



Microsoft®

System Center  
Configuration Manager

Configuration Manager  
System Center



DEPLOYING CUSTOM  
MICROSOFT OFFICE TEMPLATES  
WITH SYSTEM CENTER  
CONFIGURATION MANAGER

Author Trevor Jones  
Version 1.0  
Date 2017-03-07  
Blog [smsagent.wordpress.com](http://smsagent.wordpress.com)

# Contents

Introduction .....	5
Requirements .....	5
Deploying Custom Office Templates the Microsoft Way .....	5
How This Solution Works .....	6
Solution Creation Flow .....	6
Compliance Item Process Flow .....	7
Remote Template Store .....	8
Local System Template Store .....	9
Local User Template Store (XML files).....	9
Compliance Setting .....	10
Machine-based Compliance Setting .....	10
Discovery Script.....	10
Hash Check.....	10
Remediation Script.....	10
User-based Compliance Setting.....	11
Discovery Script.....	11
HKCU Registry Changes.....	11
Remediation Script.....	11
Multiple Office Versions .....	12
Deploying the Solution.....	12
Create the Remote Template Store.....	12
Create the Root Path .....	12
Categorise Your Templates .....	12
Template Directories.....	13
XML Directory.....	13
Creating the Template Files .....	13
The Custom Template.....	13
The Preview and Thumbnail Files.....	13
File Versioning.....	14
XML Configuration File .....	14
Example XML Configuration .....	15
Example XML Configuration File .....	15
Setting the Correct Template Source.....	16
Multiple Architectures .....	16
Naming the XML Configuration File .....	16
Adding the Template Files to the Remote Template Store .....	17

- Creating the Compliance Setting ..... 17
  - Create a Configuration Item..... 17
    - Create the Machine-based Compliance Setting ..... 18
    - Create the User-based Compliance Setting..... 20
    - Add the Configuration Item to a Configuration Baseline ..... 22
- Changing the Script Execution Timeout Setting ..... 23
- Managing the Custom Template Lifecycle ..... 24
  - Adding New Templates ..... 24
  - Updating a Template ..... 24
  - Retiring a Template ..... 24
- Multiple Language Support ..... 25

## Introduction

Many organisations that use Microsoft Office like to make custom templates available to their employees. These templates are usually unique to their organisation and allow their employees to create consistent documentation following a pre-defined style and document flow. For enterprises that use System Center Configuration Manager (aka SCCM), such custom templates can be deployed to end users using SCCM's Windows management capabilities, and this guide presents a solution for achieving that.

SCCM allows more than one way of deploying software, scripts and the like to end user's computers. In this guide, we will use SCCM's **Compliance Settings** feature. This will allow us both to deploy the templates, and also to manage them through their lifecycle including updating, adding or removing templates, and ensuring that all targeted systems remain compliant with the current templates available at all times.

## Requirements

- ☑ Microsoft Office 2010 (32-bit) or later on targeted clients
- ☑ System Center Configuration Manager 2012 or later
- ☑ PowerShell 4.0 or later on targeted clients
- ☑ Familiarity with Compliance Settings in SCCM
- ☑ Familiarity with PowerShell scripting

*Note: this guide is intended for locally-installed versions of Microsoft Office. It has not been tested for Office 365.*

## Deploying Custom Office Templates the Microsoft Way

For Office 2010, Microsoft created some documentation on TechNet describing a supported way to deploy custom Office templates. This documentation can be found here:

[https://technet.microsoft.com/en-us/library/cc178976\(v=office.14\).aspx](https://technet.microsoft.com/en-us/library/cc178976(v=office.14).aspx)

There is actually more than one way to deploy custom templates, but the method described in this article is the best and most reliable I have found, especially since it allows you to include thumbnail and preview images of your templates which adds a professional touch. Even though the documentation was produced for Office 2010 (and there is nothing more recent to my knowledge) it still works perfectly well through Office 2013 and Office 2016.

As the article describes, there is a high-level overview to deploying custom templates:

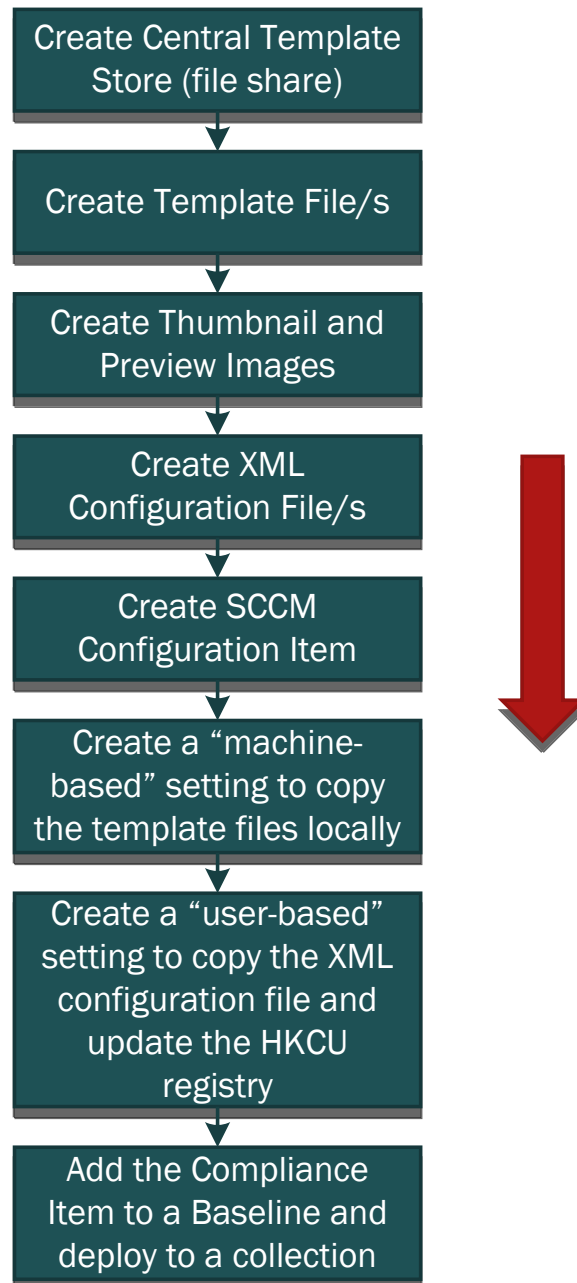
1. Create your templates.
2. Create thumbnail and preview files.
3. Create an XML configuration file that describes the custom template, thumbnail, and preview files.
4. Deploy a registry key that enables Office to read the XML configuration file.

We will look at each of these. First, however, let's have a look at how this solution works.

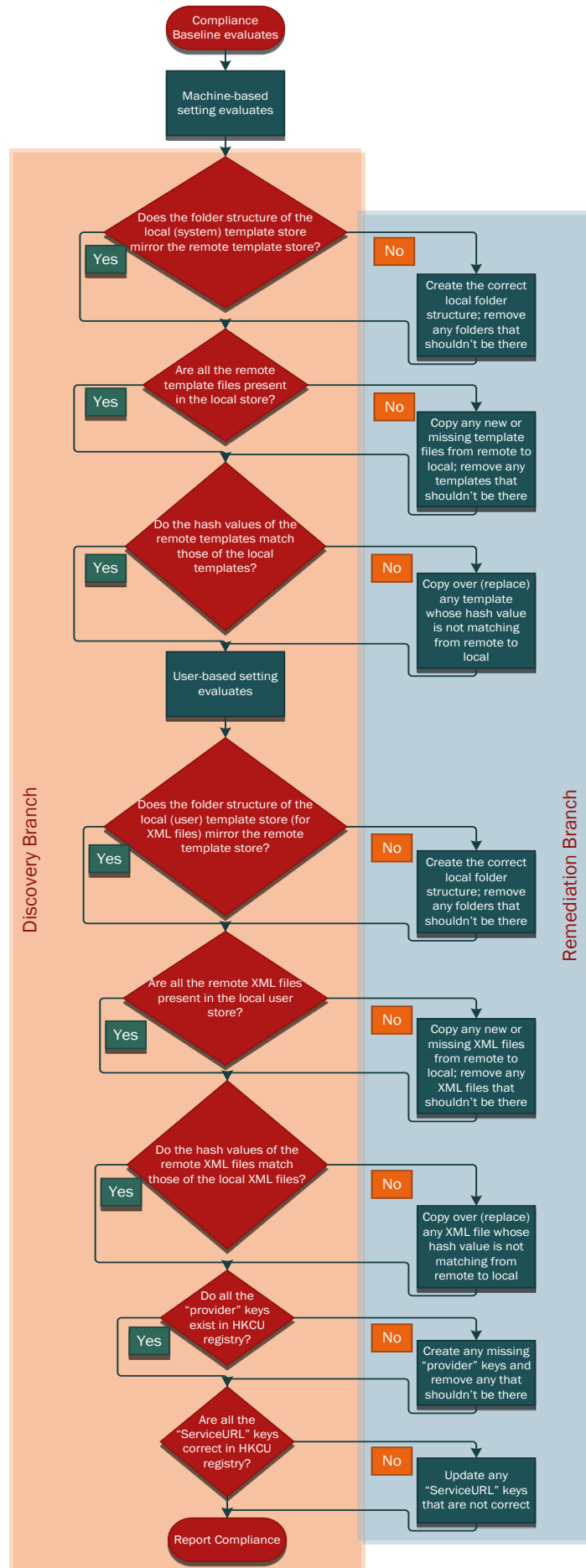
## How This Solution Works

The following is an overview of the solution and how it works. Below are two flow diagrams – the first gives an overview of what is required to create this solution, the second gives a logical overview of how the compliance item (that does all the 'heavy lifting') works.

### Solution Creation Flow



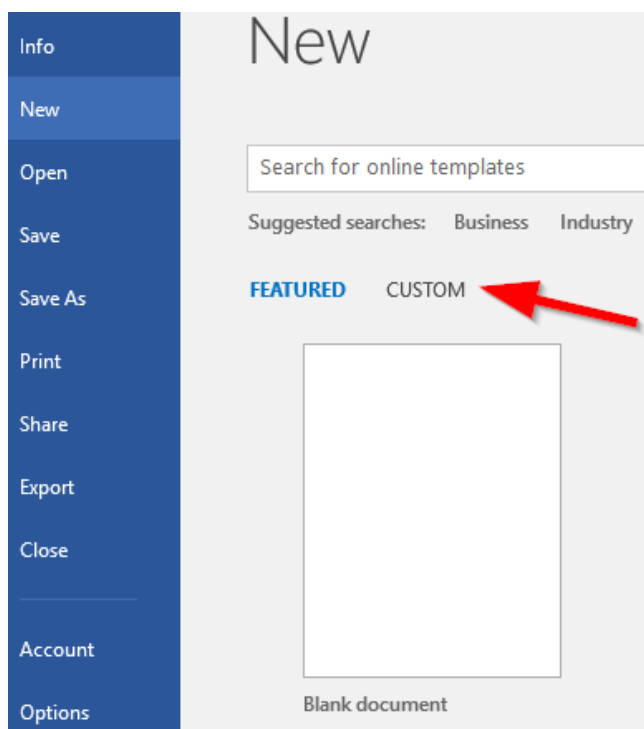
## Compliance Item Process Flow



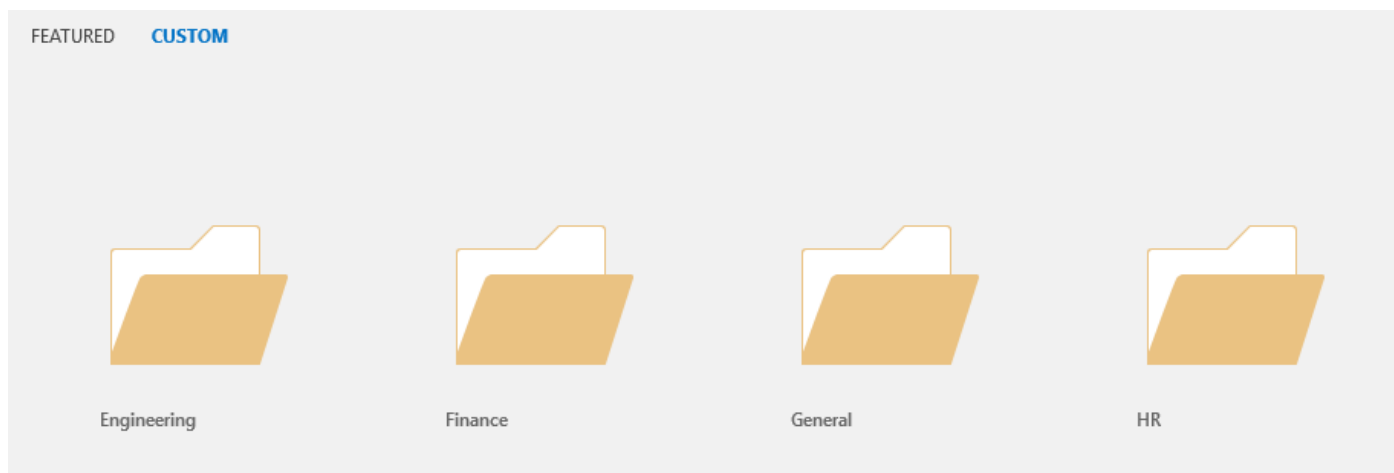
## Remote Template Store

We will place our template files (templates, preview and thumb images, XML configuration files) in a remote template store. This store can be a file server share that is accessible to all. If you have several global offices and slow network links, you may consider replicating this store using DFS. The directory structure we use is important as it will be mirrored to the local template store on the client. Under this share we organise our templates by grouping them together into subfolders, for example, one subfolder per department (HR, Finance, Engineering etc). These subfolder (provider) names will be used by your Office applications to create template collections.

To illustrate, here is an example from Word 2016. Once the custom templates are deployed, when you open Word and click New, you will see a list of default templates. Next to the **Featured** category, you will also find **Custom**.



When you click on **Custom**, you will see your templates grouped into the relevant subfolders:



In the remote template store, these subfolders are defined:



This PC > Data (G:) > RemoteFiles > Office Templates >			
Name	Date modified	Type	
Engineering	03/03/2017 16:25	File folder	
Finance	03/03/2017 16:25	File folder	
General	03/03/2017 16:25	File folder	
HR	03/03/2017 16:25	File folder	

If we go into the **General** subfolder, there are further subfolders containing template files and XML files. We create one subfolder for **each template type** (eg PowerPoint, Word, Excel, Visio), as well as one subfolder containing the XML file that defines the templates for that provider.

This PC > Data (G:) > RemoteFiles > Office Templates > General >			
Name	Date modified	Type	
PowerPoint Templates	03/03/2017 16:26	File folder	
Word Templates	03/03/2017 16:26	File folder	
XML	03/03/2017 16:26	File folder	

The Compliance Setting is configured such that any changes to providers, subfolders or files in this remote store will be mirrored on the client. This makes it easier for us to manage template updates or additions or retirements.

## Local System Template Store

The local template store is a local cache of all these templates contained in the remote store. The reason we cache them locally rather than link to the remote store directly is that our templates will always be available even when working offline.

Microsoft Office creates a system-wide template directory which can be found in your Program Files location. On a typical x64 system, for example, the template directory is found here:

### C:\Program Files (x86)\Microsoft Office\Templates

This is the location we will use to copy the remote template store to. The XML configuration file will reference this location. Changes to this location require administrative privileges, therefore one of the Compliance Item settings will run in administrative context to manage the templates files.

## Local User Template Store (XML files)

We will also use an additional location in the user profile area to store the XML files. The reason for separating the XML files from the templates is that when we add a new XML file, or update an existing XML file, we must make a change in the HKCU (current user) branch in the registry. Therefore, one of the Compliance Item settings will run in the logged-on user context to manage the XML files and the HKCU registry changes.

There are a couple of locations that can be used for the local user template store, for example since Office 2013 we have a **Custom User Templates** directory contained in the **My Documents** area of the user profile. We also have **%AppData%\Microsoft\Templates** directory, and this is the one we will use for the local user template store since it is also compatible with Office 2010.

I am calling it a template store, but in fact it will just be used to store the XML configuration files, nothing else.

### Compliance Setting

This solution uses SCCM's **Compliance Settings** feature to deploy and manage our templates. Once set up, we will be able to deploy the templates to our targeted users, and have any easy way to manage template changes in future. Templates can get updated, or retired, or new templates created, so we need an easy mechanism to manage the custom template lifecycle. Using Compliance Settings, we can deploy custom PowerShell scripts that will make this possible.

Compliance Settings in SCCM require one or more **Compliance Baselines**. These baselines are targeted for deployment to **collections** (groups of devices or users), and contain one or more **Compliance Items**. Each compliance item contains one or more **Compliance Settings**. A compliance setting can be used to check whether the target system is compliant with a particular requirement and optionally perform some remediation activity if it is not.

This solution uses **one Compliance Item** with **two Compliance Settings**. One setting runs in **administrative** context, which we are calling a **Machine-based setting**, and one setting runs in the **logged-on user** context, which we are calling a **User-based setting**.

These two settings with their custom **discovery** and **remediation** scripts will help manage the lifecycle of our deployed custom templates.

### Machine-based Compliance Setting

The machine-based compliance setting runs in the default administrative context that compliance settings use. Since our targeted users may or may not have administrative rights, this setting allows us to access the local template store on the client (which is read-only to standard users) and add, remove or update template files.

### Discovery Script

The **discovery script** for this setting will check the following things:

- Does the directory structure of the local template store mirror what exists in the remote template store?
- Do the files present in the local template store mirror the files present in the remote template store?
- Does the hash value of the files in the local template store match the hash values of the files in the remote template store?

If any of these are not true, it will report non-compliance and then attempt remediation.

### Hash Check

The hash value of the files is used to check whether a template has been updated in the remote template store. For example, if you have deployed a template, and the author has now updated the template but used the same file name, the hash value comparison will detect that the file is different even though the file name has not changed. This way, an existing template can be updated without the need to update the XML configuration file with a new template file name. The template version can instead be stored in the metadata of the file itself, and not in the file name.

### Remediation Script

The **remediation script** will do the following:

- Create any directories or files that are present in the remote template store but missing from the local template store.
- Remove any directories or files that are present in the local template store but are not present in the remote template store.
- Replace any files in the local template store whose hash values do not match those in the remote template store.

### User-based Compliance Setting

The user-based compliance setting runs in the context of the logged-on user. This is because we need to access the profile of the logged-on user to store the XML configuration files and because we need to access the HKCU (current user) branch in the registry. Remember that in terms of files, this setting only manages the XML configuration files in the local *user* template store, while the machine-based setting manages the other template files in the local *system* template store.

### Discovery Script

The **discovery script** for the user-based setting will check the following things:

- Does the directory structure of the local user template store mirror what exists in the remote template store?
- Do the XML files present in the local user template store mirror the files present in the remote template store?
- Does the hash value of the XML files in the local user template store match the hash values of the files in the remote template store?
- For each version of Microsoft Office installed:
  - Do the providers (ie categories) in the HKCU registry match the providers that exist in the remote template store?
  - Does the ServiceURL key for each provider in the HKCU registry point to the correct location of the relevant XML file?

### HKCU Registry Changes

The HKCU registry branch we are referring to is located here:

```
HKCU:\SOFTWARE\MICROSOFT\OFFICE\<OFFICEVERSIONCODE>\COMMON\SPOTLIGHT
```

Under this registry branch are two further branches, **Providers** and **Content**. This will be discussed later but a key needs to be created for each provider (ie Engineering, HR etc) under the Provider branch, and a **ServiceURL** key defined. This ServiceURL key tells Office where to locate the XML configuration file for that provider. Once it finds the XML file, it can create the custom templates folder you see in your Office application based on the contents of this file.

The Content branch is where Office stores the locations of the template files based on what it has read from the XML configuration. If the XML file has been updated, this branch needs to be updated or regenerated. To accomplish this we delete the branch so that Office is forced to re-read the XML file, recreate the Content branch and store the updated details there.

### Remediation Script

The **remediation script** will do the following:

- Create any directories or XML files that are present in the remote template store but missing from the local user template store.
- Remove any directories or XML files that are present in the local user template store but are not present in the remote template store.
- Replace any XML files in the local user template store whose hash values do not match those in the remote template store.

- For each version of Microsoft Office installed:
  - Create any missing provider branch in the HKCU registry, including the ServiceURL key
  - Remove any provider that is present in the HKCU registry but does not exist in the remote template store
  - Update the ServiceURL key is for each provider in the HKCU registry that is not pointing to the correct location
- For any registry change or XML file change, delete the Content branch in the HKCU registry so that Office is forced to re-read the XML files for each provider and recreate the Content branch pointing to the correct locations of the template files

### Multiple Office Versions

In some (hopefully rare) cases users may have more than one version of Office or Office applications installed, for example they may have Office 2013/6 and use Excel 2010 for some backwards compatibility purpose. The User-based compliance setting will update the HKCU registry for each version of Microsoft Office installed, provided you have listed the applicable Office Version Keys in the User-set Variables (more on this later). This will make the custom templates available in each Office version.

## Deploying the Solution

Now let's go ahead and create and deploy this solution.

### Create the Remote Template Store

First, we need to create the remote template store. This will be a location on a file share that is accessible to everyone. The template files will be copied from this remote store to the local clients. The directory structure of this store is important as it will be mirrored exactly on the client, and any changes to files or directories will also be changed on the client.

### Create the Root Path

The **root path** will be defined in the compliance scripts as the **\$RemoteTemplatePath** variable. It can be a normal SMB file share, eg:

```
\\FILESERVER01\OFFICETEMPLATES
```

### Categorise Your Templates

Underneath the root path, create subdirectories for each template **category** you want to create. This could be departments, projects etc. For example:

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING
```

```
\\FILESERVER01\OFFICETEMPLATES\FINANCE
```

These categories are important as they will be used to group your custom templates together in your Office application, as we have seen before.



## Template Directories

Underneath each template category, create subdirectories for each template **application type**, for example, Word Templates, PowerPoint Templates etc:

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\WORD TEMPLATES
```

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\POWERPOINT TEMPLATES
```

## XML Directory

Underneath **each** category, create one **XML** directory. All the XML files for that category will be located here. Eg:

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\XML
```

## Creating the Template Files

Now we need to create the relevant template files for each template we want to deploy.

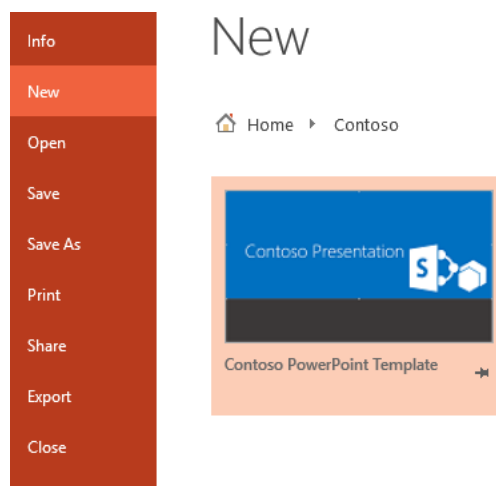
## The Custom Template

Save the template file in a template format, for example .potx for a PowerPoint template, .dotx for a Word template etc.

*Note: The Microsoft Technet article we referenced earlier states that we can deploy Word, Excel or PowerPoint templates, but you can actually deploy Visio templates too.*

## The Preview and Thumbnail Files

The Preview and Thumbnail files are used to provide a preview image of the template so you get a quick visual of what the template actually looks like:



Create two image files with different dimensions using the Microsoft recommendations below for the image sizes.

Image Type	Width (px)	Height (px)
Thumbnail	100	120
Preview	256	350

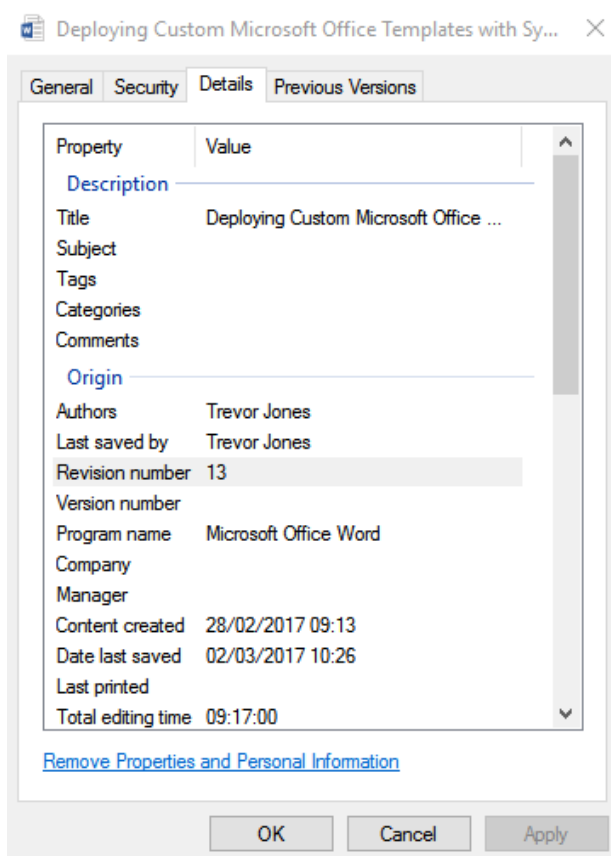
To do this, you can open the template and use a screenshot tool (like Greenshot) to capture an image of the first page. You can then resize the image (using MS Paint for example) and save the image to a supported format (.jpg, .png, .bmp, .gif).

I recommend to name these files using a similar descriptor, for example:

- Custom Template File: MyTemplate.dotx
- Preview Image File: MyTemplate\_Preview.png
- Thumbnail Image File: MyTemplate\_Thumbnail.png

### File Versioning

With this solution, it is not necessary to use version numbers in the file names. For example, you might want to use *MyTemplate\_v1.dotx*, then when it gets updated in future you can use *MyTemplate\_v2.dotx*. This is not necessary as the file hash checks performed by the compliance setting scripts will detect any changes in the source template file even if the file name has not changed. Therefore, to manage file versions you do not need to increment version numbers in the file name, you could instead use the metadata of the file itself to document the version number, for example the **Revision number** in the screenshot below:



### XML Configuration File

The XML configuration file is used to define which application/s and languages the templates are for, where the template source files can be found and how the template is referenced. Office will

read this configuration file when an Office application is opened and add the relevant custom templates within the application.

Since the location of the template files is a "hard-coded" path in the configuration file, it may be necessary to create two XML files per category, one for each OS architecture in your environment. This is because 32-bit software is installed to a different "Program Files" directory on a 64-bit OS, therefore one XML file will need to reference the "Program Files" directory on a 32-bit system, and one will need to reference the "Program Files (x86)" directory on a 64-bit system.

### Example XML Configuration

Below is an example of an XML configuration file for the x64 architecture. It contains three templates: one for PowerPoint, and two for Word.

In the headers we have some parameters such as **lcid**, which is the language code for the version of Office you are targeting. 1033 represents US English. Other supported language codes can be found here: <https://technet.microsoft.com/en-us/library/cc179219.aspx>

The **id** parameter in the **o:application** tag indicates the application you are targeting, for example:

WD = Word  
PP = PowerPoint  
XL = Excel  
VO = Visio

The **startdate** and **enddate** in the **o:featuredtemplates** tag indicate the validity period of the template. Always set the **enddate** to far in the future if you plan to use the template indefinitely.

Within each featured template you need to specify the template title that will appear in the Office application, the location of the source files for the template and image files etc.

You can have *multiple templates per application*, and *multiple applications* all in one XML file.

### EXAMPLE XML CONFIGURATION FILE

```
<?xml version="1.0" encoding="utf-8"?>
  <o:featuredcontent lcid="1033" xmlns:o="urn:schemas-microsoft-com:office:office">
    <o:application id="PP">
      <o:featuredtemplates startdate="2011-03-01" enddate="2099-03-01">
        <!-- PP TEMPLATE 1 -->
        <o:featuredtemplate title="Contoso PowerPoint Template" source="C:\Program Files
(x86)\Microsoft Office\Templates\Contoso\Engineering\PowerPoint
Templates\Contoso_PP_Template.potx" >
          <o:media mediatype="png" filename="Contoso_PP_Thumb.PNG" source="C:\Program
Files (x86)\Microsoft Office\Templates\Contoso\Engineering\PowerPoint
Templates\Contoso_PP_Thumb.PNG" />
          <o:preview filename="Contoso_PP_Preview.PNG" source="C:\Program Files
(x86)\Microsoft Office\Templates\Contoso\Engineering\PowerPoint
Templates\Contoso_PP_Preview.PNG" />
        </o:featuredtemplate>
      </o:featuredtemplates>
    </o:application>
    <o:application id="WD">
      <o:featuredtemplates startdate="2011-03-01" enddate="2099-03-01">
        <!-- WD TEMPLATE 1 -->
        <o:featuredtemplate title="Design" source="C:\Program Files (x86)\Microsoft
Office\Templates\Contoso\Engineering\word Templates\Design_Template.potx" >
          <o:media mediatype="png" filename="Design_Thumb_2015.PNG" source="C:\Program
Files (x86)\Microsoft Office\Templates\Engineering\Contoso\Design_Thumb.PNG" />
          <o:preview filename="Design_Preview_2015.PNG" source="C:\Program Files
(x86)\Microsoft Office\Templates\Engineering\Contoso\Design_Preview.PNG" />
        </o:featuredtemplate>
        <!-- WD TEMPLATE 2 -->
```



```

        <o:featuredtemplate title="Memo" source="C:\Program Files (x86)\Microsoft
Office\Templates\Contoso\Engineering\word Templates\Memo_Template.potx" >
        <o:media mediatype="png" filename="Memo_Thumb_2015.PNG" source="C:\Program Files
(x86)\Microsoft Office\Templates\Contoso\Engineering\word Templates\Memo_WD_Thumb.PNG" />
        <o:preview filename="Memo_Preview_2015.PNG" source="C:\Program Files
(x86)\Microsoft Office\Templates\Contoso\Engineering\word Templates\Memo_WD_Preview.PNG" />
        </o:featuredtemplate>
    </o:featuredtemplates>
</o:application>
</o:featuredcontent>

```

### Setting the Correct Template Source

As mentioned previously, the local template store is using the Microsoft default location for system-wide templates in the program files installation directory. Within this directory, all the custom templates will be placed underneath a single subfolder, using the same directory structure that exists in the remote template store. This subfolder is defined in the compliance scripts as the **\$RootFolderName** variable. The subfolder name could be a company name, or just "Custom Templates" – anything you decide, but all template files will be placed here. For example, on an x64 system with 32-bit Microsoft Office installed, using a root folder name of "Contoso", the templates would be located here:

```
C:\PROGRAM FILES (X86)\MICROSOFT OFFICE\TEMPLATES\CONTOSO
```

The directories underneath this would then mirror what is in the remote template store. For example, the following remote files and directories:

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING
```

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\WORD TEMPLATES
```

```
\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\WORD TEMPLATES\CUSTOM_TEMPLATE.DOTX
```

would become the following local files and directories:

```
C:\PROGRAM FILES (X86)\MICROSOFT OFFICE\TEMPLATES\CONTOSO\ENGINEERING
```

```
C:\PROGRAM FILES (X86)\MICROSOFT OFFICE\TEMPLATES\CONTOSO\ENGINEERING\WORD TEMPLATES
```

```
C:\PROGRAM FILES (X86)\MICROSOFT OFFICE\TEMPLATES\CONTOSO\ENGINEERING\WORD
TEMPLATES\CUSTOM_TEMPLATE.DOTX
```

So when specifying the source of the MS Word Custom\_Template.dotx for the Engineering department, you would use the following in the XML file:

```
SOURCE="C:\PROGRAM FILES (X86)\MICROSOFT OFFICE\TEMPLATES\CONTOSO\ENGINEERING\WORD
TEMPLATES\CUSTOM_TEMPLATE.DOTX"
```

### MULTIPLE ARCHITECTURES

If you are deploying to both x86 and x64 systems, **create one XML file for each architecture**. Use the correct program files directory location in each file, for example in an XML for a 32-bit OS you would use the following for the Custom\_Template.dotx:

```
SOURCE="C:\PROGRAM FILES\MICROSOFT OFFICE\TEMPLATES\CONTOSO\ENGINEERING\WORD
TEMPLATES\CUSTOM_TEMPLATE.DOTX"
```

### NAMING THE XML CONFIGURATION FILE

The name of the XML configuration file **must include the OS architecture being targeted**, even if you are only targeting a single architecture. For example, if targeting both 32-bit and 64-bit OS architectures, you would create two XML files:

```
Engineering_x64.xml
```



Engineering\_x86.xml

The compliance scripts will only copy the relevant XML for the OS architecture of the client, filtering by the keywords "x86" or "x64" in the file name.

## Adding the Template Files to the Remote Template Store

Once you have created all the templates files, add them to the remote template store. Let's use the MS Word Custom\_Template for the Engineering department as an example.

The template file (**Custom\_Template.dotx**), and the two image files (**Custom\_Template\_Preview.png** and **Custom\_Template\_Thumbnail.png**) go into the following location:

\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\WORD TEMPLATES

If we had an Excel template, the template files would go into:

\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\EXCEL TEMPLATES

...and so on.

The XML configuration files (**Engineering\_x64.xml** and **Engineering\_x86.xml**) would go into:

\\FILESERVER01\OFFICETEMPLATES\ENGINEERING\XML

## Creating the Compliance Setting

Now let's go to Configuration Manager and create the Compliance Setting.

### Create a Configuration Item

In the ConfigMgr console, navigate to **Assets and Compliance > Compliance Settings**. Click on **Configuration Items** and create a new Configuration item.

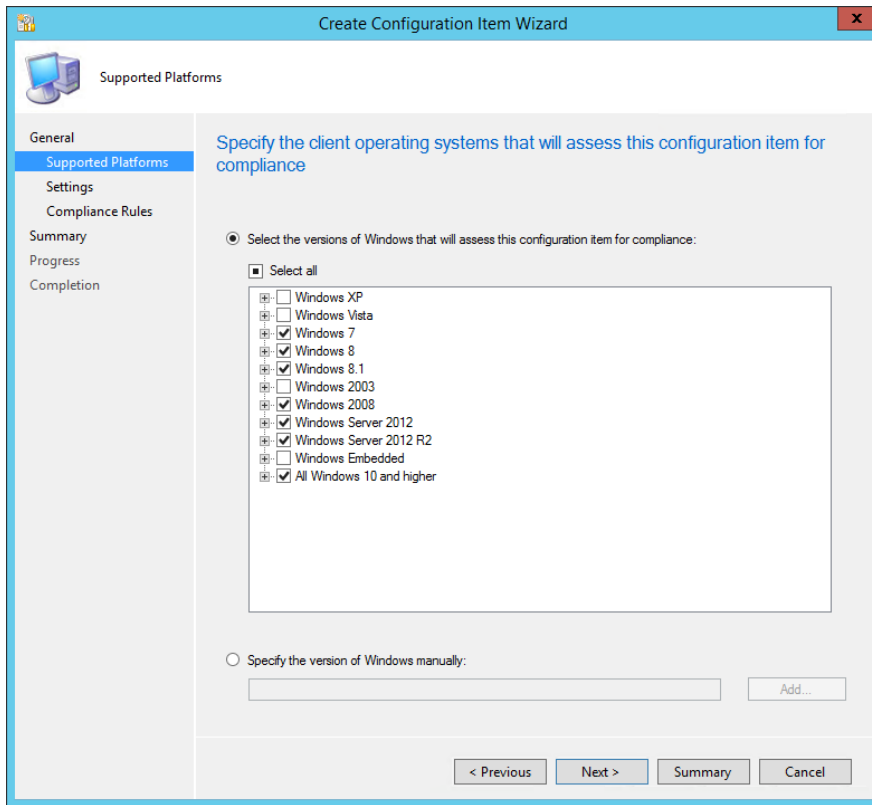
Give it a **Name** and **Description** and choose the **Windows** type.

The screenshot shows the 'Create Configuration Item Wizard' window. The 'General' tab is active, displaying the following information:

- Name:** Custom Microsoft Office Templates
- Description:** Used to deploy and manage custom templates for Microsoft Office applications.
- Specify the type of configuration item that you want to create:** Windows
- This configuration item contains application settings
- Assigned categories to improve searching and filtering:** Client

At the bottom of the wizard, there are navigation buttons: < Previous, Next >, Summary, and Cancel.

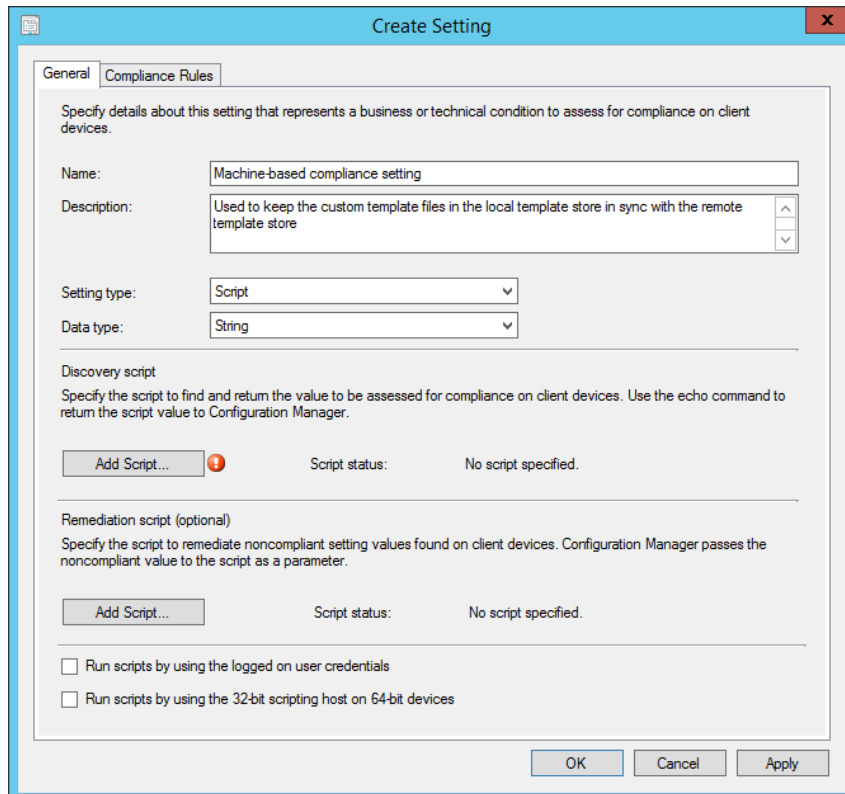
Click Next. Select the applicable Windows versions. Remember that PowerShell 4.0 minimum is required on the client.



On the **Settings** pane, click **New**.

### Create the Machine-based Compliance Setting

Enter a **Name** and **Description**, choose **Script** for the setting type and **String** for the data type.



Download the discovery and remediation scripts from my Github repo. Use the **Machine Setting** scripts for this setting.

<https://github.com/SMSAgentSoftware/ConfigMgr/tree/master/PowerShell%20Scripts/Compliance%20Settings/Microsoft%20Office%20Templates>

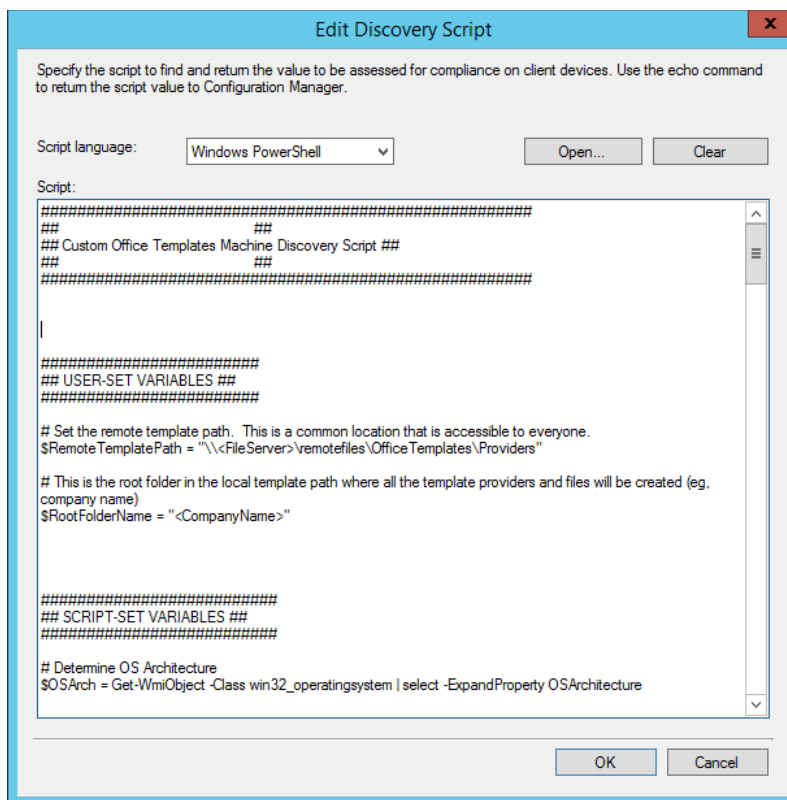
In both the discovery and remediation scripts, set the following two variables found at the top in the **User-Set Variables** section:

**\$RemoteTemplatePath** – This is the file-share being used for the remote template store, eg  
 \\FILESERVER01\OFFICETEMPLATES

**\$RootFolderName** – This is name of the directory in the local template store where all the template files will be copied to (see the “Setting the Correct Template Source” section), eg “Contoso”

In the **Create Setting** window, click **Add Script** and paste the discovery script into the window (or click **Open** and locate the script). Make sure that **Windows PowerShell** is set as the script language.

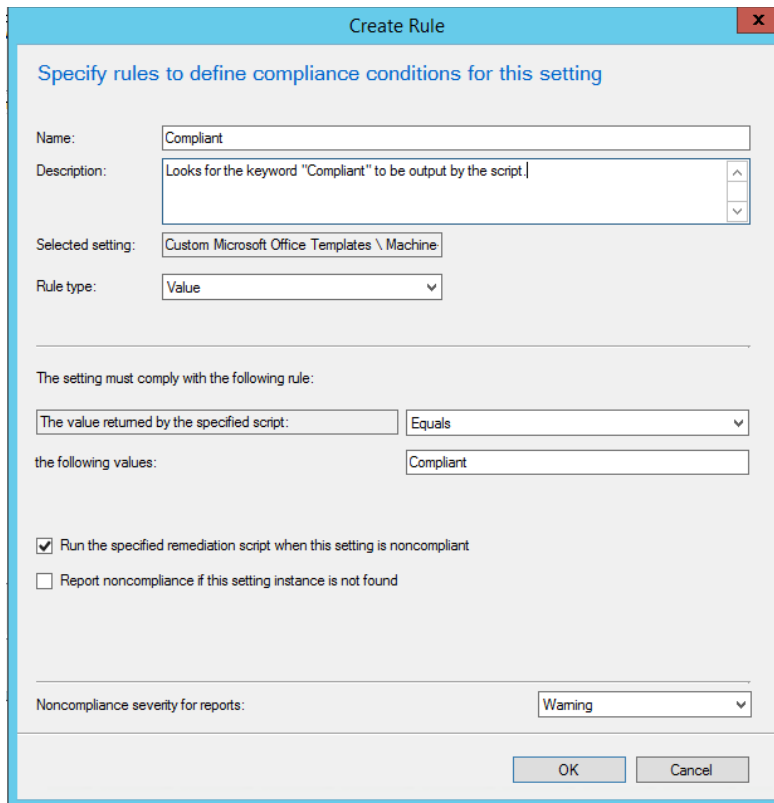
Do the same for the remediation script.



Next, click on the **Compliance Rules** tab. Click **New** to create a new rule.

Enter the **Name** and **Description**. Use the rule type **Value** and enter the value **Compliant**. The rule will look for the string “Compliant” to be output by the script to determine compliance.

Be sure to check the box to **run the specified remediation script**, and optionally set the **noncompliance severity**.



Click Ok and Ok again to save the machine-based setting.

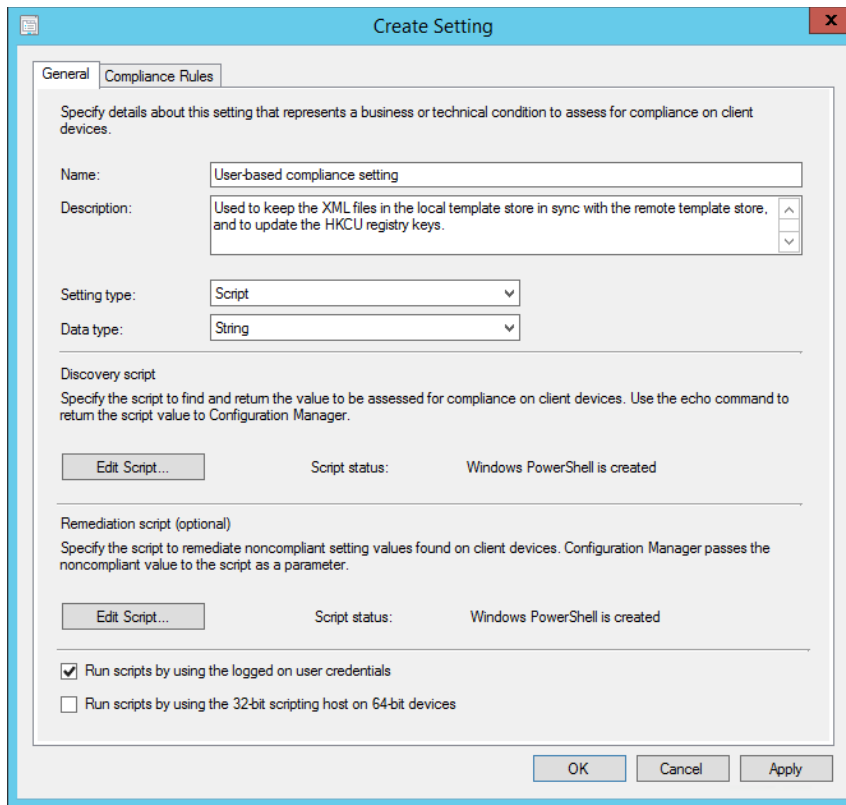
### Create the User-based Compliance Setting

Create the User-based compliance setting in the same way. Use the **User Setting** scripts from my Github repo.

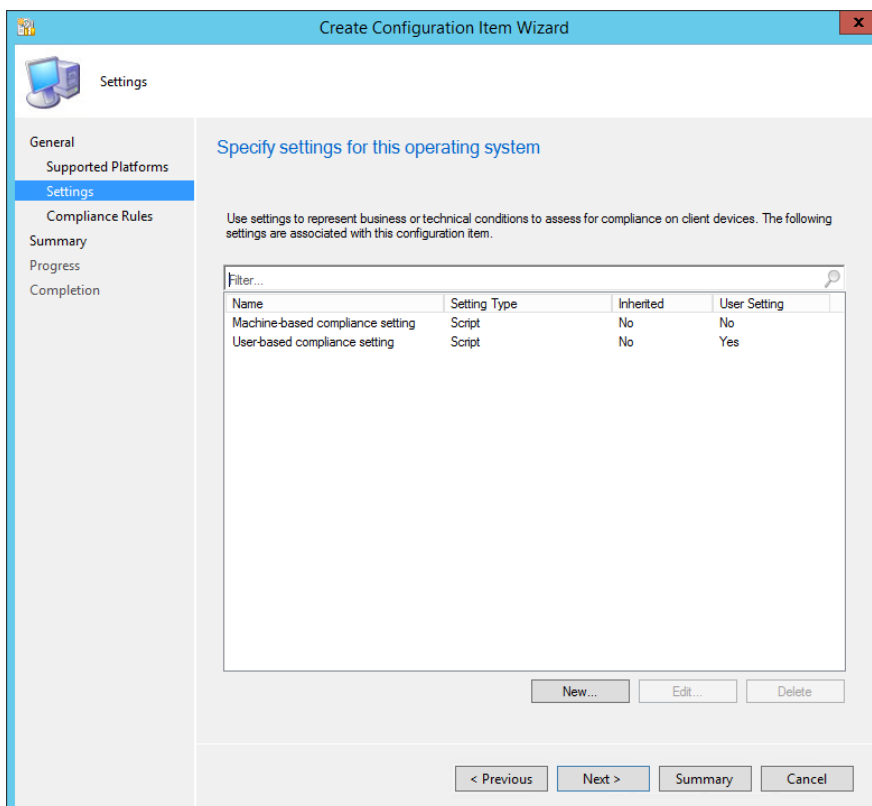
These scripts have one additional user-set variable called **\$OfficeVersionKeys**. Here you enter in an array the **version number** for each version of Microsoft Office you may be targeting in your environment.

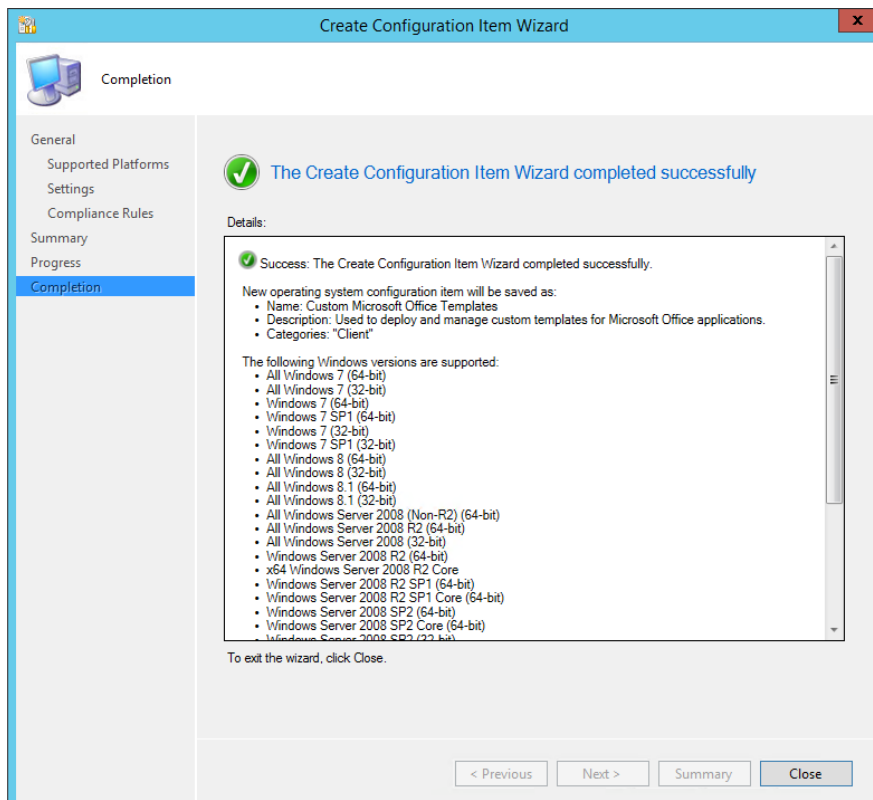
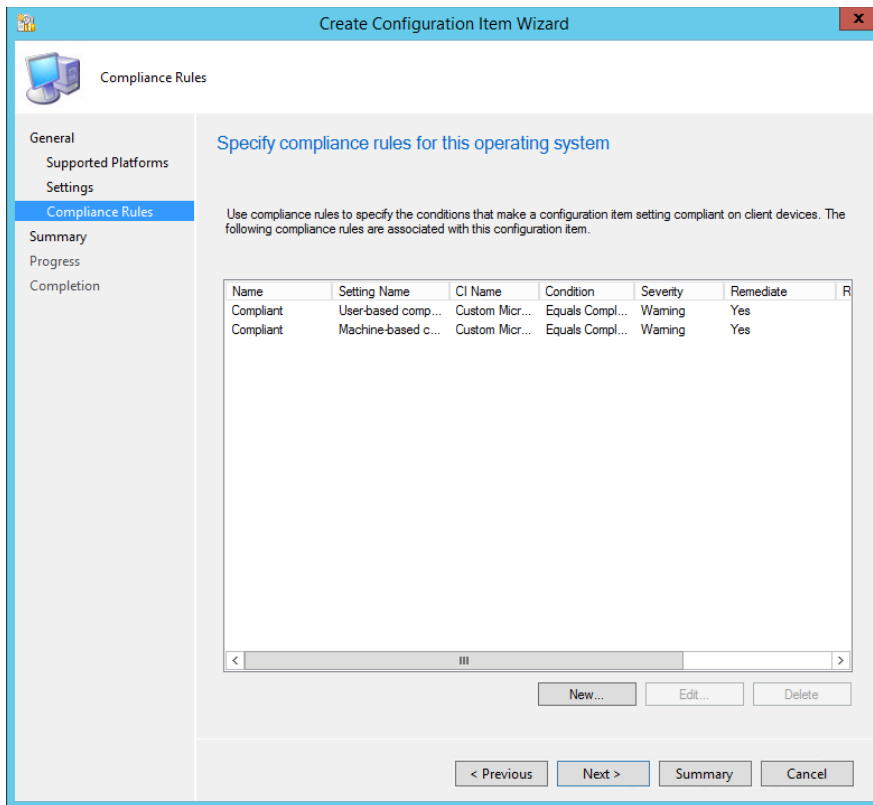
```
$OfficeVersionKeys = @(
    "16.0", # Office 2016
    "15.0", # Office 2013
    "14.0" # Office 2010
)
```

This setting must be run in the context of the logged-on user, so be sure to check the box **Run scripts by using the logged-on user credentials**.



Create a **compliance rule** identical to that of the machine-based setting.  
 When finished, click Next through the rest of the Configuration Item wizard.





### Add the Configuration Item to a Configuration Baseline

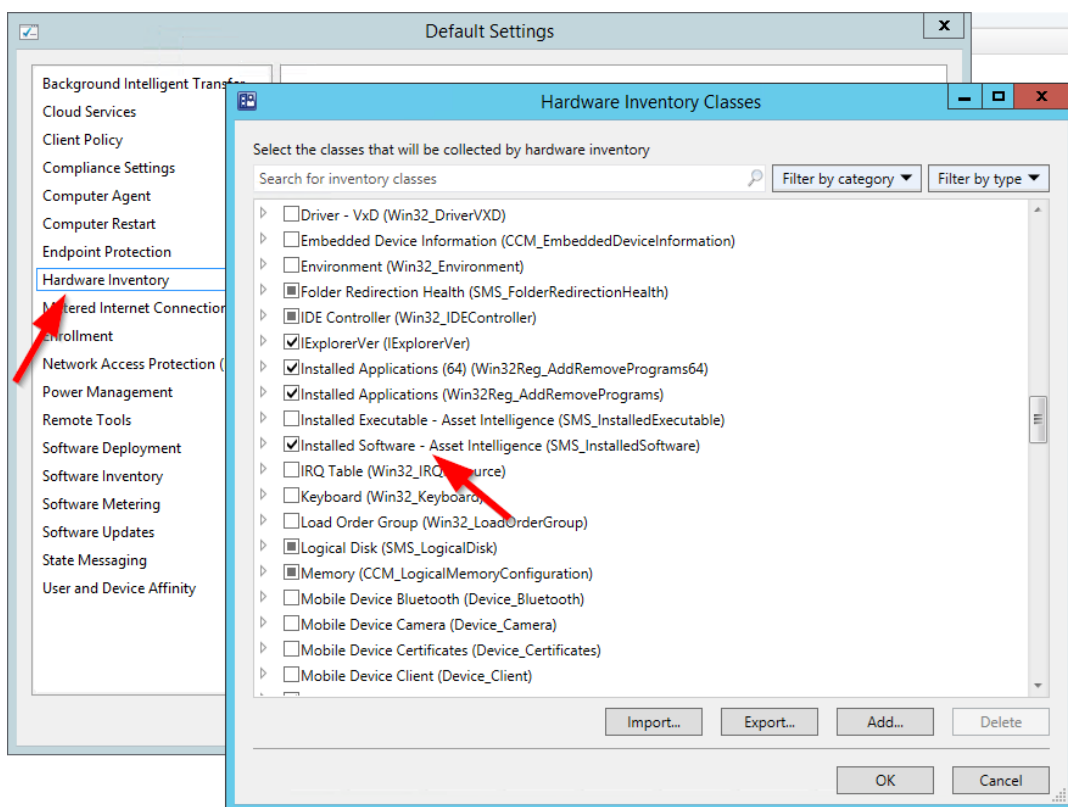
Now you can add the newly-created configuration item to an existing configuration baseline, or you can create a new baseline or baselines targeted at specific collections.

For example, you can create a collection with all Microsoft Office users using a WQL query such as the following:

**select**

**SMS\_R\_SYSTEM.ResourceID,SMS\_R\_SYSTEM.ResourceType,SMS\_R\_SYSTEM.Name,SMS\_R\_SYSTEM.SMS UniquelIdentifier,SMS\_R\_SYSTEM.ResourceDomainORWorkgroup,SMS\_R\_SYSTEM.Client from SMS\_R\_System inner join SMS\_G\_System\_INSTALLED\_SOFTWARE on SMS\_G\_System\_INSTALLED\_SOFTWARE.ResourceID = SMS\_R\_System.ResourceId where SMS\_G\_System\_INSTALLED\_SOFTWARE.ProductName like "Microsoft Office Enterprise%" or SMS\_G\_System\_INSTALLED\_SOFTWARE.ProductName like "Microsoft Office Home and Business%" or SMS\_G\_System\_INSTALLED\_SOFTWARE.ProductName like "Microsoft Office Professional%" or SMS\_G\_System\_INSTALLED\_SOFTWARE.ProductName like "Microsoft Office Home and Student%" or SMS\_G\_System\_INSTALLED\_SOFTWARE.ProductName like "Microsoft Office Standard%"**

Note: to use this WQL query, you will need to enable the **Installed Software** class in your **Client Settings > Hardware Inventory** (it is not enabled by default).



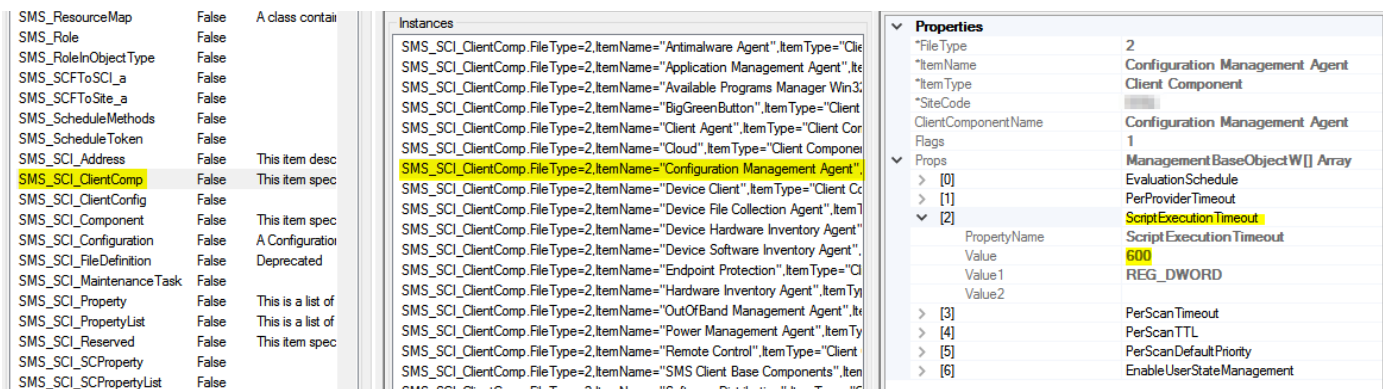
## Changing the Script Execution Timeout Setting

The default value for script execution timeout in SCCM for Compliance Items is 60 seconds. This means that if your script takes longer than 60 seconds to run, it will result in an error and compliance will be incorrectly assessed and reported. The scripts used in this solution may well take longer than 60 seconds, therefore it is recommended to increase this value by editing the site control setting. This can be done in WMI with a VBScript, or PowerShell:

[https://blogs.msdn.microsoft.com/fei\\_xias\\_blog/2013/10/20/system-center-2012-configmgr-using-vbs-to-extend-the-dcm-script-execution-timeout-value/](https://blogs.msdn.microsoft.com/fei_xias_blog/2013/10/20/system-center-2012-configmgr-using-vbs-to-extend-the-dcm-script-execution-timeout-value/)

<https://thedesktopteam.com/raphael/sccm-compliance-setting-script-error-0x87d00321/>

Here I have changed the value to 600 seconds:



## Managing the Custom Template Lifecycle

Your custom templates may get periodically updated, new templates may be added, or existing templates retired. Because of this we need to manage the lifecycle of these templates, and this section describes how that can be done with this solution.

### Adding New Templates

To add a new template, the following procedure should be followed:

1. Create the template files (template, preview image, thumbnail image)
2. Add them to the remote template store in the appropriate location
  - a. If you wish to put the templates in a new location, for example creating a new category, or adding a new subfolder such as "Excel Templates", you can do this. Simply create the required folders in the remote template store and place your templates there. These new folders will be created on the targeted clients the next time the compliance cycle is run. If you create a new category, remember to also create a new XML file for that category.
3. Update the XML file for the category. Add an entry for a new template defining the relevant parameters such as the location of the template file, the title etc.

Any changes to the directory structure or XML files in the remote template store will be detected by the compliance scripts, and the client will be updated. If an XML file is updated, or a new one created, the scripts will delete the "content" key in the HKCU registry to force the Office applications to read the XML file again and create the necessary custom templates in the application, otherwise the application will not reference the new template defined in the XML file.

### Updating a Template

To update a template, simply make the required changes and save the template to the remote template store, overwriting any existing file of the same filename. The compliance scripts will detect that the hash value of the file has changed and copy over the new file to the client.

If you decide to change the filename of the template, remember to also update this in the XML file for that category, otherwise the old filename will still be referenced in the registry and by the Office application.

### Retiring a Template

To retire or remove a template, simply remove the template files from the remote template store. Then also remove the template entry from the XML file for that category. Again the compliance scripts will detect these changes and update the client accordingly.



## Multiple Language Support

This solution will work with a single language version of Microsoft Office deployed in your environment. Some organisations however need to support multiple display languages for Microsoft Office, and this solution could be customised to support that.

The language version of Office you are deploying to is indicated by the **lcid** value in the XML configuration file. An **lcid** value of **1033** for example indicates **US English**. To support additional languages, you would need to create an XML file for each language version you wish to support containing the appropriate **lcid** code. Then save each of your XML files with the language code in the filename, for example:

Engineering\_x64\_1033.xml

Engineering\_x64\_1046.xml

Now you would need to update both the **discovery** and **remediation** scripts for the **User-based setting** to read the installed display language for Office and copy the appropriate XML file.

The following registry key in the current user branch should (in most cases) give you the language version of Office being used on the client:

```
HKCU:\SOFTWARE\MICROSOFT\OFFICE\<OFFICEVERSIONCODE>\COMMON\LANGUAGERESOURCES\
UILANGUAGE
```

Search for the following code lines in the scripts. In the **discovery** script this will be in the **Registry Key Checks** section, in the Providers foreach loop. In the **remediation** script, this will be in the **Registry Key Creation / Cleanup** section, again in the Providers foreach loop.

```
$RemoteXMLFile = (Get-ChildItem "$RemoteTemplatePath\$Provider\XML" -File -
ErrorAction Stop).Name | where {$_ -match $Architecture}
```

Just above this line, put the following code:

```
$UILanguage = (Get-ItemProperty -Path
"HKCU:\SOFTWARE\Microsoft\Office\$OfficeVersionCode\Common\LanguageResources"
-Name UILanguage).UILanguage
```

Then change the first line to include a filter for the UILanguage, eg:

```
$RemoteXMLFile = (Get-ChildItem "$RemoteTemplatePath\$Provider\XML" -File -
ErrorAction Stop).Name | where {$_ -match $Architecture -and $_ -match
$UILanguage}
```

This will detect the UI language version of Office, then find the XML file in the remote template store that contains this language code in the filename. This XML file will then be set in the registry as the ServiceURL key for the relevant provider (category).

Another way to do it would be to create a new remote template store for each language you need to support. Then create a new Compliance Item for each language, specifying the path to the relevant remote template store for that language in the PowerShell scripts. Then deploy each Compliance Item in separate Compliance Baselines which are targeted at collections where that Office language version is installed.