

Synthèse et reconnaissance vocale en .Net

par [MORAND Louis-Guillaume](#)

Date de publication : 18/05/2005

Dernière mise à jour :

Tutoriel pas à pas expliquant le principe, l'utilisation et l'intégration de la synthèse et la reconnaissance vocale dans une application en .Net

Avant-Propos

1 - L'utilisation vocale

1.1 - Introduction

1.2 - Utilisation du sdk

2 - La synthèse vocale

3 - La reconnaissance vocale

3.1 - Composants principaux

3.2 - Grammaire

3.3 - Grammaire XML

3.3.1 - Syntaxe de la grammaire

3.3.2 - Utilisation d'une grammaire dynamique

3.4 - Interaction

4 - Améliorations de la reconnaissance

Conclusion

Remerciements et liens

Avant-Propos

Qui ne s'est jamais demandé si son application ne pourrait pas être adaptée pour les malvoyants, par exemple, en "parlant" toute seule ou réagissant oralement à certaines interactions (boutons, etc); ou mieux, une application capable de réagir au son de la voix? Qui n'a jamais voulu améliorer l'interactivité de son application?

Nous verrons au long de cet article qu'il est relativement facile de mettre en oeuvre ces deux principes.

1 - L'utilisation vocale

1.1 - Introduction

La reconnaissance vocale est un concept utilisé pour interagir avec une application (homme parle à machine). Et pour avoir une interaction complète, il convient d'utiliser la synthèse vocale (machine parle à homme).

Ces deux concepts seront identifiés en tant que **SR** et **TTS**. SR pour la reconnaissance vocale (Speech Recognition) et TTS pour la synthèse vocale (Text-To-Speech).

L'utilisation vocale n'est généralement utilisé que par les "grandes" entreprises pour certaines de leurs applications internes ou alors dans des applications basées sur la reconnaissance vocale (Dragon Naturally Speaking, etc). Toutes ces applications utilisent généralement leur propre moteur vocal, comme il existe également des sociétés qui se sont spécialisées dans la création et la vente de ces moteurs.

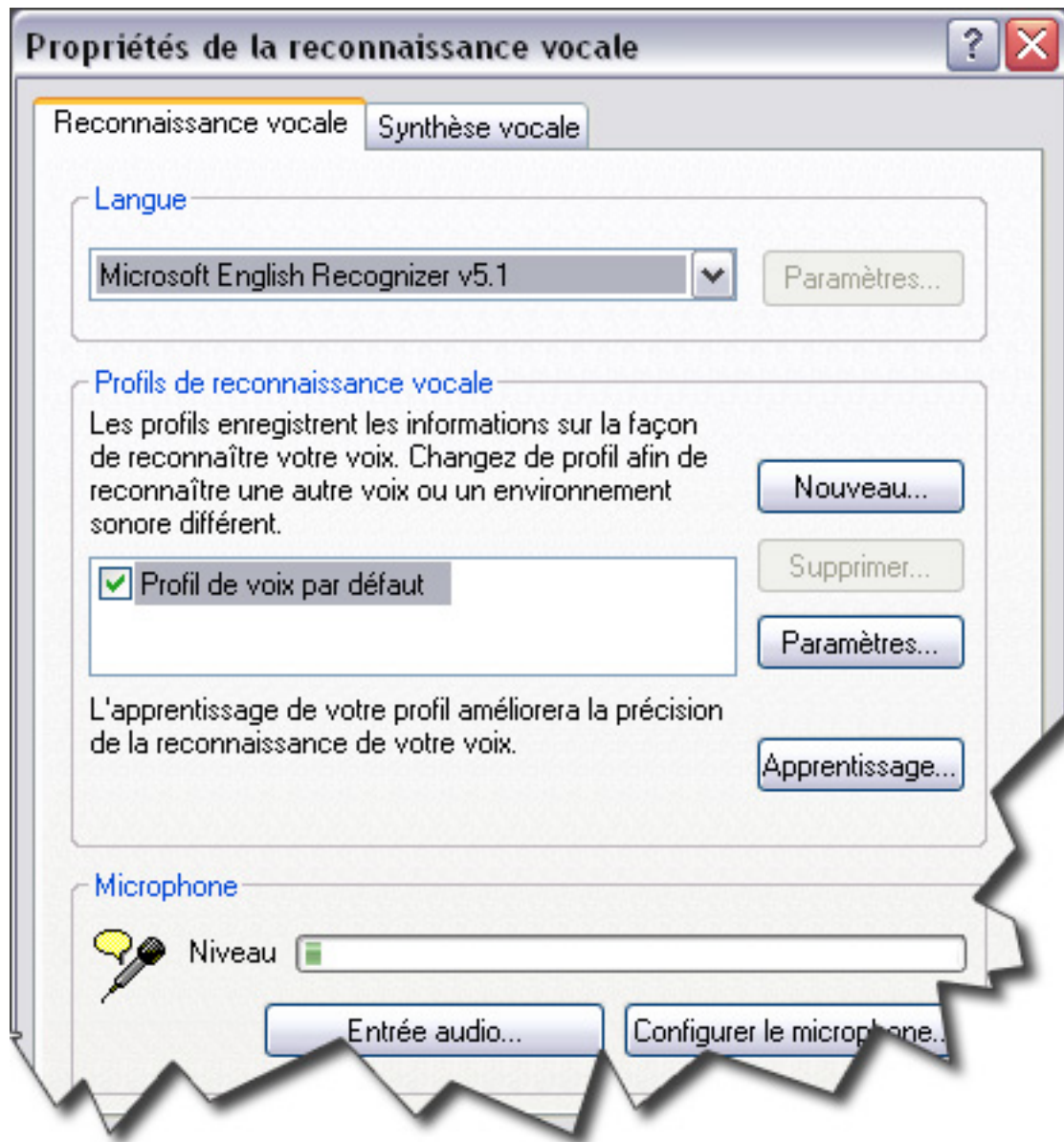
1.2 - Utilisation du sdk

Avant d'aller plus loin dans la lecture de cet article, il vous est nécessaire de télécharger et d'installer le [Speech SDK 5.1](#).

Décompressez et installez le.

Nous ne nous intéressons qu'à un fichier d'extension dll: le fichier *sapi.dll* qui se trouve par défaut dans le répertoire "**C:\Program Files\Fichiers communs\Microsoft Shared\Speech**"

Windows possède de base un moteur de reconnaissance vocale mais plus limité. Vous devez alors configurer Windows pour qu'il utilise le moteur du Speech SDK.



Panneau de configuration > Voix

Libre à vous d'utiliser le moteur vocal de votre choix, néanmoins même si le speech SDK est limité (langue anglaise uniquement (+chinois et japonais avec un "addon" complémentaire). **Ne gère pas le français** ou toute autre langue), ce dernier est gratuit et donc plus facilement utilisable.

Chaque personne a une voix bien différente de celle de son voisin, c'est pourquoi le moteur vocal de base peut ne pas reconnaître les mots de certaines personnes. D'ailleurs, certaines personnes ont des voix tellement particulières qu'elles ne seront jamais reconnues correctement par des moteurs vocaux. La plupart des moteurs dont le Speech SDK, possèdent un assistant d'apprentissage, et je vous encourage donc à créer dans le panneau

de config (voir image ci-dessus) un nouveau profil personnel. Vous devrez alors suivre un assistant d'apprentissage, qui prend à peu près 10 minutes, mais qui améliorera sensiblement la qualité de la reconnaissance du moteur.

Pensez aussi à configurer le microphone, en configurant sa sensibilité, le bruit ambiant sera moins gênant.

Votre SDK est installé et configuré, il ne vous reste plus qu'à l'appeler dans votre application. Pour cela, ouvrez votre solution, et ajoutez une référence en parcourant les dossiers et en allant chercher la dll *sapi.dll*. N'oubliez pas la clause *using* dans votre code source.

```
using SpeechLib;
```

2 - La synthèse vocale

Nous verrons tout d'abord la synthèse vocale, tout d'abord parce que celle-ci est bien plus simple d'utilisation et surtout que son implémentation est, je pense, plus courante, notamment pour les applications souhaitant implémenter des fonctionnalités pour les malvoyants.

Voici un bout de code complet et fonctionnel:

Mon premier Hello World

```
SpVoice voix = new SpVoice();
SpeechVoiceSpeakFlags flags = SpeechVoiceSpeakFlags.SVSFlagsAsync;
voix.Voice = voix.GetVoices("", "").Item(Convert.ToInt32(numericUpDown1.Value));
voix.Speak("hello world", flags);
voix.Volume=50;
```

Nous créons tout d'abord un objet Voice qui contient peu de méthodes et dont la plus utilisée sera toujours Speak(). Cette dernière prend en paramètre une chaîne de caractères à prononcer ainsi qu'un "flag" (drapeau de type *SpeechVoiceSpeakFlags*) qui sert en fait d'option pour la synthèse vocale.

Voici les différents flags ainsi que leur signification:

Nom du flag	Description
SVSFVoiceMask	Ce masque contient tous les bits de flag
SVSFIsFilename	La chaîne passée en entrée est un nom de fichier dont le contenu est lu
SVSFIsXML	Le texte passé en entrée sera parsé pour un marquage XML. En fait, la chaîne passée contient des balises XML qui servent à configurer la synthèse (ex: <VOLUME LEVEL="100">,<RATE SPEED="+10">,<PITCH MIDDLE="+10">,etc)
SVSFIsNotXML	Le texte passé en entrée ne sera pas parsé pour un marquage XML
SVSFFlagsAsync	La synthèse vocale sera asynchrone
SVSFNLPMask	Ce masque est utilisé pour enlever tous les flags liés à SAPI avant l'appel de la méthode <i>Speak()</i>
SVSFNLPSpeakPunc	Tous les symboles de ponctuation seront prononcés par leur nom. ("," devient "comma")
SVSFPersistXML	Les changements d'états globaux dans le marquage XML resteront à travers les différents appels de la méthode <i>Speak()</i>
SVSFPurgeBeforeSpeak	Purge toutes les synthèses en attente, par rapport à celle-ci
SVSFUnusedFlags	Ce masque contient tous les bits inutilisés
SVSFDefault	Les paramètres par défaut sont utilisés: <ul style="list-style-type: none"> - synthèse vocale synchrone - pas de purge - parse le texte en XML seulement s'il commence par le symbole "<"

Nom du flag	Description
	- ne transforme pas la ponctuation en mots

L'objet SpVoice contient d'autres méthodes qui vous permettent d'affiner la synthèse vocale. Les plus utilisées sont

- la méthode *WaitUntilDone()* qui bloque le moteur jusqu'à ce qu'il reçoive l'évènement *SpeakCompleteEvent*
- la méthode *Pause()* qui met en pause la diction et la méthode *Resume()* qui la reprend.
- la propriété *Volume* qui permet de définir la puissance de la voix synthétisée

3 - La reconnaissance vocale

3.1 - Composants principaux

La reconnaissance vocale se fait à partir de deux composants, ici deux objets. Nous avons tout d'abord le contexte de reconnaissance (Recognition Context) ainsi que la grammaire; cette dernière gère les mots qui seront reconnus.

Le composant est assez simple à implémenter et pourtant, il fait tout le travail d'écoute (quand la grammaire analysera l'acceptation des terminaux ou non). Ce composant, c'est l'objet *ISpeechRecoContext*. Plus précisément, nous allons instancier une classe qui implémente cette interface : *SpSharedRecoContextClass*. Son implémentation est on ne peut plus simple:

```
SpSharedRecoContextClass
```

```
monRecoContext = new SpSharedRecoContextClass();
```

Et... c'est tout. Nous rappellerons cet objet par la suite mais aucune modification ne nécessite de lui être apportée.

3.2 - Grammaire

Le deuxième composant important de la reconnaissance vocale est la grammaire. Une grammaire est une suite de règles et de terminaux ("mots") qui définissent le comportement du moteur utilisant cette grammaire. Sans le savoir, vous utilisez des grammaire régulièrement, même en parlant, et pas simplement au sens de faute de "grammaire" mais toute logique du langage (agencement des mots, etc). De même, les compilateurs utilisent des grammaires pour la reconnaissance syntaxique, ou lexicale par exemple.

Pour notre reconnaissance vocale, nous utiliserons donc également une grammaire.

La première étape consiste à créer un objet grammaire: *ISpeechRecoGrammar* puis à l'initialiser à partir de notre contexte de reconnaissance:

```
maGrammaire = monRecoContext.CreateGrammar(0);
```

Notre grammaire sera de type statique puisque nous la définissons à l'exécution et qu'elle est définie dans le code:

```
maGrammaire.DictationLoad("", SpeechLoadOption.SLOStatic);
```

Puis nous devons impérativement l'activer:

```
maGrammaire.DictationSetState(SpeechRuleState.SGDSActive);
```

A ce moment précis, notre contexte de reconnaissance est fonctionnel, il réagira à chaque règle de notre grammaire. Néanmoins, notre grammaire est vide, et il nous faut la remplir.

Créons une première règle :

```
ISpeechGrammarRule maRègleGood = maGrammaire.Rules.Add("RegleGood",
    SpeechRuleAttributes.SRATopLevel |
    SpeechRuleAttributes.SRADefaultToActive,0);
```

Puis nous allons définir le mot pour lequel cette règle sera déclenchée:

```
object proper = "";
maRègleGood.InitialState.AddWordTransition(null,"good",
    ",SpeechGrammarWordType.SGLexical,"proGood",0,ref proper,(float)1);
```

Voyons plus en détails ce que nous avons codé. Nous avons d'abord ajouté une règle dont le nom est "RegleGood", qui est active par défaut (*SRADefaultToActive*) et qui est "Top Level"(*SRATopLevel*), ce qui signifie qu'elle pourra être activée ou désactivée par programmation.

Nous avons alors ajouté un mot à cette grammaire. Ce mot est le mot "good" et dont le nom de propriété est "proGood".

L'application est alors fonctionnelle et réagira lorsqu'elle reconnaitra le mot "good". Néanmoins, cette manière de déclarer tous les mots dans le code peut être longue et fastidieuse et nous allons voir dans le chapitre suivant comment s'en passer.

3.3 - Grammaire XML

Il existe une autre grammaire: la grammaire dynamique. Celle-ci se fait à l'aide d'un fichier XML et a l'avantage de pouvoir être modifiée sans recompiler l'application. Plus encore, il est possible de doter l'application d'un panneau "options" permettant de modifier cette grammaire.

Voyons plus avant, son fonctionnement et son utilisation.

3.3.1 - Syntaxe de la grammaire

La première chose à savoir est qu'une grammaire, pour qu'elle soit fonctionnelle et reconnue par le moteur, doit suivre une structure bien précise. Ici, nous n'étudierons que la grammaire spécifique au moteur microsoft; nous ne parlerons pas des grammaires de type BNF (Backus Naur Form) ou autres, utilisées par d'autres moteurs de reconnaissance.

Voici un tableau (non exhaustif) des tags utilisables dans un fichier de grammaire XML.

Tag	Explication
<GRAMMAR>	Tout le contenu des fichiers de grammaire doit être contenu entre cette balise
<P> ou <PHRASE>	Réprésente le mot à reconnaître. Une fois ce mot reconnu, une paire (nom de propriété, valeur) est générée
<L> ou <LIST>	Définit une expression composée de différents "mots"
<O> ou <OPT>	L'élément OPT est identique à l'élément PHRASE si ce n'est qu'il n'a pas besoin d'être reconnu par la règle pour être déclenché.

Tag	Explication
<RULE>	Le tag règle (RULE) est le tag noyau d'une grammaire. Une grammaire doit contenir au moins une règle top-level et chaque règle (RULE) doit contenir une référence de règle ou un texte à reconnaître.
<RULEREf>	Le tag est utilisé pour faire référence à une règle présente dans la même grammaire.

Il existe d'autres tags qui correspondent à des cas bien précis que vous rencontrerez plus rarement dans une reconnaissance vocale (DICTATION, TEXTBUFFER, etc).

Voici donc un exemple de grammaire utilisant les tags les plus courants:

```
<GRAMMAR LANGID="409">
  <RULE NAME="developpez" TOPLEVEL="ACTIVE">
    <P>developpez</P>
  </RULE>
  <RULE NAME="message" TOPLEVEL="ACTIVE">
    <P>message</P>
  </RULE>
  <RULE NAME="hide" TOPLEVEL="ACTIVE">
    <P>hide </P>
  </RULE>
  <RULE NAME="exit" TOPLEVEL="ACTIVE">
    <L>
      <P>exit</P>
      <P>close</P>
    </L>
  </RULE>
</GRAMMAR>
```

Notez pour finir l'attribut *LANGID* qu'il est nécessaire d'ajouter au tag *GRAMMAR* sans quoi, une erreur sera générée. Ce *LANGID* d'une valeur de 409 correspond à la langue anglaise.

3.3.2 - Utilisation d'une grammaire dynamique

Nous devons maintenant charger cette grammaire XML. Vous remarquerez rapidement que nous n'aurons pas à passer par toutes les étapes fastidieuses de la création de grammaire (vues dans le chapitre 3.2). Comme dans la grammaire dite "statique", nous devons créer ("instancier") une grammaire que l'on greffe à notre *contexte*.

Instanciation

```
maGrammaireXML = monRecoContextXML.CreateGrammar(0);
```

Nous allons maintenant charger notre grammaire depuis notre fichier

Chargement

```
maGrammaireXML.CmdLoadFromFile(Application.StartupPath +
@"\grammaire.xml", SpeechLoadOption.SLODynamic);
```

Vous noterez tout d'abord le chemin du fichier, mais surtout le paramètre *SLODynamic* utilisé justement parce que cette grammaire peut-être modifiée.

Enfin, n'oubliez pas d'activer la grammaire, sans quoi, elle ne sera pas prise en compte:

Activation

```
maGrammaireXML.CmdSetRuleIdState(0, SpeechRuleState.SGDSActive);
```

3.4 - Interaction

Vous devez savoir qu'à partir du moment où vous créez votre contexte et vous activez une grammaire, le moteur de reconnaissance "tourne" et écoute constamment. Néanmoins, nous n'avons pas encore défini quand interagir et surtout, comment interagir.

Nous allons donc ajouter des événements à notre contexte de reconnaissance. L'évènement qui nous intéressera le plus est l'évènement *Recognition*:

Evenement recognition

```
monRecoContext.Recognition += new _ISpeechRecoContextEvents_RecognitionEventHandler(HandlerReco);
```

Il nous reste plus qu'à écrire la méthode appelée par notre évènement (méthode *HandlerReco()*)

Méthode appelée par l'évènement

```
#region HandlerReco
private void HandlerReco(int StreamNumber, object StreamPosition, SpeechRecognitionType
RecognitionType,
                        ISpeechRecoResult result)
{
    ...
}
#endregion
```

Cette méthode est appelée à CHAQUE fois qu'un mot est "reconnu", pas autrement. C'est à dire, que pour une grammaire de 2,3 ou 100000 mots à reconnaître, nous n'aurons qu'une seule méthode.

Dans les paramètres de cette méthode, nous nous intéresserons principalement au dernier: l'objet *ISpeechRecoResult*. A partir de ce dernier, nous pourrons récupérer le mot reconnu et la règle déclenchée (puisqu'il peut y avoir plusieurs mots pour une même règle).

```
string entrée = result.PhraseInfo.GetText(0,-1,true);
string regle = result.PhraseInfo.Rule.Name;
```

Il ne vous reste plus, probablement à l'aide d'un *switch case*, qu'à exécuter les méthodes souhaitées en fonction de la règle déclenchée:

```
switch(regle)
{
    case "RegleGood":MessageBox.Show("le mot reconnu est "+entrée);
                    break;
    case "RegleHello":MessageBox.Show("Hello");
                    break;
    default : break;
}
```

A ce niveau là, vos libertés sont infinies, du simple *MsgBox*, à la synthèse vocale en passant par une suite complète de méthodes diverses et variées.

4 - Améliorations de la reconnaissance

La reconnaissance vocale peut parfois être assez décevante, soit parce qu'elle ne comprend pas ce qu'on lui dit, soit parce qu'elle comprend quelque chose qu'elle ne devrait pas comprendre. Dans l'exemple précédent, nous aurions été en mauvaise posture, si lorsque nous souhaiterions que l'application se cache en prononçant le mot "hide", cette dernière entende le mot "exit" et se ferme. La commande exécutée, ne serait donc pas celle attendue.

Il existe alors deux moyens simples de résoudre ce problème. Le premier, je l'ai cité en introduction; il consiste à exécuter une ou plusieurs fois l'assistant d'apprentissage du moteur de reconnaissance vocal. Plus vous l'exécutez, plus la reconnaissance sera précise.

Le deuxième moyen est de travailler nous-même la précision de la reconnaissance. Pour cela, nous disposons de deux propriétés:

```
float precision = (float)result.PhraseInfo.Elements.Item(0).EngineConfidence;  
float precision2 = (float)result.PhraseInfo.Elements.Item(0).ActualConfidence;
```

La première propriété représente le score que donne le moteur à la reconnaissance qu'il vient de faire. Ce score est très aléatoire et la documentation de sapi déconseille d'ailleurs de lui faire trop confiance. Nous pouvons faire plus confiance à la deuxième propriété, qui représente le "niveau" de validité. Il existe trois niveaux: bas, moyen et élevé, représentés par les des entiers (-1,0 et 1).

Nous allons donc n'accepter que les reconnaissances jugées assez parfaites.

```
float precision = (float)result.PhraseInfo.Elements.Item(0).ActualConfidence;  
if(precision >= 1)  
{  
    switch(regle)  
    {  
        case "message":MessageBox.Show("Nouveau message");  
        break;  
        case "exit" : MessageBox.Show("L'application va être fermée");  
        ..  
    }  
}
```

Ainsi, pour chaque mot reconnu, nous exigeons que son niveau de reconnaissance soit élevé pour agir en conséquence. Attention, ce niveau de restriction peut apporter des problèmes avec des mots difficilement reconnaissables.

Conclusion

Il vous est maintenant possible d'implémenter la reconnaissance et la synthèse vocale dans toutes vos applications. Il reste néanmoins clair, que la reconnaissance vocale est encore très limitée et loin d'être parfaite. Mais si vous optimisez la reconnaissance, le résultat devrait être plus que satisfaisant.

Remerciements et liens

J'aimerais remercier tout particulièrement [Freegreg](#) pour ses corrections apportées à cet article.

Voici les sources du programme utilisé tout du long de cet article.

[Sources](#) (110ko)

Je vous encourage également à télécharger le [fichier d'aide](#) du SDK (sapi.chm)

le [Language Pack](#) si vous voulez utiliser le chinois ou le japonais (on sait jamais :D)

et le [SDK](#) NECESSAIRE!!