Chapter 11

Getting Started Developing SDK Administrator Application

Objectives

The *PowerScribe® SDK* provides you with a standard *SDK Administrator* application, but it also provides you with tools to create your own custom *SDK Administrator* program. The first section of this chapter summarizes the features available in the *SDK Administrator* program provided:

- SDK Administrator Provided vs.Custom Administrator Ap plication

The remainder of this chapter uses JavaScript to present fundamental steps to begin developing an *SDK Administrator* application:

- Instantiating the PSAdminSDK Object
- Initializing Administrator Application
- Configuring Administrator Application to Listen to Events
- Releasing Event Maps Before Exiting
- Logging Out of Application
- Code Summary

SDK Administrator Provided vs. Custom Administrator Application

Before you bother to develop your own *SDK Administrator* application, you should become acquainted with the *SDK Administrator* that Nuance provides with the product. If you then decide you would prefer to have a custom application, you can proceed to develop one.

Running the SDK Administrator

To run the *SDK Administrator* , open a web browser and enter the following for the URL, using the name of your server: **http://**<*servername*>**/pscribesdk/admin**

Understanding Features of SDK Administrator

After you log in to the product using the **admin** login and password provided for your site, you then can use it to take several actions under the tabs shown in the illustration below:

- **Users**
  - — Create, edit, disable, and delete users for *PowerScribe SDK* applications
  - — Create other administrator level users
  - — Set the initial values that apply to a new user by default
- **Logins** — Logout users, especially dangling logged in users
- **Shortcuts** and **Words**
  - — Create global shortcuts or words to be available to all users
  - — Create categories and assign them to shortcuts

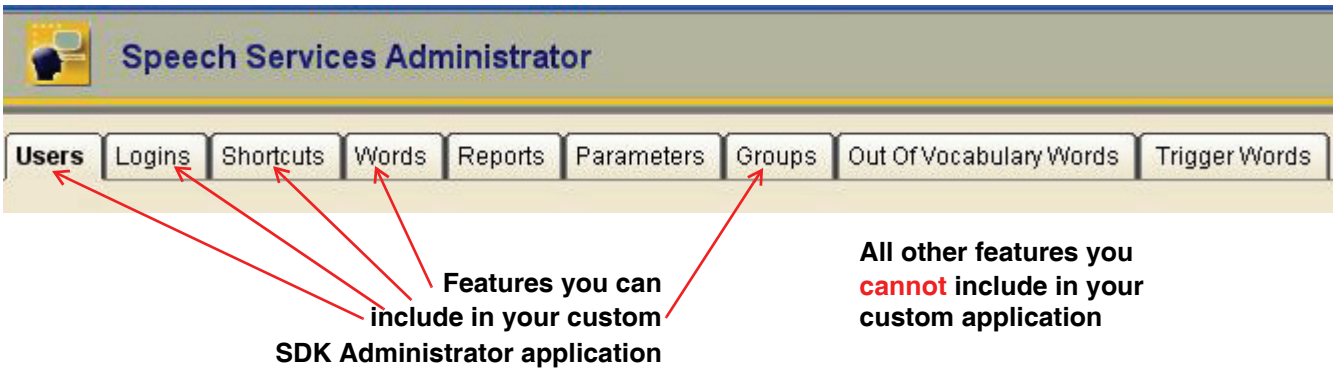
  If you are not sure what a shortcut is or need more information on shortcuts/words, refer to Chapter 12, Working with Shortcuts, Categories, and Words on page 277.
- **Reports**
  - — View a list of all reports in process, including who originally dictated the report, who last edited it, when it was originally dictated, the processing state of the report, and whether or not the report is locked
  - — Edit the properties of the report, execute recognition on the report, export the report, unlock the report, and delete or purge the report

- **Parameters** — Enable and disable PS parameters, such as those that determine the maximum number of days before audio is purged from the system
- **Groups** — Create groups and add shortcuts to those groups
- **OutOfVocabularyWords** — Find a list of out-of-vocabulary-words that the recognition engine automatically creates (refers to words that are frequently dictated but not in the language model); add these words to the language model or remove them from the list; export the entire list to a file
- **Trigger Words** — Work with custom trigger words—words that you dictate to indicate custom commands follow (an example of a trigger word is *Dictaphone*)

For more information on these features, refer to the **Help** available in the product GUI.

Features You Can Include in Custom Administrator

Your custom *SDK Administrator* application can include only some of the features of the *SDK Administrator* provided. The features you can include are under the the five tabs indicated below:



- **Users**
- **Logins**
- **Shortcuts** (**Categories** are also under this tab)
- **Words**
- **Groups**

Your custom application cannot include the features available under the **Reports**, **Parameters**, **OutOfVocabularyWords**, or **Trigger Words** tabs. To use these capabilities, you must work with the *SDK Administrator* provided.

If you decide that you want to develop your own application, proceed with the remainder of this chapter, then go through the appropriate subsequent chapters for information on:

- Creating users—Chapter 13
- Creating shortcuts and words—Chapter 12
- Creating groups—Chapter 14
- Carrying out advanced administrator tasks—Chapter 16

Instantiating the PSAdminSDK Object

To begin interacting with the *SDK Administrator Web Server*, your application must instantiate a **PowerscribeSDK** object, the main object in the object model, then instantiate a **PSAdminSDK** object, a parallel object in the model that you use to take administrative level actions. You instantiate the two objects in JavaScript as follows:

```
pscribeSDK = new ActiveXObject("PowerscribeSDK.PowerscribeSDK");
objAdmin = new ActiveXObject("PSAdminSDK.PSAdminSDK");
```

Both **PSAdminSDK** and **PowerscribeSDK** objects are required in your *SDK Administrator* application. You need not call the **Initialize()** method of the **PowerscribeSDK** object in this type of application; instantiating the **PowerscribeSDK** object is sufficient.

You cannot use methods and properties of any objects under the **PSAdminSDK** object in the object model until you have initialized the **PSAdminSDK** object and logged in to the *SDK Administrator* application.

Initializing Administrator Application

Now that you have the **PSAdminSDK** object, you initialize the object using its **Initialize()** method. This method has two arguments, the URL to your *SDK Administrator* application on the server and a unique application name that identifies the context for logging in and out:

```
var url = "http://server2/myadminApp/";
objAdmin.Initialize(url, "UserConfigs");
```

You normally take this action in a login function or script, after an administrative level user with access to the application has entered a login and password in a startup dialog box. The function or script can then continue the process of logging in that administrator by calling the **Login()** method of the **PSAdminSDK** object and passing it the login and password:

```
objAdmin.Login(login.text, password.text);
```

Logging Out Dangling Logged In User

Note: *The login for a particular user never expires. As a result, if the user exits from the application without logging off, the user remains logged in until your application logs that user out.*

In an *SDK Administrator* application, just as in a *PowerScribe SDK* application, if the administrator has shut down the application without first logging out, he or she remains logged in indefinitely and cannot log in again unless the application logs him or her out first. Your application should handle this *dangling logged in user* (error **-2146778148**, the same error code for both the **PowerscribeSDK** and the **PSAdminSDK** objects) by:

- Logging in another administrator user

- Logging out the dangling logged in user
- Logging out the other administator user
- Relogging in the no-longer-dangling user

Logging Out Dangling Logged In User Programmatically

To log out a dangling logged in user inside your application:[1]

1. To log in another administrator user, you log in again, this time passing another login name to the **Login()** method of the **PSAdminSDK** object:

   ```
   objAdmin.Login("admin", admPassword);
   ```

2. To then log off the dangling logged in administrator user, you would call the **LogoffUser()** method of the **PSAdminSDK** object.

   ```
   objAdmin.LogoffUser("userJ", strPasswd);
   ```

3. To log out the second administrator user, who is now the only user logged in, you would call **Logoff()** method of the **PSAdminSDK** object.

   ```
   objAdmin.Logoff();
   ```

4. Finally, now that you have successfully logged out the no-longer-dangling user, you can log that user back in using the **Login()** method of the **PSAdminSDK** object:

   ```
   objAdmin.Login("userJ", strPasswd);
   ```

5. On successful login, you might open the main application window:

   ```
   window.open("main.htm", "_self", "height=100, width=50, status=yes",
      true);
   ```

The entire block of code might look like this:

```
try
{
   objAdmin.Login(user.text, passwd.text);
}
catch(error)
{
   if (error == -2146778148)
   {
      objAdmin.login("admin", "");
      objAdmin.LogoffUser(user.text, passwd.text);
      objAdmin.Logoff();
      objAdmin.Login(user.text, passwd.text, "UserConfigs");
      window.open("main.htm", "_self", "height=100, width=50, status=yes",
         true);
   }
```

1. You can log out a dangling logged in user by logging in as the original **admin** user to the *SDK Administrator* application provided and, from there, logging the dangling user out. Although this action quickly resolves the situation, manual intervention of this kind is not as efficient as programmatic control of the situation.

```
    else
    {
       // Handle Other Errors
    }
}
```

Note: *Note that the error code for* ***No user is logged in*** *is the same for the* ***LogoffUser()*** *methods of both the* **PowerscribeSDK** *and the* **PSAdminSDK** *objects: -2146798283.*

Configuring Administrator Application to Listen to Events

After you have initialized the *SDK Administrator* application, first determine what kinds of actions you plan to take with your application. The only actions that result in **PSAdminSDK** object events are exporting and importing user voice models. If you plan to take those types of actions with this application, you need to set up the application to listen to events.

In JavaScript, you set up an *SDK Administrator* application to listen to events much the way you set up a *PowerScribe SDK* application to listen to events, as covered in Chapter 3, under Configuring the Application to Listen to Events on page 25. Some of the differences for an *SDK Administrator* application are delineated below.

Instantiating EventMapper for PSAdminSDK Objects

In JavaScript applications, you map events with the ***Map Creation*** facility of the product. To use ***Map Creation***, you must instantiate an **EventMapper** object to catch events that **PSAdminSDK** objects trigger. To indicate that the **EventMapper** is catching events fired by objects in the **PSAdminSDK** branch of the object model, let's name it **AdminSink**:

```
AdminSink = new ActiveXObject("PSAdminSDK.EventMapper");
```

There are only a few events that objects in the **PSAdminSDK** branch of the object model fire. Two of them are shown below, both events that the **User** object fires when the application has exported voice models to be copied or moved to another system or has imported voice models from another system. If you plan to work with those events, you map the event handlers to the events using the **EventMapper** object, just as you did for **PowerscribeSDK** object events:

```
AdminSink.EndExport = HandleEndExport;
AdminSink.EndImport = HandleEndImport;
```

Once you have the events mapped, you can call the **Advise()** method of the **EventMapper** object and pass it the **PSAdminSDK** object:

```
AdminSink.Advise(objAdmin);
```

Once you call the **Advise()** method, the application starts receiving events in response to the associated methods.

Creating Event Delegates in C#

In C#, you create the event delegates using the **PSAdminSDK** object. For instance, if the object is named **m_adminSDK**, you create the **EndImport** and **EndExport** delegates as follows:

```
m_adminSDK.EndExport +=new PSAdminSDKLib._IPSAdminSDKEvents
      _EndExportHandler(Handle_EndExport);

m_adminSDK.EndImport +=new PSAdminSDKLib._IPSAdminSDKEvents
      _EndImportHandler(Handle_EndImport);
```

When you create the actual handler for the **EndExport** event, you pass it the arguments for the file location string, error number, and error message from the **PSAdminSDKLib**:

```
private void Handle_EndExport(string FileLocation, int ErrNum, string ErrMessage)
   { // Take appropriate action  }
```

Note: *For samples showing how to set up events in Visual Basic 6.0, Visual Basic .NET, C++, and Java, refer to the sample code in the* ***Samples*** *directory under the root on the CD.*

Releasing Event Maps Before Exiting

Before you log off the administrator user of the application, you release the event maps by calling **Unadvise()** on the **EventMapper** object and passing it the name of the object whose events you want to release:

```
AdminSink.Unadvise(objUser);
```

You usually call **Unadvise()** in your user defined shutdown procedure, inside a **Try/Catch** block:

```
function cleanup()
{
   try
   {
      if (AdminSink)
         AdminSink.Unadvise();
   }
   catch(error)
   {
      alert("Cleanup " + error.description);
   }
}
```

Logging Out of Application

You usually log out the logged in user as part of a sequence of actions you take during a shutdown of the application. To log out the user, you use the **Logoff()** method instead of the **LogoffUser()** method (discussed earlier under *Initializing Administrator Application on page 270*).

Code Summary

```
<HTML>
<HEAD>

<TITLE>SDK Administrator</TITLE>
<SCRIPT type="text/javascript">


function InitializeAdminApp()
{
   try
   {
      // Instantiate Admin SDK Object
         pscribeSDK = new ActiveXObject("PowerscribeSDK.PowerscribeSDK");
         objAdmin = pscribeSDK.AdminSDK;
         objAdmin = new ActiveXObject("PSAdminSDK.PSAdminSDK");

      // Initialize Admin SDK SDK Server
         var url = "http://server2/adminsdk/";
         objAdmin.Initialize(url, "UserConfigs");

      // Create EventMapper Object for Admin SDK Object
         AdminSink = new ActiveXObject("PSAdminSDK.EventMapper");

      // Map Handlers for Each Event to SDK EventMapper Object
         AdminSink.EndExport = HandleEndExport;
         AdminSink.EndImport = HandleEndImport;
         SDKsink.Advise(objAdmin);
   }
   catch(error)
   {
      alert("InitializeAdmin: " + error.number + error.description);
   }
   finally
   {
      DoLogin()
   }
}
```

```
function DoLogin()
{
   try
   {
      objAdmin.Login(user.text, passwd.text);
      var lastUsername = user.text;
      var lastPassword = passwd.text;
   }
   catch(error)
   {
      // If the user is already logged in (is dangling logged in user)
      if (error == -2146778148)
      {
         objAdmin.login(alt_user.text, alt_passwd.text);
         objAdmin.LogoffUser(lastUsername, lastPassword, "UserConfigs");
         objAdmin.Logoff();
         objAdmin.Login(lastUsername, lastPassword);
      }
      else
      {
         // Handle Other Errors
      }
   }
   finally
   {
      window.open("main.htm", "_self", "height=100, width=50,
         status=yes", true);
   }
}
...


function DoLogoff()
{
   try
   {
      objAdmin.Logoff();
      Cleanup();
   }
   catch(error)
   {
      alert(error.description);
   }
}


function cleanup()
{
   try
```

```
   {
      if (AdminSink)
      {
         AdminSink.Unadvise();
      }
   }
   catch(error)
   {
      alert("Cleanup Error: " + error.description);
   }
}


</SCRIPT>
</HEAD>


<BODY bgcolor="#ffff" language="javascript" onload="InitializeAdminApp()"
onunload="DoLogoff()">

<form>

...
   <input language="javascript" type="button" name="btnLogin"
         value="Login" onClick="DoLogin()"><br>

   <input language="javascript" type="button" name="btnLogout"
         value="Logout" onClick="DoLogoff()"><br>


...

</form>

...

</BODY>
</HTML>
```