Sakari Ruotsalainen

Microsoft OS and Office patching procedure in an enterprise environment

Bachelor's Thesis
Information Technology

May 2018

Kaakkois-Suomen
ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät Sakari Ruotsalainen	**Tutkinto** Insinööri (AMK)	**Aika** Maaliskuu 2018
Opinnäytetyön nimi Microsoftin käyttöjärjestelmä ja Office päivitysprosessi enterprise ympäristössä.		58 sivua 10 liitesivua
Toimeksiantaja Salainen		
Ohjaaja Martti Kettunen		

Tiivistelmä

Microsoft käyttöjärjestelmien ja Office tuotteiden päivitysten hallinta loppukäyttäjien laitteiden osalta on tärkeä osa modernia kyberturvallisuutta yritysympäristössä. Laitteiden pitäminen ajan tasalla on tärkeää, mutta altistaa niiden järjestelmät ja sovellukset uusien päivitysten aiheuttamille ongelmille. Riskien vähentämiseksi on mahdollista luoda testiympäristöjä, joissa päivitykset testataan ennen niiden asentamista tuotantoon. Testiympäristö on toimiva ratkaisu päivitysten testaamiseen, mutta ei monessa tapauksessa tuota erillistä palautetta loppukäyttäjiltä. Palautteen puute voi aiheuttaa ongelmien pääsyn seulan läpi, joka on tapahtunut useaan otteeseen uusien Microsoft päivitysten kanssa.

Tämä opinnäytetyö keskittyy päivitysprosessin kuvaamiseen Microsoft System Center Configuration Managerin kanssa ja sen päätavoitteena on luoda prosessi, jossa loppukäyttäjä tuodaan mukaan päivitysten testausprosessiin. Ympäristönä on suuri, kansainvälinen tuotantotekniikkaa kehittävä ja valmistava yritys. Prosessin luontiin sisältyy lomakkeiden luominen, joilla loppukäyttäjät voivat liittyä mukaan ja raportoida takaisin sekä applikaatio, jonka avulla järjestelmänylläpitäjät voivat hallinnoida prosessia.

Lopputuloksena saavutettiin toimiva prosessi, jolla loppukäyttäjät voidaan tuoda mukaan päivitysten testausprosessiin. Se mahdollistaa loppukäyttäjien liittymisen, raportoinnin sekä prosessin hallinnoinnin. Prosessi on mahdollista ottaa käyttöön monissa eri ympäristöissä ja auttaa vähentämään päivitysprosessin aiheuttamia ongelmia merkittävästi. Yrityksen osalta suosituksena on prosessin käyttöönottaminen hieman muunneltuna versiona, joka käyttää työkaluja, jotka eivät olleet saatavilla tähän opinnäytetyöhön.

Asiasanat

päivitykset, testaaminen, loppukäyttäjien integraatio, prosessin luominen, Microsoft SCCM, PowerShell, InfoPath

XAMK
South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Sakari Ruotsalainen	Bachelor of Engineering	March 2018

Thesis Title

Microsoft OS and Office patching procedure in an enterprise environment.

58 pages
10 pages of appendices

Commissioned by

Classified

Supervisor

Martti Kettunen

Abstract

Managing Microsoft operating system and Office updates for end user clients is an important part of modern cybersecurity in an enterprise environment. Keeping client systems updated is extremely important, but this exposes them and their applications to issues that the new updates themselves may cause. In order to reduce this risk, testing environments can be implemented, and the updates are tested there before deployment into production. While this system works on its own, it usually does not provide feedback from the end users. This can cause issues to slip through detection as has happened on many occasions with new Microsoft updates.

This thesis is focused on the updates process using Microsoft System Center Configuration Manager, with the main objective being the creation of a process to integrate end users to the testing of new updates. The environment is a large, international industrial engineering and manufacturing company. Ideally, the user base would cover the entire array of applications in use and as such would maximize the probability to expose any issues with the updates before they are deployed onto the actual production environment. The creation of the process for user reporting includes creating a form for users to join the testing group, form for the users to report back information each month new updates are released and constructing an application for administrators to manage the process.

As an outcome of this thesis study, a working process to include end users in the testing of new updates was constructed. It enables the users to join the group and report back any issues, and the administrators to view the data created by the users and to manage the process. The process can be implemented in its current form in most environments and should yield results that significantly mitigate the problems caused by the updating process. The recommendation for the company is to implement an altered version of this process using different tools that were unavailable for this thesis.

Keywords

updates, testing, end user integration, process creation, Microsoft SCCM, PowerShell, InfoPath

CONTENTS

APPENDICES

TERMS AND ABBREVIATIONS

Active Directory – Microsoft directory service for Windows Domain networks.

IIS – Internet Information Services, Microsoft developed web server for Windows operating systems.

InfoPath – Microsoft software used to design forms with structured data.

PowerShell – A task automation and management framework, with a command shell and a scripting language.

Office 365 - A subscription based service for Microsoft Office products.

SCCM – System Center Configuration Manager is a Microsoft product for asset management and configuration of large groups of clients.

SharePoint – Microsoft collaboration platform which integrates with Office products. Mainly used in document management and as storage.

Software Update Point – or SUP, an SCCM site role for software updates.

Wake-On-Lan – A technology that allows a server to “wake” dormant clients using specific packets sent over a network.

WSUS – Windows Server Update Services, a Windows server role that handles the delivery of updates to clients.

WUA – Windows Update Agent, an agent program that works with WSUS to enable automated update delivery to clients.

WUSA – Windows Update Standalone Installer, an update package installer that uses WUA API to install/uninstall operating system update packages.

1 INTRODUCTION

1.1 Research problems and the scope and objectives of the thesis

This thesis concerns Microsoft operating system and Office updates and their effect before deployment to production environments. The end objective is to create a process with which the updates can be appropriately tested before deployment with the help of end user feedback. The process created here is completely new, as only a small group of clients are currently used as a testing environment with no established feedback functionality. The objective is achieved by studying user reporting and inclusion in different tasks and processes, and then applying the needed functionality into the process created in this thesis. Finally, a proposition of how to implement this process in the most efficient way will be made. Because the scope of this thesis is somewhat large, it will be limited to end user devices and not servers.

The thesis is completed as a functional thesis, as I worked on the research problem alone.

The client company is a global industrial engineering and manufacturing company that specializes in pumping solutions and services for rotating equipment. It employs about 15500 people and has a turnover of about €3.1 billion. Any information about company IT infrastructure, computer configurations and systems has been altered for security purposes.

The idea for this thesis was suggested for me in 2017 when I worked at the company as a summer intern and asked for possible topics for my thesis. My instructor in the company with his peers had been concerned about the update deployment process as being insufficient in terms of testing, and this topic was given to me. This thesis is written in English because the client is an international company.

There are other bachelor’s theses that are somewhat related to this, as they were completed for the same company by students from the same university, Application Request and Approval in Enterprise (2016) by Antti Hartikainen and Application Deployment in an Enterprise (2016) by Samuli Laamanen.

These cover different aspects of SCCM and other processes and as such are not referenced in this thesis.

The first part of this thesis is focused on the mechanics regarding how Microsoft updates are delivered to clients in a large enterprise environment, including operating system updates and updates for Office products. It will also cover the underlying infrastructure used in the process and is done for a client company which will be referred to as *the company*.

The second part will examine integrating the end user to the updates process patch testing to gather relevant information on update functionality and possible issues they may cause. Here, the methods for group maintenance and user reporting are created. This part is completed only as a Proof of Concept.

1.2 The importance of updating systems

Patching as a process involves remedying discovered vulnerabilities or faults in a system that has been released on the market. In the scope of this thesis, the systems are Microsoft operating systems and Office products. However, the patching process can be applied to any system supported by System Center Configuration Manager.

Microsoft provides new update content on the first Tuesday of each month which is known as “Patch Tuesday”.

The importance of patch management was highlighted in the summer of 2017 when WannaCry and Petya ransomware attacks were made possible by vulnerabilities in Windows operating systems. The attacks caused enormous problems in cases where patching of critical systems was not up to date. The WannaCry attack alone hit over 300,000 computers worldwide (Chappell, 2017).

This thesis only covers end user clients and their updating, not the server side, although the process is similar in most ways.

1.3 Including the end user in the update testing process

It is extremely important to update systems as soon as possible, after the publisher releases new updates, especially in the case of security related updates. This can, however, lead to updates being deployed to the production environment, which can cause unforeseen issues with the software it is intended for, or with other unconnected applications.

In order to avoid this, a testing environment is required. Here, a pre-selected subsection of the company users would act as the testing environment where users will receive patches early and report back with any issues using the method provided, as will be described later in the thesis. Optimally, the number of devices in the testing group would be 5-10% of all clients. Therefore, if a company has 15000 devices, a thousand should suffice as a base for testing the updates. This was the requested number in the company where there are approximately 15000 devices.

The objective is to have a live number of acceptance from the members of the updates testing group, if users report about problems, and the number goes down, under a pre-determined limit, action can be taken as fast as possible. Reports from the updates testing group should expose any issues with the updates as multiple reports should contain similar problem descriptions. This process also reduces stress on the ticketing system in use as the update feedback exists as a separate process. In many cases, multiple tickets have been created for an issue caused by a recent update, but it takes time for the tickets to find the correct group to solve the problem. With the process created in this thesis, the problems would be immediately visible for the correct department.

2 SYSTEM CENTER CONFIGURATION MANAGER

Because this thesis is focused on client updating, this section will cover SCCM basics and describe the underlying infrastructure that enables the updating process.

System Center Configuration Manager or ConfigMgr is Microsoft's solution for the administration of large number of client devices, including user devices, servers and mobile devices. It is a part of Microsoft's System Center product family. Previously named Systems Management Server (SMS), ConfigMgr is optimized for Windows environments but is very capable of managing for example Mac or Linux devices. The most recent version of SCCM is the System Center Configuration Manager Current Branch (SCCM CB) version 1711.

SCCM can be used to increase productivity by increasing automation of manual tasks and simplifying the delivery of software to end users through software center. SCCM offers a robust asset management of servers, mobile devices and client computers (desktops and laptops). With SCCM, it is easy to enforce client-side configuration and compliance on managed applications. (Microsoft, 2016a)

Other Microsoft products can be integrated to run with ConfigMgr such as Intune (for mobile device management), WSUS, Certificate services, Exchange, DNS, Group Policy and Remote desktop (Microsoft, 2016a).

An SCCM environment is based on sites and their roles. There are variations on how an SCCM environment can be built, depending on the scope and required need for expansion. The first ConfigMgr site installed determines the layout of the SCCM hierarchy. The site installed first must be a central administration site (CAS) or a stand-alone primary site (Microsoft, 2016b). It is possible to install a new CAS as a parent site to an existing stand-alone primary site, so it is possible to support a more complex design. The central administration site is only used for reporting and management. Secondary sites can be installed under primary sites which can exist as stand-alone primary sites or under a CAS as regular primary sites. CAS is always the top-level site in a hierarchy and can only have primary sites as child sites. A primary site can have secondary sites as child sites and can exist as the top-level site when configured as a stand-alone primary site. For example, a secondary site can have multiple roles installed such as a distribution point, management point and software update point, so a client under the secondary site can communicate with a “local” site which then communicates with a parent site (stand-alone primary or a CAS). This can reduce network traffic as clients under the secondary site receive their content from a closer source. The more intricate hierarchy configurations and considerations for different topologies are outside the scope of this thesis.

ConfigMgr manages devices through a client that is installed on all managed devices. The client then executes the tasks instructed to it from the ConfigMgr side, such as software installations and task sequences. Data on clients and their properties is stored in the ConfigMgr site database which is replicated across the environment. The client discovery (and other asset discovery) is achieved through Active Directory system discovery. ConfigMgr is tied to Active Directory which gives it access to all specified AD data.

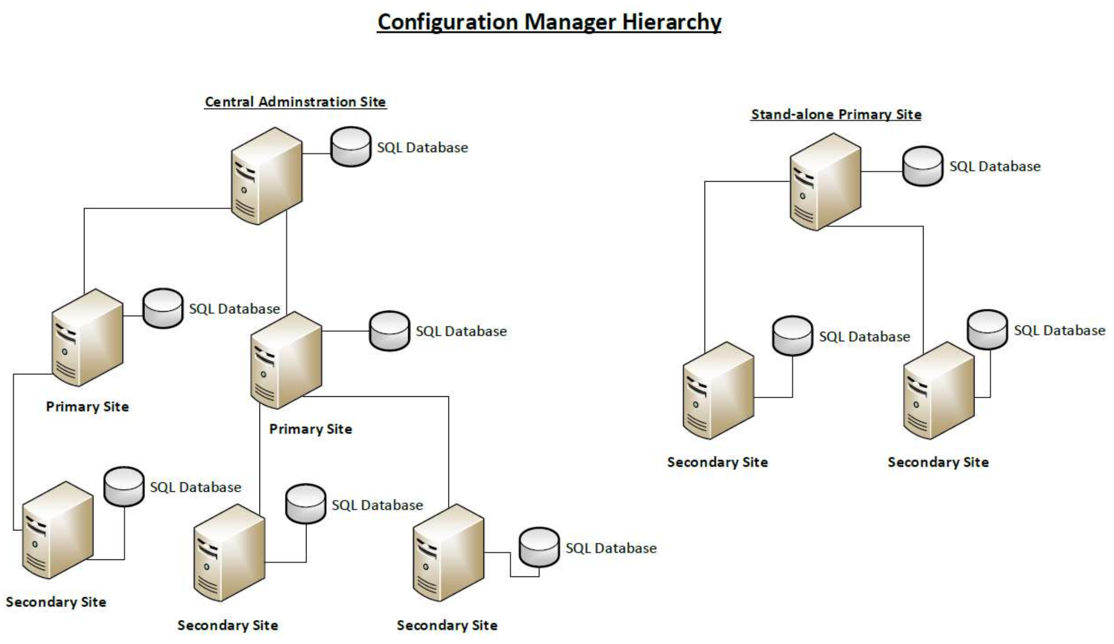


Figure 1: Configuration Manager Hierarchies.

3 OPERATING SYSTEM AND OFFICE UPDATES IN SCCM

3.1 Prerequisites for software updates

There are some prerequisites and requirements for implementing software updates in ConfigMgr. Software updates are enabled for clients by default, but the location of the software update point must be specified for the clients via Group Policy. These specific Group Policy settings are used by Windows Update Agent or WUA on the clients to connect to the WSUS running on the SUP. When the software update point in configured for the site, a machine policy is distributed to the clients that provides them with the software update point server name and configures the “Specify intranet Microsoft update service location” local policy on the clients (Microsoft, 2016c). Also, “Allow signed content from intranet Microsoft update service location” Group Policy setting must be enabled before the WUA on the clients scans for software updates that have been published with System Center Updates Publisher.

Reporting services point site role can display reports on software updates. While this role is optional, it is recommended that it is installed.

3.1.1 Internal SCCM dependencies

The following list comprises the internal dependencies that must exist in an SCCM hierarchy in order for the software update process to function correctly.

- Management points. These transmit information between the ConfigMgr site and the client devices.
- Software update point. This site system role is necessary to deploy updates to clients.
- Distribution points. These store the content of the software updates for the clients to access them more easily.

3.1.2 Software dependencies

The following external software are required for complete functionality of the ConfigMgr update process.

- IIS (Internet Information Services) must be installed. IIS is a Microsoft web server. IIS is also required to run management and distribution points. IIS is required to run software update point, management point and distribution point roles.
- A WSUS (Windows Server Update Services) instance must be present, either a fresh install or an existing one that has been cleaned up in preparation.
- When installing ConfigMgr, the latest version of WUA (Windows Update Agent) is downloaded. Then, when the Configuration Manager client is installed, WUA is upgraded if necessary.
- WSUS administration console must be installed on the ConfigMgr site server if the SUP is being installed on a remote site system server and no WSUS installation is present.

3.2 Configuration Manager Software Update Point

The software update point (SUP) is the System Center Configuration Manager role that is responsible for managing software updates and their deployment to clients. The component is installed as a site system role via the Configuration Manager console. A software update point role must be configured first on the top-level site (stand-alone primary or a CAS).

A central administration site and primary sites are required to have the software update point role to deploy software updates to clients and to enable software updates compliance assessment. On secondary sites, the role is optional. (Microsoft, 2017a). The clients on a secondary site can also connect directly to an active software update point further upstream, such as on a primary site.

SUP installation has the following prerequisites:

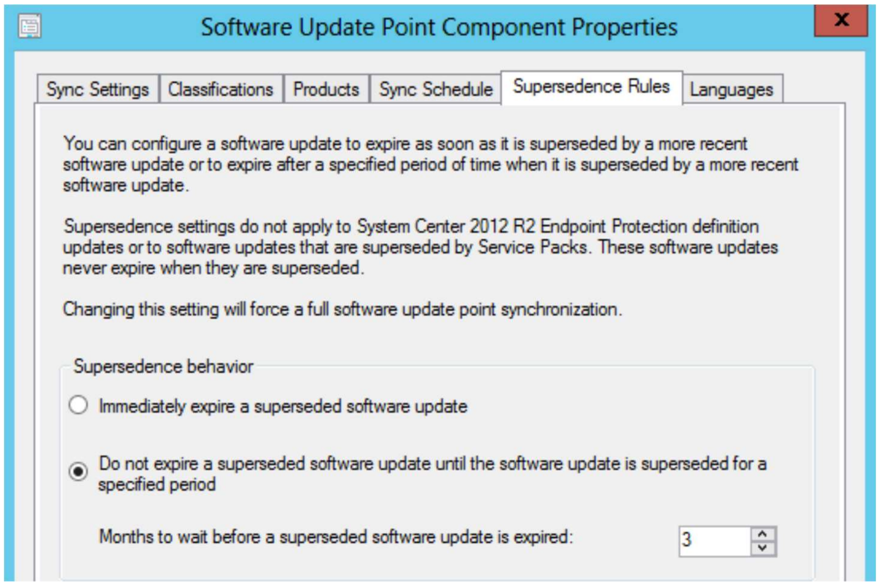
- 64-bit Operating system
- Server Core not supported.
- After installation, the following cannot be changed:
  - Domain name of the domain used in the installation.
  - Domain membership of the computer.
  - Computer name.
- Windows Server roles and features
  - .NET framework 4.5.2
  - Default IIS installation.
- If there are multiple SUPs at a site, it is important that they are all running the same version of WSUS (Microsoft, 2016d).
- KB3095113 must be installed before synchronizing the upgrades classification, otherwise the feature upgrades for Windows 10 will not display.

A software update point is required to be installed on the top-level site in the hierarchy (central administration site/stand-alone primary site). It is also required on the primary sites to deploy software updates to clients and enable the software updates compliance assessment as mentioned previously. The first software update point that is installed is also configured as the synchronization source.

If a primary site uses multiple SUPs, they should use a shared WSUS database. This can reduce network load when clients switch from one SUP to another because the database remains the same. (Keränen, 2014, 71).

The software updates must be synchronized on the SUP on the top-level site. In this process, software update metadata is retrieved from either Microsoft Update or a WSUS server that is not included in the ConfigMgr hierarchy (if, for example the CAS is not connected to the internet, it will synchronize from an intranet WSUS server). It is also possible to use WSUSUtil tool to import/export the necessary update metadata for the disconnected SUP. The SUPs further down in the hierarchy then retrieve the update metadata from the SUP that is defined as their upstream update source. (Microsoft, 2015). Child sites can only synchronize updates from upstream sources in the ConfigMgr hierarchy. The synchronization occurs on schedule or when the synchronization process is manually started from the ConfigMgr console (Microsoft, 2017b).

It is also important to specify Supersedence Rules in the Software Update Point Component properties. This can be set to immediately expire superseded updates, meaning they will not be deployed anymore, or to specify the number of months one must wait before superseded updates are expired.



You can configure a software update to expire as soon as it is superseded by a more recent software update or to expire after a specified period of time when it is superseded by a more recent software update.

Supersedence settings do not apply to System Center 2012 R2 Endpoint Protection definition updates or to software updates that are superseded by Service Packs. These software updates never expire when they are superseded.

Changing this setting will force a full software update point synchronization.

Supersedence behavior

○ Immediately expire a superseded software update

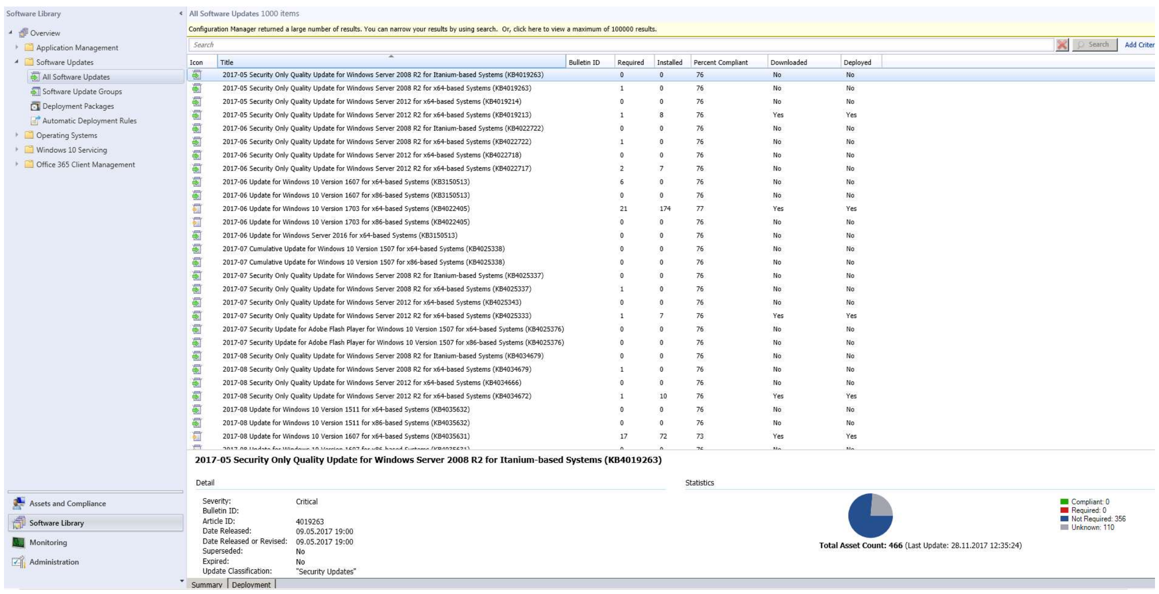
● Do not expire a superseded software update until the software update is superseded for a specified period

Months to wait before a superseded software update is expired: 3

Figure 2: Supersedense rules.

3.3 Deploying updates

In the ConfigMgr console, updates are managed from the Software Library workspace. All software updates that are available for products specified in the SUP installation are shown there.



All Software Updates 1000 items

Configuration Manager returned a large number of results. You can narrow your results by using search. Or, click here to view a maximum of 100000 results.

Icon	Title	Bulletin ID	Required	Installed	Percent Compliant	Downloaded	Deployed
	2017-05 Security Only Quality Update for Windows Server 2008 R2 for Itanium-based Systems (KB4019263)		0	0	76	No	No
	2017-05 Security Only Quality Update for Windows Server 2008 R2 for x64-based Systems (KB4019263)		1	0	76	No	No
	2017-05 Security Only Quality Update for Windows Server 2012 for x64-based Systems (KB4019214)		0	0	76	No	No
	2017-05 Security Only Quality Update for Windows Server 2012 R2 for x64-based Systems (KB4019213)		1	8	76	Yes	Yes
	2017-06 Security Only Quality Update for Windows Server 2008 R2 for Itanium-based Systems (KB4022722)		0	0	76	No	No
	2017-06 Security Only Quality Update for Windows Server 2008 R2 for x64-based Systems (KB4022722)		1	0	76	No	No
	2017-06 Security Only Quality Update for Windows Server 2012 for x64-based Systems (KB4022718)		0	0	76	No	No
	2017-06 Security Only Quality Update for Windows Server 2012 R2 for x64-based Systems (KB4022717)		2	7	76	No	No
	2017-06 Update for Windows 10 Version 1607 for x64-based Systems (KB3150513)		6	0	76	No	No
	2017-06 Update for Windows 10 Version 1607 for x86-based Systems (KB3150513)		0	0	76	No	No
	2017-06 Update for Windows 10 Version 1703 for x64-based Systems (KB4022405)		21	174	77	Yes	Yes
	2017-06 Update for Windows 10 Version 1703 for x86-based Systems (KB4022405)		0	0	76	No	No
	2017-06 Update for Windows Server 2016 for x64-based Systems (KB3150513)		0	0	76	No	No
	2017-07 Cumulative Update for Windows 10 Version 1507 for x64-based Systems (KB4025338)		0	0	76	No	No
	2017-07 Cumulative Update for Windows 10 Version 1507 for x86-based Systems (KB4025338)		0	0	76	No	No
	2017-07 Security Only Quality Update for Windows Server 2008 R2 for Itanium-based Systems (KB4025337)		0	0	76	No	No
	2017-07 Security Only Quality Update for Windows Server 2008 R2 for x64-based Systems (KB4025337)		1	0	76	No	No
	2017-07 Security Only Quality Update for Windows Server 2012 for x64-based Systems (KB4025343)		0	0	76	No	No
	2017-07 Security Only Quality Update for Windows Server 2012 R2 for x64-based Systems (KB4025333)		1	7	76	Yes	Yes
	2017-07 Security Update for Adobe Flash Player for Windows 10 Version 1507 for x64-based Systems (KB4025376)		0	0	76	No	No
	2017-07 Security Update for Adobe Flash Player for Windows 10 Version 1507 for x86-based Systems (KB4025376)		0	0	76	No	No
	2017-08 Security Only Quality Update for Windows Server 2008 R2 for Itanium-based Systems (KB4034679)		0	0	76	No	No
	2017-08 Security Only Quality Update for Windows Server 2008 R2 for x64-based Systems (KB4034679)		1	0	76	No	No
	2017-08 Security Only Quality Update for Windows Server 2012 for x64-based Systems (KB4034666)		0	0	76	No	No
	2017-08 Security Only Quality Update for Windows Server 2012 R2 for x64-based Systems (KB4034672)		1	10	76	Yes	Yes
	2017-08 Update for Windows 10 Version 1511 for x64-based Systems (KB4035632)		0	0	76	No	No
	2017-08 Update for Windows 10 Version 1511 for x86-based Systems (KB4035632)		0	0	76	No	No
	2017-08 Update for Windows 10 Version 1607 for x64-based Systems (KB4035631)		17	72	73	Yes	Yes

2017-05 Security Only Quality Update for Windows Server 2008 R2 for Itanium-based Systems (KB4019263)

Detail

Severity:	Critical
Bulletin ID:	
Article ID:	4019263
Date Released:	09.05.2017 19:00
Date Released or Revised:	09.05.2017 19:00
Superseded:	No
Expired:	No
Update Classification:	"Security Updates"

Statistics

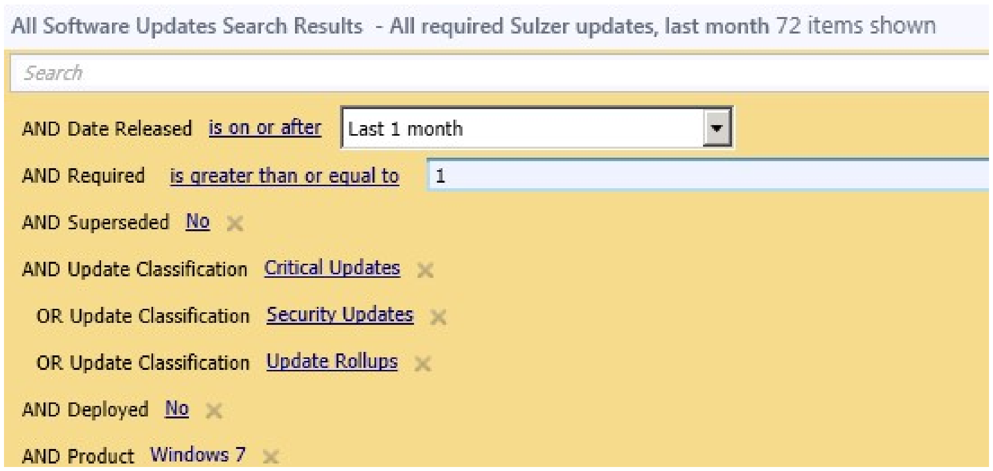
🟩 Compliant: 0
🟥 Required: 0
🟦 Not Required: 356
⬜ Unknown: 110

Total Asset Count: 466 (Last Update: 28.11.2017 12:35:24)

Figure 3: ConfigMgr Software Library.

In the search field, search criteria must be specified to list the updates that are needed. There are many fields which can be defined. However, with software updates the following are the most important:

- **Date Released**.
- **Required**. Displays updates that are applicable and required on the clients.
- **Superseded**. Specifies whether the update has been superseded by another update.
- **Update Classification**. Selects desired classifications, most commonly Critical Update, Security Updates and Update Rollups.
- **Deployed**. Specifies if the selected updates that are a part of any other Software Update deployment.
- **Product**. Selects the products that are being updated, such as Windows 7, Windows Server 2016 and Office 2013.



All Software Updates Search Results - All required Sulzer updates, last month 72 items shown

Search

AND Date Released is on or after Last 1 month

AND Required is greater than or equal to 1

AND Superseded No

AND Update Classification Critical Updates

OR Update Classification Security Updates

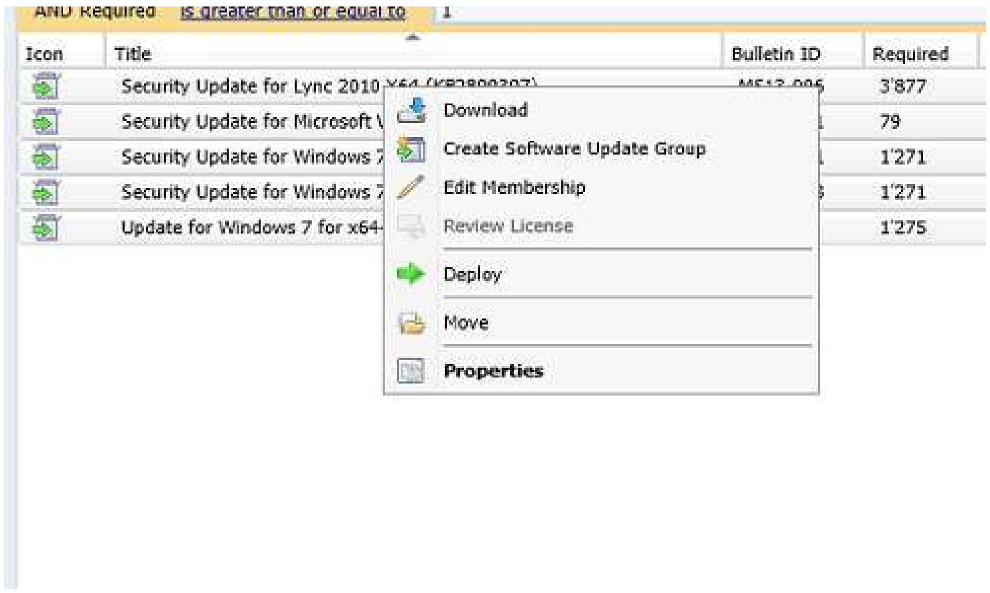
OR Update Classification Update Rollups

AND Deployed No

AND Product Windows 7

Figure 4: Criteria for updates

The search shows the specified updates, after which a Software Update Group (SUG) can be created for the updates in question.



AND Required is greater than or equal to 1

Icon	Title	Bulletin ID	Required
	Security Update for Lync 2010 X64 (KB2899397)	MS13-096	3'877
	Security Update for Microsoft V		79
	Security Update for Windows 7		1'271
	Security Update for Windows 7		1'271
	Update for Windows 7 for x64-		1'275

Figure 5: Software Update Group creation.

A name must be selected and a folder must be created for the deployment package. The name should describe the Software Update Group, such as the date deployed, the collection of devices it is deployed to (clients, servers) and the contents of the SUG. Next, the created SUG must be selected and its content downloaded using the download option. In the download Software Updates wizard, a new deployment package must be created and named (preferrably using the same naming convention as in the SUG). One must select the desired distribution points the content should be deployed to (normally all/distribution point group that contains all DPs) and set the distribution priority (determines in which order packages are sent to other sites). Next, deployment package properties must be specified, such as package source (a network path to where the content is to be located) and the download location from where updates are to be downloaded (an intranet location or the Internet). After that, appropriate languages must be selected, after which the summary of the process can be viewed. On the completion page, it must be made certain that the content has been downloaded succesfully, after which the wizard can be closed. A single update deployment can contain up to a maximum of 1000 software updates (Microsoft, 2016e). As a note, deployment packages contain the actual updates files and are not related to deployments in other ways, they simply contain the source files for distribution to the DPs. Also, updates are not duplicated if they occur in different packages, they are only referenced from the deployment package that contains them.

After a Software Update group has been successfully created and content downloaded, the SUG can be deployed to clients. An update group must be selected and selecting deploy will open the Deploy Software Updates wizard. A name, description, SUG (verify it is correct), possible deployment template and collection must be configured on the general page.

A collection represents a grouping of users or devices (never both) that update deployments are targeted towards but can be used in many other ways such as in application deployments, grouping resources, managing client or power settings and maintenance windows. Collections can be created in the Assets and Compliance workspace. On either user collections or device collections, clicking create collection will open the Create Device Collection wizard. On the general page, collection name, possible comments and a limiting collection can be configured. The limiting collection defines a collection from which all members of the collection being created can come from. On the next page, the membership rules for the collection must be defined. The rules can be based on:

- **Direct rules**, in which the members are manually defined/excluded.
- **Include/Exclude** collection(s) rule.
- **Query rules**, where a WMI query based on WQL (WMI Query Language) defines include/exclude rules for the membership. These can be created using the wizard UI or written in WQL.

```
select SMS_R_System.NetbiosName,
SMS_R_System.OperatingSystemNameandVersion from
SMS_R_System where
SMS_R_System.OperatingSystemNameandVersion like "%Workstation 6.1%"
```

Figure 6: Example WQL query that returns all Windows 7 machines.

To make queries based on AD properties like Group Membership or Organizational Unit, ConfigMgr must be collecting information on all of the desired properties (Jupe, 2015). AD discovery can be configured from Administration -> Hierarchy Configuration -> Discovery methods.

After the membership rules have been defined, the collection is created succesfully and can be used to deploy updates.

3.3.1 Manual Update Deployment

Updates are usually deployed some time after Microsoft releases them on what is called “Patch Tuesday”, the second Tuesday of every month. New releases are made available to Microsoft update and are gathered into software update groups. In order to deploy software updates to a collection, one must select the collection and choose deploy -> software updates. In the wizard, a deployment name must be input using the same naming convention as when creating the software update group and the SUG must be specified for the deployment. Next, the deployment type can be set to ‘required’, so all clients will see the updates as mandatory as opposed to available which would show the updates as available in the software center but would not automatically install them. A detail level of the messages to the user must be specified and a timeline for when the updates are available and the deadline for installation must be selected. The deadline behaviour can be set to forced installation and a forced restart if necessary. It is also possible to suppress restart behaviour on servers, which is a good option as servers are normally patched during their own maintenance window. In the choose deployment options, “download software updates distribution point and install” option should be selected. Fallback options and the need for BranchCache should be specified, ticking the box will allow clients to share content with each other on the same subnet. Using BranchCache can heavily reduce network traffic in locations without its own DP. If clients are offline, they can be permitted to download update content from Microsoft Update. Next specify the correct deployment package for the update group and select the the dowload location for your site, an intranet location or the internet. Last select the languages that the updates should be available in in your environment. It is also possible to use Wake-On-Lan setting to start dormant clients to install updates to them, however this requires that Wake-On-Lan is already configured on the clients and on the network.

After a deployment is created, an associated software update policy is also sent to client computers. Content for the software updates is downloaded from the distribution point to the local cache on the clients, after which the updates are available for installation. (Microsoft, 2016f). For internet-based clients, the download location can be set to Microsoft Update as this is a faster way compared to speeds over a standard VPN (Virtual Private Network) connection to the company intranet. WUA sees the state of the updates and sends information on compliance back to its Management Point.

3.3.2 Automatic Deployment Rules

Automatic Deployment Rules or ADRs can also be used in ConfigMgr to deploy updates automatically. When rules are created, the software updates that should be included in the deployment and running frequency must be specified. It is also important to specify whether the updates are included in a new SUG or added to an existing one. It is important to note that if an existing SUG is chosen, all updates in it will be removed and new ones matching the rule criteria will be added to it. It is not possible to create a SUG manually and then have an ADR add new updates to it, instead the ADR will always create a new SUG when the rule first runs (Smpyrakis, 2012). When an ADR runs, software updates meeting the previously set criteria are added to a SUG and source installer files are downloaded and copied to selected DPs (Rimmerman et al. 2015, 48-49). A target collection or collections must be specified and after the rule is enabled, the updates contained in the SUG are deployed to clients in the target collection automatically. If new clients are added to the target collection, updates are automatically deployed to them. When new updates are added to the SUG via the ruleset in the ADR, they are also deployed to the clients in the collection automatically. It must be specified whether the deployment is enabled when the rule runs. If the deployment is enabled, the updates are deployed in the manner as described. Otherwise, the SUG is created, and the software update policy configured, but the updates are not deployed. Of course, the same deployment characteristics apply to ADRs as other deployments such as user experience and deadline behavior. Also, the same Wake-On-Lan settings are required for clients and network if the technology is to be used.

In the company, ADRs are currently not used in operating system or Office update deployment as the rules add new updates and fixes to deployment packages, which then grow and become more difficult to distribute to DPs all over the world. As an exception, the SCEP (System Center Endpoint Protection) definition updates are deployed using ADRs, as the definitions are updated frequently and need to be deployed to almost all systems. Also, they are much smaller in size than application or OS updates so the SUG size for SCEP definition updates never becomes too large.

3.4 Operating system and Office updates

Operating system update categories include:

- **Critical Updates**: Wide release update for a specific problem addressing a critical, nonsecurity-related bug.
- **Definition Updates**: Update to virus or other definition files.
- **Feature Packs**: New product features that are distributed outside of a product release and that are normally included in the next full product release.
- **Security Updates**: A broadly distributed update for a product-specific security issue.
- **Service Packs**: A set of hotfixes concerning an application. These hotfixes can include: security updates, critical updates, software updates, and others.
- **Tools**: A utility or feature that helps to complete one or more tasks.
- **Update Rollups**: A cumulative set of hotfixes that are packaged together. These hotfixes can include security updates, critical updates and others. These normally address a specific area, like a security or a product component.
- **Updates**: An update to a currently installed application or file.
- **Upgrade**: Upgrade for Windows 10 features and functionality. Software update points and sites must run a minimum of WSUS 4.0 with the hotfix 3095113 to get the Upgrade classification.

Office product updates such as hotfixes for Word, Excel and PowerPoint are included in the product search criteria when the software update groups are created for the client base. In the company, the Office 2010/2016 updates are deployed in the same software update groups as the operating system updates, so there are no specific Office update deployments. The clients receive their OS and Office patches through the same channel. This is a good solution, as separating the two update categories would needlessly create more SUGs and therefore more work.

3.5 Office 365 updates

Office 365 is a subscription based service for Microsoft Office products, which is not a set version but a constantly changing one with Microsoft delivering updates and feature upgrades on a steady schedule.

Office 365 patching process in SCCM is separated from the normal security updates and standard Office updates method. Clients using O365 have an O365 client installed, which in turn updates and manages the features available to the user.

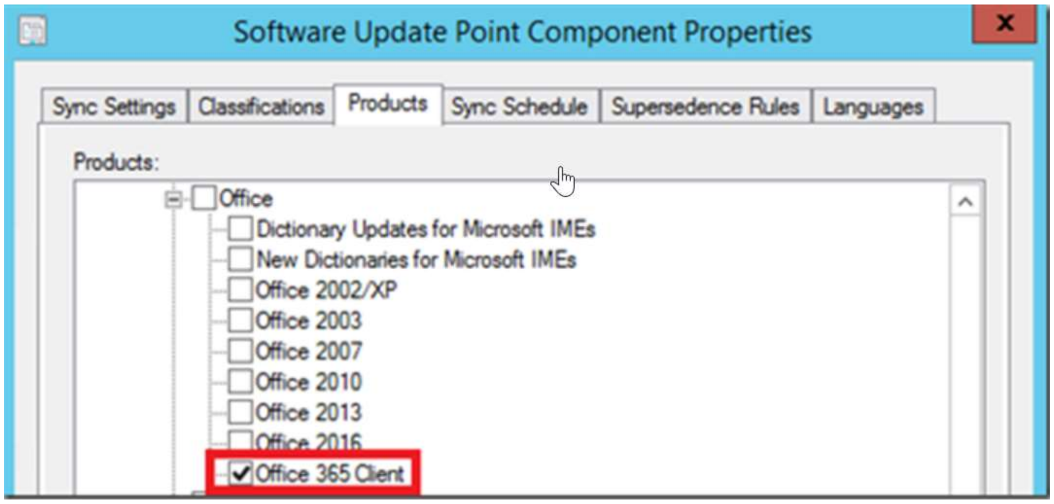
3.5.1 Prerequisites for Office 365 ProPlus updates with Configuration Manager

Updating and managing O365 with ConfigMgr requires the following features:

- Configuration Manager version 1602 or later to deploy O365 updates through ConfigMgr
- Working Software Update Point
- Enabled O365 updates in ConfigMgr.
- Supported Office 365 Client.
- Clients enabled to receive O365 updates from ConfigMgr.

3.5.2 Enable Office 365 updates in ConfigMgr

In ConfigMgr software update point properties, the Office 365 client must be selected from the products list in the SUP component properties, and classifications must be included such as security and definition updates.


Software Update Point Component Properties

Sync Settings | Classifications | Products | Sync Schedule | Supersedence Rules | Languages

Products:

- [ ] Office
  - [ ] Dictionary Updates for Microsoft IMEs
  - [ ] New Dictionaries for Microsoft IMEs
  - [ ] Office 2002/XP
  - [ ] Office 2003
  - [ ] Office 2007
  - [ ] Office 2010
  - [ ] Office 2013
  - [ ] Office 2016
  - [✓] Office 365 Client

Figure 7: SUP Component configuration for O365.

Office 365 Client cannot be directly deployed using WSUS, so ConfigMgr must be used for the task. The O365 update packages are numbered as for example 1701 and increment with new releases.

Clients can be enabled to receive O365 patches from SCCM in three ways:

- **Group Policy setting**:

```
“Computer Configuration\Policies\Administrative Templates\Microsoft
Office 2016 (Machine)\Updates\Office 365 Client Management = Enable”
```

- **Office Deployment Tool**: Xml-configuration file

```
“HKLM\SOFTWARE\Microsoft\Office\ClickToRun\Configuration\OfficeMgmtC
OM = (REG_SZ) True”
```

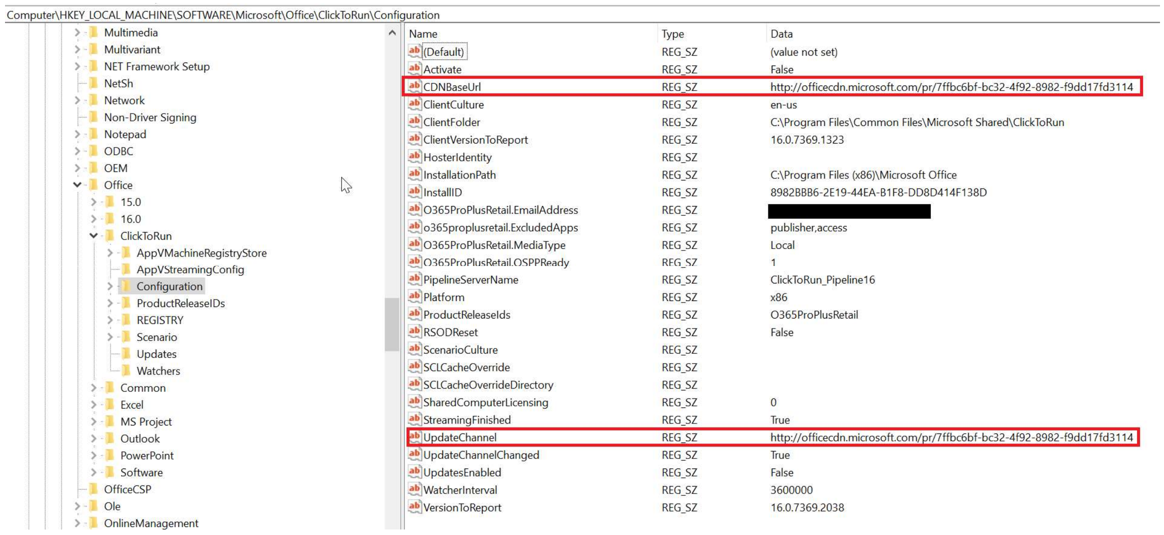
- **Modify client registry manually** with configuration item, script, task sequence etc.

```
“HKLM\SOFTWARE\Microsoft\Office\ClickToRun\Configuration\OfficeMgmtC
OM = (REG_SZ) True”
```

Then, an update channel must be configured by modifying the following registry key.

```
 “HKLM\SOFTWARE\Microsoft\Office\ClickToRun\Configuration\CDNBaseUr
l (REG_SZ)”
```

This can be done via Group Policy Preference, Script, Configuration Item or manually with registry modifications. The CDNBaseURL value specifies the update channel URL as one of the available channels. An example of a CDNBaseURL http://officecdn.microsoft.com/pr/7ffbc6bf-bc32-4f92-8982-f9dd17fd3114. This URL corresponds to the Monthly Channel.



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\ClickToRun\Configuration

Name	Type	Data
(Default)	REG_SZ	(value not set)
Activate	REG_SZ	False
CDNBaseUrl	REG_SZ	http://officecdn.microsoft.com/pr/7ffbc6bf-bc32-4f92-8982-f9dd17fd3114
ClientCulture	REG_SZ	en-us
ClientFolder	REG_SZ	C:\Program Files\Common Files\Microsoft Shared\ClickToRun
ClientVersionToReport	REG_SZ	16.0.7369.1323
HosterIdentity	REG_SZ	
InstallationPath	REG_SZ	C:\Program Files (x86)\Microsoft Office
InstallID	REG_SZ	8982BBB6-2E19-44EA-B1F8-DD8D414F138D
O365ProPlusRetail.EmailAddress	REG_SZ	[redacted]
o365proplusretail.ExcludedApps	REG_SZ	publisher,access
O365ProPlusRetail.MediaType	REG_SZ	Local
O365ProPlusRetail.OSPPReady	REG_SZ	1
PipelineServerName	REG_SZ	ClickToRun_Pipeline16
Platform	REG_SZ	x86
ProductReleaseIds	REG_SZ	O365ProPlusRetail
RSODReset	REG_SZ	False
ScenarioCulture	REG_SZ	
SCLCacheOverride	REG_SZ	
SCLCacheOverrideDirectory	REG_SZ	
SharedComputerLicensing	REG_SZ	0
StreamingFinished	REG_SZ	True
UpdateChannel	REG_SZ	http://officecdn.microsoft.com/pr/7ffbc6bf-bc32-4f92-8982-f9dd17fd3114
UpdateChannelChanged	REG_SZ	True
UpdatesEnabled	REG_SZ	False
WatcherInterval	REG_SZ	3600000
VersionToReport	REG_SZ	16.0.7369.2038

Figure 8: Registry values for Office365 deployment.

Here, the UpdatesEnabled value is set to False as this only affects the way users see available updates. The updates are still delivered to clients from ConfigMgr even if this is set to False (Fors, 2017).

Office Deployment Tool is used to deploy Office 365 and its features in large scale and to repackage O365 content. An example of the content of an .xml configuration file for ODT is presented below.

```
<Configuration>
<Add OfficeClientEdition=”32” Channel=”Current” OfficeMgmtCOM=”TRUE
” Version=”16.0.7571.2072”>
<Product ID=”O365ProPlusRetail”>
<Language ID=”en-us”/>
<Language ID=”fi-fi”/>
<ExcludeApp ID=”PowerPoint”/>
<ExcludeApp ID=”OneDrive”/>
<ExcludeApp ID=”Outlook”/>
<ExcludeApp ID=”OneNote”/>
<ExcludeApp ID=”Lync”/>
<ExcludeApp ID=”Groove”/>
<ExcludeApp ID=”Excel”/>
<ExcludeApp ID=”Access”/>
<ExcludeApp ID=”Publisher”/>
</Product>
</Add>
<Updates Enabled=”TRUE” />
<Display Level=”Full” AcceptEULA=”TRUE”/>
</Configuration>
```

The above configuration file specifies all necessary information mentioned previously and information that can be modified such as language. It is also possible to create and deploy either a legacy package or a script application for clients to install Office 365 applications.

3.5.3 Office 365 Update Channels

There are currently three different update channels for O365 updates, Monthly channel, Semi-Annual channel and Semi-Annual (Targeted) channel.

Monthly Channel (formerly Current Channel) provides users with the newest features as soon as they are released. This is the default update channel for Visio Pro for Office 365, Project Online Desktop Client and Office 365 Business.

There is no set schedule for new feature releases, but the Monthly Channel provides them as they are released. The releases might also contain non-security related updates, such as quality-of-life improvements that affect stability or performance.

Security updates for this channel are released on Patch Tuesday, the second Tuesday of every month. If a critical non-security related fix needs to be released, Microsoft will provide a separate Monthly channel release for this.

Releases for the Monthly Channel are cumulative. The most recent release contains all features such as security- and non-security updates from previous Monthly releases.

A Monthly Channel release is only supported until the next Monthly Channel update is released. New security updates will not be provided for previous Monthly Channel releases.

Semi-Annual Channel provides users with new features twice a year, in January and July. This is the default channel for Office 365 ProPlus.

The Semi-Annual channel releases are normally provided on the second Tuesday on January and July. If an organization has many business applications, office macros or other similar products in use that need to be tested thoroughly with new features and releases, the Semi-Annual channel makes this possible. As new features are provided only twice a year, here is more time for problems to arise and be fixed, in comparison to the Monthly Channel.

Microsoft provides builds of the next release in the four months before the new Semi-Annual Channel feature release, so that pilot users and compatibility testers are able to work with the new release. Microsoft (2018a). Additional security updates and critical updates will be added to the Semi-Annual Channel if needed, after the main releases in January and July.

Semi-Annual Channel (Targeted) provides pilot users and application compatibility testers with early opportunity to test the next Semi-Annual channel. Releases for this channel are made in March and September. While this channel is intended for testing purposes, it is a fully functional version of Office 365.

New content for the Semi-Annual Channel (Targeted) is only released twice in a year, March and September. The Semi-Annual Channel (Targeted) will receive no new content at other times. The included features in the Semi-Annual Channel (Targeted) are the same features that have already been released in the Monthly Channel. Microsoft (2018a).

The Semi-Annual Channel (Targeted) acts as a testing channel for enterprises with strong requirements towards application functionality. Semi-Annual Channel (Targeted) will also be updated with all the latest security and critical updates as they become available.

If an organization has a large enough testing platform and personnel to monitor the deployment of O365 and other updates, the Monthly channel can also be utilized for testing purposes. When new updates are made available on Office CDN, they can be deployed to a test group, which then screens for possible problems the updates create. If no problems arise, the updates can be deployed into the production environment.

In most cases, it is easier and more reliable to use the Semi-Annual channels, either for one’s own testing or simply deploying the updates that have been in use for monthly channel users. Any issues they might cause should have been resolved and patched at this point.

This must be evaluated on the grounds whether it is desirable to have the newest updates and thus the most security and possibly compatibility issues with versions elsewhere.

3.6 Removing Updates

There are times when an update causes severe enough issues that it must be removed from the clients it was deployed to.

In the case of operating system updates, this can be done by using WUSA, Windows Update Standalone Installer. WUSA uses the WUA (Windows Update Agent) API to install/uninstall update packages.

Here are examples of an install- and an uninstallstring:

Install:

```
wusa.exe d:\934307\Windows6.0-KB934307-x86.msu
```

Uninstall:

```
wusa.exe /kb:updateID /uninstall /quiet /norestart
```

The commands can be delivered to clients via a ConfigMgr task sequence set to run a command line. This can execute the uninstallation silently with no user interaction specified with the switches in the example. It is important to note that before the uninstallation is implemented, it must be made sure that the update in question is not a part of an active deployment. Otherwise the result is an install/uninstall loop.

The target operating system must be taken into consideration as ConfigMgr is a 32-bit application and calls a 32-bit version of the WUSA.exe. This will fail on 64-clients, so a 64-redirection must be specified in the task sequence. The redirection should be made when needed on a client, but as most task sequences in this scope are delivered to a mostly uniform client base, the command can be set to a constant.

Office updates, however, cannot be uninstalled using WUSA. However, all Office updates are provided with an uninstall string that can be found in the registry.

- Native applications:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninst
all
```

- 32-bit applications on 64-operating system:

```
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVe
rsion\Uninstall
```

Registry keys in question have a complex GUID which is constructed from Office version and other parameters, and the uninstallation string is contained as a sub key in the ‘uninstallstring’ value. The GUID can vary from client to client depending on configuration. The first part of the GUID refers to the Office product and is based on properties such as version and build. The second part of the GUID is related to the specific update. Thereby, the same uninstall string cannot be automatically used in a task sequence on all affected clients. In order to locate the uninstall string for a certain update on each client, each sub key in the correct uninstall key location must be examined. This can be done with PowerShell.

```
$Patchnumber = 'ExampleKB'
$reg = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMa-
chine', $env:COMPUTERNAME)
$software = $reg.OpenSubKey("Software\Wow6432Node\Microsoft\Win-
dows\CurrentVersion\Uninstall").GetSubKeyNames() |
? {$reg.OpenSubKey("Software\Wow6432Node\Microsoft\Windows\Cur-
rentVersion\Uninstall\" + $_).getvalue('DisplayName') -like "*$Patch-
number*"} |
% {$reg.OpenSubKey("Software\Wow6432Node\Microsoft\Windows\Cur-
rentVersion\Uninstall\" + $_).getvalue('UninstallString')}
$reg.close()
```

The code opens the LocalMachine base key on the client registry and searches sub keys of ‘*Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall*’ for a match to the provided patch number and returns the uninstallstring value for the ones that match. This is for 32-bit Office products running on 64-bit operating systems.

Example of an uninstallstring:

```
"C:\Program Files (x86)\Common Files\Microsoft Shared\OFFICE15\Oarp
many.exe" /removereleaseinpatch "{90150000-0011-0000-0000-0000000FF
1CE}" "{53EA601B-E127-4D6B-9093-3B2797A6B773}" "1033" "0"
```

Running the command retrieved from registry on remote clients will require manual intervention as a popup will be shown during the uninstall. The command must be converted to a quiet form such as:

```
msiexec /uninstall /package {90150000-0011-0000-0000-0000000FF1CE
} MSIPATCHREMOVE={53EA601B-E127-4D6B-9093-3B2797A6B773}/qb /noresta
rt /l+ C:\Windows\CCM\Logs\KB3172519_uninstall.txt
```

Here are the msiexec parameters.

```
/uninstall [/quiet][/passive][/q{n|b|r|f}]
```

The /q parameter specifies the User interaction level. In the example, command /qb means basic UI. The /qn parameter means no UI while the separate /quiet parameter also means no UI. Using the /passive parameter shows only a progress bar. Also, the logging specifics and /norestart parameter are shown.

The uninstallstring retrieved from the registry can be converted to the msiexec- format using PowerShell:

```
$Patchnumber = 'ExampleKB'
$uistring = '"C:\Program Files (x86)\Common Files\Microsoft
Shared\OFFICE15\Oarpmany.exe" /removereleaseinpatch
"{90150000-0011-0000-0000-0000000FF1CE}" "{53EA601B-E127-4D6B-9093-
3B2797A6B773}" "1033" "0"'
$GUID1 = $uistring.Split('"')[3]
$GUID2= $uistring.Split('"')[5]
$MsiString = "msiexec.exe /uninstall /package $GUID1 MSIPATCHRE-
MOVE=$GUID2 /qb /norestart /l+ c:\windows\system32\ccm\logs\$Patch-
number" + "_uninstall.txt"
```

The code splits the uninstallstring with a “, locates the GUIDs and returns a new uninstall string which also logs the process to a text file in a specified path.

A script/command such as this can be delivered with a task sequence and run on individual clients, so the uninstallstring is always obtained from the client the update is being uninstalled from, ensuring that the correct patch GUID is used. The patch being removed should also be removed from any active deployments in order to avoid an install/uninstall loop.

In some cases, when the user base is not using a singular version of Office, it may be necessary to determine which version of Office the client has. This is because Office 365 updates are not removed in the same manner as the previous versions.

The Office version in use can be determined with a PowerShell script that looks at the keys in '*HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office*' and returns the Office version based on the sub keys that are present. The operating system is not relevant when uninstalling Office updates in versions 2013 and 2016 as the updates would be removed either by a script that is deployed to a collection of devices which usually contain only clients with a certain OS. As mentioned earlier, Office 365 updates must be removed by reverting the deployed version of the O365 client. The Office version in use can also be obtained using PowerShell.

```
$reg = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMa-
chine', $env:COMPUTERNAME)
$regkey = $reg.OpenSubKey("SOFTWARE\Microsoft\Office").GetSubKey-
Names()
if($regkey -contains '14.0' -and $regkey -notcontains ('15.0' -or
'16.0' -or 'ClickToRun')){$OfficeVersion = 'Office 2010'}
if($regkey -contains '15.0' -and $regkey -notcontains ('16.0' -or
'ClickToRun')){$OfficeVersion = 'Office 2013'}
if($regkey -contains '16.0' -and $regkey -notcontains
'ClickToRun'){$OfficeVersion = 'Office 2016'}
if($regkey -contains 'ClickToRun'){$OfficeVersion = 'Office 365'}
$reg.close()
```

The code opens the LocalMachine base key on the client registry and then the subkey for '*SOFTWARE\Microsoft\Office*' where the presence of the different Office version subkeys can be used to determine the Office version on the client. The $OfficeVersion variable will then contain the Office version of the client. It is also easy to run this for a remote client by simply switching the $env:COMPUTERNAME to the targeted client. In this way, it is possible to ascertain the versions of Office in use.

4 INTEGRATING THE END USER TO THE UPDATE PROCESS

This part of the thesis will focus on creating the functionality to integrate end users to the update process via an Active Directory group which contains user devices that will receive Microsoft updates early for testing purposes.

The objective is to gather end user feedback from a pool of test users that will optimally contain users for all important applications to find any negative impacts the new updates may cause. The procedure is to send all test group members an email which contains a link to an intranet form used to report back any issues shortly after new updates have been deployed to the group. From the received data, it can be determined whether the updates are safe to deploy to the production environment. The process relies on active feedback on the part of the end users. The testing group would ideally contain up to 1000 devices/users (from 15000 devices in the company), to maximize the probability to detect any possible issues with the updates before deploying them into production.

The end users are allowed to join the group, using a form which adds the user and their client to a list. Administrators will have the option to add any user and their device to the group, so people that use important applications are included in the update testing.

There were other possible candidates for the software that could have been used to build the forms/applications, such as K2 Blackpearl. At first, it was intended that the K2 platform would be used to construct the forms and logic for the process, but the scope of the process kept growing, so it was decided that it would be best if this part of the thesis would be done as a Proof of Concept using Microsoft InfoPath and SharePoint as the platform.

In the process, SharePoint lists act as the backend databases from which the end users are added to the AD Test Group and where the reports from the end users are sent. The forms are constructed with InfoPath and the management application is coded with PowerShell, while Windows Presentation Foundation (WPF) is used for the GUI which is created in Visual Studio. XAML (Extensible Application Markup Language) allows PowerShell to interact with the

GUI objects and run the code. XAML is a part of the WPF, and WPF in turn is the category of features in .NET Framework 3.5, which enables the visual presentation of Windows applications and browser-based client applications (Microsoft, 2018b).

The Active Directory group is managed via the SharePoint list that contains all the members and their clients. When a member is added to the SharePoint list, a script running as a scheduled task will read the list and add all clients to the AD group if they are not already there. Each entry on the SharePoint list has a property ‘member’ which is set to true when the client/user are added to the list and set to false when an admin “removes” user from the AD group. The admin is only setting the user’s ‘member’ property to false, and a script executes the actual task of removing the client from the AD group and the SharePoint list.

In this thesis, the user/device connection is made through a SharePoint list which contains a list of users and their devices. In an actual implementation, the data would come from ConfigMgr and a method called User Device Affinity which creates a connection between user and device. In the end, the resulting functionality is the same.

4.1 InfoPath

Microsoft InfoPath is a software application used to design, submit and fill forms. The forms can contain data from various sources. Data connections can be made to SharePoint, SQL and Access databases. The forms can then be used to update, create and delete data. InfoPath can also capture data from sources such as web services, XML and other forms.

The forms are based on controls, views, rules and bindings. It is possible to construct very complicated forms using these methods to interact with multiple different systems simultaneously.

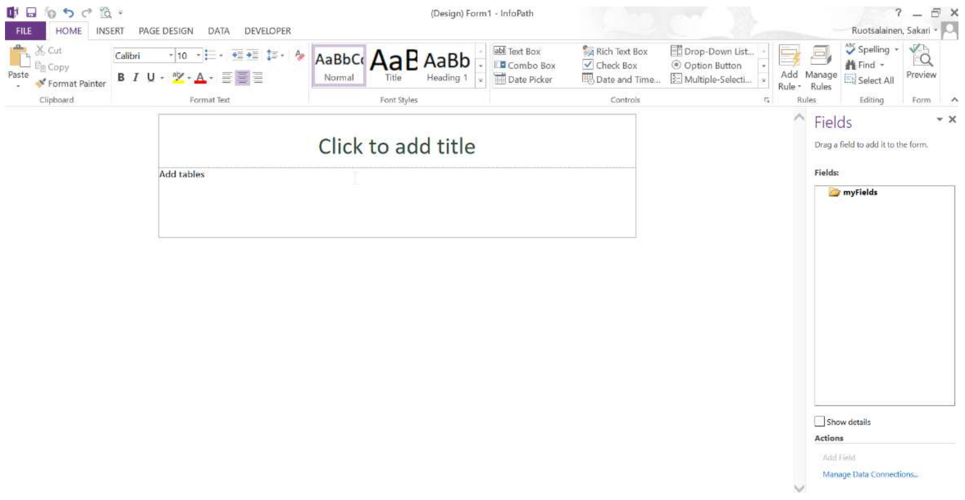
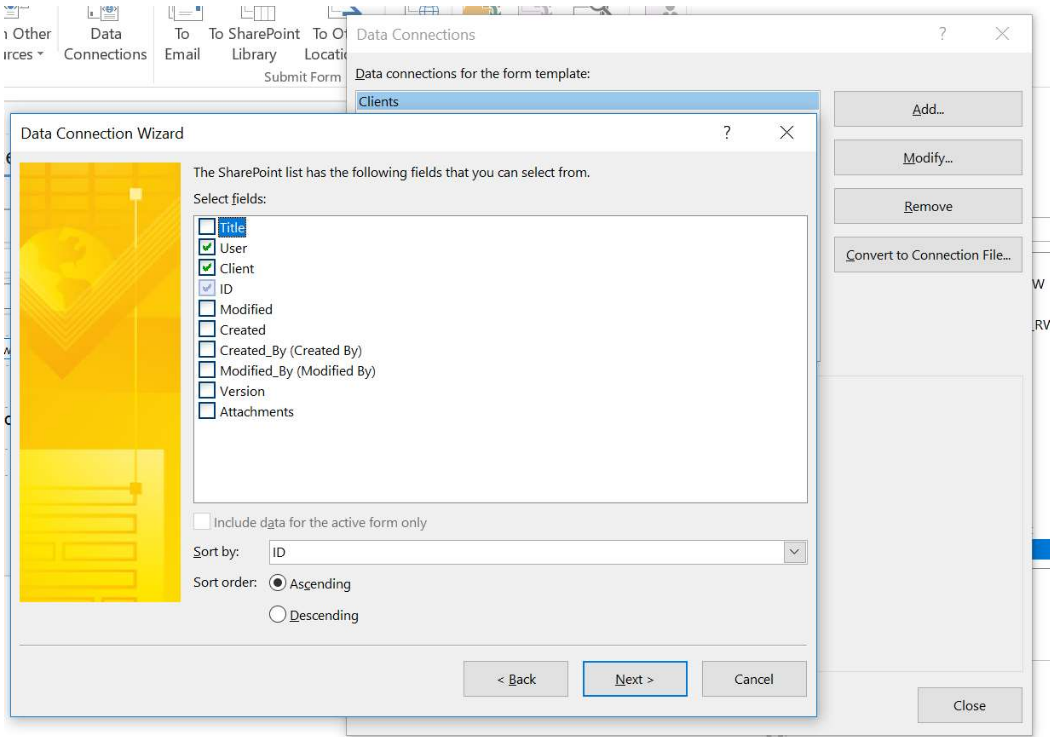


Figure 9: InfoPath Designer.

It is easy to implement InfoPath forms to any SharePoint list as the form that is presented when clicking the add item can be opened and modified in InfoPath designer and then published back on the SharePoint site for use.

4.2 User Join Form

The form which will be used to join the testing group for Microsoft updates is based on a SharePoint list “Test Group Members”. The list contains the following properties: ‘username’, ‘client’, ‘member’ and ‘manualclient’ for each entry. The ‘manualclient’ property is defined only if the user enters a device that is not found in the dropdown menu for personal devices. The properties on this list are all single lines of text, but the data type can be for various others. The form contains a data connection to another SharePoint list Clients to present the users a list of their own clients for ease of use.



Data connections for the form template:
Clients

The SharePoint list has the following fields that you can select from.

Select fields:

- [ ] Title
- [x] User
- [x] Client
- [x] ID
- [ ] Modified
- [ ] Created
- [ ] Created_By (Created By)
- [ ] Modified_By (Modified By)
- [ ] Version
- [ ] Attachments

[ ] Include data for the active form only

Sort by: ID

Sort order: [✓ Ascending] Descending

Figure 10:Data connection to 'Clients' SharePoint list.

The SharePoint list add-item form can be modified and opened in InfoPath designer from the list in SharePoint, in the top ribbon.

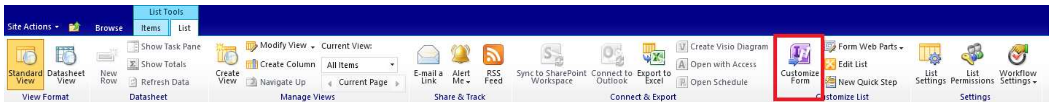
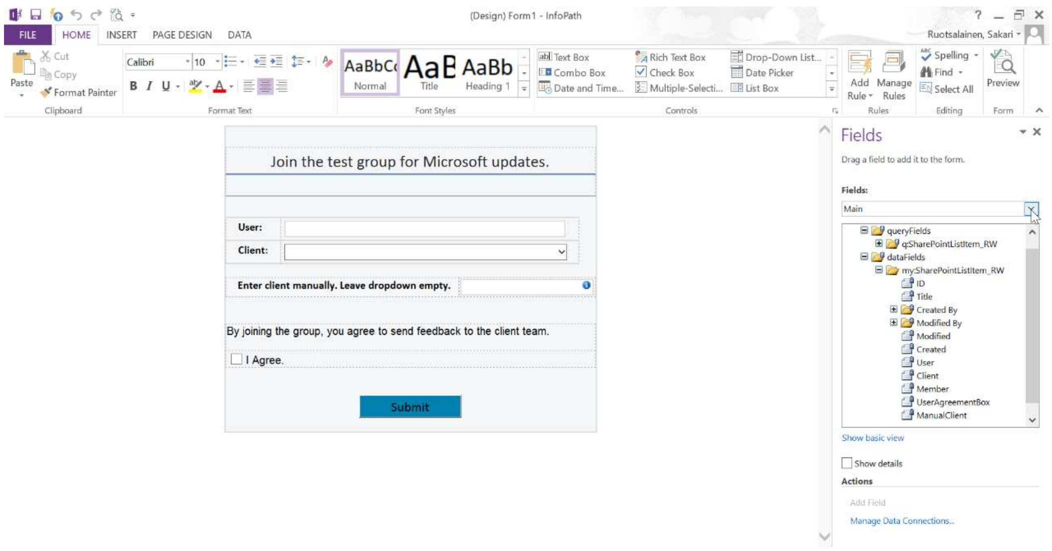


Figure 11: Access SharePoint form in InfoPath.



Join the test group for Microsoft updates.

User:

Client:

Enter client manually. Leave dropdown empty.

By joining the group, you agree to send feedback to the client team.

☐ I Agree.

Submit

Fields

Drag a field to add it to the form.

Fields: Main

- queryFields
  - q:SharePointListItem_RW
- dataFields
  - my:SharePointListItem_RW
    - ID
    - Title
    - Created By
    - Modified By
    - Modified
    - Created
    - User
    - Client
    - Member
    - UserAgreementBox
    - ManualClient

Show basic view

☐ Show details

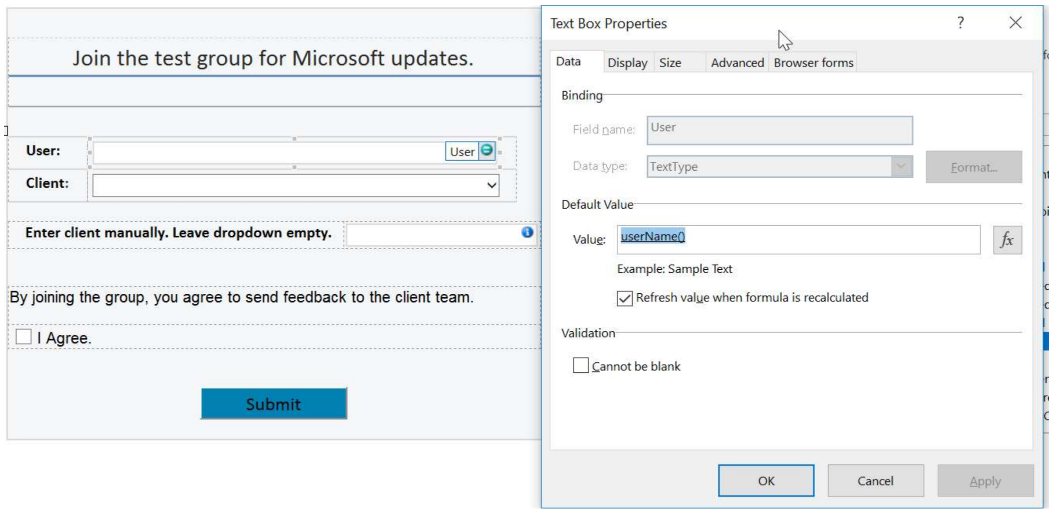
Actions

Add Field

Manage Data Connections...

Figure 12: User join form in InfoPath.

The form was created using the controls for the different data columns on the SharePoint list it is based on. The user control on the form is populated by an InfoPath function that retrieves the username of the user accessing the form, so an end user can only add clients assigned to them to the testing group.



Join the test group for Microsoft updates.

User: User

Client:

Enter client manually. Leave dropdown empty.

By joining the group, you agree to send feedback to the client team.

☐ I Agree.

Submit

Text Box Properties

Data | Display | Size | Advanced | Browser forms

Binding

Field name: User

Data type: TextType

Format...

Default Value

Value: userName()

Example: Sample Text

☑ Refresh value when formula is recalculated

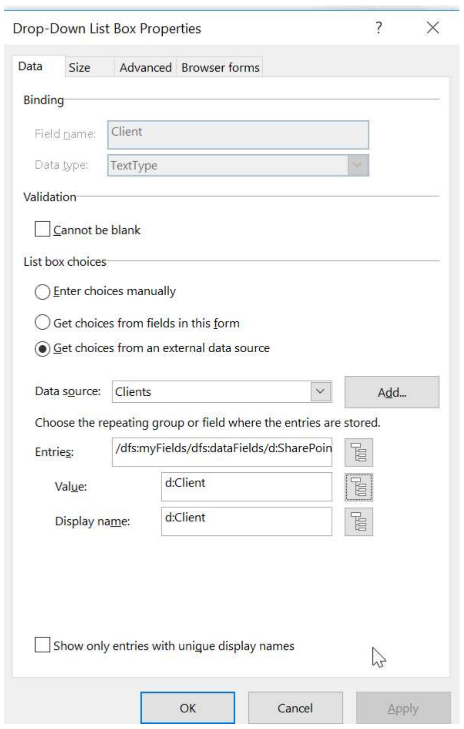
Validation

☐ Cannot be blank

OK | Cancel | Apply

Figure 13: User control information.

The device dropdown menu control is populated by data from an external source, the ‘Clients’ SharePoint list. The data is filtered in such a way that only items shown on the dropdown menu will be items on the ‘Clients’ list where the user matches the content of the username control.


Drop-Down List Box Properties

Data | Size | Advanced | Browser forms

Binding

Field name: Client

Data type: TextType

Validation

☐ Cannot be blank

List box choices

○ Enter choices manually

○ Get choices from fields in this form

● Get choices from an external data source

Data source: Clients

Choose the repeating group or field where the entries are stored.

Entries: /dfs:myFields/dfs:dataFields/d:SharePoin

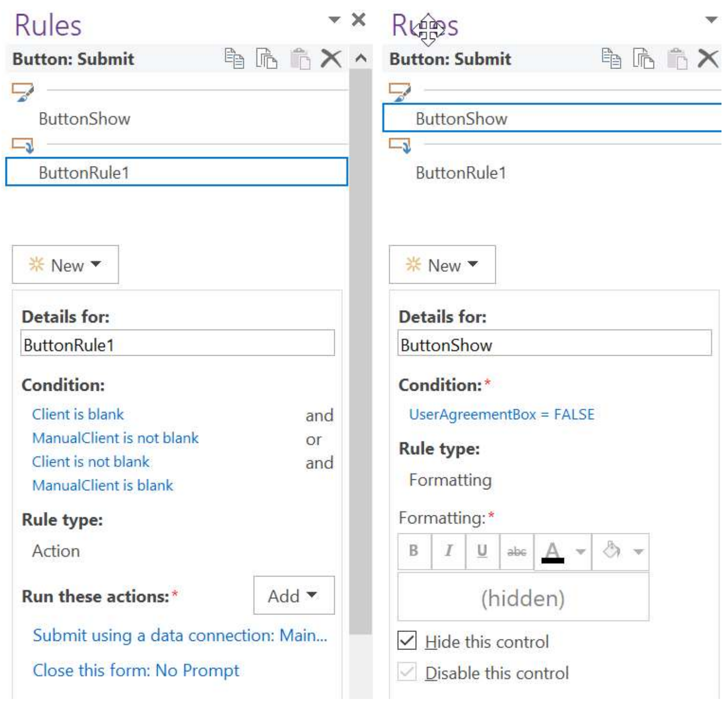
Value: d:Client

Display name: d:Client

☐ Show only entries with unique display names

Figure 14:Clients data filter.

Rules are set in such a way that the ‘Submit’ button will be displayed if the ‘User Agreement’ box is ticked. On the ‘Submit’ button, the form will not be submitted unless either the client dropdown menu control is not empty and the manual client control is empty or vice versa. If the conditions are met, clicking the ‘Submit’ button will submit the form and add an item to the SharePoint list ‘Test Group Members’.


Rules

Button: Submit

ButtonShow

ButtonRule1

New

Details for: ButtonRule1

Condition:

Client is blank and

ManualClient is not blank or

Client is not blank and

ManualClient is blank

Rule type: Action

Run these actions:* Add

Submit using a data connection: Main...

Close this form: No Prompt

Rules

Button: Submit

ButtonShow

ButtonRule1

New

Details for: ButtonShow

Condition:*

UserAgreementBox = FALSE

Rule type: Formatting

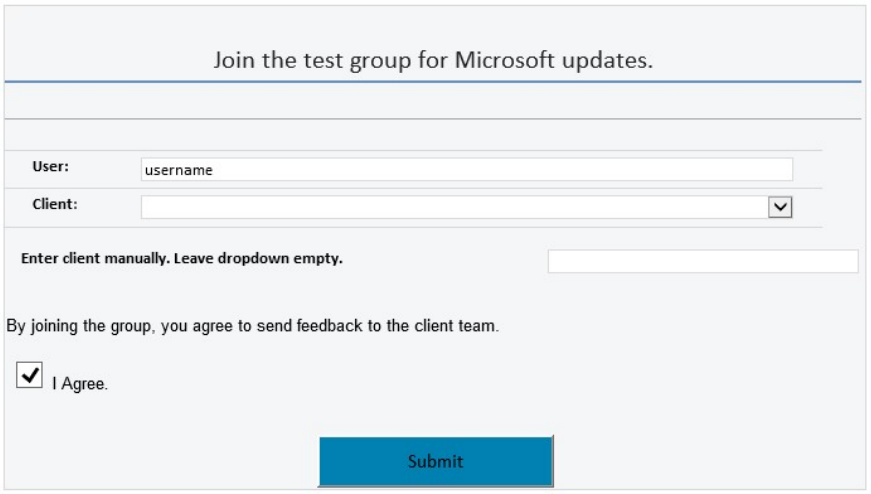
Formatting:*

(hidden)

☑ Hide this control

☑ Disable this control

Figure 15:Form rules.


Join the test group for Microsoft updates.

User:	username
Client:	

Enter client manually. Leave dropdown empty.

By joining the group, you agree to send feedback to the client team.

☑ I Agree.

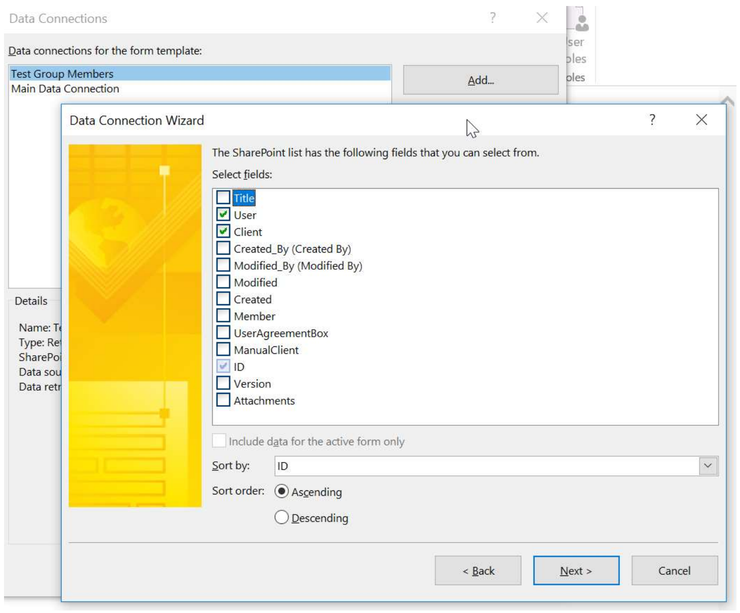
Submit

Figure 16:The join form as shown to end users.

4.3 User Report Form

The user report form is based on a SharePoint list “Report DB” which contains data for user, client, attachment, description, status and date for each entry. The attachment column data type allows for files to be attached to the list, so users can attach a screenshot of a problem situation. The description column data type is set multiple lines of text, so users can write as much as is needed for the description of the problem. The date contains the date of the item submission and will be used later to monitor the status of new updates.

The only external data connection made in this form was to the original ‘Test Group Members’ list, to retrieve the client name of the user who is submitting the report.



Data Connections

Data connections for the form template:

Test Group Members
Main Data Connection

Data Connection Wizard

The SharePoint list has the following fields that you can select from.

Select fields:

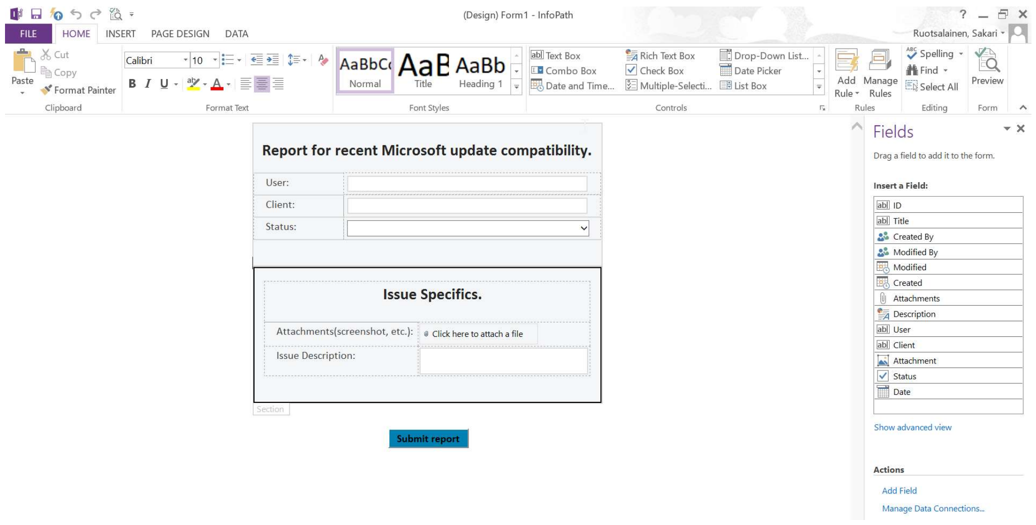
- [ ] Title
- [✓] User
- [✓] Client
- [ ] Created_By (Created By)
- [ ] Modified_By (Modified By)
- [ ] Modified
- [ ] Created
- [ ] Member
- [ ] UserAgreementBox
- [ ] ManualClient
- [✓] ID
- [ ] Version
- [ ] Attachments

[ ] Include data for the active form only

Sort by: ID

Sort order: [✓ Ascending] Descending

Figure 17:Data connection to 'Test Group Members'.



Report for recent Microsoft update compatibility.

User:
Client:
Status:

Issue Specifics.

Attachments(screenshot, etc.): Click here to attach a file
Issue Description:

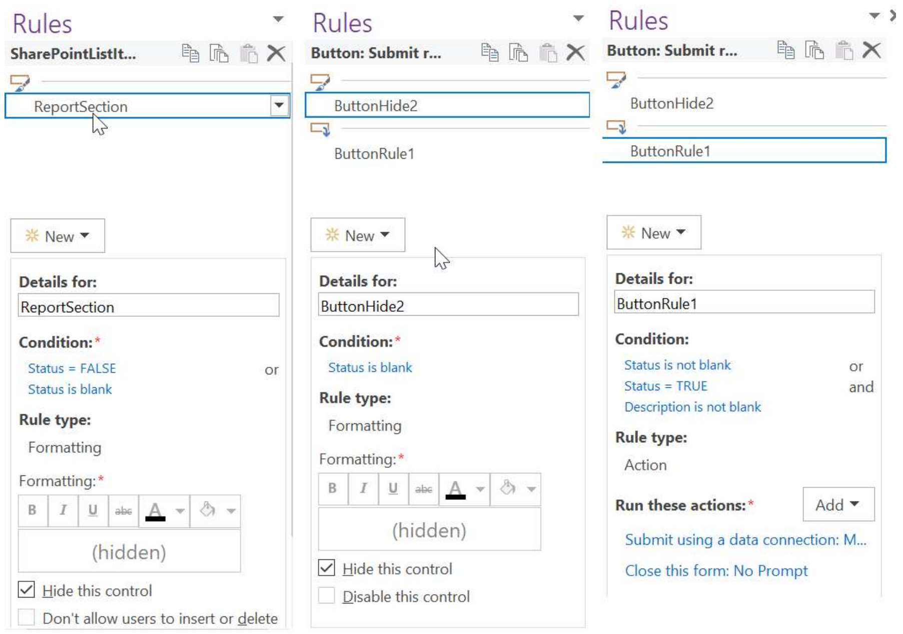
Section

Submit report

Figure 18:The user report form in InfoPath.

The report form is divided into two sections. The upper section contains the username of the user who has opened the form using the same InfoPath function as in the join form, the client name associated with the username from the 'Test Group Members' list and a dropdown menu where the user specifies the state of the recent updates (On the form: 'Issues'/'No issues', on the list: 'Yes'/'No', No means no issues). The lower section contains the controls with which the user can specify the issues affecting their computer. The attachment control will allow the user to attach a file (such as a screenshot of the problem) and the description control will allow for text description of the issue to be submitted.

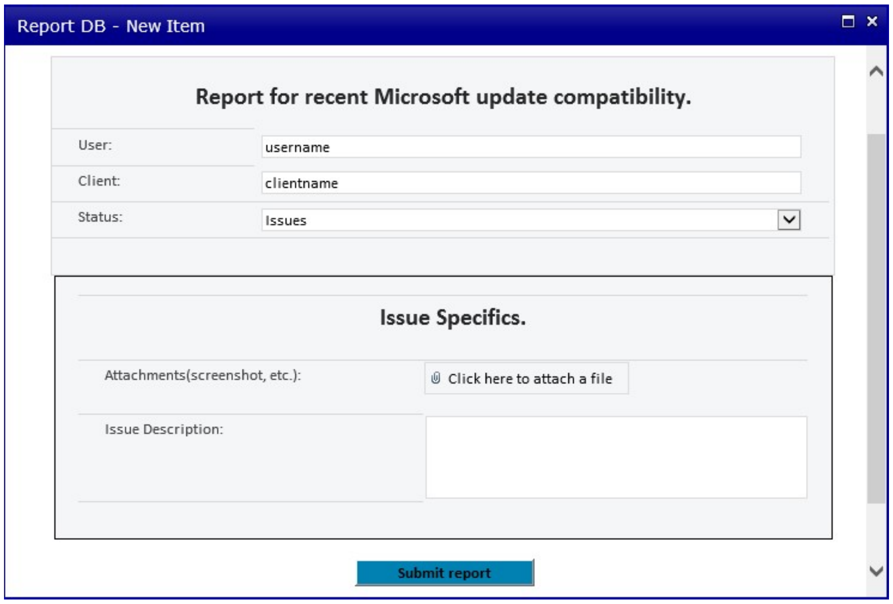
The rules implemented on this form specify that if the status dropdown is empty, the submit button will be hidden. If the status dropdown is selected as 'Issues', the lower section and the Submit button will become visible, and if status is selected as 'No issues', only the 'Submit' button will become visible. The submit button will submit the form fields to the 'Report DB' list and close the form when clicked.


Rules
SharePointListIt...
ReportSection
New
Details for: ReportSection
Condition:*
Status = FALSE or
Status is blank
Rule type: Formatting
Formatting:*
(hidden)
☑ Hide this control
☐ Don't allow users to insert or delete

Rules
Button: Submit r...
ButtonHide2
ButtonRule1
New
Details for: ButtonHide2
Condition:*
Status is blank
Rule type: Formatting
Formatting:*
(hidden)
☑ Hide this control
☐ Disable this control

Rules
Button: Submit r...
ButtonHide2
ButtonRule1
New
Details for: ButtonRule1
Condition:
Status is not blank or
Status = TRUE and
Description is not blank
Rule type: Action
Run these actions:* Add
Submit using a data connection: M...
Close this form: No Prompt

Figure 19:Report form rules.


Report DB - New Item

Report for recent Microsoft update compatibility.

User:	username
Client:	clientname
Status:	Issues

Issue Specifics.

Attachments(screenshot, etc.):	Click here to attach a file
Issue Description:	

Submit report

Figure 20:Reporting form as shown to the end user.

4.4 Admin Application

The application created here is used to manage the SharePoint list “Test Group Members” and obtain data from the reporting list “Report DB”.

4.4.1 Application User Interface

The GUI for the application was created using Visual Studio. Creating a new project and selecting .NET Framework based WPF application is somewhat simple.

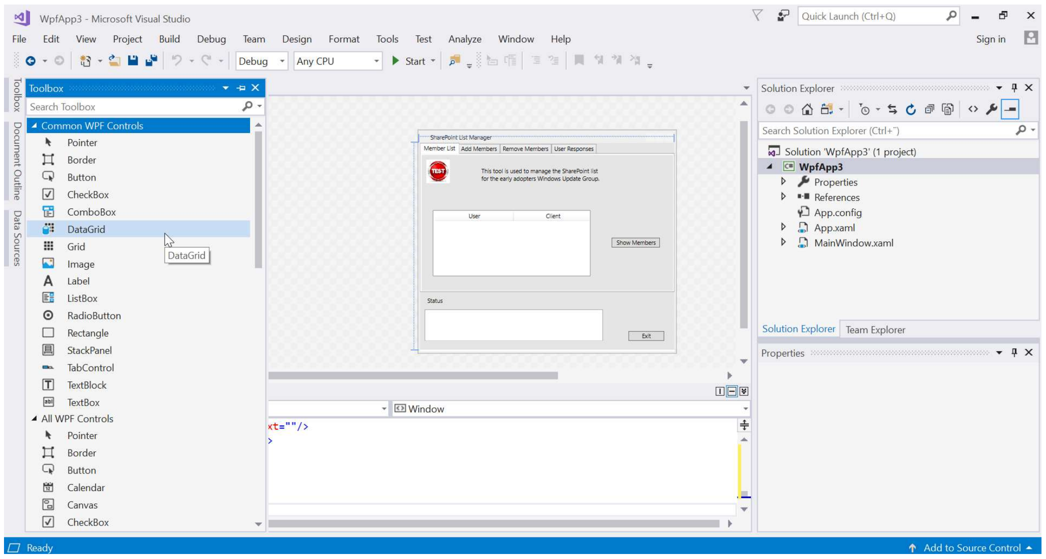


Figure 21:Application UI in Visual Studio.

All the controls that are needed can be added from the toolbox to the application. Controls that are used in this application include TabControl, TextBox, RichTextBox, Button, RadioButton, Label, TextBlock and Image. After the UI design is complete, the XAML code can be copied from Visual Studio to PowerShell. The XAML is saved as a here-string which allows for multiple lines of text and saves the formatting, so the code is properly indented and more readable.

Some of the XAML code must be modified so PowerShell can read it properly and load the XAML objects.

```
$inputXML = $inputXML.replace('mc:Ignorable="d"','').re-
place("x:N",'N').replace('^<Win.*','<Window')
```

Here the XAML is put into an xml variable and then read using the [Windows.Markup.XamlReader] class and its load method.

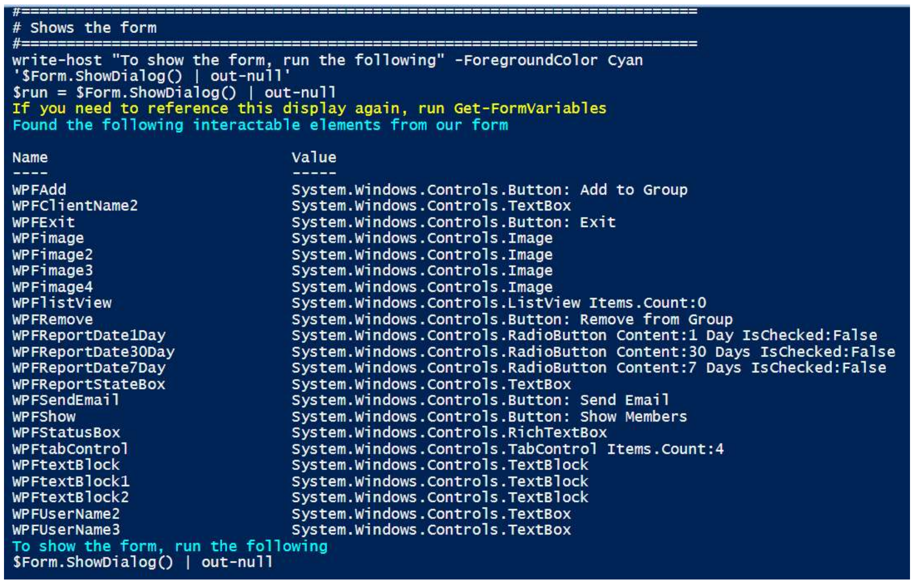
```
[xml]$XAML = $inputXML
$reader=(New-Object System.Xml.XmlNodeReader $XAML)
$Form=[Windows.Markup.XamlReader]::Load($reader)
```

The following converts the objects in the XAML code to interactable variables in PowerShell (FoxDeploy, 2015).

```
$XAML.SelectNodes("//*[@Name]") | % {Set-Variable -Name
"WPF$($_.Name)" -Value $Form.FindName($_.Name)}

Function Get-FormVariables{
if ($global:ReadmeDisplay -ne $true){Write-host "If you need to ref-
erence this display again, run Get-FormVariables" -ForegroundColor
Yellow;$global:ReadmeDisplay=$true}
write-host "Found the following interactable elements from our form"
-ForegroundColor Cyan
get-variable WPF*
}
Get-FormVariables
```

The function returns all objects in the XAML as variables that can then be used in the application. An added 'WPF' prefix is added to the objects. As shown in Figure 23, $Form.ShowDialog() | out-null will run the form.


```
#===========================================================================
# Shows the form
#===========================================================================
write-host "To show the form, run the following" -ForegroundColor Cyan
'$Form.ShowDialog() | out-null'
$run = $Form.ShowDialog() | out-null
If you need to reference this display again, run Get-FormVariables
Found the following interactable elements from our form

Name                           Value
----                           -----
WPFAdd                         System.Windows.Controls.Button: Add to Group
WPFClientName2                 System.Windows.Controls.TextBox
WPFExit                        System.Windows.Controls.Button: Exit
WPFimage                       System.Windows.Controls.Image
WPFimage2                      System.Windows.Controls.Image
WPFimage3                      System.Windows.Controls.Image
WPFimage4                      System.Windows.Controls.Image
WPFlistView                    System.Windows.Controls.ListView Items.Count:0
WPFRemove                      System.Windows.Controls.Button: Remove from Group
WPFReportDate1Day              System.Windows.Controls.RadioButton Content:1 Day IsChecked:False
WPFReportDate30Day             System.Windows.Controls.RadioButton Content:30 Days IsChecked:False
WPFReportDate7Day              System.Windows.Controls.RadioButton Content:7 Days IsChecked:False
WPFReportStateBox              System.Windows.Controls.TextBox
WPFSendEmail                   System.Windows.Controls.Button: Send Email
WPFShow                        System.Windows.Controls.Button: Show Members
WPFStatusBox                   System.Windows.Controls.RichTextBox
WPFtabControl                  System.Windows.Controls.TabControl Items.Count:4
WPFtextBlock                   System.Windows.Controls.TextBlock
WPFtextBlock1                  System.Windows.Controls.TextBlock
WPFtextBlock2                  System.Windows.Controls.TextBlock
WPFUserName2                   System.Windows.Controls.TextBox
WPFUserName3                   System.Windows.Controls.TextBox
To show the form, run the following
$Form.ShowDialog() | out-null
```

Figure 22: Variables in the WPF application.

After the objects are made into variables, PowerShell code can be added to enable the required functionality. Version 5.1 of PowerShell was used in the construction of this application.

4.4.2 The Code Behind

The application interactions with the SharePoint lists are based on New-WebServiceProxy cmdlet from the Microsoft.PowerShell.Management module. This cmdlet creates a web service proxy object that enables the use and management of a web service (Microsoft PowerShell Team, 2010).

```
# The path of the service description, e.g. the .asmx page of the
site
$URI = "http://test.test.com/si-
tes/sakke/Lists/testi/_vti_bin/Lists.asmx?WSDL"
```

If the web service used was created using ASP.NET, “?WSDL” must be appended to the end of the URL of the web service (Microsoft, 2017c).

```
#Creates the service
$Service = New-WebServiceProxy -Uri $URI  -Namespace SpWs  -UseDe-
faultCredential

#The name of the SharePoint list
$List = "Test Group Members"

#Create XML query to retrieve the contents of the list.
$XmlDoc = new-object System.Xml.XmlDocument
$Query = $XmlDoc.CreateElement("Query")
$ViewFields = $XmlDoc.CreateElement("ViewFields")
$QueryOptions = $XmlDoc.CreateElement("QueryOptions")
$Query.set_InnerXml("FieldRef Name='Full Name'")
$RowLimit = "10"


try{
    $List = $Service.GetListItems($List, "", $Query, $ViewFields,
$RowLimit, $QueryOptions, "")
}
catch{
    Write-Error $_ -ErrorAction SilentlyContinue
}
```

The $List variable contains all information from the queried list and can be viewed by accessing $List.data.row where the list items can be found. The next example would print all items where the username is Sakari.

```
$List.data.row | ? {$_.ows_User -eq 'Sakari'}
```

Updating list items or creating/deleting them requires that that account used has the necessary permissions for the list on SharePoint (Microsoft PowerShell Team, 2010). In this part, permissions to modify the lists in question were only set for the author and people testing the application.

In the $List variable, the name attribute values or GUIDs for the list and the view can be found and set in other variables.

```
$ndlistview = $Service.getlistandview($List, "")
$strlistid = $ndlistview.childnodes.item(0).name
$strviewid = $ndlistview.childnodes.item(1).name
```

These are necessary when modifying SharePoint lists. In order to modify the content of a list, an xmldocument object and an associated batch element must be created.

```
# Create the xmldocument object
$xmldoc = new-object system.xml.xmldocument

#The batch element. An empty viewname parameter causes the method to
use the default view.
$batchelement = $xmldoc.createelement("Batch")
$batchelement.setattribute("Onerror", "Continue")
$batchelement.setattribute("Listversion", "1")
$batchelement.setattribute("Viewname", $strviewid)

#Create the xml content for the batch element.
$xml = ""
$xml += "<Method ID='1' Cmd='Update'>" +
        "<Field Name='ID'>$item</Field>" +
        "<Field Name='Member'>false</Field>" +
        "</Method>"
```

The ID of the item that is updated can be retrieved using the previous methods of obtaining list content and locating the ID of the desired item. For example, the ID of all items where the username is ‘sakari’ can be retrieved using the following code.

```
$UserData = $List.data.row  | ? {$_.ows_User -eq 'sakari'}
[array]$IDList = $UserData | % {$_.ows_ID}
```

The returned array would contain all the ID of the items that have ‘sakari’ as the username.

After all necessary information is gathered, the xml content can be added to the batch element.

```
#Add the xml content to the batch element.
$batchelement.innerxml = $xml
```

Now the list can be updated using the updatelistitems method.

```
$ndreturn = $Service.UpdateListItems($List, $batchelement)
```

The New-WebServiceProxy is specific about case sensitivity, especially in the batch element. Modification of the lists failed numerous times before succeeding.

Deleting an item from a list:

```
#The ID must be known and Cmd in the xml must be set to delete.
$xml = ""
$xml += "<Method ID='$id' Cmd='Delete'>" +
        "<Field Name='ID'>$RowID</Field>" +
        "</Method>"
```

Creating a new item works almost in the same manner. The ‘Cmd’ in the xml is specified as New and no ID number for the item is needed as it is created automatically for new items. The desired content for the item is specified in the xml.

```
#New item xml content.
$xml = ""
$xml += "<Method ID='1' Cmd='New'>" +
        "<Field Name='User'>$User</Field>" +
        "<Field Name='Client'>$Client</Field>" +
        "<Field Name='Member'>true</Field>" +
        "</Method>"
```

4.4.3 Processing script

The process is run by a PowerShell script that runs as a scheduled task on a server. The script goes through the items in the ‘Test Group Members’ SharePoint list and adds computers to the Active Directory group that are not members of the group already.

The lists ‘member’ property determines the membership to the AD group and is set to ‘true’ when a new item is added. The script skips an item and continues to the next one if the computer is already present in the AD group. If the computer is not yet a member, it is added to the group and an email is sent to the user the computer is bound to on the list. The user email is retrieved from AD.

An administrator can set the member property to false using the application (on the application it is stated as ‘remove from group’). When the property is set to ‘false’, the script first removes the computer from the AD group and then deletes the item from the SharePoint list.

The processing script code can be found in Appendix 2.

4.4.4 Functions for the Application Controls

All the controls are now defined as variables in the PowerShell/WPF application and functionality can be added to them. There is much more that can be done regarding this portion, but as a Proof of Concept project, the main functionality was the priority.

Code and conditions are applied to six controls:

- Button that retrieves the group members and displays them on a list view ($WPFShow).
- Button that adds user/client to the group ($WPFAdd).
- Button that removes user/client to the group ($WPFRemove).
- Exit button ($WPFExit).
- Button that sends the report form email to all members ($WPFSendEmail).
- Reporting data radio buttons ($WPFReportDate(1|7|30) Day) and Show button ($WPFShowStats).
- Clear the statusbox button ($WPFClear).

Using the Add_Click method, it is possible insert a script block into the () overload (FoxDeploy, 2015). Using this, the PowerShell code that interacts with the SharePoint lists can be added to the buttons and other controls.

```
#List Users Button
$WPFShow.Add_Click({$ListContent = Get-SPList -Listname "Test Group
Members"
[array]$ListInfo = @()
$ListContent.data.row | % {$TempObject = New-Object psobject -Prop-
erty @{'User' = $_.ows_User;'Client' = $_.ows_Client}; $ListInfo +=
$TempObject}
$WPFlistView.Items.Clear()
$ListInfo | % {$WPFlistView.AddChild($_)}})
```

By clicking the button, the list of information is retrieved using the custom Get-SPList function that returns the list data as a variable. From the variable, user/client objects are created and shown on the list view ($WPFListView). In order to refresh the information, the list view content is cleared with each button click.

```
#Add User Button
if($WPFUserName2.Text -ne $Null -and $WPFClientName2.Text -ne $Null){
   $WPFAdd.Add_Click({

           try{Add-ToSPList -User $WPFUserName2.Text -Client $WPFCli-
           entName2.Text
           Write-ToTextBox -RichTextBox $WPFStatusBox -Text "User
$($WPFUserName2.Text) with client $($WPFClientName2.Text) has been
added to the group." -TextColor Green}

           catch {Write-ToTextBox -RichTextBox $WPFStatusBox -Text
"An error occurred!" -TextColor Red}
$WPFUserName2.clear();$WPFClientName2.clear()})
}
```

The add user button is set to only function if the controls with the username and client name are not empty. If content is present, a click of the button attempts to add a new item to the SharePoint members list using the custom Add-ToSPList function which takes the username and clientname from the associated controls as input. If successful, a new item is added, and a message confirming this is printed on the statusbox. If an error occurs, a message "An error occurred!" is printed. Additional error messages with more information would be added in a production version.

```
#Remove User Button
if($WPFUserName3.Text -ne $Null){
           $WPFRemove.Add_Click({Write-ToTextBox -RichTextBox $WPF-
           StatusBox -Text "Removing $($WPFUserName3.Text) from the
           test group." -TextColor Green

    try{Set-SPListMember -User $($WPFUserName3.Text)
           Write-ToTextBox -RichTextBox $WPFStatusBox -Text "User
           $($WPFUserName3.Text) has been removed from the group." -
           TextColor Green
                      }

     catch {Write-ToTextBox -RichTextBox $WPFStatusBox -Text "An er
     ror occurred!" -TextColor Red}
                      $WPFUserName3.Clear()})
}
```

The ‘remove user’ button functions in the same way as the ‘add user’ one, except that it uses the Set-SPListMember function that changes the previously mentioned ‘member’ property to ‘false’, after which the control script removes the list item and the client from the AD Group.

```
#Exit button
$WPFExit.Add_Click({$Form.Close()})
```

The exit button closes the application.

```
#Send Email with a link to the reporting form.
$WPFSendEmail.Add_Click({Write-ToTextBox -RichTextBox $WPFStatusBox -
Text "Sending Email to Test Group Members." -TextColor Green

           try{Send-ReportEmail;Write-ToTextBox -RichTextBox $WPFSta-
           tusBox -Text "Emails successfully sent." -TextColor Green}

           catch{Write-ToTextBox -RichTextBox $WPFStatusBox -Text "An
           error occurred!" -TextColor Red}
})
```

The ‘Send email’ button is set to send emails with a link to the reporting form to all user in the ‘Test Group Members’ SharePoint list using custom Send-ReportMail function that uses another custom function New-ReportEmail that in turn does a singular email, whereas Send-ReportMail uses New-ReportEmail to loop over the members list.

```
#Clear textbox
$WPFClear.Add_Click({Clear-TextBox -RichTextBox $WPFStatusBox})
```

The clear button clears the contents of the statusbox.

```
#Show report stats for 1,7 and 30 day windows.
$WPFShowStats.Add_Click({
    Clear-TextBox -RichTextBox $WPFStatsBox

    $Stats = Get-ReportDBStats
    $1D = $Stats[0]
    $7D = $Stats[1]
    $30D = $Stats[2]

    $1DMessage = @"
    Statistics for 1 Day period:
    -----------------------------
    Number of reports: $($1D.Count)
    Issues reported: $($1D.Issue)
    No Problems for: $($1D.Percentage) %
"@
    $7DMessage = @"
    Statistics for 7 Day period:
    -----------------------------
    Number of reports: $($7D.Count)
    Issues reported: $($7D.Issue)
    No Problems for: $($7D.Percentage) %
"@
    $30DMessage = @"
    Statistics for 30 Day period:
    ------------------------------
    Number of reports: $($30D.Count)
    Issues reported: $($30D.Issue)
    No Problems for: $($30D.Percentage) %
"@

if($WPFReportDate1Day.IsChecked){Write-ToTextBox -RichTextBox $WPF-
StatsBox -text $1DMessage -textcolor Black}
elseif($WPFReportDate7Day.IsChecked){Write-ToTextBox -RichTextBox
$WPFStatsBox -text $7DMessage -textcolor Black}
elseif($WPFReportDate30Day.IsChecked){Write-ToTextBox -RichTextBox
$WPFStatsBox -text $30DMessage -textcolor Black}

})
```

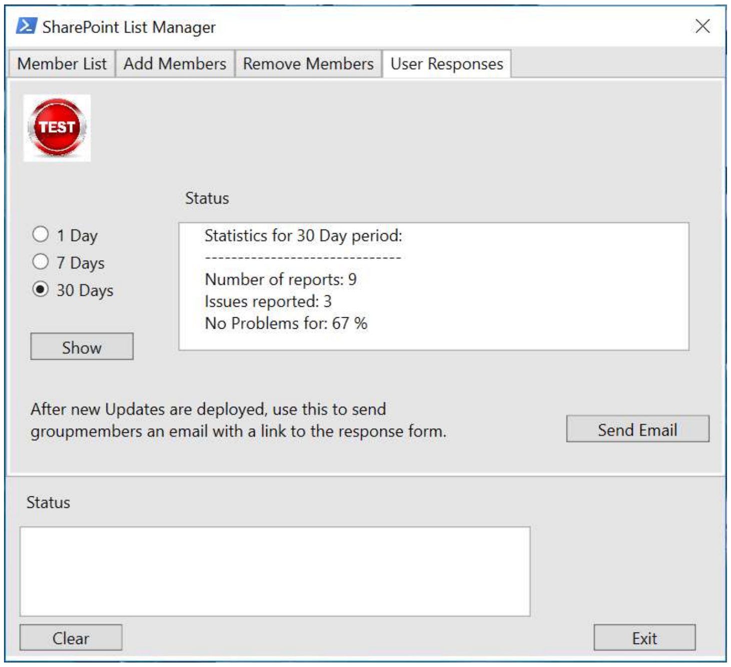
The button ‘Show’ in the User Responses tab retrieves information using the Get-ReportDBStats function that returns three items containing the relevant information on each timespan (1 day, 7 days and 30 days). The timespans are chosen as an example and can be altered easily in the function. When using the Write-ToTextBox function, a string containing the information is printed on the statistics richtextbox.

The application is contained in the .ps1 file but using a shortcut and specifying a ’-WindowStyle Hidden’ parameter in the target, the application is shown without the background PowerShell window while PowerShell is still executing the application.

This is an example shortcut target field:
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -NoProfile -WindowStyle Hidden -file "C:\MyScripts\WpfAppendix.ps1"

The functions Set-SPListMember and Add-ToSPList contain regular expression pattern validation. Set-SPListMember only accepts combinations of 8-word characters for its $User parameter and Add-ToSPList $Client parameter must begin with a A, B or C followed by 6 numbers due to the company naming convention in place.

All the functions can be found in the application code in Appendix 3.



Statistics for 30 Day period:

Number of reports: 9
Issues reported: 3
No Problems for: 67 %

After new Updates are deployed, use this to send groupmembers an email with a link to the response form.

Figure 23:Management application screenshot showing statistics.

5 FURTHER DEVELOPMENT

The process created in this thesis can be equated to an alpha version as there are many aspects that can be improved.

First, while SharePoint lists work well, they are not an optimal database for a larger implementation with thousands of reports flowing into a constantly growing database. A SQL database is much more suited towards this. InfoPath can easily work with and update a SQL database, so the functionality remains unchanged.

Second, there has been some discussion in the company of whether the self-signup for the end users is needed. IT managers would manage the entire operation and the membership of the testing group. Arguably, if there is a tool or a process that can keep track of the user devices and their owners, the option for the users to join the group voluntarily is very positive. This would further improve user commitment to the process and improve its accuracy for detecting issues.

Another platform could be used to build the required components used in this thesis study. For example, K2 Blackpearl is a process automation platform that would dramatically decrease the parts needed for the same functionality as achieved in this thesis study. K2 would eliminate the need for InfoPath, SharePoint (although an external database is still needed) and some of the scripting as it can tie directly into services such as SQL, AD and Azure AD. The forms created with K2 would have all the functionality built in them as the services would be tied together by K2. Originally the process was supposed to be created using K2, but after it was decided that the thesis was to be completed as a Proof of Concept, InfoPath/SharePoint was the easier choice.

These are aspects of the process that can be improved, but if the process is developed further in the future, the entire technical structure would likely change in some way.

6 CONCLUSIONS

The beginning of this thesis study was not easy as the scope of the implementation and my role in the development was constantly changing. Originally, the process presented in this thesis was supposed to be implemented into use at the company, but due to various reasons, it was decided that the reporting process section of this thesis would be done as a Proof of Concept.

Update management is extremely important in any enterprise, large or small, as new updates can wreak havoc on a range of vital business applications. While the delivery of the updates is easier to monitor, their effects are not. New updates deployed to production environments can cause problems as they interact with different applications in different. In many environments, updates are deployed to testing environments first but lack the user feedback and rely on tickets submitted to helpdesk. This method is functional but lacks the real-time feedback from the testing environment. Also, the connection between problem and cause can be more difficult to ascertain when tickets for problems regarding testing of updates are mixed with other tickets. Using a separate system for testing environment data (users, devices, feedback), the problems can be identified much faster. This, of course, relies on active user participation which should be made as simple as possible. Ideally, clicking a link in an email for an ‘all clear’ signal would suffice on the part of the user.

The study of the inner layers of System Center Configuration Manager was also quite enlightening. While the operation and finer details of the many aspects of ConfigMgr are still unclear to me, I feel that I have a good grasp on the updating process in ConfigMgr. The process description in the first part of the thesis covers the subject well and gives some insight into the update process when operating with ConfigMgr.

Different platforms used in this thesis such as InfoPath, SharePoint and PowerShell also required quite some research. Originally, the platform discussed in the creation of the end user reporting functionality was K2, which I studied in quite some detail but unfortunately was unable to use. Using K2 could have eliminated some aspects of the project as it has extensive integration to services, such as Active Directory. InfoPath forms took some time to study and create but were eventually quite simple. More complex creations are certainly possible with InfoPath. They simply were not required in this thesis, as simple database connections, forms and logic were sufficient. With SharePoint there was much less to study as the lists used were simple and many of the features in SharePoint were not relevant to this thesis. PowerShell scripting has become something I have tried to constantly develop in, starting last summer when I was working in the local IT support at the company. It has proven to be quite useful as understanding a little coding is a great help in in many instances. The application to control the content of the SharePoint lists is done entirely in PowerShell and while it would have been possible to implement the management functionality on a SharePoint/InfoPath form or make it HTML-based, it was extremely useful as a further learning experience on PowerShell scripting to construct the application. Also, it is easier to include many different functionalities into an application instead of a form.

It is recommended that a process such as this would be in place in any large environment that relies on the functionality of various applications, but at the same time systems must be constantly kept up to date with the newest patches. In order to gather a reliable sample data each month, a significant portion of the client base should be involved a process like this. A number somewhere between 5-10% has been recommended by experts inside the company.

Finally, the task to include the end user to the update process is functionally sound and could be implemented into any environment. Of course, in its current form the process is unrefined and should be examined more closely before implementation. Also, as mentioned earlier, better tools exist for this task such as K2 and should be considered instead of the tools used in this thesis. For the commissioner company it is recommended that the process be implemented using K2 as a platform as its integration with other services far exceeds the capabilities of InfoPath.

Creating this process in this thesis study was very interesting and quite the learning experience on enterprise environment mechanics and development tools. In the future, this process is hopefully implemented in some form to the company infrastructure, adding a layer of protection to the updates process.

REFERENCES

Chappell, B. 2017. WannaCry Ransomware: What We Know Monday. [Online]
Available at: https://www.npr.org/sections/thetwo-way/2017/05/15/528451534/wannacry-ransomware-what-we-know-monday
[Accessed 6 February 2018].

Fors, M. 2017. Deploy and Troubleshoot Office 365 ProPlus Updates with ConfigMgr. [Online]
Available at: https://deploywindows.info/2017/03/03/deploy-and-troubleshoot-office-365-proplus-updates-with-configmgr-12/
[Accessed 14 December 2017].

FoxDeploy. 2015. Deploying PowerShell GUIs in Minutes using Visual Studio. [Online]
Available at: https://foxdeploy.com/2015/04/16/part-ii-deploying-powershell-guis-in-minutes-using-visual-studio/
[Accessed 7 February 2018].

Jupe, H. 2015. SCCM Collections – The basics. [Online]
Available at: http://www.hayesjupe.com/sccm-collections-the-basics/
[Accessed 29 November 2017].

Keränen, J. 2014. Centralized IT management using SCCM in a large multina-tional company.
Available at: https://www.doria.fi/bitstream/handle/10024/101925/MasterThe-sis_Keranen_Jesse.pdf?sequence=2
[Accessed 7 February 2018].

Microsoft. 2018a. Overview of update channels for Office 365 ProPlus. [Online]
Available at: https://docs.microsoft.com/en-us/deployoffice/overview-of-up-date-channels-for-office-365-proplus
[Accessed 7 February 2018].

Microsoft. 2018b. What is XAML. [Online]
Available at: https://msdn.microsoft.com/en-us/library/cc295302.aspx
[Accessed 7 February 2018].

Microsoft. 2017a. Install and configure a software update point. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/sum/get-started/install-a-software-update-point
[Accessed 24 November 2017].

Microsoft. 2017b. Introduction to software updates in System Center Configu-ration Manager. [Online]

Available at: https://docs.microsoft.com/en-us/sccm/sum/understand/software-updates-introduction
[Accessed 7 February 2018].

Microsoft. 2017c. New-WebServiceProxy. [Online]
Available at: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/new-webserviceproxy?view=powershell-5.1
[Accessed 8 February 2018].

Microsoft. 2016a. Introduction to System Center Configuration Manager. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/core/understand/introduction
[Accessed 20 October 2017].

Microsoft. 2016b. Fundamentals of sites and hierarchies for System Center Configuration Manager. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/core/understand/fundamentals-of-sites-and-hierarchies
[Accessed 23 November 2017].

Microsoft. 2016c. Prerequisites for software updates in System Center Configuration Manager. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/sum/plan-design/prerequisites-for-software-updates
[Accessed 27 November 2017].

Microsoft. 2016d. Prerequisites for Software Updates in Configuration Manager. [Online]
Available at: https://technet.microsoft.com/fi-fi/library/hh237372.aspx
[Accessed 29 November 2017].

Microsoft. 2016e. Manually deploy software updates. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/sum/deploy-use/manually-deploy-software-updates
[Accessed 29 November 2017].

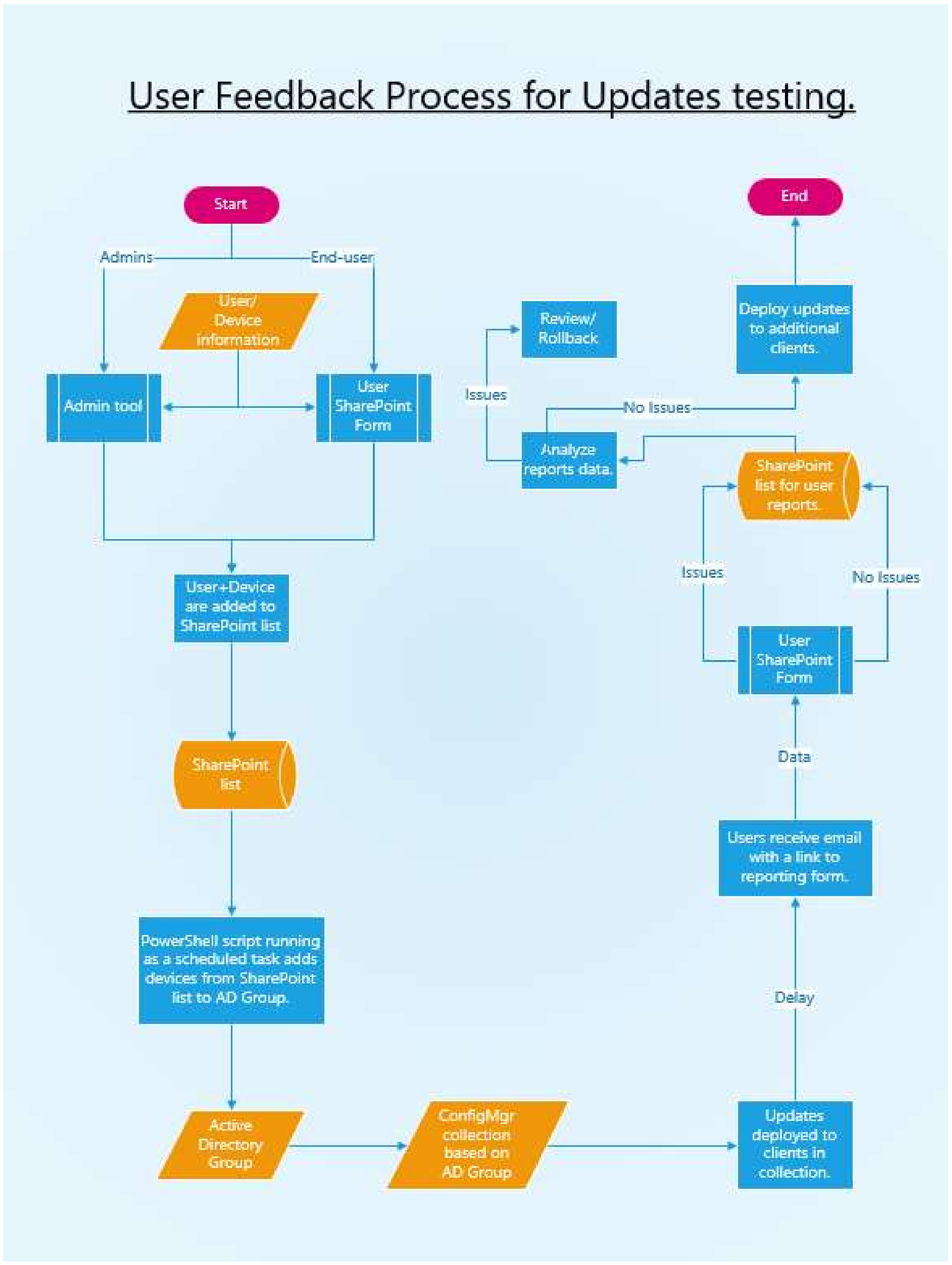
Microsoft. 2016f. Deploy software updates. [Online]
Available at: https://docs.microsoft.com/en-us/sccm/sum/deploy-use/deploy-software-updates
[Accessed 4 December 2017].

Microsoft. 2015. Configuring Software Updates in Configuration Manager. [Online]
Available at: https://technet.microsoft.com/en-us/library/gg712312.aspx
[Accessed 27 November 2017].

Microsoft PowerShell Team. 2010. Using New-WebServiceProxy to get, modify, and add items to a list in SharePoint 2007. [Online]
Available at: https://blogs.msdn.microsoft.com/powershell/2010/06/24/using-new-webserviceproxy-to-get-modify-and-add-items-to-a-list-in-sharepoint-2007/
[Accessed 7 February 2018].

Rimmerman, Shilt, Della Monica, Faldu. 2015. Microsoft System Center Software Update Management Field Experience.
Available at: https://mva.microsoft.com/ebooks#9780735695849

Smpyrakis, G. 2012. ConfigMgr 2012 Automatic Deployment Rules. [Online]
Available at: https://blogs.technet.microsoft.com/configmgrdogs/2012/05/07/configmgr-2012-automatic-deployment-rules/
[Accessed 20 October 2017].


User Feedback Process for Updates testing.
Start
Admins
End-user
User/ Device information
Admin tool
User SharePoint Form
User+Device are added to SharePoint list
SharePoint list
PowerShell script running as a scheduled task adds devices from SharePoint list to AD Group.
Active Directory Group
ConfigMgr collection based on AD Group
Updates deployed to clients in collection.
Delay
Users receive email with a link to reporting form.
Data
User SharePoint Form
Issues
No Issues
SharePoint list for user reports.
Analyze reports data.
Issues
Review/ Rollback
No Issues
Deploy updates to additional clients.
End

```
#This script manages the membership of an Active Directory group related to Microsoft updates
#testing. It retrieves information from a SharePoint list and adds/removes group members
#accordingly.


#Import active directory module
Import-Module ActiveDirectory

#Function to send email to added groupmembers.
Function New-JoinEmail (){

    Param([parameter(Mandatory=$true)]
          [string]$User,
          [parameter(Mandatory=$true)]
          [string]$Client)

    $SmtpServer = "mail.test.com"
    $EmailFrom = "sakari.ruotsalainen@test.com"
    $EmailTo = (Get-ADUser $User -properties *).mail
    $MailMessage = New-Object System.Net.Mail.MailMessage $EmailFrom, $EmailTo
    $MailMessage.Subject = "Microsoft Updates Test Group Membership."
    $MailMessage.IsBodyHTML = $True
    $MailMessage.Body = "<body style='font-family:arial; font-size: 13px; font-style: nor-
mal'><b>Dear Recipient,</b><p><p>Computer <b>$Client</b>, has been added to the testing group
for Microsoft updates. You will receive a link to report any problems you may encounter as a re-
sult and are expected to report back.<p><p><p>Best Regards,<p> GROUP IT</body>"
    $NewMessage = New-Object Net.Mail.SmtpClient($SmtpServer)
    $NewMessage.Send($MailMessage)

}

$ADTestGroup = "ADTESTGROUP"
$GroupMembers = Get-AdGroupMember -Identity $ADTestGroup | select name
$ListName = "Test Group Members"

#Uses Get-SPList function that returns the data for a specified list.
$SPListData = Get-SPList -Listname $ListName

#Go through the items in the SharePoint List
$SPListData | % {

    $Client = $_.ows_Client
    $Member = $_.ows_Member
    $ManualClient = $_.ows_ManualClient
    $RowID = $_.ows_ID
    $User = $_.ows_User

    #Use ManualClient property if client has been added manually.
    if($ManualClient -ne $null -and $ManualClient.length -eq 8){

            try {   Get-ADComputer $ManualClient

                    #If the computer is already in the AD group, continue to next item.
                    if($GroupMembers -like $ManualClient){
                          Write-Output "Client $ManualClient is already in the group.";continue
                        }

                 #If the computer is not in the AD group, add it and send an email to the user.
                    else {
                          Write-Output "Client $ManualClient is not in the group. Adding to
$ADTestGroup."

                          Add-ADGroupMember $Group -Members $Client
                          New-JoinEmail -User $User -Client $ManualClient

                        }
                }

            catch {Write-Error $_}

            }

    #Else use the $Client property
    else{
            #If the computer is already in the AD group, continue to next item.
            if($GroupMembers -like $Client){
                    Write-Output "Client $Client is already in the group.";continue
                  }
             #If the computer is not in the AD group, add it and send an email to the user.
             else {
                    Write-Output "Client $Client is not in the group. Adding to $ADTestGroup."
                    Add-ADGroupMember $Group -Members $Client
```

```
                        New-JoinEmail -User $User -Client $ManualClient
                    }

                }

        #If Member property has been set to false, remove client from AD Group and SP list.
        #This property is set by administrators using the application.
        if($Member -eq 'false'){
                Remove-ADGroupMember -Identity $Group -Member $Client
                $xml = ""
                $xml +=
                   "<Method ID='$id' Cmd='Delete'>" +
                   "<Field Name='ID'>$RowID</Field>" +
                   "</Method>"

                $ndreturn = $null

                #Modify the batchelement content with the $xml variable
                $batchelement.innerxml = $xml

                #Implement the modifications to the SharePoint list.
                    try {
                        $ndreturn = $Service.UpdateListItems($ListName,$batchelement)
                    }
                    catch {
                        write-error $_ -ErrorAction 'SilentlyContinue'
                    }

               }
   }
```

```
#The Entire code for the application to interact with SharePoint lists used in the thesis.

#The URI of the SharePoint lists has been changed to a generic one.

#XAML content set to a here string.
$inputXML = @"
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApplication1"
        mc:Ignorable="d"
        Title="SharePoint List Manager" ResizeMode="NoResize" Height="485.39" Width="539.918">
    <Grid Margin="0,0,-0.333,3.333" Background="#FFE5E5E5">
        <TabControl x:Name="tabControl" Height="314" Margin="0,0,-0.333,0" VerticalAlign-
ment="Top">
            <TabItem Header="Member List">
                <Grid Background="#FFE5E5E5">
                    <Image x:Name="image" HorizontalAlignment="Left" Height="49" Mar-
gin="10,10,0,0" VerticalAlignment="Top" Width="53" Source="C:\Users\ruotsak\Desktop\icon.jpg"/>
                    <TextBlock x:Name="textBlock" HorizontalAlignment="Left" Margin="125,26,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="260" Height="33"><Run Text="This tool is used
to manage the SharePoint list for the early adopters Windows Update Group"/><Run
Text="."/></TextBlock>
                    <ListView x:Name="listView" HorizontalAlignment="Left" Height="139" Mar-
gin="24,116,0,0" VerticalAlignment="Top" Width="330">
                        <ListView.View>
                            <GridView>
                                <GridViewColumn Header="User" DisplayMemberBinding ="{Binding
'User'}" Width="165"/>
                                <GridViewColumn Header="Client" DisplayMemberBinding ="{Binding
'Client'}" Width="165"/>
                            </GridView>
                        </ListView.View>
                    </ListView>
                    <Button x:Name="Show" Content="Show Members" HorizontalAlignment="Left" Mar-
gin="399,174,0,0" VerticalAlignment="Top" Width="101" RenderTransformOrigin="-0.409,0.777"/>
                </Grid>
            </TabItem>
            <TabItem Header="Add Members" Margin="0">
                <Grid Background="#FFE5E5E5" >
                    <Image x:Name="image2" HorizontalAlignment="Left" Height="49" Mar-
gin="10,10,0,0" VerticalAlignment="Top" Width="53" Source="C:\Users\ruotsak\Desktop\icon.jpg"/>
                    <TextBox x:Name="UserName2" HorizontalAlignment="Left" Height="23" Mar-
gin="108,96,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="112"/>
                    <TextBox x:Name="ClientName2" HorizontalAlignment="Left" Height="23" Mar-
gin="108,141,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="111"/>
                    <Button x:Name="Add" Content="Add to Group" HorizontalAlignment="Left" Mar-
gin="381,187,0,0" VerticalAlignment="Top" Width="99" Height="25" RenderTrans-
formOrigin="0.309,0.441"/>
                    <TextBlock x:Name="textBlock1" HorizontalAlignment="Left" Mar-
gin="108,29,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="15" Width="298"><Run
Text="Add "/><Run Text="a "/><Run Text="member to the "/><Run Text="Windows Update test
group."/></TextBlock>
                    <Label Content="Username" HorizontalAlignment="Left" Margin="28,93,0,0" Ver-
ticalAlignment="Top" RenderTransformOrigin="-0.906,-0.157" FontWeight="Bold"/>
                    <Label Content="Clientname" HorizontalAlignment="Left" Margin="28,138,0,0"
VerticalAlignment="Top" FontWeight="Bold"/>
                </Grid>
            </TabItem>
            <TabItem Header="Remove Members" Margin="0">
                <Grid Background="#FFE5E5E5">
                    <Image x:Name="image3" HorizontalAlignment="Left" Height="49" Mar-
gin="10,10,0,0" VerticalAlignment="Top" Width="53" Source="C:\Users\ruotsak\Desktop\icon.jpg"/>
                    <TextBox x:Name="UserName3" HorizontalAlignment="Left" Height="23" Mar-
gin="94,129,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="120"/>
                    <TextBlock x:Name="textBlock2" HorizontalAlignment="Left" Margin="94,19,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Height="40" Width="228"><Run Text="Input the
username you wish to remove from the "/><Run Text="Windows Update tes"/><Run Text="t "/><Run
Text="group"/><Run Text="."/></TextBlock>
                    <Button x:Name="Remove" Content="Remove from Group" HorizontalAlign-
ment="Left" Margin="350,129,0,0" VerticalAlignment="Top" Width="123" Height="23" RenderTrans-
formOrigin="0.158,0.46"/>
                    <Label Content="Username" HorizontalAlignment="Left" Margin="25,126,0,0"
VerticalAlignment="Top" FontWeight="Bold"/>
                </Grid>
            </TabItem>
            <TabItem Header="User Responses" Margin="0">
                <Grid Background="#FFE5E5E5">
```

```
                    <Image x:Name="image4" HorizontalAlignment="Left" Height="49" Mar-
gin="10,10,0,0" VerticalAlignment="Top" Width="53" Source="C:\Users\ruotsak\Desktop\icon.jpg"/>
                    <Button x:Name="SendEmail" Content="Send Email" HorizontalAlignment="Left"
Margin="406,244,0,0" VerticalAlignment="Top" Width="105"/>
                    <TextBlock HorizontalAlignment="Left" Margin="15,231,0,0" TextWrap-
ping="Wrap" Text="After new Updates are deployed, use this to send groupmembers an email with a
link to the response form." VerticalAlignment="Top" Height="33" Width="311"/>
                    <RadioButton x:Name="ReportDate1Day" Content="1 Day" GroupName="ReportDate"
IsChecked="True" HorizontalAlignment="Left" Margin="15,105,0,0" VerticalAlignment="Top"/>
                    <RadioButton x:Name="ReportDate7Day" Content="7 Days" GroupName="ReportDate"
HorizontalAlignment="Left" Margin="15,125,0,0" VerticalAlignment="Top"/>
                    <RadioButton x:Name="ReportDate30Day" Content="30 Days" GroupName="Re-
portDate" HorizontalAlignment="Left" Margin="15,145,0,0" VerticalAlignment="Top"/>
                    <RichTextBox x:Name="StatsBox" IsReadOnly="True" HorizontalAlignment="Left"
Height="94" Margin="123,103,0,0" VerticalAlignment="Top" Width="373">
                        <FlowDocument>
                            <Paragraph>
                                <Run Text=""/>
                            </Paragraph>
                        </FlowDocument>
                    </RichTextBox>
                    <Label Content="Status" HorizontalAlignment="Left" Margin="123,72,0,0" Ver-
ticalAlignment="Top" RenderTransformOrigin="0.079,0.561"/>
                    <Button x:Name="ShowStats" Content="Show" HorizontalAlignment="Left" Mar-
gin="15,184,0,0" VerticalAlignment="Top" Width="75"/>
                </Grid>
            </TabItem>
        </TabControl>
        <Button x:Name="Exit" Content="Exit" HorizontalAlignment="Left" Margin="429,421,0,0"
VerticalAlignment="Top" Width="75"/>
        <Label Content="Status" HorizontalAlignment="Left" Margin="10,319,0,0" VerticalAlign-
ment="Top"/>
        <RichTextBox x:Name="StatusBox" IsReadOnly="True" Grid.Column="1" HorizontalAlign-
ment="Left" Height="66" Margin="10,350,0,0" VerticalAlignment="Top" Width="373" Grid.Col-
umnSpan="2">
            <FlowDocument>
                <Paragraph>
                    <Run Text=""/>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
        <Button x:Name="Clear" Content="Clear" HorizontalAlignment="Left" Margin="10,421,0,0"
VerticalAlignment="Top" Width="75"/>

    </Grid>
</Window>

"@

$inputXML = $inputXML -replace 'mc:Ignorable="d"','' -replace "x:N",'N'  -replace '^<Win.*',
'<Window'

[void][System.Reflection.Assembly]::LoadWithPartialName('presentationframework')
[xml]$XAML = $inputXML
#Read XAML

$reader=(New-Object System.Xml.XmlNodeReader $xaml)

try{$Form=[Windows.Markup.XamlReader]::Load( $reader )}
catch{Write-Host "Unable to load Windows.Markup.XamlReader. Double-check syntax and ensure .net
is installed."}

#===========================================================================
# Load XAML Objects In PowerShell
#===========================================================================

$xaml.SelectNodes("//*[@Name]") | % {Set-Variable -Name "WPF$($_.Name)" -Value $Form.Find-
Name($_.Name)}

Function Get-FormVariables{
if ($global:ReadmeDisplay -ne $true){Write-host "If you need to reference this display again,
run Get-FormVariables" -ForegroundColor Yellow;$global:ReadmeDisplay=$true}
write-host "Found the following interactable elements from our form" -ForegroundColor Cyan
get-variable WPF*
}
Get-FormVariables

#Function to add new item to the SharePoint list manually.
Function Add-ToSPList(){

    Param([Parameter(Mandatory=$true)]
          [string]$User,
          [Parameter(Mandatory=$true)]
          [ValidatePattern("^[abcABC]{1}\d{6}$")]
```

```
            [string]$Client)

    $Uri = "http://test.test.com/sites/sakke/_vti_bin/Lists.asmx?WSDL"
    $Service = New-WebServiceProxy -Uri $Uri -Namespace SpWs -UseDefaultCredential
    $List = "Test Group Members"

    $ndlistview = $Service.getlistandview($List, "")
    $strlistid = $ndlistview.childnodes.item(0).name
    $strviewid = $ndlistview.childnodes.item(1).name

    #Create xml object for list modification with batchelement attributes.
    $xmldoc = new-object system.xml.xmldocument
    $batchelement = $xmldoc.createelement("Batch")
    $batchelement.setattribute("Onerror", "Continue")
    $batchelement.setattribute("Listversion", "1")
    $batchelement.setattribute("Viewname", $strviewid)

    $xml = ""
    $xml += "<Method ID='1' Cmd='New'>" +
            "<Field Name='User'>$User</Field>" +
            "<Field Name='Client'>$Client</Field>" +
            "<Field Name='Member'>true</Field>" +
            "</Method>"

    #Modify the batchelement content with the $xml variable
    $batchelement.innerxml = $xml

    #Implement the modifications to the SharePoint serviceobject defined at the start.
    try {
        $ndreturn = $service.UpdateListItems($List, $batchelement)
    }
    catch {
        write-error $_
    }

}

#Function to set the member property to false on a SharePoint list item. Used in the management
of the Active Directory group.
Function Set-SPListMember() {

    Param([Parameter(Mandatory=$true)]
            [ValidatePattern("^\w{8}$")]
            [string]$User)

    $Uri = "http://test.test.com/sites/sakke/_vti_bin/Lists.asmx?WSDL"
    $List = "Test Group Members"
    $Service = New-WebServiceProxy -Uri $Uri -Namespace SpWs -UseDefaultCredential

    $ndlistview = $Service.getlistandview($List, "")
    $strlistid = $ndlistview.childnodes.item(0).name
    $strviewid = $ndlistview.childnodes.item(1).name

    #Create xml object for list modification with batchelement attributes.
    $xmldoc = new-object system.xml.xmldocument
    $batchelement = $xmldoc.createelement("Batch")
    $batchelement.setattribute("Onerror", "Continue")
    $batchelement.setattribute("Listversion", "1")
    $batchelement.setattribute("Viewname", $strviewid)

    $UserInfo = Get-SPList -Listname $List
    $UserData = $UserInfo.data.row  | ? {$_.ows_User -eq $User}
    [array]$IDList = $UserData | % {$_.ows_ID}

    foreach ($item in $IDList) {

        $ndreturn = $null
        $batchelement.innerxml = $null

        $xml = ""
        $xml += "<Method ID='1' Cmd='Update'>" +
                "<Field Name='ID'>$item</Field>" +
                "<Field Name='Member'>false</Field>" +
                "</Method>"

        #Modify the batchelement content with the $xml variable
        $batchelement.innerxml = $xml

        #Implement the modifications to the SharePoint serviceobject defined at the start.

        $ndreturn = $service.UpdateListItems($List, $batchelement)

    }
```

```
}

#Function to retrieve SharePoint list data.
Function Get-SPList (){

    Param([Parameter(Mandatory=$true)]
          [string]$Listname)

    $Uri = "http://test.test.com/sites/sakke/_vti_bin/Lists.asmx?WSDL"
    $Service = New-WebServiceProxy -Uri $Uri  -Namespace SpWs  -UseDefaultCredential

    #Create xml object for the list query.
    $xmlDoc = new-object System.Xml.XmlDocument
    $Query = $xmlDoc.CreateElement("Query")
    $ViewFields = $xmlDoc.CreateElement("ViewFields")
    $QueryOptions = $xmlDoc.CreateElement("QueryOptions")
    $Query.set_InnerXml("FieldRef Name='Full Name'")
    $RowLimit = "0"

    $SPList = $Service.GetListItems($Listname, "", $Query, $ViewFields, $RowLimit, $QueryOp-
tions, "")

    return $SPList

}

#Function to write text to the richtextboxes in the application.
Function Write-ToTextBox
{
    Param(
         $RichTextBox,
         $Text,
         $TextColor,
         $TextFontWeight = "Normal"
         )

    $NewParagraph = New-Object System.Windows.Documents.Paragraph
    $NewParagraph.Margin = 0

    $NewParagraph.AddText("$Text")

    $NewParagraph.Foreground = $TextColor
    $RichTextBox.Document.Blocks.Add($NewParagraph)
    $NewParagraph.FontWeight = $TextFontWeight
    $RichTextBox.ScrollToEnd()
}

#Clear the contents of a richtextbox.
Function Clear-TextBox
{
    Param($RichTextBox)
    $RichTextBox.Document.Blocks.Clear()
}

#Function to generate an email to a user containing a link to the reporting form.
Function New-ReportEmail (){

    Param([parameter(Mandatory=$true)]
          [string]$User)

    $SmtpServer = "mail.test.com"
    $EmailFrom = "sakari.ruotsalainen@test.com"
    $EmailTo = (Get-ADUser $User -properties *).mail
    $MailMessage = New-Object System.Net.Mail.MailMessage $EmailFrom, $EmailTo
    $MailMessage.Subject = "Microsoft Updates Test Group Membership."
    [string]$Body = @"
        <body style='font-family:arial; font-size: 13px; font-style: normal'><b>Dear Recipi-
ent,</b><p><p>As a member of the Microsoft Updates testing group you are requested to report
back any problems you may have encountered or to just report nothing is wrong. You will find a
link to the reporting form below.<p><p><p>Best Regards,<p> GROUP IT</body>
        <a href="http://test.test.com/sites/sakke/_layouts/listform.aspx?PageType=8&Lis-
tId={D5142E11-2683-48A0-95E4-09FD6E3D84AB}&RootFolder=">Reporting Form</a>
"@
    $MailMessage.IsBodyHTML = $True
    $MailMessage.Body = $Body
    $NewMessage = New-Object Net.Mail.SmtpClient($SmtpServer)
    $NewMessage.Send($MailMessage)

}


#Function to send report emails to all list member, uses the New-ReportEmail function.
function Send-ReportEmail (){
    $a = Get-SPList -Listname "Test Group Members"
    $Users = $a.data.row.ows_User

    $SmtpServer = "mail.test.com"
    $EmailFrom = "sakari.ruotsalainen@test.com"

    $Users | % {New-ReportEmail -User $_}

}

#Function to get the statistics from the reports SharePoint list and output relevant data.
Timespans are just examples and easy to modify.
function Get-ReportDBStats () {
  #Get list data
  $Stats = Get-SPList -Listname "Report DB"

  $Date = (get-date).Date

  #Separate into diffrent variables by timespan.
  $1days =   $Stats.data.row | ? {[datetime]$_.ows_Created -ge $date.AddDays(-1)}
  $7days =   $Stats.data.row | ? {[datetime]$_.ows_Created -ge $date.AddDays(-7)}
  $30days =   $Stats.data.row | ? {[datetime]$_.ows_Created -ge $date.AddDays(-30)}

  #Create objects containing information of the reports during the diffrent timespans.

  #1 Day
  if($1days.count -gt 0){
      [array]$1daysOK = $1days | ? {($_.ows_Status -eq 0)}
      [array]$1daysIssue = $1days | ? {($_.ows_Status -eq 1)}

      $1daypercentage = [math]::Round(($1daysOK.Count / $1days.Count)*100)
      $1DayStats = New-Object psobject -Property @{'ID' = '1Days'
                                                   'Count' = $1days.count
                                                   'OK' = $1daysOK.count
                                                   'Issue' = $1daysIssue.count
                                                   'Percentage' = $1daypercentage}
  }
  else{  $1DayStats = New-Object psobject -Property @{'ID' = '1Days'
                                                      'Count' = 'N/A'
                                                      'OK' = 'N/A'
                                                      'Issue' = 'N/A'
                                                      'Percentage' = 'N/A'}}
  #7 Days
  if($7days.count -gt 0){
      [array]$7daysOK = $7days | ? {($_.ows_Status -eq 0)}
      [array]$7daysIssue = $7days | ? {($_.ows_Status -eq 1)}

      $7DayPercentage = [math]::Round(($7daysOK.Count / $7days.Count)*100)
      $7DayStats = New-Object psobject -Property @{'ID' = '7Days'
                                                   'Count' = $7days.count
                                                   'OK' = $7daysOK.count
                                                   'Issue' = $7daysIssue.count
                                                   'Percentage' = $7DayPercentage}
  }

  else{  $7DayStats = New-Object psobject -Property @{'ID' = '7Days'
                                                      'Count' = 'N/A'
                                                      'OK' = 'N/A'
                                                      'Issue' = 'N/A'
                                                      'Percentage' = 'N/A'}}
  #30 Days
  if($30Days.count -gt 0){
      [array]$30daysOK = $30days | ? {($_.ows_Status -eq 0)}
      [array]$30daysIssue = $30days | ? {($_.ows_Status -eq 1)}

      $30DayPercentage = [math]::Round(($30daysOK.Count / $30days.Count)*100)
      $30DayStats = New-Object psobject -Property @{'ID' = '30Days'
                                                   'Count' = $30days.count
                                                   'OK' = $30daysOK.count
                                                   'Issue' = $30daysIssue.count
                                                   'Percentage' = $30DayPercentage}
  }

  else{  $30DayStats = New-Object psobject -Property @{'ID' = '30Days'
                                                       'Count' = 'N/A'
                                                       'OK' = 'N/A'
                                                       'Issue' = 'N/A'
                                                       'Percentage' = 'N/A'}}
  #Output data
  $1DayStats
  $7DayStats
  $30DayStats
}
```

```
#============================================================================
# Make the objects work
#============================================================================

#List users tab
#---------------

#Get user list Button
$WPFShow.Add_Click({$ListContent = Get-SPList -Listname "Test Group Members"
                    [array]$ListInfo = @()
                    $ListContent.data.row | % {$TempObject = New-Object psobject -Property
@{'User' = $_.ows_User;'Client' = $_.ows_Client}; $ListInfo += $TempObject}
                    $WPFlistView.Items.Clear();$ListInfo | % {$WPFlistView.AddChild($_)}})


#Add user tab
#---------------

#Add User Button
if($WPFUserName2.Text -ne $Null -and $WPFClientName2.Text -ne $Null){
$WPFAdd.Add_Click({
                   try{
                   Add-ToSPList -User $WPFUserName2.Text -Client $WPFClientName2.Text
                   Write-ToTextBox -RichTextBox $WPFStatusBox -Text "User $($WPFUserName2.Text)
with client $($WPFClientName2.Text) has been added to the group." -TextColor Green}

                   catch {Write-ToTextBox -RichTextBox $WPFStatusBox -Text "An error occurred!"
-TextColor Red}
                   $WPFUserName2.clear();$WPFClientName2.clear()})
}

#Remove user tab
#---------------

#Remove user Button
if($WPFUserName3.Text -ne $Null){
                      $WPFRemove.Add_Click({Write-ToTextBox -RichTextBox $WPFStatusBox -Text
"Removing $($WPFUserName3.Text) from the test group." -TextColor Green
                      try{#Set-SPListMember -User $($WPFUserName3.Text)
                          Write-ToTextBox -RichTextBox $WPFStatusBox -Text "User
$($WPFUserName3.Text) has been removed from the group." -TextColor Green
                      }
                      catch {Write-ToTextBox -RichTextBox $WPFStatusBox -Text "An error oc-
curred!" -TextColor Red}
                      $WPFUserName3.Clear()})
}

#Exit button
$WPFExit.Add_Click({$Form.Close()})

#User responses tab
#---------------

#Send Email with a link to the reporting form.
$WPFSendEmail.Add_Click({Write-ToTextBox -RichTextBox $WPFStatusBox -Text "Sending Email to Test
Group Members." -TextColor Green
                        try{Send-ReportEmail}

                        catch{Write-ToTextBox -RichTextBox $WPFStatusBox -Text "An error oc-
curred!" -TextColor Red}
                        })

#Show report stats for 1,7 and 30 day windows.
$WPFShowStats.Add_Click({
    Clear-TextBox -RichTextBox $WPFStatsBox

    $Stats = Get-ReportDBStats
    $1D = $Stats[0]
    $7D = $Stats[1]
    $30D = $Stats[2]

    $1DMessage = @"
    Statistics for 1 Day period:
    -----------------------------
    Number of reports: $($1D.Count)
    Issues reported: $($1D.Issue)
    No Problems for: $($1D.Percentage) %
"@
    $7DMessage = @"
    Statistics for 7 Day period:
    -----------------------------
    Number of reports: $($7D.Count)
    Issues reported: $($7D.Issue)
    No Problems for: $($7D.Percentage) %
"@
    $30DMessage = @"
    Statistics for 30 Day period:
    ------------------------------
    Number of reports: $($30D.Count)
    Issues reported: $($30D.Issue)
    No Problems for: $($30D.Percentage) %
"@

if($WPFReportDate1Day.IsChecked){Write-ToTextBox -RichTextBox $WPFStatsBox -text $1DMessage -
textcolor Black}
elseif($WPFReportDate7Day.IsChecked){Write-ToTextBox -RichTextBox $WPFStatsBox -text $7DMessage
-textcolor Black}
elseif($WPFReportDate30Day.IsChecked){Write-ToTextBox -RichTextBox $WPFStatsBox -text $30DMes-
sage -textcolor Black}

})

#Clear statusbox button
$WPFClear.Add_Click({Clear-TextBox -RichTextBox $WPFStatusBox})

#===========================================================================
# Shows the form
#===========================================================================
write-host "To show the form, run the following" -ForegroundColor Cyan
'$Form.ShowDialog() | out-null'
$run = $Form.ShowDialog() | out-null
```