



Text2Speech

September 14, 2026

maXbox Starter Tutorial 196

My problem was to change the voice in Microsoft Speech Object Library MSSOL as late binding in a script to sound in french or japanese?

And I found the SAPI. Some SAPI installations expose language values with a leading 0, or include several language IDs separated by semicolons or hyphen:

Microsoft Hazel Desktop - English (Great Britain)

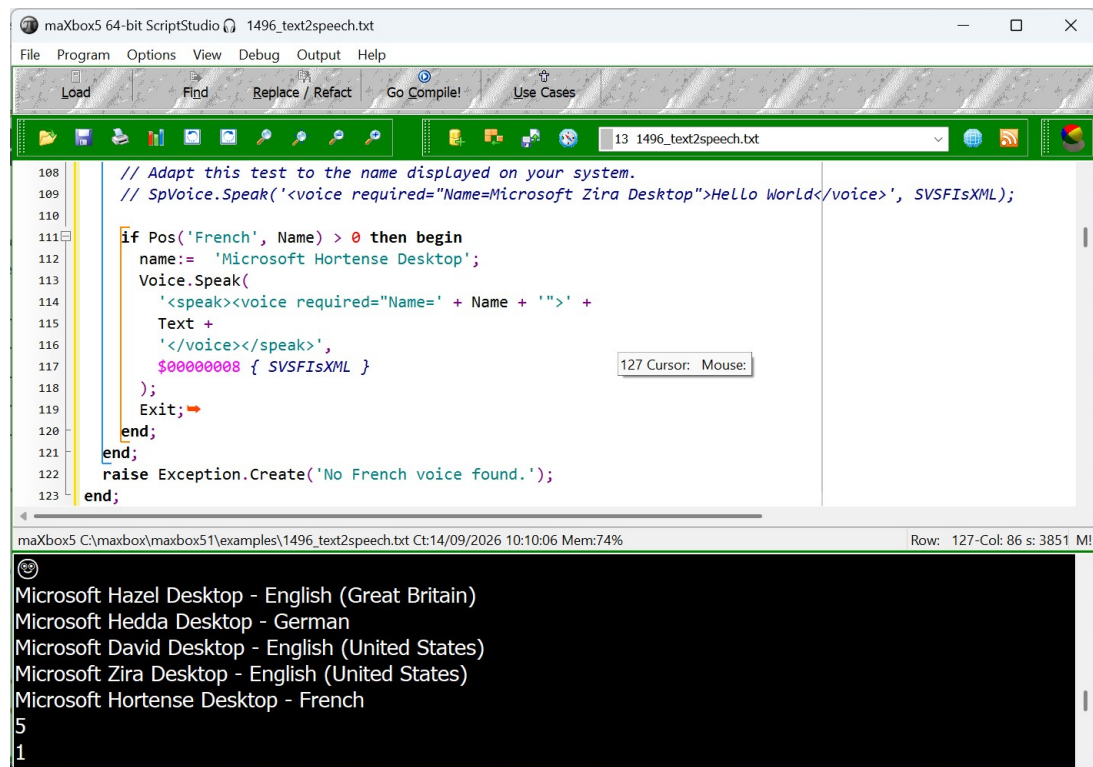
Microsoft Hedda Desktop - German

Microsoft David Desktop - English (United States)

Microsoft Zira Desktop - English (United States)

Microsoft Hortense Desktop - French

But we got some problem. **GetVoices** accepts selection criteria such as **Language**, **Gender**, **Name**, **Vendor**, and **Age**; the **Voice** property must then receive one of the returned voice-token objects.



```

108 // Adapt this test to the name displayed on your system.
109 // SpVoice.Speak('<voice required="Name=Microsoft Zira Desktop">Hello World</voice>', SVSFI/XML);
110
111 if Pos('French', Name) > 0 then begin
112   name:= 'Microsoft Hortense Desktop';
113   Voice.Speak(
114     '<speak><voice required="Name=' + Name + '">' +
115     Text +
116     '</voice></speak>',
117     $00000000 { SVSFI/XML }
118   );
119   Exit;
120 end;
121 end;
122 raise Exception.Create('No French voice found.');
```

Microsoft Hazel Desktop - English (Great Britain)
 Microsoft Hedda Desktop - German
 Microsoft David Desktop - English (United States)
 Microsoft Zira Desktop - English (United States)
 Microsoft Hortense Desktop - French

https://sourceforge.net/projects/maxbox5/files/examples/1496_text2speech.txt/download



Free short stories to download!

Download my index. It lists all the stories available on my blog.
[wineinducedshortstories.blog/2025.](https://wineinducedshortstories.blog/2025/)
 Adventure, Romance, Science, Travel.

[Explore Stories](#)

Voice is a write-by-reference COM property. Delphi's `OleVariant` late binding may fail to resolve that property assignment and report "Member not found." Use SAPI's XML `<VOICE>` tag instead, or use early binding for `SetVoice`. We use the the `<VOICE>` tag which is specifically supported by SAPI synthesis markup for selecting a voice.[documentation](#)

Most reliable late-bound solution

Select the French voice by name in SSML/SAPI markup:

```

1  if Pos('French', Name) > 0 then begin
2      name:= 'Microsoft Hortense Desktop';
3      Voice.Speak(
4          '<speak><voice required="Name=' + Name + '">' +
5          Text +
6          '</voice></speak>',
7          $00000008 { SVSFIIsXML }
8      );
9      Exit;

```

Voice or getVoice() is valid in some Automation clients, but Delphi's late-bound **OleVariant** invocation does not always handle SAPI's **Voice** property correctly because the property is declared as a COM object property with a reference-style setter. The SAPI documentation describes the property as getting and setting an **ISpeechObjectToken**, while the lower-level API exposes the equivalent operation as **SetVoice**. Therefore the SSML/SAPI markup XML should almost work:

```

1  const SVSFIIsXML = $00000008;
2
3  procedure setVoiceTest;
4  var SpVoice, Voices: Variant;
5  begin
6      SpVoice:= CreateOleObject('SAPI.SpVoice');
7      Voices:= SpVoice.GetVoices('', ''); // Enumerate installed voices
8      for it := 0 to Voices.Count - 1 do
9          writ(Voices.Item(it).getDescription);
10
11     // Specify the desired voice attribute directly in XML:
12     SpVoice.Speak('<voice = "Name=Microsoft Zira Desktop">Hell
13     SpVoice.Speak('<voice required="Name=Microsoft Hortense Desktop">
14     SpVoice.Speak('<voice required="Name=Microsoft Hedda Desktop">
15     SpVoice.Speak('<voice required="Name=Microsoft Hortense Desktop">
16
17     //SVSFIIsXML flag ($00000008
18
19     if Voices.Count > 0 then begin
20         // Use SetVoice() method rather than property assigner
21         //SpVoice.SetVoice(Voices.Item(0));
22         SpVoice.Speak('Voice successfully changed.', 0);
23     end;
24 end;

```

Note: when the XML markup syntax is not valid you get an exception to find a fix:

Exception: : SSML input not valid; cannot find 'speak' element. at 868.1198



Would You Speak Out?

If you had the most vital truth for humanity but no one believed you, would you still share it? Explore this challenge.

[Discover the story](#)

SVSFIsXML has the value 8 or in a const as hex \$00000008; . It tells SpVoice.Speak to interpret the supplied text as SAPI XML rather than ordinary text.

The flags are:

```
const
SVSFDefault = 0;
SVSFlagsAsync = 1;
SVSFPurgeBeforeSpeak = 2;
SVSFIsFilename = 4;
SVSFIsXML = 8;
SVSFIsNotXML = 16;
SVSFPersistXML = 32;
SVSFNLPSpeakPunc = 64;
```

These values are bit flags, so combine them with bitwise **or**, not arithmetic addition.

```
1  const
2      SVSFIsXML = 8;
3  var
4      Voice: OleVariant;
5  begin
6      Voice := CreateOleObject('SAPI.SpVoice');
7      Voice.Speak(
8          '<speak><voice name="Microsoft Hortense Desktop">' +
9          'Bonjour, comment allez-vous ?' +
10         '</voice></speak>',
11         SVSFIsXML or SVSFlagsAsync
12     );
13 end;
```

So now comes the big trick, you can combine these flags to talk for example, XML plus asynchronous playback means voices in a group:

```
Voice.Speak(XmlText, SVSFIIsXML or SVSFlagsAsync);
```

That is equivalent to:

```
delphiVoice.Speak(XmlText, 9);
```



Would You Speak Out?

If you had the most vital truth for humanity but no one believed you, would you still share it? Explore this challenge.

[Discover the story](#)

because $8 \text{ or } 1 = 9$

but in my script it won't work for the moment, see Appendix Async Flags!

Conclusion

The OLE exception `Member not found (0x80020003 / DISP_E_MEMBERNOTFOUND)` occurs because the SAPI COM type library defines the **Voice** property as a **propputref** (property assignment by reference). When accessing **SpVoice** via late-bound OLE **Variant** (`CreateOleObject('SAPI.SpVoice')`), Delphi's late-binding dispatch attempts a standard value assignment (**propput**) rather than a reference assignment, causing the OLE dispatch lookup to fail.

- Use **propput** for *value types* (strings, numbers, structs) where you want a copy.
- Use **propputref** for *object references* (interfaces, dispatch pointers) where you want to share the same instance.

💡 **Tip:** In Delphi's Type Library Editor, if you define a property of an interface

type, Delphi will automatically generate **propputref** for the setter. For simple types, it uses **propput**. In maXbox theres no Type Library Editor so the late binding with flags combination works.

Here's a clear breakdown:

1. propput (Value Assignment)

- **Meaning:** The property setter receives a *copy* of the value.
- **Effect:** The COM object gets its own copy, so changes to the original variable after the call won't affect the stored property.
- **Delphi Equivalent:** Passing a parameter **by value**.

2. propputref (Reference Assignment)

- **Meaning:** The property setter receives a *reference* to the value.
- **Effect:** The COM object can hold onto the same reference, so if the caller changes the original object/value later, the property reflects that change.
- **Delphi Equivalent:** Passing a parameter **by reference** (like **var** or object references).

So theres only one option in the end with a runtime script: **Option A:** Use SAPI XML Voice Tags (Easiest & Most Reliable). Bypass object token assignment entirely by embedding SAPI XML selection tags directly in the text passed to **Speak()**, using the **SVSFIsXML** flag (8):

```

1  begin
2      SpVoice:= CreateOleObject('SAPI.SpVoice');
3      // Specify the desired voice attribute directly in XML:
4      SpVoice.Speak('<voice required="Name=Microsoft Zira Desktop" />');
5  end;
```

Note: when you forget the Desktop in the name you get an:

Exception: : SSML input not valid; cannot find 'speak' element. at 877.1832

So change the voice dynamically without token object assignments by embedding SAPI XML tags directly in the text passed to **Speak()**, using the **SVSFIsXML** flag (**\$00000008**) must be checked for the syntax.

Script to download:

https://sourceforge.net/projects/maxbox5/files/examples/1496_text2speech.txt/download

Appendix: Async Flags

`SVSFlagsAsync` has the value 1. Pass it as the second argument of `SpVoice.Speak`; the call returns immediately while SAPI continues speaking in the background.[\[learn.microsoft\]](#)

Equivalent:

```
Voice.Speak('Bonjour.', 1);
```

XML and asynchronous speech

Combine `SVSFlagsXML` and `SVSFlagsAsync` with bitwise `or`:

```
const
    SVSFlagsAsync = 1;
    SVSFlagsXML   = 8;

var
    Flags: Integer;
begin
    Flags := SVSFlagsXML or SVSFlagsAsync; // 9

    Voice.Speak(
        '< speak>' +
        '< voice name="Microsoft Hortense Desktop">' +
        'Bonjour, comment allez-vous ?' +
        '< /voice>' +
        '< /speak>',
        Flags
    );
end;
```

The flags should be combined with `or`; Microsoft's examples use the same combination for XML plus asynchronous speech.[\[learn.microsoft\]](#)



Free short stories to download!

Download my index. It lists all the stories available on my blog.
wineinducedshortstories.blog/2025.
 Adventure, Romance, Science, Travel.

[Explore Stories](#)

Wait until speech finishes

Although speech is asynchronous, you can optionally wait later:

```
const
    SVSFlagsAsync = 1;
    SVEWaitUntilDone = -1;

begin
    Voice.Speak('Bonjour.', SVSFlagsAsync);

    // Wait indefinitely until the queued speech is complete.
    Voice.WaitUntilDone(SVEWaitUntilDone);
end;
```

A timeout is specified in milliseconds:

```
Voice.WaitUntilDone(10000); // Wait at most 10 seconds
```

This is useful in a console program or before releasing the **SpVoice** object. In a GUI application, avoid waiting indefinitely on the main thread, because it can make the user interface appear frozen. **WaitUntilDone** is intended to let the program wait for completion after an asynchronous **Speak** call.

[[learn.microsoft](https://learn.microsoft.com)]

Stop queued speech

Use **SVSFPurgeBeforeSpeak** together with asynchronous speech when you want a new utterance to replace anything already queued:

```
const
  SVSFlagsAsync      = 1;
  SVSFPurgeBeforeSpeak = 2;

begin
  Voice.Speak(
    'This replaces queued speech.',
    SVSFlagsAsync or SVSFPurgeBeforeSpeak
  );
end;
```

The combined value is 3.

Without `SVSFPurgeBeforeSpeak`, repeated asynchronous calls are normally queued:

```
Voice.Speak('First sentence.', SVSFlagsAsync);
Voice.Speak('Second sentence.', SVSFlagsAsync);
```

Important object-lifetime issue

Keep the `Voice` variable alive while asynchronous speech is running:

```
procedure StartSpeech;
var
  Voice: OleVariant;
begin
  Voice := CreateOleObject('SAPI.SpVoice');
  Voice.Speak('This is asynchronous.', SVSFlagsAsync);

  // Voice is released when this procedure ends.
end;
```

For a short script this may work inconsistently because the COM object can be released as soon as the procedure exits. Prefer a longer-lived object, for example a global or form field:

```
type
  TForm1 = class(TForm)
  private
    FVoice: OleVariant;
  end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FVoice := CreateOleObject('SAPI.SpVoice');
end;

procedure TForm1.ButtonSpeakClick(Sender: TObject);
begin
  FVoice.Speak(
    'Bonjour, ceci est une lecture asynchrone.',
    SVSFlagsAsync
  );
end;
```

For a one-shot console script, use:

```
Voice.Speak('Bonjour.', SVSFlagsAsync);
Voice.WaitUntilDone(-1);
```

This preserves asynchronous SAPI processing while ensuring that the script does not terminate before the speech has finished.



Belge Polytension Locs as Multilanguage Power