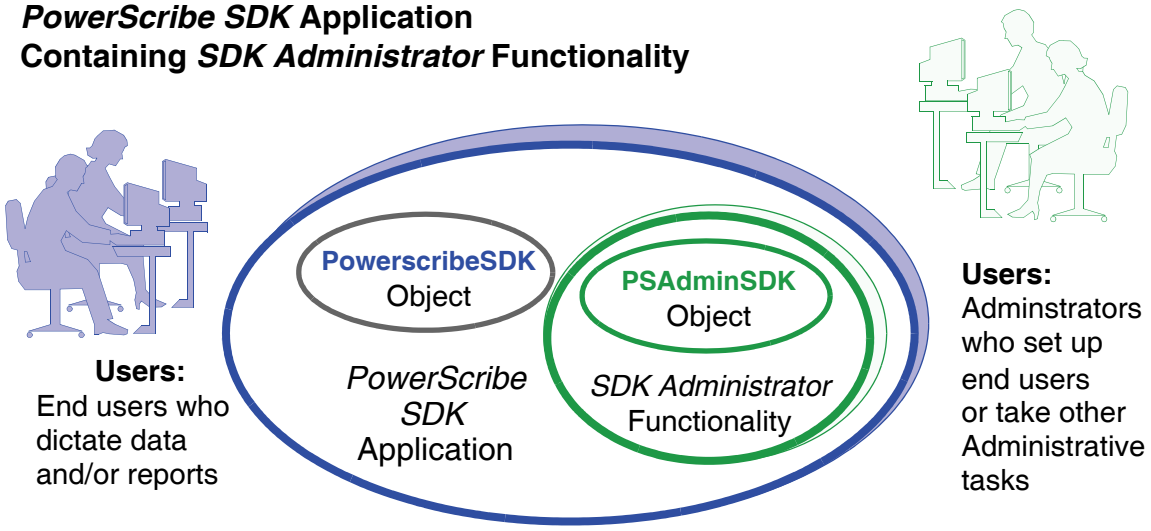
In your custom *SDK Administrator*, you can manage users, shortcuts, categories, groups, and custom words. In the course of managing users, you also handle assigning each user administrative privileges, a particular language model ID, and a postprocessor profile for formatting that user's reports. For more detail, refer to SDK Administrator Provided vs. Custom Administrator Application on page 268.

As shown in the illustration below, *SDK Administrator* and *PowerScribe SDK* applications each have a different top level object.



Although the *SDK Administrator* and *PowerScribe SDK* applications can function separately, side-by-side, you might find it convenient to create a single application that lets a select group of users who are assigned appropriate administrative privileges carry out the administrative tasks, and all other users dictate data, dictate reports, or carry out related tasks.

Combining the two types of functionality in a single application is straightforward, because as shown in the illustration below, the **PSAdminSDK** object and **PowerscribeSDK** object can both be in the same application.



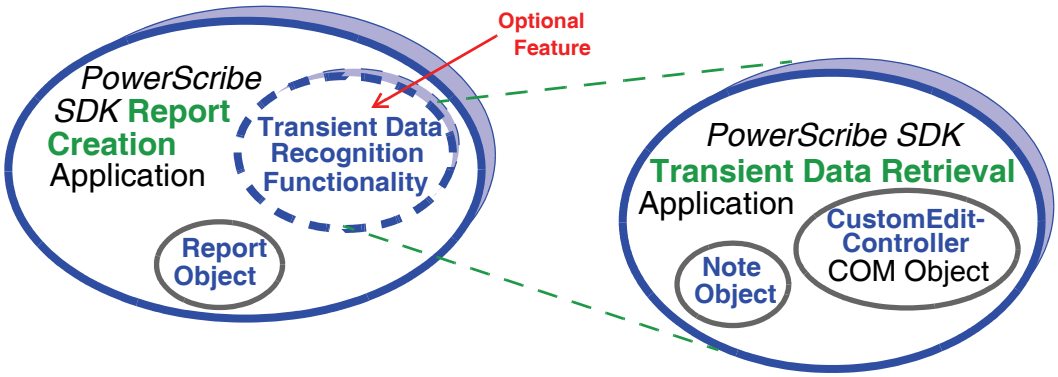
If you choose not to write a full *SDK Administrator* application, having an **PSAdminSDK** object inside your *PowerScribe SDK* application gives you access to user data and other administrative data that you set up with the *SDK Administrator* program provided.

Types of Speech Recognition Technology

PowerScribe SDK applications can utilize one or both of two major types of technology:

- Report creation—Recognize speech from dictation for creating and developing reports using the **Report** object. Stores text and audio of the report on the server.
- Transient data retrieval—Recognize speech from dictation to obtain transient data (such as a dictated part number to search for) using the **Note** object and **CustomEditController** creatable COM object. Does *not* store audio or recognized text of this data.

  Note: *These two types of technology can be integrated (see circle to left below), so that applications that deploy report creation can also deploy realtime transient data retrieval.*



The **Note** object is available in JavaScript as well as in C#, Visual Basic, C++, and Java on Windows. The **CustomEditController** object is available in all of the same languages, except JavaScript, where you can generate the same functionality using the **Note** alone.

PowerscribeSDK Objects for Creating Reports

Let's begin by taking a look at the objects you would use in any *PowerScribe SDK* application designed for the purpose of dictating, transcribing, and editing reports. These objects are all under the **PowerscribeSDK** object in the object model.

Report Object and Types of Reports

You create reports with the *SDK* using the **Report** object. Each report can be one of two types:

- Structured
- Plain

Structured Reports

Structured reports contain predefined headings and sections created by applying an XML formatted template. The template determines the text that each heading should contain and the style of that heading text, as well as the style of the paragraphs entered under the heading. The structured report template can have up to nine heading levels. The end user can quickly enter text under each heading by typing or dictating, without having to enter the actual headings for each new report. The *SDK* provides this functionality as well as the ability to navigate the structure using the microphone.

A structured report also contains data about the source of the report text. The source of the text might be, for instance, typed, dictated, or reused. You can choose to display the source of the text by assigning different highlight colors and fonts to each source, then turning on the *display of source markup* to see each piece of text's source.

Plain Reports

Plain reports, by contrast, are text reports with no structure. If you do not provide an XML structure for the report, it defaults to a plain report. The plain report is not just ASCII text (although it can be), but normally contains fonts, and paragraph and heading styles you assign. What it does *not* contain is predefined headings and subheadings with predefined heading and paragraph fonts and styles that force it into a particular structure.

RTF Reports

When you save a plain report, the *SDK* saves it in *Rich Text Format* (**.rtf**), so it retains fonts and styles you are using.

Word Format Reports

You can also create a plain report in a Microsoft Word editor. When you do, the *SDK* saves the report in Microsoft Word Document (**.doc**) format, so that you can open the file in Word or a compatible editor.

Other Objects for Use When Receiving Dictation

Other objects that you can use in the course of creating reports include:

- **Microphone**
- **PlayerEditorCtl** (or **WordPlayerEditorCtl)**
- **LevelMeterCtl**
- **XmlTemplateEdit**

The **PlayerEditorCtl**, **WordPlayerEditorCtl**, and **LevelMeterCtl** are all ActiveX controls, while the **XmlTemplateEdit** object is a creatable COM object.

Microphone Object

When you create a report in your *SDK* application, you can type text or dictate into a microphone. To work with a microphone, you are not required to have a **Microphone** object, but you can create this object to program customizable buttons on the microphone or even take over full control of all the buttons on the microphone in your application.

Another alternative you could take would be to have GUI buttons that initiate microphone actions, like **Record**, **Play**, **Stop**, **FastForward**, and **Rewind**. You can create such buttons and activate them using **Microphone** object methods.

PlayerEditorCtl ActiveX Control

After you dictate the report, the next step for a **Speech Recognition** user (a user with permission to dictate, transcribe, and edit) is to transcribe the audio—so the text appears in an editor. In *PowerScribe SDK* applications, you use a special editor called a **PlayerEditor**. The **PlayerEditor** is an editor that not only allows the user to see the text and modify it, but allows the user to play back the audio and watch the cursor track the text that corresponds to that audio. To integrate a **PlayerEditor** with all its capabilities into your application, you embed a **PlayerEditorCtl** ActiveX control in your Windows form or HTML.

WordPlayerEditorCtl ActiveX Control

If you would like to use a Microsoft Word-compatible editor to create plain reports that you can later open in Word, you can deploy a unique **PlayerEditor** designed to work with Word documents by embedding a **WordPlayerEditorCtl** ActiveX control in your Windows or HTML form. This **PlayerEditor** has not only all of the functionality of a **PlayerEditorCtl**, but the ability to utilize Microsoft Word's editing tools as well.

The *SDK* also provides a method you can call to retrieve a Microsoft Word document object, so that your application can work with Word commands that Microsoft provides.

LevelMeterCtl ActiveX Control

While dictating audio into a report in your application, a user needs to know that action is taking place. An object that shows changes in the volume of vocalizations spoken into the microphone is a **LevelMeter**, a bar that shows more or fewer tick marks as the volume of the speaker rises and falls. You can insert a **LevelMeter** in your application by embedding the **LevelMeterCtl** ActiveX control in your Windows form or HTML.

Creating Report Structures with XmlTempalteEdit Object

You can create an XML formatted template manually and pass it to a method when you create the report, or you can create the template programmatically using a special object called an **XmlTemplateEdit** object.

Creating Your Own Grammars for Reports

A grammar is a collection of specialized custom voice commands that you define in an XML file and can use only in **Command** mode. (In contrast with commands built in to the product that are operational in **Dictation** mode.)

A grammar might, for instance, include commands to create and edit reports or pop up a list of available shortcuts:

- Create Report
- Edit Report
- Display Shortcuts

Another grammar might be a set of voice commands for checking off options on a form, pressing buttons on a form, or using other specific phrases.

Grammar and Rules Objects

To create a grammar, you use the **Grammar** object.

When you define the grammar, you delineate the **Rules** of the grammar inside an XML file. Your application activates the collection of **Rules**, which contains the individual **Rule** objects, to activate the grammar.

You can create several sets of voice commands by creating a **Grammar** object for each one. Your application can then deploy one or more **Grammar** objects, depending on the needs of the person who is dictating.

PowerscribeSDK Objects for Transient Data Retrieval

When you create any *PowerScribe SDK* application not necessarily designed for the purpose of generating reports, you can embed in that application speech recognition for purposes other than reporting. Your application might, for instance, contain a form where a user can dictate the content of the fields, dictate a name or number to search for, or retrieve dictated numbers, letters, or words for other purposes.

Objects you would us in a realtime data retrieval application include two you also use for creating reports:

- **Microphone**
- **LevelMeterCtl**

In addition, a realtime data retrieval application also deploys two additional objects:

- **Note**
- **CustomEditController**

Note and CustomEditController Objects

To create fields in an application that receive dictation from a speaker for purposes other than creating a report, you use a **Note** object. The **Note** object, unlike a report, does not save the recognized text of the dictation; instead, your application uses the **Note** object to hold the recognized text in memory and use it to, for instance, look up a name or part number in a database. The **Note** is for text that is transient, like a lookup key, that can be disposed of when it is no longer needed.

The **Note** object is available in both JavaScript and *Visual Studio* applications.

In JavaScript, you can associate a **PlayerEditor** with the **Note** object and dictate into the editor. However, in *Visual Studio* applications using Windows components, you do not use the **PlayerEditor** to retrieve dictation into a **Note**.

Instead, for a GUI developed in a *Visual Studio* language (such as Visual Basic or C#) to receive dictation from a speaker, after you create a Windows form, you can set up any component on the form to receive dictation by deploying an *SDK* creatable COM object called a **CustomEditController** and using the **Note** object with that controller object.

The **CustomEditController** associates the component in the GUI with the **Note** object that receives the dictation.

Other PowerScribe SDK Objects/Tasks

Deploying an EventMapper

If you are developing your application in JavaScript, you can take advantage of a unique feature of the *PowerScribe SDK* product. This feature, called *Map Creation*, was designed to map events that the *SDK* COM objects fire, so that your JavaScript application can receive those events.

EventMapper Creatable COM Object

To use *Map Creation*, you instantiate an **EventMapper** creatable COM object. You then use that **EventMapper** to map each event you expect from a particular object to its event handler.

Later, when the event occurs, the associated handler responds to that event.

You use the **EventMapper** object to receive events from COM objects that are not ActiveX controls.

Creating Custom Training Module

Another object in the arsenal of the **PowerscribeSDK** object is the **Training** object. You use this object to apply a custom look and feel to the training pages in the product. The **Training** object provides the text for the pages, while you provide the custom colors and text styles, then determine the conditions under which the next page of text displays.

Accessing Information About Logged In User

Often when a user is logged in to a *PowerScribe SDK* application, the application needs to take action or know something about that user. To work with the logged in user, the application uses a **UserProfile** object. This object lets the application change the user's login ID and password, but also lets the application determine other aspects of that user—whether the user is an administrator, whether to use automatic puntuation in this user's recognized text, whether to stream text into the application for this user, and other aspects of the logged in user's profile.

SDK Administrator Tasks

Administrative tasks are those that an administrator level user normally takes. You can carry out many of these actions using the *SDK Administrator* application provided; however, if you want to customize the administrator functionality, you might want to create an *SDK Administrator* application of your own.

Let's take a look at the objects you use to carry out those tasks in your application. These objects are under the **PSAdminSDK** object in the object model.

You might write a separate *SDK Administrator* application where the user can carry out these tasks or you might integrate the tasks into a *PowerScribe SDK* application.

Creating Users and Assigning User Privileges

Before anyone can use either your *PowerScribe SDK* application or your *SDK Administrator* application, you must create users with the **PSAdminSDK** object.

Users and User Objects

You create users by retrieving the **Users** collection object (that already contains the original **admin** user) and adding new users to the collection. Each user has a **User** object that you retrieve to access that user's property settings and, in some cases, to modify those property settings.

AdminPrivileges Objects

One of the properties of a **User** object is the **AdminPrivilege** property. You use the **AdminPrivilege** property to return an **AdminPrivileges** object that has several properties of its own. You set the properties of the **AdminPrivileges** object to give the associated user particular types of administrative privileges.

PostProfiles Collection Object

Another object that you use when assigning **User** property values is the **PostProfile**, an object that you can define to set up how all the reports of that particular user are later formatted during postprocessing. You can utilize the default postprofile or you can configure a profile format using the *AutoformatProfile Customization* tool. Later, you can retrieve that profile from a **PostProfiles** collection and use it to set the **AutoFormatProfileID** property of a particular user.

LanguageModels and LanguageModel Objects

A language model is a vocabulary for a particular specialty that the *Speech Recognition* engine understands. For instance, the Radiology and Pathology models would be for specific fields in medicine.

Your application can retrieve the collection of all **LanguageModels** and then select a single **LanguageModel** object from the collection to assign to a particular speech recognition user (who can dictate reports and have the speech recognition engine transcribe them).

When the speech recognition user later logs in, the *PowerScribe SDK* application loads the associated language model from the site server into the memory of the machine where the user is running the *PowerScribe SDK* application, storing a copy of the model as the *in-memory language model*.

UserAccents and UserAccent Objects

PowerScribe SDK provides a collection of accents that you can choose from to assign an accent to a particular user. To retrieve that collection, your application can use the **UserAccents** collection object. The application can then retrieve a particular **UserAccent** from the collection and assign it to a particular user.

The collection includes accents such as **Indian English**.

AcousticModels and AcousticModel Objects

Each time a user logs in to your *PowerScribe SDK* application, that user speaks into an audio device such as a microphone, but the user can log in to another login session using a different type of device, such as a recorder or bluetooth device. Each time the user logs in with a different audio device, the *SDK* has the user teach the system his or her voice with that device, then generates a unique acoustic voice model for that user with that audio device. The user's voice model with that audio device is stored in an **AcousticModel** object in the **AcousticModels** collection. Because the voice model requirements may vary based on the audio device being used, if you want to know the training state of a user, you do not retrieve that data from the **User** or **UserProfile** object. Instead, the application retrieves the data about the training state of the logged in user through the **AcousticModel** of that user.

Working with Custom Shortcuts and Words

Shortcuts are single words or short phrases that trigger *SDK* functionality to insert entire predefined sections or multiple predefined paragraphs into a report, a process called *expanding the shortcut*. By saying a single word like **Start**, you could have your application insert several boilerplate paragraphs into a report. Similar short words or phrases might expand to include information that applies only in particular cases.

Your application can facilitate the process of creating shortcuts by helping the user define them, associating them with an audio file, and loading them for use. It can take similar actions for custom words that are not in your usual language model, but that you might define for a particular situation, such as generic drug names for a pharmaceutical application.

You use the **Shortcut** object to deploy this kind of functionality.

Shortcuts and Shortcut Objects

Although shortcuts can be dictated words or phrases (called *voice* shortcuts), they can also be typed words or groups of letters (called *text* shortcuts). For instance, instead of typing out a paragraph of standard input, you could type a series of letters that the application would automatically expand into a paragraph or complete section of the report.

To create shortcuts, you start by creating a **Shortcuts** collection object. You then add each **Shortcut** to the collection. Any shortcut in that collection can be for the currently logged in user only or globally available to all users.

Categories and Category Objects

Your *SDK Administrator* application can create user-defined purposes or types called **Categories** to assign to each shortcut. You create a category by first creating a collection of **Categories**, then adding particular **Category** objects to the collection, assigning each a name and description. Once the **Category** exists, you can assign that category to any particular shortcut using the **CategoryName** property of the **Shortcut** object. Each shortcut can have only one category.

Groups and Group Objects

Your *SDK Administrator* application can associate related or similar shortcuts with one another by putting them into groups. You create a group by first creating a collection of **Groups**, then adding particular **Group** objects to the collection, assigning each a name, description, and global status (global if available to all users, not global if specific to the creator of the group). Once the **Group** exists, you can add shortcuts to the group using the **Shortcut** object. Each shortcut can belong to more than one group.

Words and Word Objects

Sometimes the speech recognition engine has difficulty recognizing a word when a particular user dictates it. To help improve the quality of recognition, you can create custom words. You teach the speech recognition engine how those words sound when spoken by the particular user. You create such words using the **Word** object. You start by creating a **Words** collection object, and then add each individual **Word** to the collection.

PhoneticTranscriber Object

You can create an audio file to indicate how a custom shortcut or word would sound when it is dictated. You then pass the information from that file in a phonetic transcription string to the method that creates the shortcut or word. You retrieve that string from a **PhoneticTranscriber** object. This object helps you improve the quality of speech recognition for shortcuts and words.

Taking System Actions on Users, Shortcuts, Categories, Groups, and Words

Administrator level users might also take advantage of methods of the **PSAdminSDK** object that facilitate importing and exporting of users, voice models for those users, shortcuts, categories, groups, and words.

The administrator can use these capabilities to copy those objects from one *SDK* system to another.

Using XML formatted files, the administrator can also import user, shortcut, category, group, or word information from other (non-*SDK*) systems.