16 Sprachsteuerung

Es ist ein Trend erkennbar, dass Geräte weitere Bedienkonzepte anbieten – genannt seien Bewegungssteuerungen oder die Sprachsteuerung. Gerade letztere hat mit Apples Siri an Popularität gewonnen.

Windows Phone 8 bietet neben einer erweiterten Sprachsteuerung eine entsprechende API, um in eigenen Anwendungen die Sprache als Interaktionskonzept zu verwenden.

16.1 Möglichkeiten der Sprachsteuerung

Die Sprach-API bietet uns drei zentrale Bereiche, wie wir Sprache integrieren können:

- Text-zu-Sprache
- Sprache-zu-Text
- Sprachkommandos

Text-zu-Sprache erlaubt es, einen vorhandenen Text vorlesen zu lassen. Mögliche Einsatzbereiche können Hörbücher, das Vorlesen von Artikeln oder das Erzählen von Handlungen in Spielen sein.

Sprache-zu-Text geht den umgekehrten Weg und führt für eine Sprachaufzeichnung eine Erkennung durch. Denkbar wäre dies für Texterfassungen aller Art.

Sprachkommandos bieten die interessantesten Möglichkeiten, da sie ganz neue Interaktionswege eröffnen. Mit ihnen kann die App Aktionen starten und eine Art Dialog mit dem Benutzer durchführen, um erforderliche Eingaben abzufragen.

Benötigte Funktion (Capability)

Alle Bereiche der Sprach-API erfordern die Funktion IP_CAP_SPEECH_RECOGNITION.

16.2 Text-zu-Sprache

Die Ausgabe von Text als Sprache ist sehr einfach – das Beispiel zeigt Listing 16–1.

```
01 using Windows.Phone.Speech.Synthesis;
02
03 var synth = new SpeechSynthesizer();
04 synth.SpeakTextAsync("Hallo Welt!");
```

Listing 16–1 *Minimalbeispiel für Text-zu-Sprache*

Der Vorleseprozess läuft asynchron ab. Sollte die nachfolgende Programmausführung erst nach Abschluss des Vorlesens erfolgen, muss dessen Ende abgewartet werden. Dies und wie eine andere Sprache für das Vorlesen gesetzt werden kann, zeigt Listing 16–2.

```
01 var sync = new SpeechSynthesizer();
02
03 var voice = InstalledVoices
                  .All.FirstOrDefault(
                     v => v.Language == "de-DE" &&
                          v.Gender == VoiceGender.Male);
04 sync.SetVoice(voice);
05   await sync.SpeakTextAsync(TextToSpeak.Text);
```

Listing 16–2 *Sprachausgabe abwarten und Sprache setzen*

16.3 Sprache-zu-Text

Mithilfe von Sprache-zu-Text kann eine Alternative zur virtuellen Tastatur geschaffen werden. Ein Beispiel zeigt Listing 16–3.

```
01 using Windows.Phone.Speech.Recognition;
02
03 var sr = new SpeechRecognizerUI();
04 sr.Settings.ListenText = "Notiz erfassen";
05 sr.Settings.ExampleText = "Geburtstagsgeschenk kaufen";
06 sr.Settings.ReadoutEnabled = true;
07 sr.Settings.ShowConfirmation = false;
08
09 var result = await sr.RecognizeWithUIAsync();
10 if (result.ResultStatus == SpeechRecognitionUIStatus.Succeeded)
11 {
12   string spokenText = result.RecognitionResult.Text;
13   string confidence =
14     result.RecognitionResult.TextConfidence.ToString();
15 }
```

Listing 16–3 *Verwenden von Sprache-zu-Text*

Zentraler Angelpunkt ist die Klasse *SpeechRecognizerUI*. Sie zeigt den Systemdialog zur Spracherkennung, der über Eigenschaften konfiguriert werden kann (Zeilen 04-07).

ListenText (Zeile 04) setzt den Titel des Dialoges. Ein Beispieltext für den Nutzer kann über `ExampleText` (Zeile 05) definiert werden.

Soll der erkannte Text dem Nutzer vorgelesen werden, so ist *ReadoutEnabled* zu setzen (Zeile 06).

Anhand der *ShowConfirmation*-Eigenschaft (Zeile 07) legen wir fest, ob der erkannte Text noch einmal angezeigt (und ggf. vorgelesen) werden soll, was eine Korrektur des Nutzers erlaubt. Andernfalls wird das Ergebnis dem Programm direkt übergeben und die Auswertung kann erfolgen.

Die Spracherkennung beginnt mit dem Öffnen des Systemdialoges, der das Ergebnis des Vorgangs zurückliefert (Zeile 09). Da die Spracherkennung asynchron erfolgt, müssen wir mittels des *await*-Schlüsselwortes auf den Abschluss des Vorganges warten.

War die Spracherkennung erfolgreich (Zeile 10), kann der gesprochene Text (Zeile 12) und die Erkennungswahrscheinlichkeit (Zeile 13) abgerufen werden.

Soll bei den Sprachoptionen der Nutzer nur eine bestimmte Anzahl von Optionen haben, dann kann dies konfiguriert werden. Dazu muss die verwendete Spracherkennungsgrammatik erweitert werden. In Listing 16–4 wird gezeigt, wie die Erkennung auf die deutschen Werktage beschränkt werden kann.

```
01 var weekdays = new[] {"Montag",
                         "Dienstag", "Mittwoch",
                         "Donnerstag", "Freitag"};
02 sr.Recognizer.Grammars
                .AddGrammarFromList("Weekdays",
                                    weekdays);
```

Listing 16–4 *Eigene Grammatik für die Spracherkennung festlegen*

16.4 Sprachkommandos

Sprachkommandos werden mit einer XML-Datei beschrieben. Dafür bietet Visual Studio eine eigene Elementvorlage an: die Sprachkommandodefinition. Ein komplettes Beispiel zeigt das folgende Listing 16–5.

```
01 <?xml version="1.0" encoding="utf-8"?>
02
03 <VoiceCommands
       xmlns="http://schemas.microsoft.com/voicecommands/1.0">
04   <CommandSet xml:lang="de-DE">
05     <CommandPrefix>Tagplaner</CommandPrefix>
06     <Example>Welcher Tag ist heute?</Example>
07
08     <Command Name="DayToday">
```

```
09        <Example>Welcher Tag ist heute?</Example>
10        <ListenFor>Welcher Tag ist {day}</ListenFor>
11        <Feedback>Schlage Tag nach...</Feedback>
12        <Navigate Target="/MainPage.xaml"/>
13      </Command>
14
15      <PhraseList Label="day">
16        <Item> heute </Item>
17        <Item> morgen </Item>
18        <Item> übermorgen </Item>
19      </PhraseList>
20    </CommandSet>
21 </VoiceCommands>
```

Listing 16–5 *Sprachkommando-Definitionsdatei*

Jedes Sprachkommando beginnt mit einem **Befehlspräfix**, anhand dessen die Unterscheidung zwischen System- und App-Befehle getroffen wird. Dieses wird in der Definitionsdatei in Zeile 05 festgelegt.

Wischt man im Spracherkennungsdialog von Windows Phone nach links, bekommt man eine Liste aller installierten Apps, die Sprachbefehle unterstützen. Dazu wird ein Beispielbefehl angezeigt, der in Zeile 06 festgelegt wird.

Nachdem diese globalen Einstellungen festgelegt wurden, können eine oder mehrere Sprachbefehle definiert werden. Diese beginnen mit dem Element <Command> (Zeile 08) – das Name-Attribut definiert den eindeutigen Namen, der zur Identifizierung verwendet wird. Dieser Wert wird vom System mitgeliefert, sodass man den erkannten Sprachbefehl identifizieren kann, um entsprechend in der App reagieren zu können.

Jeder Sprachbefehl definiert eine beispielhafte gültige Verwendung über das <Example>-Tag (Zeile 09).

Anhand des Textes von <ListenFor> (Zeile 10) legt man den zu erkennenden Text fest. Hier können Platzhalter verwendet werden, die in geschweiften Klammern notiert werden. Diese Platzhalter müssen mit ihren gültigen Werten definiert werden (Zeilen 15–19).

Wenn ein Sprachbefehl erkannt wurde, wird eine Nachricht zur Rückmeldung dem Benutzer angezeigt. Dieser Text kann über das <Feedback>-Element (Zeile 11) bestimmt werden.

Abschließend wird festgelegt, welche Seite in der App aufgerufen werden soll (Zeile 12).

Bevor die definierten Sprachkommandos vom Nutzer verwendet werden können, müssen diese geladen werden. Dieser asynchrone Vorgang wird in Listing 16–6 gezeigt.

```
01 await VoiceCommandService
           .InstallCommandSetsFromFileAsync(
                new Uri("ms-appx:///VoiceDays.xml"));
```

Listing 16–6 *Laden einer Kommandodefinitionsdatei*

Im Rahmen der Kommandodefinition wird die Zielseite aufgerufen – somit liegt die Vermutung nahe, dass beim Seitenladen auf den Sprachbefehl reagiert werden sollte. Diese Vermutung ist auch korrekt. Insgesamt werden mindestens zwei *QueryString*-Parameter übergeben:

- voiceCommandName: Der Name des erkannten Kommandos
- reco: Der erkannte Text wie gesprochen

Daneben wird für jeden definierten Platzhalter ein *QueryString*-Parameter übergeben. Eine exemplarische Umsetzung ist in Listing 16–7 zu sehen.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  if (e.NavigationMode == NavigationMode.New)
  {
    if (NavigationContext.QueryString.ContainsKey(
         "voiceCommandName"))
    {
      string day;
     string dayParameter = NavigationContext.
              QueryString["day"].ToLower();
      DayOfWeek weekDay;

      switch (dayParameter)
      {
        case "morgen":
          weekDay = DateTime.Now.AddDays(1).DayOfWeek;
          day = "Morgen";
          break;
        case "übermorgen":
          weekDay = DateTime.Now.AddDays(2).DayOfWeek;
          day = "Übermorgen";
          break;
        default:
          weekDay = DateTime.Now.DayOfWeek;
          day = "Heute";
          break;
        }

        string dayName = CultureInfo.CurrentCulture.
                DateTimeFormat.GetDayName(weekDay);
        string answer = string.Format("{0} ist {1}",
                          day, dayName);
        Result.Text = answer;
    }
  }
}
```

Listing 16–7 *Reagieren auf einen Sprachbefehl*