**The Ultimate Guide to Building REST APIs with PowerShell**

[**PowerShell**](https://blog.ironmansoftware.com/tags/powershell/) **[PowerShellUniversal](https://blog.ironmansoftware.com/tags/powershelluniversal/)** **[UniversalDashboard](https://blog.ironmansoftware.com/tags/universaldashboard/)**

**September 28, 2020**

This guide will cover how to build REST APIs using [PowerShell Universal](https://ironmansoftware.com/powershell-universal) and PowerShell. Most of this post can be accomplished without purchasing an [API license](https://ironmansoftware.com/pricing).

**Getting started with PowerShell Universal APIs**

This video outlines how to get started with PowerShell Universal APIs.

**What is a REST API?**

[Representational state transfer](https://en.wikipedia.org/wiki/Representational_state_transfer) (REST) is a software architecture that defines how web services should be designed. In practice, it takes advantage of the HTTP verbs to dictate which actions are requested by the client and uses the URL and query string to identify which resources are being acted upon.

For example, you may have an API that provides access to users that can be addressed by the URL /api/user. You can then perform different operations on the users based on the HTTP verb. A GET request would return users. A POST request would create a new user. A DELETE request would remove a user.

REST APIs are very popular and you are probably using them already; even if you don’t realize it.

**REST APIs with PowerShell**

To create simple endpoints with PowerShell, we will use the New-UDEndpoint cmdlet and specify a URL as well as a verb. Below you’ll find all the recipes for creating REST APIs with various options.

**Parameters**

This section will cover ways to access various information about HTTP requests. All of these techniques can be used with any HTTP verb.

**Basic Endpoint**

New-PSUEndpoint -Url "/user" -Endpoint {

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

)

}

Invoke-RestMethod http://localhost:10001/api/user

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

bruce Bruce Willis

tom Tom Hanks

#>

**Route Parameters**

You can specify parameters by including a URL segment that starts with a colon (:).

New-PSUEndpoint -Url "/user/:username" -Endpoint {

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

) | Where-Object UserName -eq $username

}

Invoke-RestMethod http://localhost:10001/api/user/bruce

<#

UserName FirstName LastName

-------- --------- --------

bruce Bruce Willis

#>

**Multiple route parameters**

You can specify multiple route parameters.

New-PSUEndpoint -Url "/user/:first/:last" -Endpoint {

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

) | Where-Object { $\_.FirstName -eq $first -and $\_.LastName -eq $last }

}

Invoke-RestMethod http://localhost:10001/api/user/adam/driscoll

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

#>

**Query string**

PowerShell Universal API will set parameters based on the query string as well as route parameters.

New-PSUEndpoint -Url "/user" -Endpoint {

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

) | Where-Object UserName -eq $username

}

Invoke-RestMethod http://localhost:10001/api/user?username=adam

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

#>

**Headers**

The $Headers variable has all the information about the HTTP request.

New-PSUEndpoint -Url "/user" -Endpoint {

$username = $Headers["UserName"]

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

) | Where-Object UserName -eq $username

}

Invoke-RestMethod http://localhost:10001/api/user -Headers @{ UserName = "adam" }

**Regular Expressions**

You can also specify a regular expression within your route. You can use named groups to retrieve parameters that are part of the regular expression.

New-PSUEndpoint -Url "nodes\(agent=(?<name>[0-9]\*)\)" -Method "GET" -Endpoint {

$name

} -EvaluateUrlAsRegex

Invoke-RestMethod -Uri 'http://localhost:10001/api/nodes(agent=1234)' -Method GET

<#

1234

#>

**HTTP Verbs**

This section will cover how to expose endpoints that implement various HTTP verbs.

**GET**

The GET verb is used for returning data to the client. It should not modify state.

New-PSUEndpoint -Method GET -Url "/user" -Endpoint {

$Data

}

Invoke-RestMethod http://localhost:10001/api/user

**POST**

The POST verb is used for creating a new entity based on the data specified by the client.

New-PSUEndpoint -Method POST -Url "/user" -Endpoint {

$User = $Body | ConvertFrom-Json

$Data = @(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

)

$Data += User

$Data

}

Invoke-RestMethod http://localhost:10001/api/user -Method Post -ContentType "application/json" -Body (@{

UserName = "bill"

FirstName = "Bill"

LastName = "Clinton"

} | ConvertTo-Json)

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

bruce Bruce Willis

tom Tom Hanks

bill Bill Clinton

#>

**PUT**

The PUT verb is used for updating an existing entity.

New-PSUEndpoint -Method PUT -Url "/user/:username" -Endpoint {

$Data = @(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

)

$User = $Data | Where-Object UserName -eq $UserName

$UpdatedUser = $Body | ConvertFrom-Json

$User.FirstName = $UpdatedUser.FirstName

$User.LastName = $UpdatedUser.LastName

$Data = $Data | Where-Object UserName -ne $UserName

$Data += $User

$Data

}

Invoke-RestMethod http://localhost:10001/api/user/adam -Method Put -ContentType "application/json" -Body (@{

UserName = "adam"

FirstName = "Adam"

LastName = "Smith"

} | ConvertTo-Json)

<#

UserName FirstName LastName

-------- --------- --------

bruce Bruce Willis

tom Tom Hanks

adam Adam Smith

#>

**DELETE**

The DELETE verb is used for deleting an existing entity.

New-PSUEndpoint -Method DELETE -Url "/user/:username" -Endpoint {

$Data = @(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

)

$Data = $Data | Where-Object UserName -ne $UserName

$Data

}

Invoke-RestMethod http://localhost:10001/api/user/adam -Method Delete

<#

UserName FirstName LastName

-------- --------- --------

bruce Bruce Willis

tom Tom Hanks

#>

**Files**

PowerShell Universal API will provides a $Data variable that contains the byte array for an HTTP request. Unlike the body variable, it will not be a string. You can save this data to a file or process it will other cmdlets.

New-PSUEndpoint -Method POST -Url "/user/image" -Endpoint {

[System.IO.File]::WriteAllBytes("$env:temp\image.png", $Data)

}

Invoke-RestMethod http://localhost:10001/api/user/image -Method Post -ContentType "image/png" -InFile ".\myImage.png"

**Content Types**

You can return data of other content types, such as XML, by using the New-PSUAPIResponse cmdlet before returning your data from within your endpoint.

New-PSUEndpoint -Url "project" -Method "GET" -Endpoint {

New-PSUApiResponse -ContentType 'application/xml' -Body "<Project name=`"test`"></Project>"

}

Invoke-RestMethod -Uri http://localhost:10001/api/project -Method GET -ContentType "application/xml"

<#

Project

-------

Project

#>

**Authentication**

Authentication requires a [PowerShell Universal API license](https://ironmansoftware.com/pricing). Tokens can be generated in the PowerShell Universal Admin Console.

New-PSUEndpoint -Method GET -Url "/user" -Endpoint {

@(

[PSCustomObject]@{ UserName = "adam"; FirstName = "Adam"; LastName = "Driscoll" }

[PSCustomObject]@{ UserName = "bruce"; FirstName = "Bruce"; LastName = "Willis" }

[PSCustomObject]@{ UserName = "tom"; FirstName = "Tom"; LastName = "Hanks" }

)

} -Authentication

Invoke-RestMethod http://localhost:10001/api/user -Headers @{ "Authorization" = "Bearer $($token)"}

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

bruce Bruce Willis

tom Tom Hanks

#>

**Authorization**

Authentication requires a [PowerShell Universal API license](https://ironmansoftware.com/pricing). Roles can be configured in the PowerShell Universal Admin Console.

New-PSUEndpoint -Method GET -Url "/user" -Endpoint {

$Data | ConvertTo-Json

} -Role 'Administrator' -Authentication

Invoke-RestMethod http://localhost:10001/api/user -Headers @{ "Authorization" = "Bearer $($token)"}

<#

UserName FirstName LastName

-------- --------- --------

adam Adam Driscoll

bruce Bruce Willis

tom Tom Hanks

#>