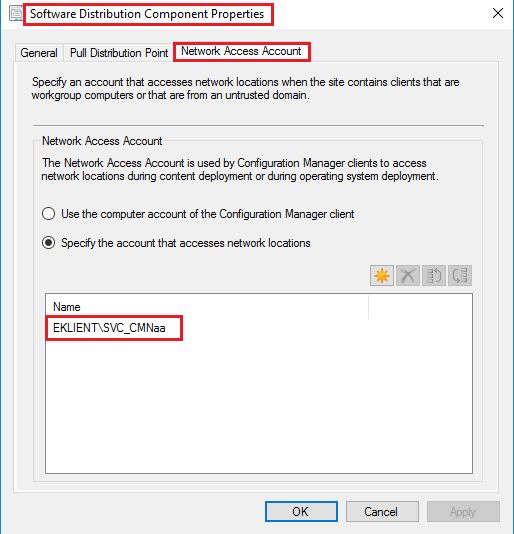
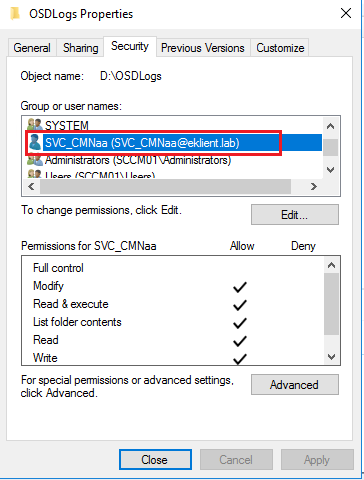
The script “Copy\_CM\_OSD\_Logs.ps1.ps1” can be run anywhere in an OSD TS but is most often used in the Error Section, thus only run in case of a failed deployment.

There are a couple of pre-requisites to make it work:

* PowerShell Optional Component in Boot image.
* We need a location (file share) to save the logs.
* A TS Variable holding the UNC path to the share.
* The “First” Network Access Account must be granted “Modify” permissions on the share.

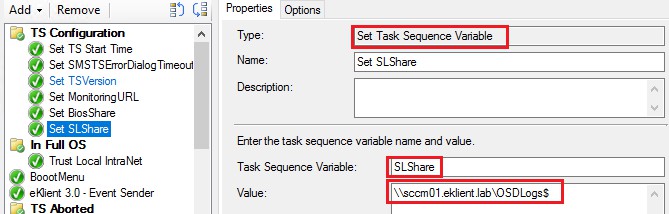
The script will use the Network Access Account for authentication; making it work also in the event of a failure during Windows PE, where we cannot use the computer account, as the machine is not yet domain joined

Check the name of your “first” NAA, if you have several it should be the one on top.

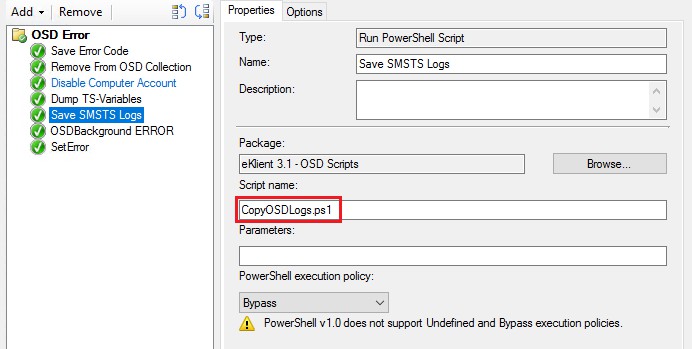


Make sure the Account has been granted "Modify" permissions on your log share:

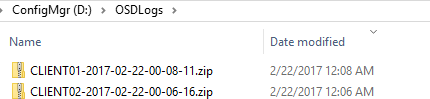
Create a TS Variable "SLShare" and assign it the UNC-Path to your log share:



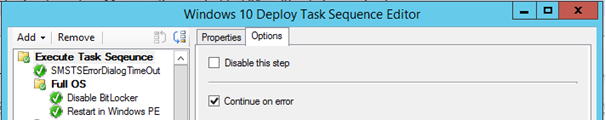
Create a Package (without program) or put the script in an existings scripts package, incorporate in TS as:



In the event of a failure the SMSTSLogs folder will now get zipped and stored as a single file on your log share:



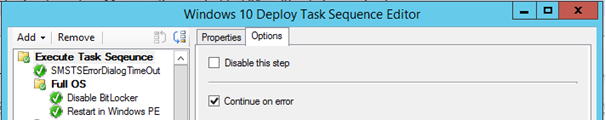
# Error handling in TS without MDT using



I promised last week I would write a blog post on how I use OSDbackground in case of a Task sequence failure, so here it is. OSDbackground gives us the possibility to show an error in case a Task Sequence fails and when doing so it also provides us with the option to read all TS variables and open a command Prompt or CMtrace without having F8 Support enabled.

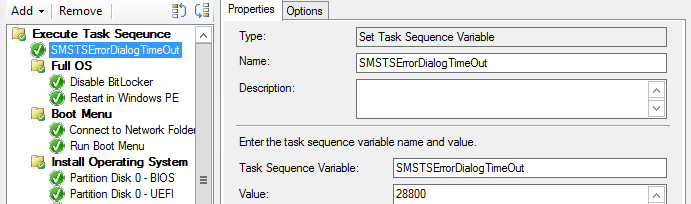
Well, to handle errors in a Task Sequence I use a couple of small scripts that I will describe here and that can be downloaded further down in the blog post. Basically, the functionality is the same as in an MDT integrated Task Sequence.

We start by using a group called “Execute Task Sequence” with the “continue on” error option selected.

[](https://ccmexec.com/wp-content/uploads/2016/12/TS-error-handling.PNG)

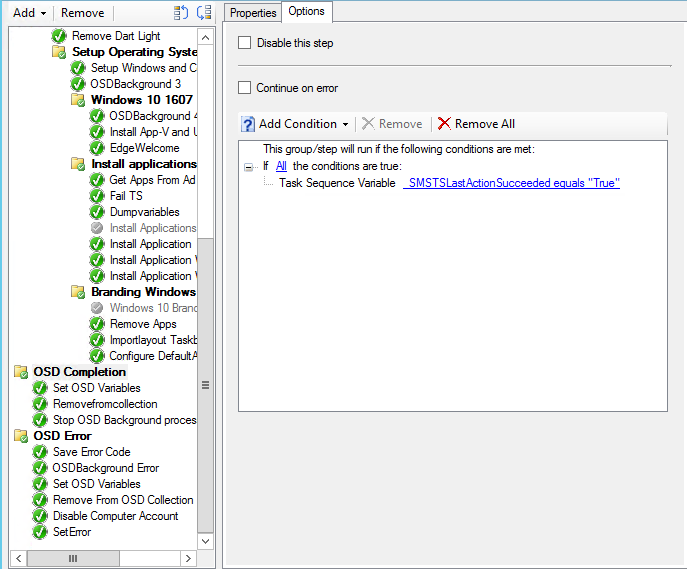
Then we run our whole Task Sequence within that group, in that way we can catch any error in a group later in the Task Sequence.

I also set the variable shown above “SMSTSErrorDialogTimeOut” to “28800” which equals 8 hours. If the task sequence fails, the countdown timer will count down from 8 hours before restarting.

[](https://ccmexec.com/wp-content/uploads/2016/12/SMSTSErrorDialogTimeOut.PNG)

In the end of the Task Sequence we have two groups, “**OSD Completion**” and “**OSD Error**”.

The OSD Completion group is run as the name indicates when the Task Sequence is successful, using the Task Sequence variable “\_SMSTSLastActionSucceeded” = “True”. We remove the computer from the OSD Collection using the Onevinn WebService in this group as well.

[](https://ccmexec.com/wp-content/uploads/2016/12/OSDCompletionGroup-.PNG)

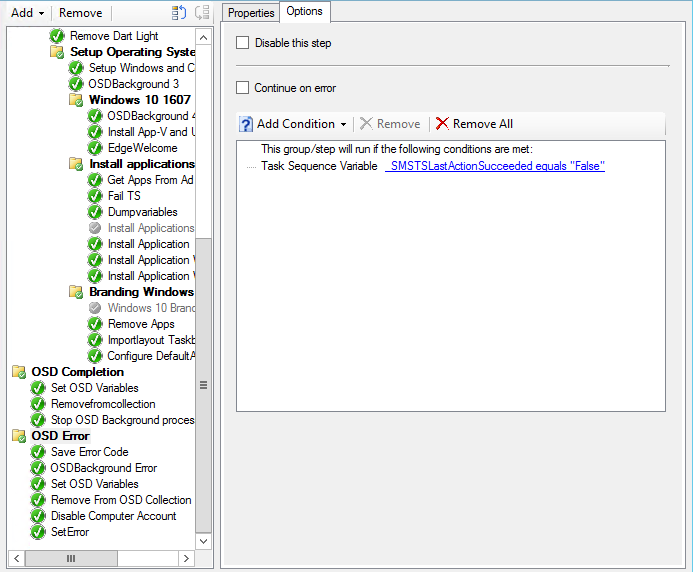
I also added a little step to stop the OSDBackground process, if we don’t restart the computer after we started OSDBackground the last time it will still show as desktop background when the user logs on.

The Powershell script used looks like this.

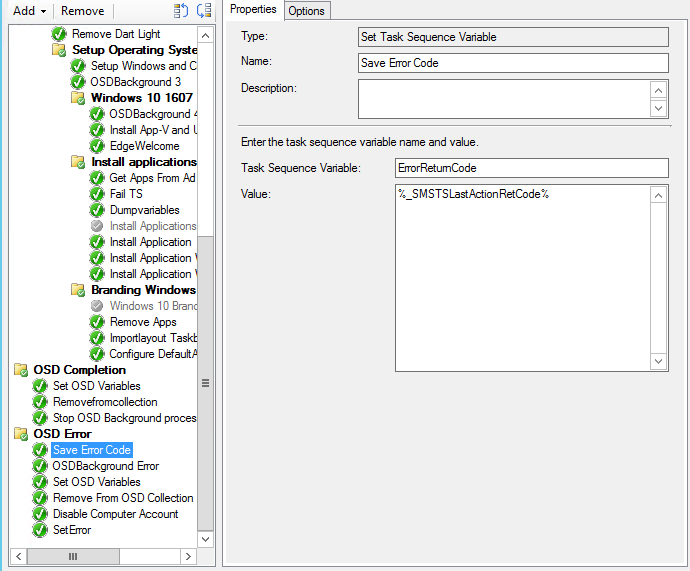
Stop-Process -Name “OSDBackground” -Force -ErrorAction SilentlyContinue

**What if the Task Sequence fails?**

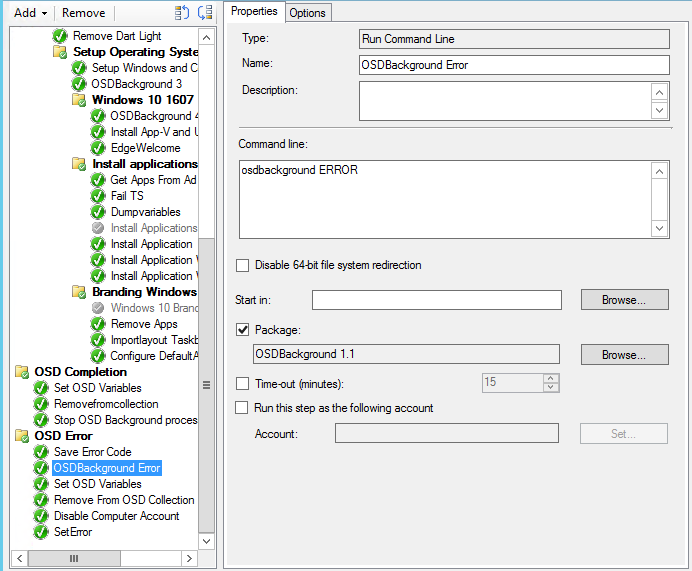
In the OSD Error group we have a couple of interesting steps as well. The OSD Error code has the following condition. Using the same Task Sequence variable as before when there is an error in the Task Sequence, “\_SMSTSLastActionSucceeded” = “False”.

[](https://ccmexec.com/wp-content/uploads/2016/12/OSDErrorGroup.PNG)

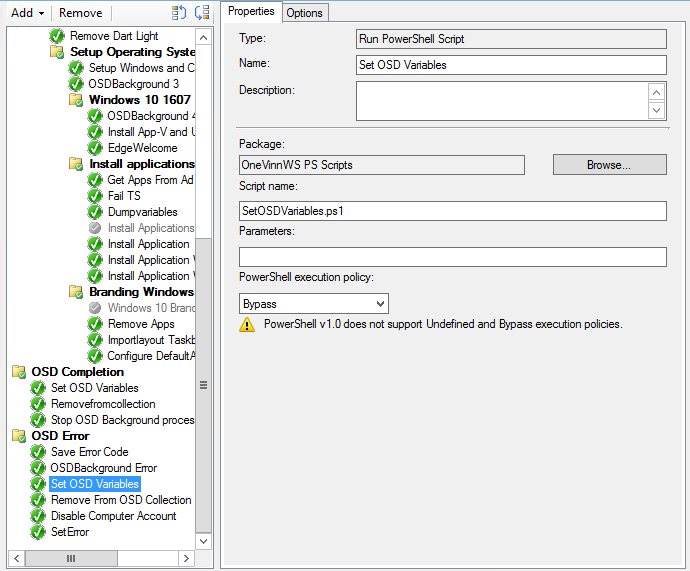
The next step saves the error code from the Task Seqeunce step that actually failed in a Task Sequence variable called “ErrorReturnCode”. We use that later to actually fail the Task Sequence using a script but with the original error code.

[](https://ccmexec.com/wp-content/uploads/2016/12/SaveErrorCode.PNG)

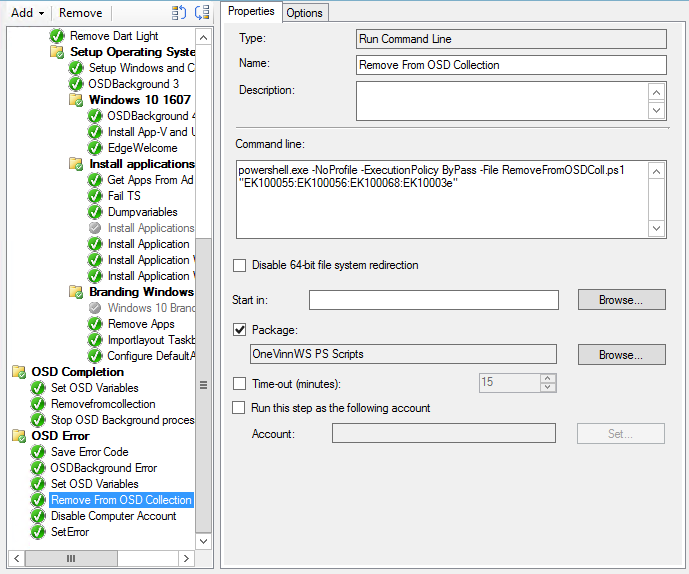
The next step uses OSDBackground to change the Background image and enable us to open a password protected debug mode with ,command prompt support without having F8 enabled.

[](https://ccmexec.com/wp-content/uploads/2016/12/OSDBackgroundError.PNG)

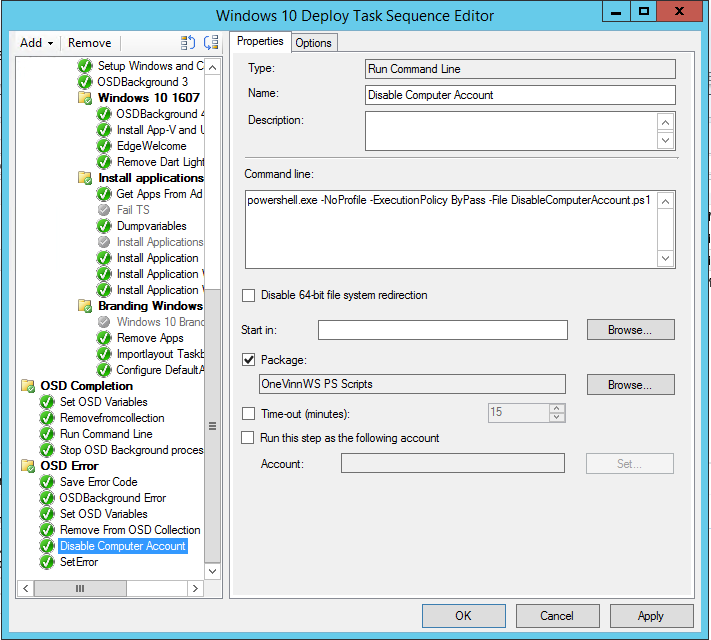
The next three steps are from the sample scripts in the Onevinn OSD WebService, and the first one sets a couple of variables we need to be able to remove it from the Collection used to target the OS deployment.

[](https://ccmexec.com/wp-content/uploads/2016/12/SetTSVariable.PNG)

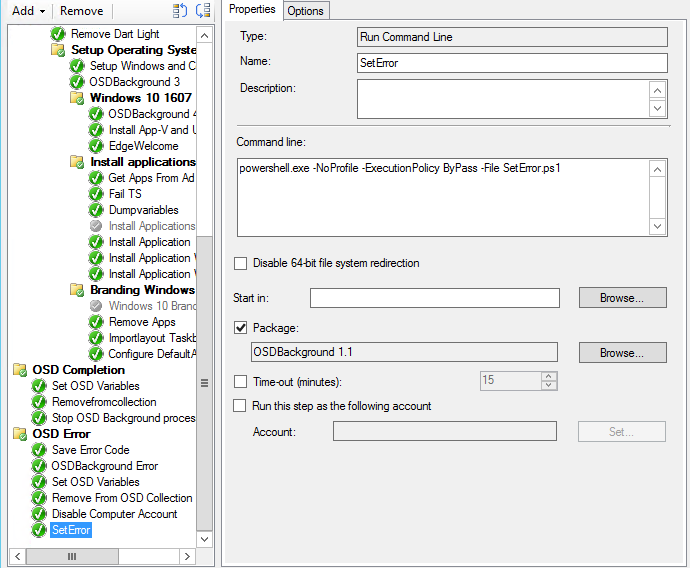
The next step remove the computer from the OSD Collection.

[](https://ccmexec.com/wp-content/uploads/2016/12/RemoveFromOSDCollection.PNG)

We can then disable the computer account using the Web Service in the domain to make sure no one uses a computer with a failed OS deployment potentially missing anti-virus and much more.

[](https://ccmexec.com/wp-content/uploads/2016/12/Disable-Computer-account.PNG)

Then we use a small script that will fail the Task Sequence with the original error code that we saved in the variable before.

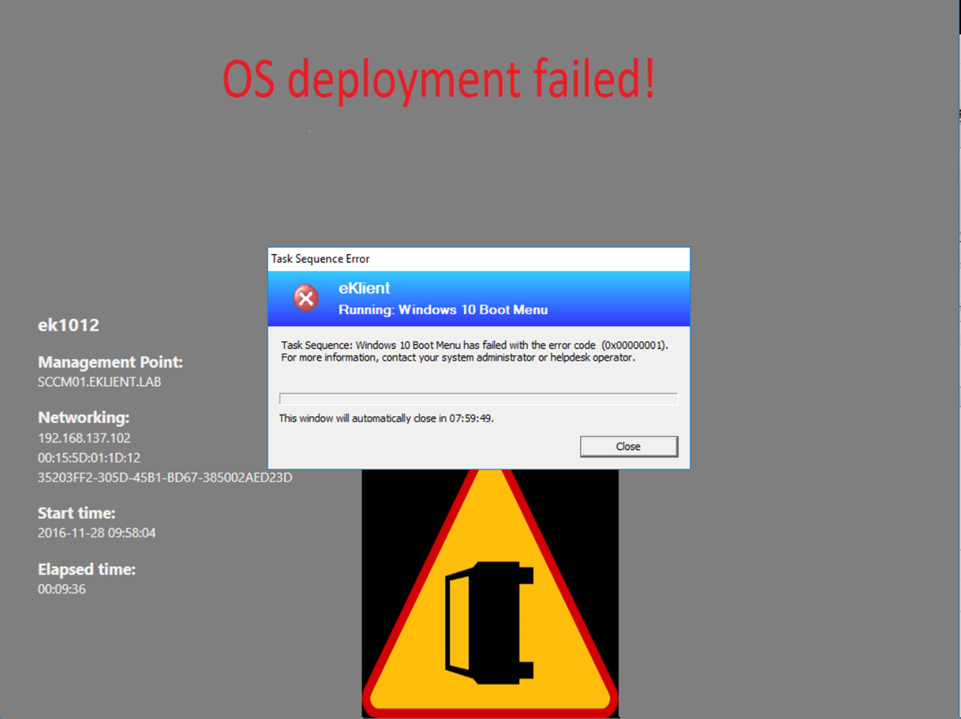
[](https://ccmexec.com/wp-content/uploads/2016/12/SetErrorCode.PNG)

The script used looks like this:

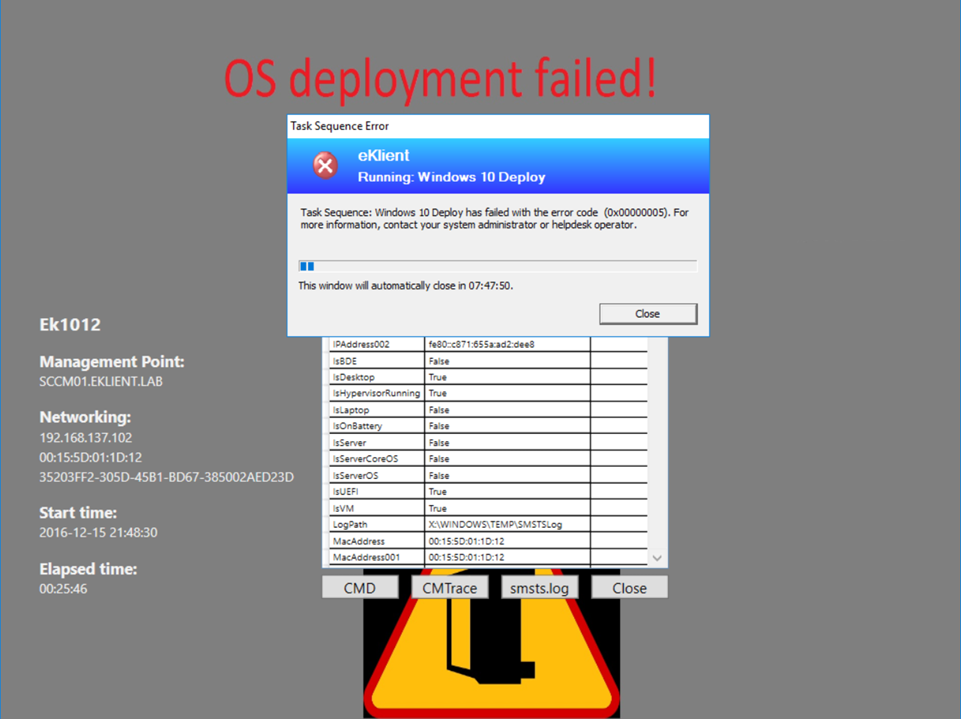
**$tsenv = New-Object -ComObject Microsoft.SMS.TSEnvironment**

**exit $tsenv.Value(“ErrorReturnCode”)**

The result is a Task Sequence that will end up with this dialog when it fails.



Now we can right-click in the upper left corner and supply the configured password to open the debug options in OSDBackground and troubleshoot our Task Sequence error without having F8 enabled in our boot image.



OSDBackground was updated on Technet yesterday as well, so if you don’t have CMtrace in your image, you can just copy Cmtrace.exe to the OSDBackground package and it will copy CMtrace to the local drive so it can be used to read the log files. A great addition by Johan!  
The two scripts used can be downloaded here and I would add them to the OSDBackground package so we can run them from the same package in our Task Sequence.