YRKESHÖGSKOLAN NOVIA

Automation of routine tasks with PowerShell

Dennis Renkonen

Degree Thesis for Engineer (Bachelor of Engineering)

Degree Programme in Information Technology

Vasa 2020

EXAMENSARBETE

Författare: Dennis Renkonen
Utbildning och ort: Informationsteknik, Vasa
Inriktningsalternativ/Fördjupning: Informationsteknik
Handledare: Kaj Wikman

Titel: Automation av rutinarbete med PowerShell

Datum 11.12.2020 Sidantal 25 Bilagor 2

Abstrakt

Det här arbetet beskriver en process för automation av rutinarbeten med hjälp av verktyg som PowerShell. Grunden för arbetet gjordes under min praktikperiod vid Wärtsilä där jag arbetade med Wärtsiläs DCM365 produkt som är byggd på Microsoft SharePoint.

Idéen för det här arbetet kom från mina egna observationer och erfarenheter med att utföra rutinarbeten mot SharePoint. Min forskning centrerades runt de problem jag försökte lösa och procedurerna jag implementerade för att lösa problemen.

Resultatet av det här arbetet är två PowerShell script som kan skräddarsys för olika användningsändamål. Teknikerna som användes är främst SharePoint och PowerShell.

Språk: Engelska Nyckelord: SharePoint, PowerShell, Automation

---

BACHELOR'S THESIS

Author: Dennis Renkonen
Degree Programme: Information Technology, Vaasa
Specialization: Information Technology
Supervisor(s): Kaj Wikman

Title: Automation of routine tasks with PowerShell

Date 11.12.2020 Number of pages 25 Appendices 2

Abstract

This thesis describes a process of automating routine tasks using tools such as PowerShell. The ground work was done during my internship for Wärtsilä where I worked on Wärtsilä's DCM365 product which is built on Microsoft SharePoint.

The idea for this project came from my own observations and experiences working with routine tasks on SharePoint. My research was centered around the problems I was trying to solve and the procedures I implemented to solve those problems.

The result of this work is a set of PowerShell scripts that can be tailored for different use cases. Technologies used are mainly SharePoint and PowerShell.

Language: English Key words: SharePoint, PowerShell, Automation

---

Table of contents

Abbreviations and definitions

SharePoint	Web-based collaborative platform
PowerShell	Task automation and management tool
Microsoft 365	Subscription service for Microsoft Office products (Formerly Office 365)
DCM365	Document Control Module 365
.NET	Application development framework by Microsoft
Cmdlet	A lightweight command used in the PowerShell environment
IDE	Integrated Development Environment
ISE	Integrated Scripting Environment
API	Application Programming Interface

1 Introduction

This thesis was developed during my time working as a trainee for Wärtsilä. I got the idea for the project from my own observations and desire to make my own work easier. The purpose of this thesis is to optimize work processes and reduce repetitive work.

1.1 Employer

Wärtsilä is a Finnish company that was originally established in 1834 as a sawmill in the village of Wärtsilä in the county of North Karelia. Commonly known for its diesel engines, Wärtsilä also produces additional equipment and services for two main business divisions. The Energy Business division, which is focused around power plants and supporting services. The Marine Business which focuses on power systems and services for the marine industry. Wärtsilä currently employs around 19000 people in over 200 locations in more than 80 countries around the world. [1]

1.2 DCM 365

During my internship for Wärtsilä I worked in the Energy Business division for the Project documentation department with a team focused on developing and maintaining IT applications. The main product that this team is working on is called Document Control Module 365 (DCM365).

DCM365 is a collaboration and document sharing platform developed and maintained by Wärtsilä. It is built and hosted on SharePoint, which integrates with Office 365. DCM365 is targeted towards Wärtsilä's power plant projects and the end users of the platform consist of Wärtsilä employees, Wärtsilä partners and project site engineers. The product is packaged on a per-project basis where every project has its own SharePoint site collection that includes subsites for project collaborators.

One of the main features of the DCM365 platform is document sharing and reviewing. The document control functions allow for submittal of documents for different purposes such as for review or for approval. The submitted documents can be reviewed and commented on by the receiver.

1.3 Investigation of how to automate work towards SharePoint

My task when working on DCM365 has been to set up new DCM365 sites and to do some maintenance and update work on existing sites. Setting up a new SharePoint site with the DCM365 templates includes several steps that are mostly done in a web browser environment. Updating the sites also usually requires some movement through the SharePoint web interface. This thesis investigates how this process could be automated.

1.4 Problem specification

Setting up a new SharePoint site with the DCM365 templates includes several steps. A DCM365 site is deployed on SharePoint from templates and according to project specifications. The specifications generally include a project number, country and partner names. The site deployment process is mostly done in a web browser environment.

After the SharePoint site collection has been set up, a new site is deployed from a template on the site collection. Project specific details are then edited. This includes project number and name, navigation links, user groups and permissions.

The following step is to set up several subsites for different collaborators in the project, such as site engineers, project managers and partners. Some settings are also updated including navigation links, groups and permissions for the subsites. Finally the document control functions are tested using test files. It is made sure that submitted documents are sent to the correct receiver and that the commenting functions work.

Some PowerShell scripts were being used to automate parts of this process but the remaining manual work still amounts to about a few hours of work. The manual work consists mostly of simple browser interactions and data field updates which can theoretically easily be automated.

DCM365 sites may also need maintenance work and updates. Currently Wärtsilä has a few hundred of these sites so an update could require accessing a few hundred sites in total. Doing this the manual way requires opening every site in a web browser one by one and navigating to where you want to make an update. This process could be further automated in the same way as the site deployment process.

2 Technologies

2.1 Microsoft SharePoint

SharePoint is a cloud-based content management system developed by Microsoft. It was first launched in 2001. SharePoint integrates with Microsoft 365 to include access to Microsoft Office tools such as Word and Excel. SharePoint serves as a solution for internal collaboration for organizations worldwide.

The system is sold in different editions. SharePoint in Microsoft 365 refers to a SharePoint solution included with Microsoft 365 subscriptions. It is hosted by Microsoft and suits businesses of all sizes. SharePoint Server is a self-hosted solution that provides more control over the implementation of the system but requires greater management. [2]

2.1.1 SharePoint front end structure

Planning of a SharePoint network can be a complicated matter. Microsoft offers design recommendatoins for how to build a SharePoint network including concepts and definitions that may change over time following new developments in SharePoint. A general idea for a company SharePoint network would be to have one company base site and dedicated site collections or hub sites for different departments or different teams which in turn can include additional subsites. [3]

The most relevant concept for this thesis is the concept of sites and site collections. A site collection can include one or several subsites. All of the subsites in a site collection will inherit properties like navigation, permissions and branding from the top-level site in the collection. Different site collections will however not share data and should be used when you want to separate functions. [4]

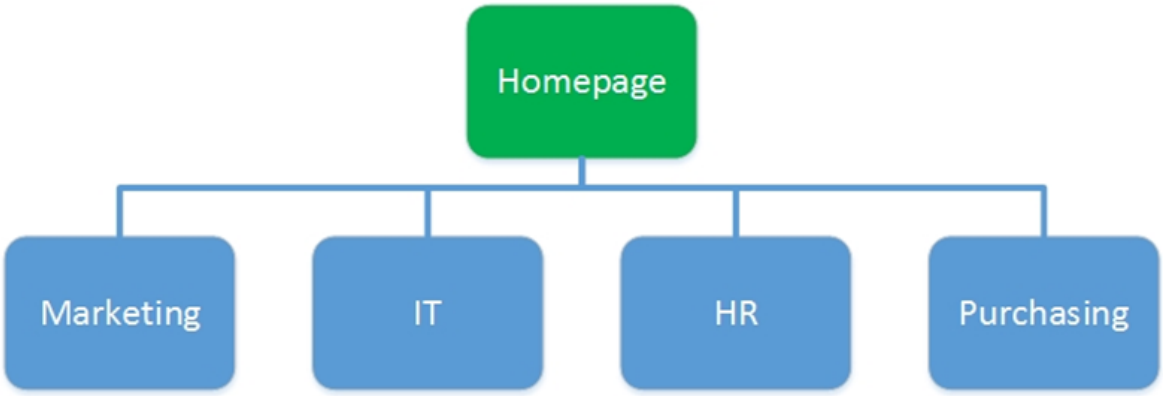

Figure 1. SharePoint site collection. [4]

The DCM365 product uses different site collections for different projects as they should function as independent units and not share information.

2.1.2 Microsoft SharePoint Designer

SharePoint Designer is a Windows application made for interaction with SharePoint websites. It allows you to build, design and access the content of SharePoint sites. SharePoint Designer enables development of features such as workflows without writing code. SharePoint workflows are usually used for automating some task such as updating one data source when another data source changes. SharePoint Designer is no longer updated by Microsoft but it is still required for maintaining older SharePoint features such as workflows. [5]

2.1.3 SharePoint REST API

SharePoint offers a REST API that allows you perform create, read, update and delete (CRUD) operations with HTTP requests to the API endpoints. One benefit of using the REST API is not having to reference additional libraries or modules in your code. The SharePoint REST API endpoints follow the pattern “https://{site_url}/_api/site” to access a specific site collection and “https://{site_url}/_api/web” to access a specific site. [6]

2.2 PowerShell

PowerShell is a task automation and configuration management framework developed by Microsoft. It consists of a command-line interface and a scripting language. Its features include aliases, pipelines and objects. [7] PowerShell can run cmdlets, PowerShell script files with the file extension .ps1, PowerShell functions and standalone executables.

Windows PowerShell 5.1 is the edition that comes packaged with Windows installations and was initially designed for Windows on top of the .NET Framework. PowerShell Core is a newer edition that is built on the .NET Core framework with multi-platform support, released as PowerShell Core 6.0 in 2018 [8]. PowerShell Core may initially lack some features and compatibility with older modules but new developments for PowerShell is focused on the multi-platform PowerShell Core edition [9].

2.2.1 Cmdlets and aliases

A cmdlet can be seen as a small command for PowerShell that follows certain rules. It is technically a .NET class that outputs a .NET object. A cmdlet usually has a single purpose. Cmdlets follow a “Verb-Noun” naming convention, for example “Get-Date” which outputs the current date. PowerShell comes bundled with a set of cmdlets but users can also create their own or download modules that contain additional cmdlets. [10]

Cmdlets can also be accessed by simpler names with the alias feature. An alias is a defined alternative name for a given cmdlet. Some of the pre-defined aliases can be familiar to UNIX users such as “man” which is an alias for Get-Help, displaying helpful information about a cmdlet. By using the Set-Alias cmdlet you can define your own aliases. All aliases for the current system can be listed with the Get-Alias cmdlet. [11]

2.2.2 Object output

The output of a PowerShell command is in the form of an object. An object is a variable with extra information attached to it. In PowerShell this information is stored in attributes that are referred to as members. The most common types of members are methods and properties. A method is a member that does something and a property usually stores something. These members can be found by using the pipeline feature (the “|” operator) and the Get-Member cmdlet. [12]

```
Get-Date | Get-Member
```
Code example 1: Cmdlets and pipeline

In this example the object result of the Get-Date cmdlet is piped through the Get-Member cmdlet and will output the members of Get-Date. This includes associated properties such as Day or Year. These properties can be accessed as in the following example.

```
$date = Get-Date
$date.Year
```
Code example 2: Variable and property

The output of the Get-Date cmdlet is stored in a variable (a variable can be identified by the dollar sign). By storing the output in a variable its properties can be accessed with the "." operator. The output of "$date.Year" will be only the value of the Year property of the Get-Date result.

PowerShell also supports filtering and sorting of objects in a fashion similar to SQL syntax with the Select-Object, Where-Object and Sort-Object cmdlets. Advanced queries can be built by utilizing the pipeline feature. [12]

2.2.3 Functions and scripts

A function in powershell is defined with the "function" command.

```
function PrintGreeting ($name) {
    $date=get-date
    write-host Hello $name, today is $date.DayOfWeek!
}
```
Code example 3: Defining function

In the example the input is stored in the $name variable and the result is a personalized message written to the output line. In the following example the user is asked to enter a value for the $name variable and the function will write a message using the user input.

```
$name=read-host "Enter your name"
PrintGreeting $name
```
Code example 4: Calling function

The output of this function looks like this.

```
Enter your name: Dennis
Hello Dennis today is Wednesday !
```
Code example 5: Function output

The previous examples can be run directly in PowerShell but if you want to run it again later you can save it as a PowerShell script file with the extension ".ps1". A PowerShell script file can be run by opening it in the PowerShell command line. Script execution will however be restricted on most Windows systems by default so the execution policy needs to be changed before you can run your scripts. The current execution policy can be checked with the Get-ExecutionPolicy cmdlet and a new one can be set with the Set-ExecutionPolicy cmdlet. [13]

2.2.4 Modules

A module in PowerShell is a package of commands. PowerShell includes some pre-installed modules but additional modules can be installed. All currently loaded modules can be listed with the Get-Module cmdlet. All modules installed on the system can be listed with "Get-Module -ListAvailable". [14]

The main module of interest for this thesis is called PnP PowerShell. PnP PowerShell is an open-source, community-driven PowerShell module that contains commands for interaction with SharePoint. [15]

2.2.5 Windows PowerShell ISE

PowerShell scripts can be written and edited in any text editor but there is a dedicated application called PowerShell ISE. PowerShell ISE comes with helpful features for writing PowerShell scripts, cmdlets and modules such as syntax coloring, tab completion and debugging functionality. Most of the development work for this thesis was done in PowerShell ISE. [16]

2.2.6 Asynchrous operations

PowerShell supports asynchronous operations, meaning several operations performed at the same time. Running asynchronous operations in PowerShell requires some extra coding structure. Any operation that you want to run asynchronously is known as a "job" in PowerShell.

An asynchronous operation is started with the Start-Job cmdlet. It will require a script block to be provided in the ScriptBlock parameter. Additionaly cmdlets are required for waiting

for the operation to finish and for receiving the result of the operation, these are Wait-Job and Receive-Job. Get-Job will get all currently active jobs. The result of the job in the following example will be the “Hello World!” message printed.

```
$helloJob = Start-Job -ScriptBlock { Write-Host "Hello World!" }
Wait-Job $helloJob
Receive-Job $helloJob
```

Code example 6: Basic asynchronous structure

2.3 Alternative technologies

2.3.1 Python

Python is a dynamically typed programming language, often called scripting language, similar to PowerShell. The design of the Python language is focused on readability and does not use curly brackets or require closing semi-colons. Python is highly extensible and comes with a large standard library that covers many application areas. Additional third-party libraries can also be downloaded. Python is supported by the Python Software Foundation. [17]

There are at least two Python libraries for interfacing with SharePoint. One called simply “sharepoint” and another one called SharePlum. The first one has not been updated since 2015 so does not seem like a reliable choice. SharePlum however is still being maintained and could be a viable alternative to PnP PowerShell. SharePlum allows for interaction with lists and libraries but it is unclear whether it has all of the features that PnP PowerShell does. [18] [19]

I decided to stick with the PnP PowerShell module and using PowerShell as an automation solution since it was a technology that was already being used within my team at Wärtsilä and since Microsoft itself is involved with the development of that particular module I expected more compatibility. I did not have any considerable previous experience with either of the two scripting languages so that didn’t influence my decision in any way.

2.3.2 SharePoint REST API

SharePoint resources can be accessed through a number of ways, including a REST API that is accessed through HTML endpoints. This means a greater freedom of how you implement your solution as you are free to choose any programming language that can make HTML requests. The downside is that this requires additional development. [6]

2.3.3 Browser automation with Selenium

Selenium is a tool for automating web browser interaction. It is mainly targeted towards testing web applications but it can also be used for automating web-based work. With Selenium you can write browser interaction seqeuences in code and run it against most modern web browsers. Selenium support several popular programming languages including C#, Java and Python. Selenium also has an IDE that allows for recording of browser interaction sequences in a dedicated language called Selenese. [20]

Selenium is a tool that I ultimately did not end up using because my goal was to first use any available forms of direct interactions through code and APIs. Browser automation can still be a solution for any web-based work that isn't exposed through an API. However, issues could surface if the web-based interface that you have written code for behaves unexpectedly.

3 Development

3.1 Starting point

Initially a step-by-step guide to manually deploy a new DCM365 site was followed. Doing the process manually served as a way of familiarization with the product. Becoming more accustomed with the process some already available scripts could be used. These scripts would update user groups, user permissions and navigation links. This would already remove a large amount of manual work but there was still a possibility for additional automation of the process.

3.2 Research and implementation

3.2.1 Connecting to SharePoint

The first step in running any commands towards SharePoint is to connect to SharePoint. When using the PnP PowerShell module the cmdlet used for connection is Connect-PnPOnline. This cmdlet requires at least the URL parameter to be specified.

```
Connect-PnPOnline -URL "https://example.sharepoint.com"
```
Code example 7: Connecting to SharePoint

If only the URL is specified PowerShell will attempt to connect by either finding credentials in the Windows Credential Manager or prompt for username and password. Authentication credentials can be directly specified with additional parameters. The UseWebLogin parameter needed to be used because Wärtsilä requires multi factor authentication and the other methods won't work in that case. [21]

3.2.2 Identifying steps that can be automated

Several tasks still had to be done manually. These included the following:

- Uploading and applying templates
- Creating subsites
- Retrieving and setting project info
- Setting navigation audience
- Uploading and removing files

The next step was to research the capabilities of PowerShell and the PnP PowerShell module. If there was a corresponding cmdlet for a given action it would mean the step can be done in a script instead of manually.

3.2.3 Uploading and applying templates

The DCM365 templates are developed and stored in a dedicated site where they can be downloaded from. For a new site the templates need to be uploaded to the site before they could be applied. The template files use the file extension ".wsp".

First it was tried to upload these template files with a cmdlet for uploading files to a SharePoint site, Add-PnPFile. This would upload the files but applying them would not work properly. The assumption is that this is because the templates require some additional step of verification on the SharePoint side that isn't handled by the Add-PnPFile cmdlet.

Another cmdlet called Install-PnPSolution was found. This cmdlet is specifically made for uploading .wsp template files but it requires specifying the PackageId property value of the solution you want to upload and that property value is different for every unique template file. The following example shows a function for uploading all templates at once using the Install-PnPSolution cmdlet.

```
function UploadTemplates{
    Install-PnPSolution -PackageId A13E68CC-AA27-4E90-8F32-1D39FBFCB4BD -
SourceFilePath C:\templates\template1.wsp
    Install-PnPSolution -PackageId 3A172862-1E43-4322-8C67-20CF10B225AE -
SourceFilePath C:\templates\template2.wsp
    Install-PnPSolution -PackageId E2FC0E72-B952-43C4-AFAC-8B41E31O6578 -
SourceFilePath C:\templates\template3.wsp
    Install-PnPSolution -PackageId 52751D37-AD10-4E67-B470-0BF6F2F26170 -
SourceFilePath C:\templates\template4.wsp
}
```

Code example 8: UploadTemplates function

3.2.4 Creating subsites

For the subsite creation there were already parts from another internally used script that could be used, with the most important parts being the New-PnPWeb cmdlet and a way of retrieving available templates on the current site. Two functions needed to be written for this part of the script. First a generic function for creating a subsite and then a specific function for creating the subsites with the right information. These functions are called CreateSubsite and CreateStandardSubsites.

The New-PnPWeb cmdlet requires at least three parameters to be specified. “Title”, “Url” and “Template” [22]. The CreateStandardSubsites will handle the values of these parameters and feed them to the CreateSubsite function.

```
function CreateSubsite ($Title, $Url, $Template){
    New-PnPWeb -Title $Title -Url $Url -Locale 1033 -Template $Template -
InheritNavigation:$false
}
```

Code example 9: CreateSubsite function

The simplest form of the CreateSubsite function accepts title, URL and template as input parameters and createst a new subsite with them. Locale and InheritNavigation are also specified according to DCM365 specifications.

The first thing that the CreateStandardSubsites function needs to do is to retrieve available templates from the current site. This can be achieved with the following code.

```
#Get the Context & Web Objects
$ClientContext = Get-PnPContext
$Web = Get-PnPWeb

#Get All Web Templates
$WebTemplateCollection = $Web.GetAvailableWebTemplates(1033,0)
$ClientContext.Load($WebTemplateCollection)
$ClientContext.ExecuteQuery()
```

Code example 10: Getting available templates

Available templates are then stored in the $WebTemplateCollection. Specific templates can then be selected by piping $WebTemplateCollection into a Where-Object (alias “where”). First I get the collection of custom templates with the following line of code. The “where” command is applied to the object that has been passed through the pipeline, specified with “$_”. The “-like” parameter specifies that the name should include a “{“ anywhere in the name. The asterisk functions as a wildcard character. It is written this way because all the custom templates happen to include a “{“ in the name. [23]

```
$CustomTemplates = $WebTemplateCollection | Where {$_.Name -like "*{*" }
```

Code example 11: Getting custom templates

The $CustomTemplates variable can be used for printing a list of templates if needed in the following way.

```
Write-Host "Available custom templates: "
    foreach ($template in $CustomTemplates){
        Write-Host $template.Name
    }
```

Code example 12: Printing available templates

For the purpose of the CreateStandardSubsites function additional operations were performed to select the right template.

```
$wesTemplate = $CustomTemplates | where { $_.Title -like "WES*" }
```
Code example 13: Selecting "Wärtsilä" template

This line retrieves the template for the "Wärtsilä" subsite. The title for this template always starts with "WES". I made a similar command for the "DCM Settings" subsite.

```
$setTemplate = $CustomTemplates | where { $_.Title -like "SET*" }
```
Code example 14: Selecting "DCM Settings" template

These templates could then be used as input for the CreateSubsite function which would look like the following example. PowerShell parameters can be used without specifying the command name if there is an expected order of parameters. The parameters in this case are "Title", "Url" and "Template".

```
CreateSubsite "Wärtsilä" "WES" $wesTemplate.Name
CreateSubsite "DCM Settings" "SET" $setTemplate.Name
```
Code example 15: Creating "Wärtsilä" and "DCM Settings" subsites

So far this solution can easily create two subsites but there was one additional subsite that needed to be created with additional project-specific requirements. The "site" subsite should include both the project ID and the project name for the site. This led to an additional challenge.

3.2.5 Retrieving project information

The project ID and names are stored in a dedicated SharePoint list on the Wärtsilä SharePoint network. Using the knowledge of how to work towards one SharePoint site meant retrieving the project info from another site wasn't too much of a problem. To get data from an entry in a SharePoint list the Get-PnPListItem cmdlet could be used. The "where" command was used again to find the right list entry. The following example retrieves the list item where the "Title" value equals "project ID", the "project ID" parameter value is unique per project. It then stores the output object in the $item variable.

```
$item=Get-PnPListItem -List "Project Information List" | where
{$_.FieldValues.Title -eq "project ID"}
```
Code example 16: Getting project ID

Additional information about the project could then be retrieved from the properties of the $item variable. The two desired values were project name and country. The project name was found in $item.FieldValues.Projectname and the country in $item.FieldValues.CountryOfRegistration. The project name and country values could then be used in the CreateStandardSubsites function for creating the “site” subsite. There were however additional places in the DCM365 sites where the project info needed to be updated so some structure for storing the project info needed to be made so that it could be reused.

3.2.6 Project class

The project information could be stored in a class. A PowerShell class can hold values in properties and perform actions with methods. A class called Project was created. The class included properties for project ID, name and country. It also included methods for getting project info and printing project info, called GetInfo and PrintInfo.

Every project requires an ID so the constructor for the Project class requires one to be defined. This is assured with the [ValidateNullOrEmpty()] tag on the $ID property and by requiring an ID to be provided in the constructor. A small method for rewriting the project ID was also created. The following example shows the properties of the class, the constructor and the SlashID method.

```
[ValidateNotNullOrEmpty()][string]$ID
[string]$Name
[string]$Country
[string]SlashID(){ return $this.ID -replace '_', '/' }

Project([string]$id) #required parameter
{ $this.ID = $id
}
```

Code example 17: Project class properties

The GetInfo method was then created. This method would perform the retrieval of project info mentioned before. An additional feature was made for this method. The country name was stored as a two letter code but the full country name should be displayed on the DCM365 site. The codes being used follow the ISO 3166-1 alpha-2 standard of country codes. Translation of the codes to full country names was then done with a list of the codes and corresponding country names.

A fitting list of country codes was found on the datahub.io website [24]. These values were then stored in a .csv file that was imported to PowerShell with the Import-Csv command. The GetInfo method would then compare the code for the current project and match it with the corresponding country name for the list. If no country code was available it would set the value to “N/A”.

```
[void]GetInfo(){
    Connect-PnPOnline https://example.sharepoint.com/ -UseWebLogin
    Write-Host "`nGetting project info for $($this.ID)..."
    $item=Get-PnPListItem -List “Project Information List” | where
{$_.FieldValues.Title -eq $this.SlashID()}
    $this.Name=$item.FieldValues.ProjectName
    if($item.FieldValues.CountryOfRegistration -eq $null){
        $this.Country="N/A"
    }
    else{
        $codes = Import-Csv -Path "$PSScriptRoot\alpha2codes.csv" -Delimiter ";"
-Encoding Default
            $code = $codes | where { $_.Code -eq
$item.FieldValues.CountryOfRegistration }
            $this.Country = $code.Name
    }
}
```

Code example 18: GetInfo method

The PrintInfo method would use the information obtained by the GetInfo method and print a formatted message with the information.

```
[void]PrintInfo(){
    Write-Host "Project info`n------------"
    Write-Host "ID: $($this.SlashID())`nName: $($this.Name)`nCountry:
$($this.Country)"
}
```

Code example 19: PrintInfo method

3.2.7 Setting navigation audience

Navigation audience is a setting for what users and user groups can see navigation links in the top bar on a SharePoint site. This setting can only be accessed through the web interface. No PowerShell commands for changing these values were found. This could theoretically still be automated with browser automation using Selenium but was not attempted in this thesis.

3.2.8 Uploading and removing files

When testing the DCM365 sites some test files needed to be uploaded. Update work also sometimes include uploading files to sites. The relevant cmdlets for this action are Add-PnPFile and Add-PnPListItem. Add-PnPFile uploads a file to a specified folder on a SharePoint site. Add-PnPListItem adds an entry to a specified SharePoint list. The file

information needs to be specified in the parameters of the command. The Remove-PnPListItem can be used to remove list entries.

These cmdlets were used to create a function called StandardTestingSetup that uploads test files and adds an email address for testing purposes. First the files are uploaded with Add-PnPFile and then the file information on SharePoint is updated with Add-PnPListItem.

3.2.9 Updating project information

When setting up a new DCM365 site from a template the project specific information still needs to be updated. This includes changing the title for a subsite and updating some list entries. A function called SetProjectInfo was created. This function takes the project information from an object created with the Project class and performs updates wherever necessary.

```
Set-PnPWeb -Title "$($project.SlashID()) - $($project.Name)"
```
Code example 20: Setting site title

The title of a site is updated with the Title parameter of the Set-PnPWeb cmdlet. The values for the title are retrieved from a Project variable called $project. The SlashID() method returns a specific format of the project ID.

```
Set-PnPListItem -List "Project info" -Identity 1 -Values
@{Title=$project.SlashID(); Project_x0020_name=$project.Name;
Country=$project.Country}
```
Code example 21: Updating list values

Additional updates are done with the Set-PnPListItem cmdlet. The cmdlet requires a list to be specified and an identity for the entry to be modified.

3.3 Looping structure

After a functional script for deploying one site was created the next step was to create functionality for running the same operations on many sites. This would require a looping structure for the script and a list of target sites.

3.3.1 Reading list of target sites

First a list of target sites needed to be made. This was done in two ways. One way was to create a function called ReadProjectList that creates a list based on user input. The function creates a new ArrayList object and inserts user input into the ArrayList object.

```
function ReadProjectList {
    $projIdList=New-Object System.Collections.ArrayList
    $projIdList=@()
    $projId=Read-Host "Enter first project id"
    do{
        $projIdList+=$projId
        $projId=Read-Host "Enter next project id (enter to continue)"
    }while (!($projId -eq ""))
    return $projIdList
}
```

Code example 22: ReadProjectList function

User input is retrieved with the Read-Host cmdlet. The function uses a do-while loop to continually ask for project IDs until an empty answer is provided. Finally the function returns an ArrayList object.

The other way of creating a list of target sites was to assemble a .csv file. Either as an exported list or from manual input. The .csv file simply includes a list of project IDs.

```
$sites = Import-Csv -Path "$PSScriptRoot\testsites.csv" -Header "Url"
$siteCount = $sites.Count
Write-Host "$siteCount sites loaded."
```

Code example 23: Importing .csv file

This example imports a .csv file called “testsites.csv” with the Import-Csv cmdlet. A header needs to be specified so that it can be access as a property of the $sites object created from the import. It will also print a message stating how many sites were imported using the Count property.

3.3.2 Foreach loop

With a list of sites assembled either through user input or an imported file a looping function could be created. The foreach loop can then simply be wrapped around the rest of the script.

```
foreach ($site in $sites) {
    #code to loop
}
```
Code example 24: foreach loop

3.4 Update and maintenance tasks

The same ideas could also be used for update and maintenance tasks. Often these tasks involved uploading or replacing a file. Sometimes a different action had to be done on different subsites so additional structure could be added for that purpose. Asynchronous operations were also tested.

3.4.1 Performing different actions on different subsites

Information about the subsites of the SharePoint site that you are currently connected to could be retrieved with the Get-PnPSubWebs cmdlet. The result is stored in a variable called $SubWebs that can then be looped through in a foreach loop.

```
$SubWebs = Get-PnPSubWebs

foreach ($SubWeb in $SubWebs){
    Write-Host "Starting job at $($SubWeb.ServerRelativeUrl)"

    Start-Job -ScriptBlock $onSubsite -ArgumentList @($SubWeb) | Out-Null
    #Get-Job | Wait-Job | Out-Null #uncomment to run synchronously
}
```
Code example 25: Looping through subsites

The previous example is written to be compatible with asynchronous operations. The operations to be performed are defined in the $onSubsite script block. It includes the currently handled subweb as a parameter $SubWeb. The $onSubsite script block connects to a subsite and then checks the title of the subsite and performs an action depending on which subsite it is.

```
$onSubsite = {
    param($SubWeb)
    Connect-PnPOnline
"https://example.sharepoint.com$($SubWeb.ServerRelativeUrl)"
    if($SubWeb.Title -eq "DCM Settings" -or $SubWeb.Title -like "*SITE"){
        Write-Host "SET or SITE-specific job completed at
$($SubWeb.ServerRelativeUrl)"
        continue
    }
    elseif($SubWeb.Title -eq "Wärtsilä"){
        Write-Host "Wärtsilä-specific job completed at
$($SubWeb.ServerRelativeUrl)"
        continue
    }
    else{
        Write-Host "Non-specific job completed at $($SubWeb.ServerRelativeUrl)"
    }
}
```

Code example 26: $onSubsite script block

3.5 Measuring script speed

As the goal of this project has been to reduce inefficencies and to save time it would naturally benefit from a way to measure how long it takes to run a script. A few methods of measuring script speed were found in an article by Adam Bertram. The Measure-Command cmdlet, using the .NET Stopwatch class or using datetime objects. [25]

3.5.1 Measure-Command

The Measure-Command cmdlet works as a wrapper around a specified command that outputs timing data after the execution of the command.

```
PS C:\Users\User> Measure-Command { 0..1000 | ForEach-Object {$i++} }


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 17
Ticks             : 174555
TotalDays         : 2,0203125E-07
TotalHours        : 4,84875E-06
TotalMinutes      : 0,000290925
TotalSeconds      : 0,0174555
TotalMilliseconds : 17,4555
```

Code example 27: Measuring ForEach-Object

The example shows the Measure-Command cmdlet wrapped around a simple for-loop. The output shows a measured execution time of 17 milliseconds.

```
PS C:\Users\User> Measure-Command { foreach ($j in 0..1000) {$i++} }


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 3
Ticks             : 39733
TotalDays         : 4,59872685185185E-08
TotalHours        : 1,10369444444444E-06
TotalMinutes      : 6,62216666666667E-05
TotalSeconds      : 0,0039733
TotalMilliseconds : 3,9733
```

Code example 28: Measuring foreach

This example shows the Measure-Command cmdlet wrapped around a different implementation of the same for-loop. By comparing the different output results we can see that the “foreach” implementation is 14 milliseconds faster. The reason for this was found to be related to ForEach-Object needing items to be sent through the pipeline [26]. This means that in this case “foreach” is a better solution. ForEach-Object includes additional features such as possiblity of further piping of the results which could make it a better choice in other situations.

3.5.2 .NET Stopwatch class

The .NET Stopwatch class works as you would expect a physical stopwatch to work. You can use the .NET Stopwatch class by importing it with the following syntax.

```
PS C:\Users\User> $StopWatch = [system.diagnostics.stopwatch]::StartNew()
```

Code example 29: Importing .NET Stopwatch class

This example creates a new stopwatch object, stores it in the $StopWatch variable and calls the StartNew() method to start the stopwatch. After the stopwatch has been started timing information can be retrieved through the Elapsed property of the stopwatch object.

```
PS C:\Users\User> $StopWatch.Elapsed


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 10
Milliseconds      : 106
Ticks             : 101060812
TotalDays         : 0,000116968532407407
TotalHours        : 0,00280724477777778
TotalMinutes      : 0,168434686666667
TotalSeconds      : 10,1060812
TotalMilliseconds : 10106,0812
```

Code example 30: Measuring with .NET Stopwatch

To measure execution time of a script a stopwatch could be started at the start of the script, stopped at the end and the total time printed from the Elapsed property.

3.5.3 Datetime objects

Execution time can also be measured by defining two datetime objects, one at the start of the script and one at the end, and then calculating the difference between them. A datetime object can be created with the Get-Date cmdlet.

```
PS C:\Users\User> $start = Get-Date

PS C:\Users\User> $end = Get-Date

PS C:\Users\User> $end - $start


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 4
Milliseconds      : 156
Ticks             : 41562810
TotalDays         : 4,81051041666667E-05
TotalHours        : 0,0011545225
TotalMinutes      : 0,06927135
TotalSeconds      : 4,156281
TotalMilliseconds : 4156,281
```

Code example 31: Measuring with datetime objects

3.5.4 Completed code segment

All three methods seemed viable to me and I did not see any obvious reasons to use one over the other. Performance differences between the different methods should be small enough to not really matter for my use case. I did settle on using datetime objects but the other methods should functionally be the same.

```
$sOuter=Get-Date
# script goes here
$eOuter=Get-Date
$completionTime=$eOuter-$sOuter
if ($completionTime.TotalMinutes -gt 2) { Write-Host "`nScript completed in
$(($completionTime.TotalMinutes).ToString("#.##")) minutes." }
else { Write-Host "`nScript completed in
$(($completionTime.TotalSeconds).ToString("#.##")) seconds." }
```

Code example 32: Script measuring structure

Some formatting and output was added to the finished code. If the execution time is greater than two minutes the result will be printed in minutes, otherwise it will be printed in seconds. This is done with an “if” statement that checks if the TotalMinutes property of the $completionTime object is greater than 2.

3.6 Error handling

Some basic error handling was added to the scripts and the GetInfo method using try-catch-finally blocks. The code to be executed is placed in the “try” block, the actions to be performed in case of an error are placed in the “catch” block and actions that will run in both cases are placed in the “finally” block. In this case the “finally” block is used for the Disconnect-PnPOnline cmdlet so that the script won’t leave a connection open.

```
try{
    Connect-PnPOnline "https://example.sharepoint.com/" -UseWebLogin
}
catch{
    Write-Host "`nAn error occured: "
    Write-Host "$_" -f Red
    pause
}
finally{
    Disconnect-PnPOnline
}
```

Code example 33: Try-catch-finally block

This example will attempt to connect to a SharePoint site. If the connection fails it will print the error message. When the script is finished it will disconnect any open connections.

3.7 Completed scripts

3.7.1 SiteDeployment.ps1

The purpose of this script is to allow as much as possible of the DCM365 site deployment process to be executed in a single script. It will read a list of project IDs and then loop through each corresponding DCM365 site. The script can be run in two or three separate executions to make sure some steps are completed correctly before continuing with the next. This is done by commenting out the functions you don’t want to run.

A number of functions related to the deployment process are included. These functions perform actions specific to DCM365 site deployment such as uploading templates and updating project information. The script calls some functions from other scripts that were already being used. Those are the scripts for permission and navigation settings and for cleaning up test files. An edited version of the finished script is found in the appendices section.

3.7.2 MaintenanceBase.ps1

The purpose of this script is to allow a given operation to run on many different SharePoint sites and subsites while adhering to special conditions. This script will also read a list of project IDs and loop through each corresponding DCM365 site. The implementation of this script is slightly different than the SiteDeployment script, it will read the project IDs from a .csv file instead of a function and it includes additional structure to enable different actions on different subsites of each DCM365 site. The finished script is found in the appendices section.

4 Results and conclusion

4.1 PowerShell scripts

The result of this thesis is two PowerShell scripts that are aimed at automating repetitive tasks. Both scripts feature a looping structure at their cores. The scripts are also customizable for other purposes. Using these scripts instead of doing the work manually will save a considerable amount of time. The total time spent doing the manual work can vary from person to person and from day to day but the scripts can potentially reduce a few hours of manual work to a few minutes.

4.2 Automating routine tasks

A general idea about automating routine tasks was also formed and the principles followed in this thesis can also be applied with regards to other technologies than SharePoint. Some key points of consideration were found:

- Identifying steps that can be automated
- Utilizing APIs and libraries
- Using looping structures

4.3 Future development

4.3.1 Further asynchronous testing

The basic structure for asynchronous operations was created in the MaintenanceBase script but was not thoroughly tested. The main limitation to using this is the connection to SharePoint, how reliable the connection is and how many open connections are allowed. Scripts could run many times more quickly if actions could be run asynchronously.

4.3.2 Improving exception handling

Running custom scripts can potentially be dangerous if the user isn't aware of what the script is actually doing. In my particular case it could also lead to disrupted work for others if a faulty change is made on many sites. Additionally unhandled errors in the middle of the script can lead to unwanted behavior and sometimes create a messy situation that is difficult to clean up. These issues can be improved upon with a better considered exception handling structure.

4.3.3 Improving user experience

Personally I like to execute my scripts in as few steps as possible but to help others use the scripts a menu structure and user prompts can considerably improve the user experience and will also improve safety of the scripts if it helps the user understand what the script is doing.

4.3.4 Logging features

When using a script to make a large amount of changes it can be hard to keep track off what exactly has been changed and when. Implementing some kind of logging structure would serve as an additional safety measure in case any changes need to be reversed. A logging feature could for example store information about what commands have been executed on what sites at what time.

4.3.5 Dedicated application

A dedicated application with a graphical interface for updating SharePoint sites could be developed. This would improve the user experience but additional investigations would need to be done to determine if the additional time required to develop and maintain such an application is worth it.

5 References

[1] Wärtsilä, "About Wärtsilä," [Online]. Available: https://www.wartsila.com/about. [Accessed 5 October 2020].

[2] Microsoft, "What is SharePoint?," [Online]. Available: https://support.microsoft.com/en-us/office/what-is-sharepoint-97b915e6-651b-43b2-827d-fb25777f446f. [Accessed 8 November 2020].

[3] Microsoft, "Planning your SharePoint hub sites," 5 June 2020. [Online]. Available: https://docs.microsoft.com/en-us/sharepoint/planning-hub-sites. [Accessed 30 November 2020].

[4] G. Zelfond, "Sites vs Site Collections in SharePoint," 12 July 2015. [Online]. Available: https://sharepointmaven.com/sites-vs-site-collections-in-sharepoint/. [Accessed 30 November 2020].

[5] Microsoft, "Introducing SharePoint Designer," [Online]. Available: https://support.microsoft.com/en-us/office/introducing-sharepoint-designer-66bf58fe-daeb-4fa6-ae84-fd600e0005c1. [Accessed 30 November 2020].

[6] Microsoft, "Get to know the SharePoint REST service," 15 January 2020. [Online]. Available: https://docs.microsoft.com/en-us/sharepoint/dev/sp-add-ins/get-to-know-the-sharepoint-rest-service. [Accessed 26 November 2020].

[7] Microsoft, "What is PowerShell?," [Online]. Available: https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7. [Accessed 12 October 2020].

[8] J. Aiello, "PowerShell Core 6.0: Generally Available (GA) and Supported!," 10 January 2018. [Online]. Available: https://devblogs.microsoft.com/powershell/powershell-core-6-0-generally-available-ga-and-supported/. [Accessed 20 November 2020].

[9] J. Vigo, "How to choose between PowerShell Core and PowerShell," 4 September 2019. [Online]. Available: https://www.techrepublic.com/article/a-tale-of-two-powershells-which-is-the-right-version-for-you/. [Accessed 20 November 2020].

[10] Microsoft, "Cmdlet Overview," 6 November 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/cmdlet-overview?view=powershell-7.1. [Accessed 22 November 2020].

[11] M. Taylor, "How to create a PowerShell alias," 29 July 2014. [Online]. Available: https://4sysops.com/archives/how-to-create-a-powershell-alias/. [Accessed 23 November 2020].

[12] B. Kindle, "Back to Basics: Understanding PowerShell Objects," 27 November 2019. [Online]. Available: https://adamtheautomator.com/powershell-objects/. [Accessed 22 November 2020].

[13] J. Petters, "Windows PowerShell Scripting Tutorial For Beginners," 14 May 2020. [Online]. Available: https://www.varonis.com/blog/windows-powershell-tutorials/. [Accessed 25 November 2020].

[14] Microsoft, "About Modules," 15 September 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_modules?view=powershell-7.1. [Accessed 28 November 2020].

[15] Microsoft, "PnP PowerShell overview," 28 November 2017. [Online]. Available: https://docs.microsoft.com/en-us/powershell/sharepoint/sharepoint-pnp/sharepoint-pnp-cmdlets?view=sharepoint-ps. [Accessed 28 November 2020].

[16] Microsoft, "The Windows PowerShell ISE," 8 August 2018. [Online]. Available: https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.1. [Accessed 25 November 2020].

[17] Python Software Foundation, "General Python FAQ," 26 November 2020. [Online]. Available: https://docs.python.org/3/faq/general.html. [Accessed 26 November 2020].

[18] IT Services, University of Oxford, "sharepoint 0.4.2," 5 August 2015. [Online]. Available: https://pypi.org/project/sharepoint/. [Accessed 28 November 2020].

[19] J. Rollins, "SharePlum 0.5.1," 22 April 2020. [Online]. Available: https://pypi.org/project/SharePlum/. [Accessed 28 November 2020].

[20] Selenium community, "The Selenium project and tools," 29 November 2020. [Online]. Available: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/. [Accessed 30 November 2020].

[21] Microsoft, "Connect-PnPOnline," 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/module/sharepoint-pnp/connect-pnponline?view=sharepoint-ps. [Accessed 30 November 2020].

[22] Microsoft, "New-PnPWeb," 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/module/sharepoint-pnp/new-pnpweb?view=sharepoint-ps. [Accessed 2 December 2020].

[23] Microsoft, "Where-Object," 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object?view=powershell-7.1. [Accessed 2 December 2020].

[24] Data Hub, "List of all countries with their 2 digit codes (ISO 3166-1)," 2018. [Online]. Available: https://datahub.io/core/country-list. [Accessed 3 December 2020].

[25] A. Bertram, "3 ways to measure your Powershell script's speed," 1 May 2011. [Online]. Available: https://www.pluralsight.com/blog/tutorials/measure-powershell-scripts-speed. [Accessed 12 November 2020].

[26] D. Scripto, "Getting to Know ForEach and ForEach-Object," 8th July 2014. [Online]. Available: https://devblogs.microsoft.com/scripting/getting-to-know-foreach-and-foreach-object/. [Accessed 9 December 2020].

Appendices

Appendix 1: SiteDeployment.ps1

```
class Project{
    [ValidateNotNullOrEmpty()][string]$ID
    [string]$Name
    [string]$Country
    [string]SlashID(){ return $this.ID -replace '_', '/' }

    Project(
        [string]$id #required parameter
    ){ $this.ID = $id
    }

    [void]GetInfo(){
        try{
            Connect-PnPOnline https://example.sharepoint.com/ -UseWebLogin
            Write-Host "`nGetting project info for $($this.ID)..."
            $item=Get-PnPListItem -List “Project Information List” | where
{$_.FieldValues.Title -eq $this.SlashID()}
            $this.Name=$item.FieldValues.ProjectName
            if($item.FieldValues.CountryOfRegistration -eq $null){
                $this.Country="N/A"
            }
            else{
                $codes = Import-Csv -Path "$PSScriptRoot\alpha2codes.csv" -
Delimiter ";" -Encoding Default
                $code = $codes | where { $_.Code -eq
$item.FieldValues.CountryOfRegistration }
                $this.Country = $code.Name
            }
        }
        catch{
            Write-Host "`nError getting project info in Project constructor"
            Write-Host "$_" -f Red
            pause
        }
        finally{
            Disconnect-PnPOnline
        }
    }
    [void]PrintInfo(){
        Write-Host "Project info`n------------"
        Write-Host "ID: $($this.SlashID())`nName: $($this.Name)`nCountry:
$($this.Country)"
    }
}

function ReadProjectList {
    $projIdList=New-Object System.Collections.ArrayList
    $projIdList=@()
    $projId=Read-Host "Enter first project id"
    do{
        $projIdList+=$projId
        $projId=Read-Host "Enter next project id (enter to continue)"
    }while (!($projId -eq ""))
    return $projIdList
}

function UploadTemplates{
    Install-PnPSolution -PackageId A13E68CC-AA27-4E90-8F32-1D39FBFCB4BD -
SourceFilePath C:\templates\Customer.wsp
    Install-PnPSolution -PackageId 3A172862-1E43-4322-8C67-20CF10B225AE -
SourceFilePath C:\templates\SET.wsp
    Install-PnPSolution -PackageId E2FC0E72-B952-43C4-AFAC-8B41E31O6578 -
SourceFilePath C:\templates\Site.wsp
    Install-PnPSolution -PackageId 52751D37-AD10-4E67-B470-0BF6F2F26170 -
SourceFilePath C:\templates\WES.wsp
}
function PrintTemplates(){
    #Get the Context & Web Objects
    $ClientContext = Get-PnPContext
    $Web = Get-PnPWeb

    #Get All Web Templates
```

```
    $WebTemplateCollection = $Web.GetAvailableWebTemplates(1033,0)
    $ClientContext.Load($WebTemplateCollection)
    $ClientContext.ExecuteQuery()

    #Get the Template Name and Title
    $CustomTemplates = $WebTemplateCollection | Where {$_.Name -like "*{*" }

    Write-Host "Available custom templates: "
    foreach ($template in $CustomTemplates){
        Write-Host $template.Name
    }
}
function CreateStandardSubsites(){
    #Get the Context & Web Objects
    $ClientContext = Get-PnPContext
    $Web = Get-PnPWeb

    #Get All Web Templates
    $WebTemplateCollection = $Web.GetAvailableWebTemplates(1033,0)
    $ClientContext.Load($WebTemplateCollection)
    $ClientContext.ExecuteQuery()

    #Get the Template Name and Title
    $CustomTemplates = $WebTemplateCollection | Where {$_.Name -like "*{*" }

    $siteTemplate = $CustomTemplates | where { $_.Title -like "Site*" }
    CreateSubsite "$($project.SlashID()) - $($project.Name) - SITE" "site"
$siteTemplate.Name

    $setTemplate = $CustomTemplates | where { $_.Title -like "SET*" }
    CreateSubsite "DCM Settings" "SET" $setTemplate.Name

    $wesTemplate = $CustomTemplates | where { $_.Title -like "WES*" }
    CreateSubsite "Wärtsilä" "WES" $wesTemplate.Name
}
function CreateSubsite ($Title, $Url, $Template){

    Write-Host "`nSubsite details:" -f Yellow
    Write-Host "`nTitle:" -f Green -NoNewLine; Write-Host "`t`t$Title" -f Cyan
    Write-Host "Url:" -f Green -NoNewLine; Write-Host "`t`t$Url" -f Cyan
    Write-Host  "Template:" -f Green -NoNewLine; Write-Host "`t$Template" -f Cyan
    #pause
    Write-Host "Creating subsite..." -f Green
    New-PnPWeb -Title $Title -Url $Url -Locale 1033 -Template $Template -
InheritNavigation:$false
    Write-Host "Subsite created" -f Green
}
function SetProjectInfo(){
    Connect-PnPOnline https://example.sharepoint.com/sites/$projId -UseWebLogin

    Set-PnPWeb -Title "$($project.SlashID()) - $($project.Name)"
    Write-Host "Title changed to '$($project.SlashID()) - $($project.Name)'"
    Set-PnPListItem -List "Project info" -Identity 1 -Values
@{Title=$project.SlashID(); Project_x0020_name=$project.Name;
Country=$project.Country}
    Write-Host "Project info list updated. Country: $($project.Country)"

    Connect-PnPOnline https://example.sharepoint.com/sites/$projId/WES/ -
UseWebLogin
    Set-PnPListItem -List "PartnerInfo" -Identity 1 -Values
@{ProjectNo=$project.SlashID()}
    Write-Host "Partner info list updated"

    Connect-PnPOnline https://example.sharepoint.com/sites/$projId/SET/ -
UseWebLogin
    Set-PnPListItem -List "SiteStructure" -Identity 2 -Values
@{UrlName="https://example.sharepoint.com/sites/$projId/"}
    Set-PnPListItem -List "SiteStructure" -Identity 3 -Values
@{UrlName="https://example.sharepoint.com/sites/$projId/site/"}
    Remove-PnPListItem -List “SiteStructure” -Identity 4 -Force -Recycle -
ErrorAction Ignore
    Remove-PnPListItem -List "SiteStructure" -Identity 5 -Force -Recycle -
ErrorAction Ignore
    Write-Host "Site structure updated"
}
function StandardTestingSetup(){
    Connect-PnPOnline "https://example.sharepoint.com/sites/$projId/" -
UseWebLogin
```

```
    #uploads test files, updates the values of the test files and adds an email
for testing

    Connect-PnPOnline https://example.sharepoint.com/sites/$projId/SET/ -
UseWebLogin
    Add-PnPListItem -List ContactList -Values @{"Email"=$email;
"CUSProjectmanagement"="to"; "CUSAll"="to"; "STEAll"="to"}
    Write-Host "Email added to contact list"
}
function AddWorkspaceLink(){
    Connect-PnPOnline "https://example.sharepoint.com/" -UseWebLogin
    Add-PnPListItem -List "Workspace links" -Values
@{URL="https://example.sharepoint.com/sites/$projId/, $($project.SlashID()) -
$($project.Name)";ProjectName="$($project.SlashID()) -
$($project.Name)";SiteAddress="https://example.sharepoint.com/sites/$projId/";Sta
mpList="N/A";doczone="2502"}
}


$email="name@example.com"
$projIdList=ReadProjectList
$projCount = $projIdList.Count
Write-Host "$projCount projects loaded."

$sOuter=Get-Date
foreach($projId in $projIdList){
    if ($projCount -eq 1) { Write-Host "$projCount site remaining" } else {
Write-Host "$projCount sites remaining." };$projCount--
    $project=[Project]::new($projId)
    $project.GetInfo()
    $project.PrintInfo()
    try{
        Connect-PnPOnline "https://example.sharepoint.com/sites/$projId" -
UseWebLogin
        Write-Host "`nConnected to https://example.sharepoint.com/sites/$projId"

        First run # check path to templates in UploadTemplates
        Write-Host "`nUploading templates";UploadTemplates
        PrintTemplates

        #Second run # check $email, script paths and path to test files in
StandardTestingSetup before running
        #Write-Host "`nCreating subsites" -f Green;CreateStandardSubsites
        #Write-Host "`nSetting project info" -f Green;SetProjectInfo
        #Write-Host "`nSetting permissions" -f Green;. "$PSScriptRoot\Permission
Scripts\permissions1to5.ps1";SetGroupsAndPermissions($projId)
        #Write-Host "`nSetting up navigation" -f Green;.
"$PSScriptRoot\GlobalNavChoice\GlobalNavStandard.ps1";SetStandardNavigation($proj
Id)
        #Write-Host "`nSetting up standard testing" -f Green;StandardTestingSetup

        #Third run # check script path before running
        #Write-Host "`nCleaning up test files" -f Green;.
"$PSScriptRoot\Cleaner\CleanStandard.ps1";CleanStandard($projId)
        #Write-Host "Setting title to |-DCM365-|";Set-PnPWeb -Description "|-
DCM365-|"
        #Write-Host "Adding workspace link";AddWorkspaceLink
    }
    catch{
        Write-Host "`nAn error occured: "
        Write-Host "$_" -f Red
        pause
    }
    finally{
        Disconnect-PnPOnline
    }
}
$eOuter=Get-Date
$completionTime=$eOuter-$sOuter
if ($completionTime.TotalMinutes -gt 2) { Write-Host "`nScript completed in
$(($completionTime.TotalMinutes).ToString("#.##")) minutes." }
else { Write-Host "`nScript completed in
$(($completionTime.TotalSeconds).ToString("#.##")) seconds." }
```

Appendix 2: MaintenanceBase.ps1

```
$onSubsite = {
    param($SubWeb)
    Connect-PnPOnline
"https://example.sharepoint.com$($SubWeb.ServerRelativeUrl)"
    if($SubWeb.Title -eq "title of first subsite" -or $SubWeb.Title -like "title
of second subsite"){
        Write-Host "first subsite or second subsite-specific job completed at
$($SubWeb.ServerRelativeUrl)"
        continue
    }
    elseif($SubWeb.Title -eq "title of third subsite"){
        Write-Host "third subsite-specific job completed at
$($SubWeb.ServerRelativeUrl)"
        continue
    }
    else{
        Write-Host "Non-specific job completed at $($SubWeb.ServerRelativeUrl)"
    }
}

clear

$sites = Import-Csv -Path "$PSScriptRoot\testsites.csv" -Header "Url"
$siteCount = $sites.Count
Write-Host "$siteCount sites loaded."

$sOuter=Get-Date
foreach ($site in $sites) {
    try{
        Connect-PnPOnline $site.Url
        $SubWebs = Get-PnPSubWebs

        #clear
        if ($siteCount -eq 1) { Write-Host "$siteCount site remaining" } else {
Write-Host "$siteCount sites remaining." }

        Write-Host "Accessing $($site.Url)"
        #pause
        #put tasks to complete on root site here

        $sInner=Get-Date
        foreach ($SubWeb in $SubWebs){
            Write-Host "Starting job at $($SubWeb.ServerRelativeUrl)"
            #put tasks to complete on subsites here

            Start-Job -ScriptBlock $onSubsite -ArgumentList @($SubWeb) | Out-Null
            #Get-Job | Wait-Job | Out-Null #uncomment to run synchronously
        }

        Write-Host "`nWaiting for all jobs to finish..."
        Get-Job | Wait-Job | Receive-Job
        $eInner=Get-Date
        Write-Host "`nJobs completed in $((($eInner-
$sInner).TotalSeconds).ToString("#.##")) seconds."
        $siteCount--
    }
    catch{
        Write-Host "`nAn error occured: "
        Write-Host "$_" -ForegroundColor Red
        pause
    }
    finally{
        Get-Job | Remove-Job
        Disconnect-PnPOnline
    }
}
$eOuter=Get-Date
$completionTime=$eOuter-$sOuter
if ($completionTime.TotalMinutes -gt 2) { Write-Host "`nScript completed in
$(($completionTime.TotalMinutes).ToString("#.##")) minutes." }
else { Write-Host "`nScript completed in
$(($completionTime.TotalSeconds).ToString("#.##")) seconds." }
```