# Création de fichier d'aide XML Powershell.

Par Laurent Dardenne, le 07/10/2013.


Conçu avec Powershell v2 sous Windows Seven.

Site de l'auteur : http://laurent-dardenne.developpez.com/

Les fichiers sources :
http://ottomatt.pagesperso-orange.fr/Data/Tutoriaux/Powershell/Creation-de-fichier-daide-XML-Powershell/Sources.zip

# Chapitres

# 1 Principe de l'aide XML

Powershell propose un mécanisme de construction d'aide qui est imbriqué dans la déclaration d'un script ou d'une fonction. La limite de cette solution est qu'elle ne peut cibler qu'une seule culture ou langage. Pour proposer une aide multi langage, Powershell s'appuie sur le format XML nommée *Microsoft Assistance Markup Language* ou MAML.

Autant l'usage du premier mécanisme de construction d'aide est simple, autant le second est plus délicat à mettre en œuvre. Ceci est dû à un manque d'outil et ceux existant ne sont pas à mon avis des plus simples à utiliser.

Il existe bien un outil de création d'aide, mais seulement pour les cmdlets développés en dotnet :

http://blogs.msdn.com/b/powershell/archive/2007/05/08/cmdlet-help-editor-tool.aspx?PageIndex=2#comments

Un projet codeplex en cours de développement semble être une solution intéressante, mais pour le moment c'est une bêta :

https://pscmdlethelpeditor.codeplex.com/

Comme j'ai eu besoin dernièrement d'un mécanisme de construction d'aide relativement simple, j'ai complété la solution *Helps* existante, proposée ici : https://github.com/nightroman/Helps

Le Wiki de *Helps* : https://github.com/nightroman/Helps/wiki
Note : le support de la mise à jour de l'aide (Powershell v3) n'est pas pris en charge.

Son approche est de simplifier la rédaction du texte de l'aide et de s'affranchir de la connaissance du format MAML.

La déclaration de l'aide XML dans une fonction se fait de la manière suivante :

```
Function Compress-ZipFile {
# .ExternalHelp PsIonic-Help.xml
```

Sous Powershell, la construction des répertoires de l'aide se base sur celle du framework dotnet : http://msdn.microsoft.com/en-us/library/windows/desktop/dd878343(v=vs.85).aspx

### *1.1 Helps par Roman Kuzmin*

Une téléchargé le script Helps, chargez le en dotsource :

```
."C:\RépertoireInstallationDeHelps\Helps.ps1"
```

Parmi les fonctions proposées, *New-Helps* génère un script par fonction à documenter.
Son mécanisme se base sur les métadonnées renvoyées par le cmdlet **Get-Command**, car la construction de l'aide nécessite de connaitre le détail de la signature d'une fonction. C'est-à-dire ces paramètres et leurs attributs.

Prenons comme exemple la fonction suivante :

```
Function Test-CommandHelp {
 # .ExternalHelp Test-CommandHelp-Help.xml
 param (
   [Parameter(Mandatory=$true,ValueFromPipeline = $true)]
   [ValidateNotNullOrEmpty()]
  [string]$CommandName,

    [Parameter(Position=0,Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
  [string]$ModuleName,

    [Parameter(Position=1,Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [ValidateScript( {Write-warning "value= '$_'";Test-Path $_})]
  [string] $TargetDirectory
 )
  #Ne fait rien, à part déclarer des paramètres
  Write-Host "Test"
}#Test-CommandHelp
```

Récupérons les métadonnées de la fonction :

```
$CommandName=Get-Command Test-CommandHelp
```

Puis exécutons la fonction **New-Helps** en utilisant les métadonnées :

```
New-Helps -Command $CommandName
```

Celle-ci génère un script détaillant la structure de l'aide :

```
# Test-CommandHelp command help
@{
        command = 'Test-CommandHelp'
        synopsis = ''
        description = ''
        parameters = @{
```

```
                CommandName = ''
                ModuleName = ''
                TargetDirectory = ''
        }
        inputs = @(
                @{
                        type = ''
                        description = ''
                }
        )
        outputs = @(
                @{
                        type = ''
                        description = ''
                }
        )
        notes = ''
        examples = @(
                @{
                        #title = ''
                        #introduction = ''
                        code = {
                        }
                        remarks = ''
                        test = { . $args[0] }
                }
        )
        links = @(
                @{ text = ''; URI = '' }
        )
}
```

L'usage du paramètre *LocalizedData:*

```
New-Helps -Command $CommandName -LocalizedData Data
```

Insère dans le script la déclaration d'une variable de type hashtable nommée '$Data' :

```
# Test-CommandHelp command data
$Data = @{
  TestCommandHelpSynopsis = ''
  TestCommandHelpDescription = ''
  TestCommandHelpSets__AllParameterSets = ''
  TestCommandHelpParametersCommandName = ''
  TestCommandHelpParametersModuleName = ''
```

```
  TestCommandHelpParametersTargetDirectory = ''
  TestCommandHelpInputsDescription1 = ''
  TestCommandHelpOutputsDescription1 = ''
  TestCommandHelpNotes = ''
  TestCommandHelpExamplesRemarks1 = ''
}
…
```

Vous devrez renseigner la valeur des clés de cette hashtable qui constitueront le texte la documentation de la fonction *Test-CommandHelp*. Ces clés définissent certaines entrées du fichier d'aide MAML et surtout reprennent chaque paramètre déclaré par la fonction ciblée.

Les entrées de cette hashtable seront référencées plus avant dans le script généré :

```
…
# Test-CommandHelp command help
@{
  command = 'Test-CommandHelp'
  synopsis = $Data.TestCommandHelpSynopsis
  description = $Data.TestCommandHelpDescription
  parameters = @{
    CommandName = $Data.TestCommandHelpParametersCommandName
    ModuleName = $Data.TestCommandHelpParametersModuleName
    TargetDirectory = $Data.TestCommandHelpParametersTargetDirectory
  }
 …
}
```

La création de ce script suffit à décrire l'aide d'une fonction pour une langue donnée. Une fois celui-ci renseigné, il restera à construire le fichier XML qui pourra être constitué de plusieurs déclarations d'aide de fonctions. On créera un fichier XML par culture.

# 2 Principes des scripts additionnels

Afin d'éviter la duplication de code d'exemple pour chaque langage et éviter, en cas de corrections ou d'ajouts, de devoir modifier plusieurs fichiers, le script additionnel *New-CommandHelpTemplate* scinde le script d'origine en deux fichiers et crée un fichier commun.

Les fonctions additionnelles présentées ici ont été conçues à l'origine pour documenter des modules uniquement.

La suite de ce texte suppose le chargement des scripts suivant en dotsource :

```
."C:\RépertoireInstallationDeHelps\Helps.ps1"
Cd "C:\VotreRépertoireDesSources"
. .\HelpsAddon.ps1
```

## *2.1 Initialize-HelpFile*

Fonction d'initialisation des scripts de l'aide d'une fonction.

Celle-ci appelle en interne les fonctions *Initialize-Help* et *New-CommandHelp*.

Si un des modules à traiter n'est pas chargé, la fonction le charge, l'analyse puis le décharge.

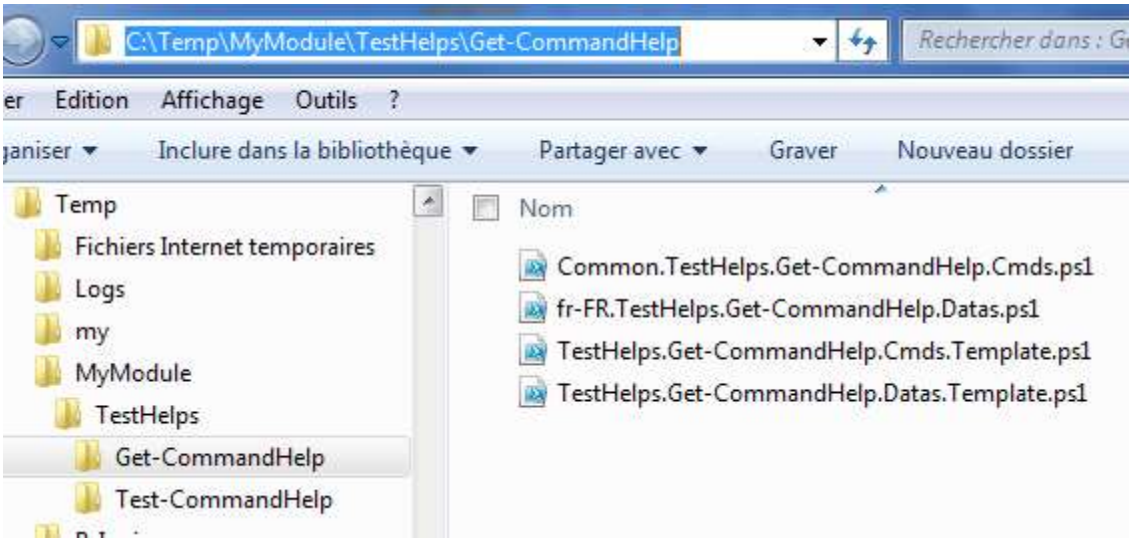
Le répertoire de destination est créé s'il n'existe pas.

Les fichiers sont écrasés s'ils existent déjà.

Par exemple pour le module TestHelps les instructions suivantes :

```
 #Répertoire du résultat
$path='C:\temp\MyModule'
Import-Module .\TestHelps.psm1 -Pass|Initialize-HelpFile $path 'Fr-fr'
```

Créent dans le répertoire indiqué par le paramètre *$Path* les scripts de création d'aide pour le Français. Plus précisément, elles créent un répertoire par fonction et chacun contient au minimum 4 fichiers :




La fonction crée un sous répertoire du nom du module (*TestHelps*) puis un sous répertoire par fonction exportée (*Get-CommandHelp* et *Test-CommandHelp*).

### 2.1.1 Détails des fichiers créés

Dans le dernier exemple, l'appel à la fonction *Initialize-HelpFile* crée les fichiers suivants :

*#Fichier de la partie exemple commun à toutes ou à plusieurs langues*
*#Fichier à modifier manuellement*
***Common.TestHelps.Get-CommandHelp.Cmds.ps1***

*#Fichier de la partie donnée pour le Français*
*#Fichier à modifier manuellement*
***fr-FR.TestHelps.Get-CommandHelp.Datas.ps1***

*#Fichier template de la partie code d'exemple*
*#Ne pas modifier ce fichier*
***TestHelps.Get-CommandHelp.Cmds.Template.ps1***

*#Fichier template de la partie donnée*
*#Ne pas modifier ce fichier*
***TestHelps.Get-CommandHelp.Datas.Template.ps1***

La construction de l'aide en Français pour la commande **Get-CommandHelp** utilisera les fichiers :
*Common.TestHelps.Get-CommandHelp.Cmds.ps1*
*fr-FR.TestHelps.Get-CommandHelp.Datas.ps1*

### 2.1.2 Nommage des fichiers utilisés lors de la construction de l'aide

Afin de construire un nom de fichier unique et reconnaissable, le nommage des fichiers est basé sur les informations suivantes :

| Contenu | Nom de fichier | Obligatoire | A modifier |
|---|---|---|---|
| Commandes communes à plusieurs cultures. | Common.**NomDeModule.NomDeFonction**.Cmds.ps1 | Oui | Oui |
| Commandes dédiées à une culture. | **NomDeCulture**.**NomDeModule.NomDeFonction**.Cmds.ps1 | Non | Oui |
| Texte de l'aide dédié à une culture. | **NomDeCulture. NomDeModule.NomDeFonction**.Datas.ps1 | Oui | Oui |
| Template du fichier du texte de l'aide. | **NomDeModule.NomDeFonction**.Datas.Template.ps1 | Oui | Non |
| Template du fichier des commandes. | **NomDeModule.NomDeFonction**.Cmds.Template.ps1 | Oui | Non |

La colonne '*A modifier'* indique les fichiers que vous devez modifier avant la génération du fichier XML.

## *2.2 New-CommandHelpTemplate*

Fonction de création des fichiers templates pour une fonction. Elle scinde le fichier d'origine, crée par la fonction New-Helps, en deux fichiers et crée un fichier commun.

Ces fichiers seront réutilisés lors de la prise en charge d'une nouvelle culture

Le répertoire de destination est créé s'il n'existe pas.

Les fichiers sont écrasés s'ils existent déjà.

Note : Si vous utilisez la fonction *Initialize-HelpFile*, l'usage de cette fonction n'est pas nécessaire.

### *2.3 New-CommandHelp*

Fonction de création des fichiers de données localisée pour une fonction.

Le fichier est écrasé s'il existe déjà.

Note : Si vous utilisez la fonction *Initialize-HelpFile* et que vous ciblé qu'une seule culture, l'usage de cette fonction n'est pas nécessaire.

Cette fonction permet de créer un fichier spécifique pour une nouvelle culture :

```
$Path='C:\temp\MyModule\TestHelps'
 #Suppose le module TestHelps.psm1 chargé
New-CommandHelp -Command 'Get-CommandHelp' TestHelps $Path 'en-US'
```

Le fichier créé est :

*#Fichier de la partie donnée pour l'Anglais US*
*#Fichier à modifier manuellement*
***en-US.TestHelps.Get-CommandHelp.Datas.ps1***

Ce fichier est créé à partir du fichier template de la partie donnée, à savoir le fichier 'TestHelps.Get-CommandHelp.Datas.Template.ps1'

La structure de la hashtable ne change pas ni ces noms de clés, seule le texte associé change. Les fichiers template ne sont pas modifiés, mais recopiés et leur date de dernier accès modifiée.

La construction de l'aide en Anglais US pour la commande **Get-CommandHelp** utilisera les fichiers :

*Common.TestHelps.Get-CommandHelp.Cmds.ps1*
*en-US.TestHelps.Get-CommandHelp.Datas.ps1*

On suppose pour cet exemple que le code commun aux deux cultures est identique et ne nécessite pas de localisation.
Dans le cas contraire utilisez le paramètre –All :

```
New-CommandHelp -Command 'Get-CommandHelp' TestHelps $Path 'en-US' -All
```

L'appel de la fonction génère deux fichiers spécifiques à l'aide US. Ainsi, la construction de celle-ci se fera à partir des fichiers suivants :

*en-US.TestHelps.Get-CommandHelp.Cmds.ps1*
*en-US.TestHelps.Get-CommandHelp.Datas.ps1*

Pour utiliser de nouveau le fichier de commande commun, supprimez le fichier de commande spécifique à la culture :

```
Remove-Item 'en-US.TestHelps.Get-CommandHelp.Cmds.ps1-Command'
```

### *2.4 ConvertTo-XmlHelp*

Fonction de conversion des scripts de données et de commandes d'une fonction en un fichier MAML au format XML.

**Pour effectuer cette étape, les scripts d'aide doivent être renseignés.**

Le répertoire de destination doit exister.

Le fichier XML généré est écrasé s'il existe déjà.

Si, dans le fichier à transformer, la valeur de la clé synopsis de la hashtable est vide, une exception est déclenchée :

```
Convert-Helps :
C:\temp\my\test\Test-CommandHelp\Common.Test.Test-CommandHelp.Cmds.ps1 :
Help of 'Test-CommandHelp': Missing or empty 'synopsis'.
```

Si la valeur d'une clé d'un des paramètres est vide un warning est émis :

```
AVERTISSEMENT : Test-CommandHelp : missing parameter description :
ModuleName
AVERTISSEMENT : Test-CommandHelp : missing parameter description :
TargetDirectory
AVERTISSEMENT : Test-CommandHelp : missing parameter description :
CommandName
```

On peut soit énumérer les noms de fonction :

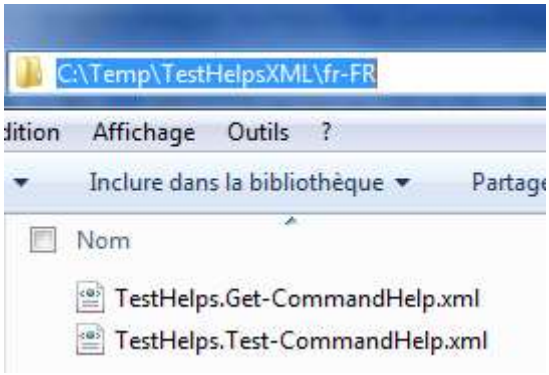
```
Cd "C:\VotreRépertoireDesSources"
$Source='C:\temp\MyModule\TestHelps'
$Target='c:\temp\TestHelpsXML'
md $Target  -ea silentlyContinue


'Test-CommandHelp','Get-CommandHelp'|
 ConvertTo-XmlHelp 'TestHelps' -Source $Source –Target $Target -Culture
'Fr-fr'
```

Soit utiliser les métadonnées du module :

```
Cd "C:\VotreRépertoireDesSources"
$Module=Import-Module .\TestHelps.psm1 -Pass
$Source='C:\temp\MyModule\TestHelps'
$Target='c:\temp\TestHelpsXML'
md $Target  -ea silentlyContinue
"fr-Fr" |
    Foreach {
      $Module.ExportedFunctions.GetEnumerator()|
       ConvertTo-XmlHelp $Module.Name -Source $Source -Target $Target -
Culture $_
    }
```

Ces instructions génèrent les fichiers suivants :




### *2.5 Join-XmlHelp*

Fonction de fusion de fichier d'aide MAML pour une culture donnée en un seul fichier XML. Elle construit le résultat final

La fonction *ConvertTo-XmlHelp* construisant un fichier XML par fonction et par culture, si on souhaite livrer un fichier d'aide commun aux fonctions d'un module ou à un ensemble de scripts, on doit regrouper toutes les déclarations MAML.

Le répertoire de source et de destination doivent exister.

Le fichier XML généré est écrasé s'il existe déjà.

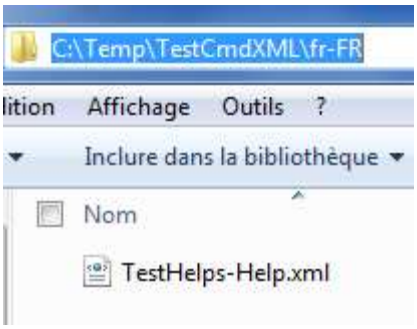
La dernière étape consiste à récupérer les fichiers xml précédemment créé :

```
Cd "C:\VotreRépertoireDesSources"
$Module=Import-Module .\TestHelps.psm1 –Pass

$Target='c:\temp\TestHelpsXML'
$ResultatFinal='C:\Temp\TestCmdXML'

md $ResultatFinal -ea silentlyContinue
  "fr-Fr" |
    Foreach {
      $Module.ExportedFunctions.GetEnumerator()|
       Join-XmlHelp $Module.Name -Source $Target -Target $ResultatFinal -
Culture $_
    }
```

Ces instructions génèrent le fichier d'aide XML final :




Contrôlons rapidement le contenu du fichier MAML :

```
[xml]$Help=get-content "$ResultatFinal\fr-Fr\TestHelps-Help.xml"
$Help.helpItems.command|% {$_.details.name}
Get-CommandHelp
Test-CommandHelp
```

Il vous reste à copier ce fichier dans le répertoire d'aide de la langue ciblée du module nommée 'TestHelps'.

### *2.6 Récapitulatif*

Ce chapitre reprend les étapes de création d'un fichier d'aide XML.

Elle est constituée de quatre étapes :

1. la construction des scripts de déclaration d'aide,
2. la modification manuelle des scripts de déclaration d'aide,
    a. l'ajout d'une nouvelle culture
3. Pour chaque culture,
    1. la construction des fichiers XML pour chaque fonction,
    2. la génération du fichier d'aide XML final.

Pensez à ajouter la clause de recherche de l'aide XML dans la signature de chaque fonction exportée :

```
Function Test-CommandHelp {
 # .ExternalHelp Test-CommandHelp-Help.xml
 param (
…
```

## 3 Autres scénarios

### *3.1 Génération de l'aide de fonction déclarée dans un script*

Une fois la fonction chargée en dotsource, l'usage reste identique, à ceci près que le nom de module doit tout de même être utilisé. Ceci ne gêne pas la construction du fichier final qui ne porte pas de référence à la notion de module :

```
New-CommandHelpTemplate -Command 'Test-CommandHelp' 'Test-CommandHelp'
c:\temp\my\test

New-CommandHelp -Command 'Test-CommandHelp' 'Test-CommandHelp'
c:\temp\my\test 'Fr-fr'
```

Complétez au moins la clé synopsis et les paramètres dans le script

*fr-FR.Test-CommandHelp.Test-CommandHelp.Datas.ps1*, puis générez le xml :

```
md c:\temp\TestCmdHelp

md c:\temp\TestCmdXML

ConvertTo-XmlHelp -Command 'Test-CommandHelp' 'Test-CommandHelp' -Source
c:\temp\my\test -Target c:\temp\TestCmdHelp  -Culture 'Fr-fr'

#puis

Join-XmlHelp -Command 'Test-CommandHelp' 'Test-CommandHelp' -Source
c:\temp\TestCmdHelp -Target c:\temp\TestCmdXML  -Culture 'Fr-fr'
```

Le nom de module est utilisé pour nommer le fichier final : Test-CommandHelp-Help.xml

### *3.2 Génération de l'aide d'un script*

Cette solution ne prend pas en compte la génération d'aide pour un script.

## 4 Liens divers

Création d'aide MAML :
http://blogs.technet.com/b/jamesone/archive/2009/07/24/powershell-on-line-help-a-change-you-should-make-for-v2-3-and-how-to-author-maml-help-files-for-powershell.aspx

Création d'aide MAML via la création de nœud XML :
http://poshcode.org/1338

National Language Support, nom des cultures :
http://msdn.microsoft.com/fr-fr/goglobal/bb896001.aspx

Fonctionnement de la recherche du fichier d'aide :
http://blogs.msdn.com/b/powershell/archive/2008/12/24/powershell-v2-external-maml-help.aspx

Voir aussi :

http://blogs.msdn.com/b/powershell/archive/2006/09/14/draft-creating-cmdlet-help.aspx

## 5 Conclusion

Il est préférable de construire l'aide en ligne une fois que vous avez figé la signature de vos fonctions, car la modification des fichiers s'avère ensuite fastidieuse.

En cas de modification de l'aide d'un module vous devez le recharger si celui-ci est déjà en mémoire.

Je trouve cette solution assez simple, bien que le nommage de fichier par l'addon puisse paraître embrouillé. Lorsque l'on a plusieurs scripts d'aide issus de plusieurs modules ce nommage prend son sens. De plus vous n'aurez pas à manipuler ces fichiers hormis dans un éditeur.

En attendant un outil plus élaboré celui-ci rendra quelques services.