Chapitre 5
Classes

1. Introduction

Les classes sont apparues depuis la version 5.0 de PowerShell. Elles ont été intégrées en très grande partie pour DSC. Leur utilisation dans DSC permet de réduire les interactions avec les fichiers de type MOF (voir le chapitre Desired State Configuration (DSC)).

La notion de classe peut paraître totalement abstraite. En vérité, avec PowerShell, on manipule des classes sans s'en rendre compte. Ou du moins, on manipule des objets instanciés par des classes du framework .NET ou mises à disposition par des éditeurs tiers. Il est très important d'avoir assimilé la notion d'objet avant d'aller plus loin.

Remarque

Attention, le but de ce chapitre n'est pas de faire un cours sur la Programmation Orientée Objet (POO).

2. Création d'une classe

Une classe permet de définir la structure d'un objet. C'est en quelque sorte son squelette. Elle définit les membres d'un objet, propriétés et méthodes notamment.

2.1 Mot-clé Class

Tous comme la déclaration d'une fonction ou d'un *workflow*, la déclaration d'une classe se réalise avec un mot-clé : `Class`. Le nom d'une classe doit être le plus représentatif possible de son fonctionnement. Il est même recommandé de faire attention au nom que l'on utilise pour éviter toute confusion avec d'autres classes.

Remarque

Pour éviter ce genre de problème, on regroupe les classes sous un même espace de noms. Exemple : `<Nom de groupement de classes>.<Nom de la classe>`.

Syntaxe

```
Class <NomDeMaClasse> {
    [Type]<Membre1>
    [Type]<Membre2>
    [Type]<Membre3>
    ...
}
```

Remarque

La notion de membre a sûrement fait écho. En effet, ce sont ces informations qu'affiche la commande `Get-Member`. Contrairement à C#, les seuls types de membres qu'il est possible de créer sont des propriétés et des méthodes. De plus, ils sont tous publics. La notion de membre privé n'a pas encore été implémentée sous PowerShell, actuellement en version 5.1. Autre syntaxe pour lister les membres d'une classe : `[Maclasse].GetMembers()`.

Exemple de classe

```
PS > Class Autor {
    $Name
    $Email
    $age
    $PostalCode
}
```

La classe `[Autor]` vient d'être créée. Les membres qui la composent sont tous des propriétés.

2.2 Propriété

Une propriété est en réalité composée de trois éléments : un champ et deux méthodes. Le champ contient la valeur. Il n'est pas définissable directement dans la classe, contrairement à C#. Les méthodes accesseur (ou *getter*) et mutateur (*setter*) permettent respectivement de lire et d'écrire la valeur contenue par le champ.

Remarque

Dans PowerShell, il n'est pas encore possible de redéfinir le comportement de ces deux méthodes. De même qu'il n'est pas possible pour le moment de retirer la méthode mutateur, et donc de passer une propriété en lecture seule. Un contournement est toutefois possible en utilisant la commande `Add-Member` à l'intérieur de la classe.

Vous aurez noté qu'une propriété est déclarée de la même manière qu'une variable ou le paramètre d'une fonction. Ce n'est pas la seule chose commune. Il est possible d'affecter un type ou une classe sur une propriété donnée.

Exemple de typage de propriété

```
PS > Class Autor {
     [String]$Name
     [String]$Email
     [int]$age
     [String]$PostalCode
}
```

Il est également possible de mettre en place un attribut de validation sur une propriété, tout comme les paramètres d'une fonction. Les attributs de validation sont donnés dans le chapitre Fonctions avancées, inutile donc d'aborder de nouveau ce point.

Exemple d'attributs de validation

```
PS > Class Autor {
        [validatenotnullorempty()][String]$Name
        [ValidatePattern("^[a-z0-9.-_]+@[a-z0-9.-_]+`.[a-z]{2,4}")]
        [String]$Email
        [ValidateRange(1,160)][int]$age
        [ValidateRange(01053,98821)][int]$PostalCode
}
```

Enfin, on peut affecter des valeurs par défaut aux propriétés.

Exemple de valeur par défaut

```
PS > Class Autor {
        [validatenotnullorempty()][String]$Name = 'Nicolas BAUDIN'
        [ValidatePattern("^[a-z0-9.-_]+@[a-z0-9.-_]+`.[a-z]{2,4}")]
        [String]$Email = 'nicolas.baudin@frpsug.com'
```

```
        [ValidateRange(1,160)][int]$age = 25
        [ValidateRange(01053,98821)][int]$PostalCode
}
```

Dans le cas où aucune valeur n'est renseignée par défaut sur une propriété, c'est la valeur par défaut du type qui est utilisée. La propriété `PostalCode` est alors affectée de la valeur 0.

2.3 Méthode

Tout comme les propriétés, les méthodes doivent avoir un type. Si une méthode ne renvoie aucune valeur, alors le type `[void]` est utilisé. Ce dernier est présent par défaut. Dans le cas d'une méthode avec un retour de valeur, le mot-clé `Return` doit être spécifié.

Méthodes UpdateAge() et GetNewBornYears()

```
PS > Class Autor {
    ...
    [int]$BornYear
    [void]UpdateAge($Age){
        $this.age = $Age
        $Now = Get-Date
        $this.BornYear = $Now.AddYears(-$age).Year
    }
    [int]GetNewBornYears($Age){
        $Now = Get-Date
        $NewBornYear = $Now.AddYears(-$age).Year
        Return $NewBornYear
    }
}
PS > $Me = [Autor]::New(25)
PS > $Me
Name       : Nicolas BAUDIN
Email      : nicolas.baudin@frpsug.com
Age        : 25
PostalCode : 0
BornYear   : 1991
```

Remarque

La méthode `New()` est ici un constructeur, cette notion sera abordée dans la section suivante.

On a ajouté la propriété `BornYear` correspondant à l'année de naissance. La méthode `UpdateAge()` affecte la propriété `Age` et calcule, à une année près, l'année de naissance, et finalement l'affecte à la propriété `BornYear`.

La méthode `GetNewBornYears()` calcule l'année de naissance et retourne la valeur. Elle n'altère pas les propriétés de l'objet.

Résultat de l'utilisation de la méthode GetNewBornYears

```
PS > $Me.GetNewBornYears(30)
1986
```

Les méthodes sont utilisables même après l'instanciation de l'objet. En C#, il est possible de rendre des membres inaccessibles via le mot-clé `private`. Malheureusement, cette possibilité n'est pas encore disponible sous PowerShell. Toutefois, nous verrons plus loin qu'il est possible de pallier cette carence.

2.4 Constructeur

Un constructeur est un type de méthode particulier. Il a pour fonction d'instancier un objet. L'instanciation d'un objet a pour but de lui réserver un espace mémoire puis d'initialiser les membres de l'objet. De ce fait, il ne dispose pas de type de retour vu qu'il ne renvoie rien. Pour définir un constructeur, il suffit de définir une méthode qui porte le même nom que la classe.

Exemple de constructeur

```
PS > Class Autor {
    ...
    Autor(){
            $this.age = 25
    }
}
PS > $Me = [Autor]::New()
```

La syntaxe `[Maclasse]::New()` permet de faire appel au constructeur par défaut et donc d'instancier l'objet à partir de la classe.

Remarque

Dans le cas présent, l'utilisation de `$this` fait appel à l'objet en cours d'instanciation. On peut donc utiliser l'ensemble de ses membres non statiques.

Il est possible de déclarer des paramètres pour les constructeurs, tout comme pour les méthodes.

Exemple de constructeur avec paramètres

```
PS > Class Autor {
        ...
        Autor([ValidateRange(1,160)][Int]$NewAge)
        {
              $this.age = $NewAge
        }
}
PS > $Me = [Autor]::New(25)
```

Remarque

Si jamais un objet doit être construit avec des variables extérieures à la classe, il est conseillé de faire appel aux paramètres du constructeur, comme dans l'exemple précédent. Une autre méthode est possible, mais elle n'est pas à privilégier.

Une classe peut définir plusieurs constructeurs. Ces différentes déclarations sont possibles grâce à la mise en place de paramètres.

Exemple de surcharge de constructeur

```
PS > Class Autor {
        ...
        Autor([ValidateRange(1,160)][Int]$NewAge)
        {
              $this.age = $NewAge
        }
        Autor(
              [ValidateRange(1,160)][Int]$NewAge,
              [ValidateRange(01053,98821)][int]$PostalCode

        )
        {
         $this.age = $NewAge
        }
}
PS > $Me = [Autor]::New(25)
PS > $Me2 = [Autor]::New(25,44000)
```

On vient de créer deux objets instanciés par la même classe, mais avec deux constructeurs différents. Cette technique est abordée plus loin dans ce chapitre.

Pour connaître la liste des constructeurs disponibles sur une classe donnée :
`[<MaClasse>].GetConstructors()`