Use of simultaneous processing using PowerShell threads

If a long script has to perform many tasks, this can take a lot of time in PowerShell. As PowerShell is not multithreaded by nature and processes are always executed sequentially, so-called threads can be used to minimise the script length.

Example Code

This code serves as a template for the parallel execution of processes. The tricky part with threads is the handling of input parameters and output values.

The script part is defined in the $Scriptblock variable. There you can work with arguments that are passed to the thread. They can be used with the $args variable and the element number. Values can be returned using the keyword "return".

```
$Scriptblock = {
	Start-Sleep 5
	return $args[0]
}

$MaxThreads = 10
$RunspacePool = [runspacefactory]::CreateRunspacePool(1, $MaxThreads)
$RunspacePool.Open()

$Jobs = @()
$InputObjects = @(
	"<yourinputobject1>",
	"<yourinputobject2>",
	"<yourinputobject3>",
	"<yourinputobject4>",
	"<yourinputobject5>"
```

```
)
$OutputObjects = @()

foreach ($Object in $InputObjects) {
	$Instance = [powershell]::Create()
	$Instance.RunspacePool = $RunspacePool
	$Instance.AddScript($ScriptBlock).AddArgument($Object) | Out-Null
	$Jobs += New-Object PSObject -Property @{
		Instance = $Instance
		State    = $Instance.BeginInvoke()
	}

}
while ($Jobs.State.IsCompleted -contains $false) {
	Start-Sleep 1
}
foreach ($job in $jobs) {
	$OutputObjects += $job.Instance.EndInvoke($job.State)
}

$Instance.Dispose()
```

Revision #2
Created 18 January 2024 17:10:19
Updated 21 July 2024 15:11:16