# Guide for Upload-IntuneWin.ps1 script process

The Upload-IntuneWin.ps1 PowerShell script has been coded to simplify the creation and upload of Intune Win32 Client packages. It supports MSI, EXE or PS1 packages and has been designed to take input from an XML answer (config) file, making it repeatable and extensible.

This document serves to highlight how to use the script and accompanying files.

The script requires AzureADPreview PowerShell module to be installed from the PowerShell Gallery (using the Install-Module -Name AzureADPreview command). Failure to install this module may result in failed operation of the script.

For devices that are running in a restricted mode, using AppLocker – the AppLocker policy must include an EXE publisher rule for the C:\Scripts\IntuneWinAppUtil.exe path. You also need to ensure scripts are allowed to be run from the C:\Scripts path.

# Process Flow

The graphic details the process flow for

using the script:

# Quick-Start

For those who are familiar with application packaging and also with preparing Intune Win32 (.IntuneWin) packages, the following can be used as a quick-start guide. More in-depth walkthroughs are available later in this guide, for this less familiar with this process.

1. Download and extract the Scripts.zip – for example to the root of C:\ drive *(this will automatically create the C:\Scripts folder structure that is required for the correct operation of the Upload-IntuneWin.ps1 script)*. You can extract the file to a path of your choosing – but for the purposes of these instructions we assume you extracted to the root of C:\ drive *(so remember to amend the paths to suit where you extracted to on your computer)*.
2. The Scripts.zip contains the IntuneWinAppUtil.exe tool and several example packages to show the expected folder structure and examples of the different Config.xml files.
3. Review the folder structure for the C:\Scripts\Install-OoBUpdates example first – notice the Source subfolder, which contains the content to be packaged and uploaded to Intune.
4. Review the Config.xml file in the C:\Scripts\Install-OoBUpdates folder and notice the entries that should be modified for future script uploads to Intune – the items to pay particular attention to are:
   1. Username – this must be edited to suit your environment (or removed if you want the script to prompt for this at run-time) – ensure you use an account that has enough rights to create Win32 Client App packages in Intune (such as the Intune Service Administrator role, with active PIM)
   2. AppType – can be MSI, EXE or PS1 but in this example is PS1
   3. installCmdLine – only used for MSI or EXE installs. For MSI – only add custom switches as MSIEXEC /q already added by the script. For EXE add the executable name and any install commands
   4. uninstallCmdLine – as per installCmdLine – only for MSI or EXE
   5. InstallExperience – User or System, depending on how the package should be installed on the client device – the example uses System
   6. PackageName – this must match the name of the object in the source folder, without any file extension – so the name of the MSI/EXE/PS1 file but without the extension
   7. displayName – this is the name that will appear in the Intune console and Company Portal
   8. Description – this is the description that will appear in the Intune console and Company Portal
   9. Category – this is not currently implemented in the script, so you can ignore this for now
   10. Publisher – this is the publisher that will appear in the Intune console and Company Portal
   11. LogoFile (ensure you put the logo.png format file in the same folder as the Config.xml file)
   12. AADGroupName – the script uses the name specified here and automatically creates 3 groups – with Required, Available and Uninstall. It then configures the associated targeting for the object in Intune. If an object is added (or nested) to the Uninstall group, it is automatically excluded from the Required and Available groups. Note: make sure to restrict the total group name length to 55 characters or less. This will avoid script failures.
5. Open a PowerShell console and change to the C:\Scripts folder (type: cd C:\Scripts and then hit return).
6. Run the script with .\Upload-IntuneWin.ps1 and it will prompt for the packagepath (which is the full path to the package to be uploaded – like C:\Scripts\Install-OoBUpdates) and intuneWinAppUtilPath (which is the full path to the IntuneWinAppUtil.exe file – like C:\Scripts\IntuneWinAppUtil.exe)
7. Hit return and the process will run the upload.
8. If you need to re-upload anything – make sure to manually delete the AAD groups and the Intune Win32 object that also gets created.

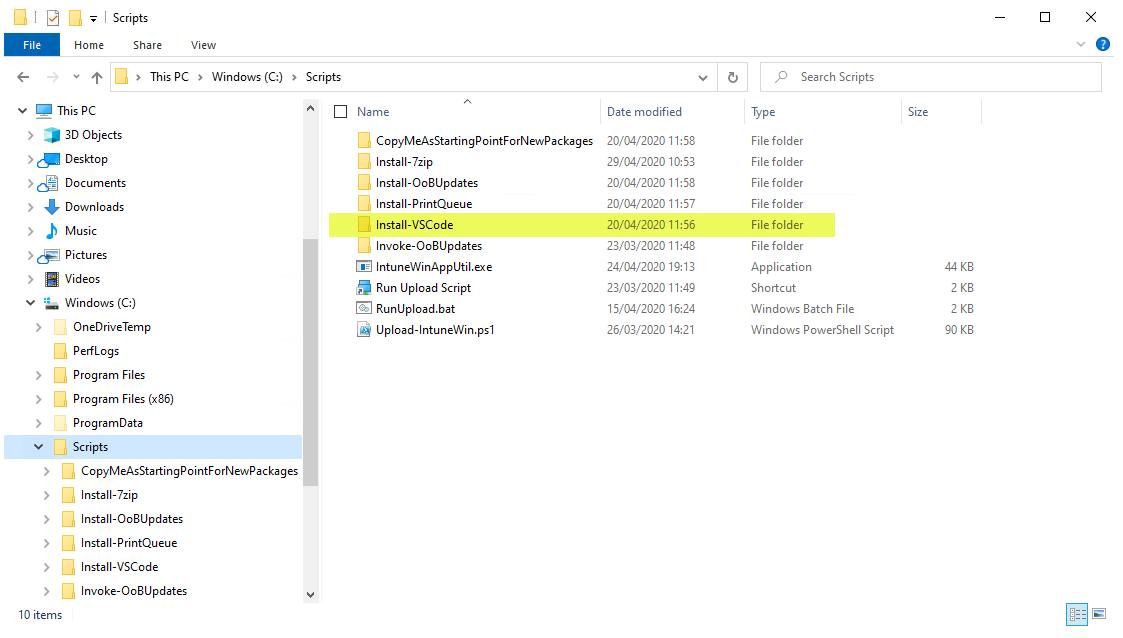
# Example walkthrough for an EXE

Before starting – ensure you have obtained the following items:

* The ‘Scripts.zip’ package
* The source content you want to package up into the resulting Intune Win32 package
* An optional icon file *(saved in .png format)* that will be shown in Company Portal *(and the Intune console)*
* The relevant silent install/uninstall command switches *(which are required so packages can be installed or uninstalled without any user prompts/wizards, as Intune requires all Win32 package installs to happen without any user interaction)*

Note – using an advanced XML Editor, such as [VSCode](https://code.visualstudio.com/) makes editing the Config.xml much easier.

1. Download and extract the Scripts.zip – for example to the root of C:\ drive *(this will automatically create the C:\Scripts folder structure that is required for the correct operation of the Upload-IntuneWin.ps1 script)*. You can extract the file to a path of your choosing – but for the purposes of these instructions we assume you extracted to the root of C:\ drive *(so remember to amend the paths to suit where you extracted to on your computer)*.
2. Copy the ‘CopyMeAsStartingPointForNewPackages’ folder and rename it to suit the package you want to manage.

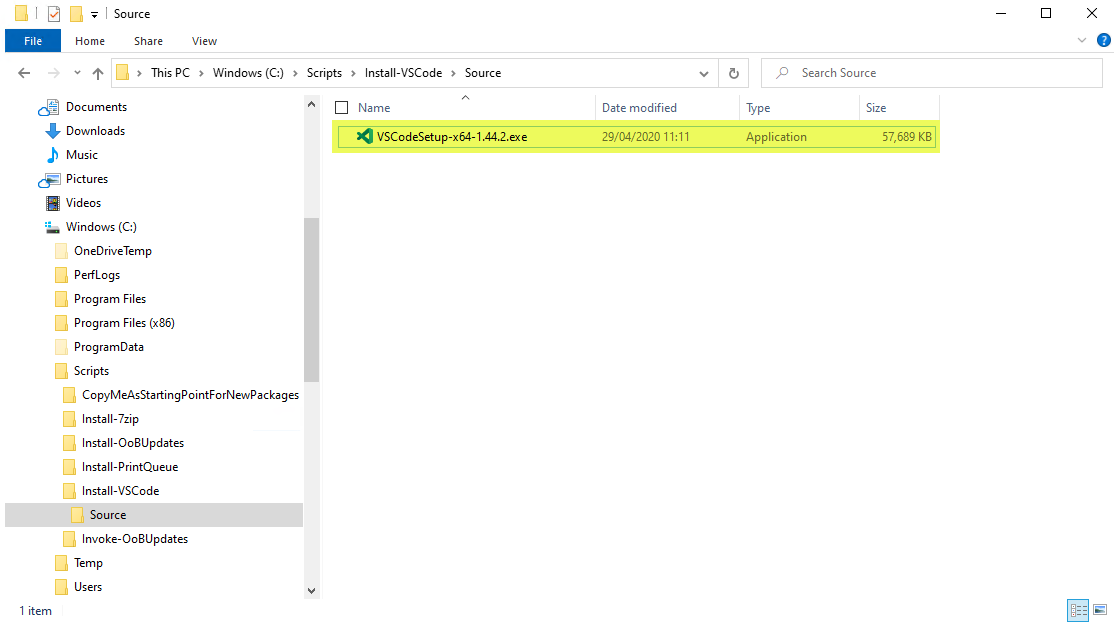
I’ll use the ‘Install-VSCode’ folder as an example:

1. Copy the source content for the package into the \Source subfolder.

For the VSCode example, it looks like this:

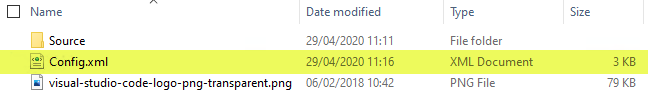
Download the VSCode 64-bit installer and save it to the \Source path



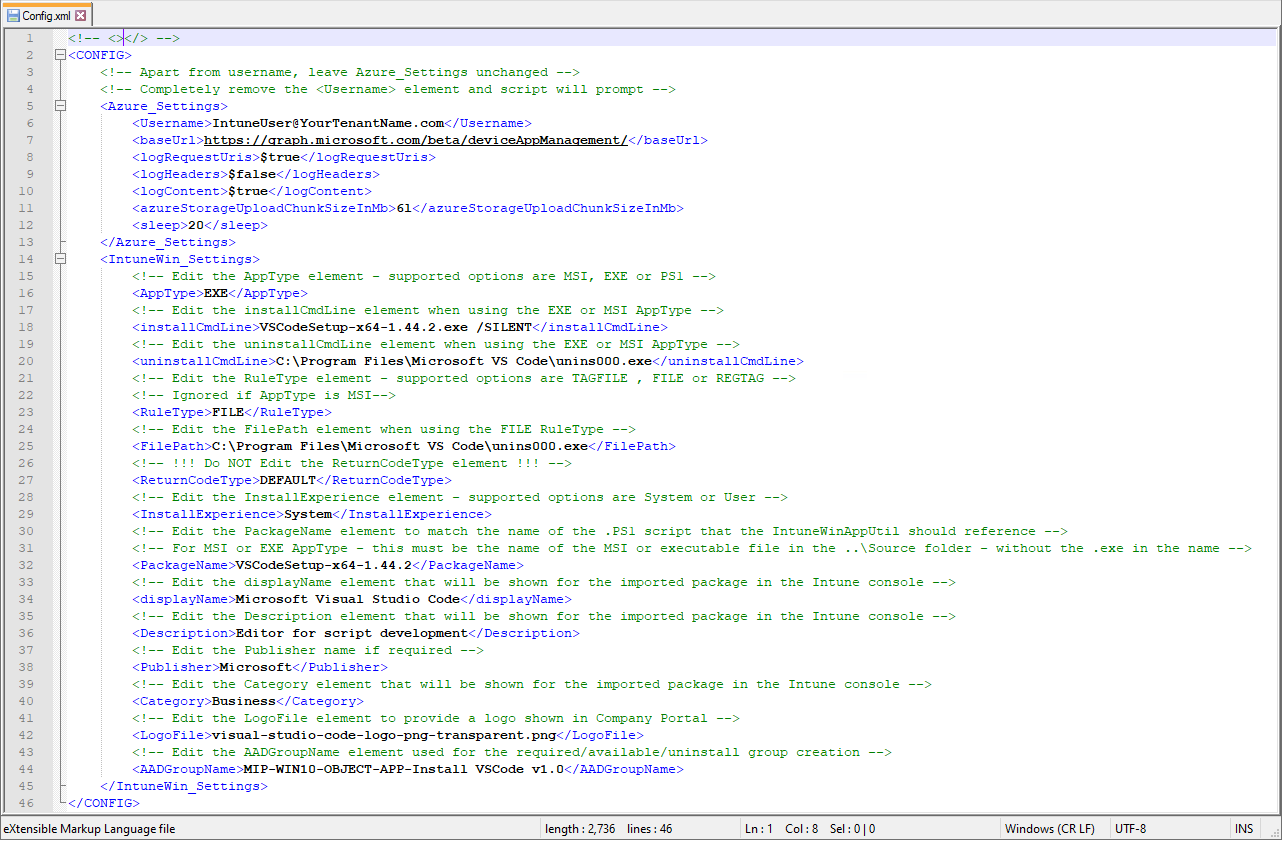


1. Edit the Config.xml file *(at the level above the folder containing the Source)*.

This is what the Install-VSCode example looks like:

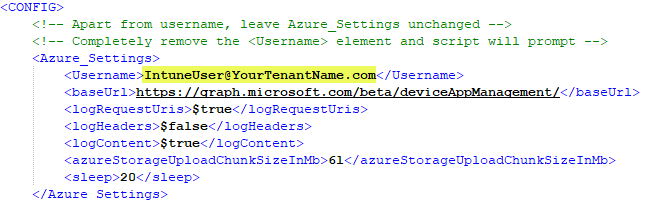


when editing the Config.xml file, it will look similar to this:



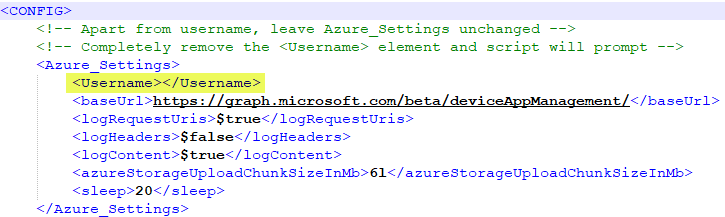
1. First, edit the ‘Username’ field to match an account in your tenant with rights to add new Intune Win32 packages *(you will likely need Intune Administrator rights for this)*.

Note the following highlighted entry confirming what you should edit in your Config.xml:



If you don’t want to specify a username in this Config.xml file – you can leave the field blank and the script will prompt for a username at runtime.

An example without the ‘Username’ field populated, looks like this:



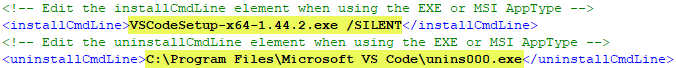
1. Next, edit the ‘AppType’ field to suit the type of package you are installing.

In the VSCode example case, we set the AppType field to EXE *(as the VSCode installer is an executable)*:



1. Then, edit the ‘installCmdLine’ field and the ‘uninstallCmdLine’ field to suit the relevant silent install/uninstall command-line switches.

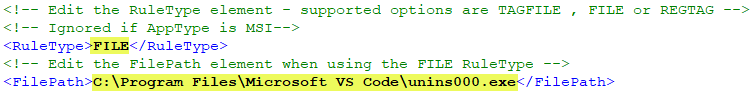
For the VSCode example, it looks like this:



1. Next, edit the ‘RuleType’ field. This controls the detection rule for the package in Intune.

See RuleType in the Appendix for more details.

For the VSCode example, we use FILE and also set the ‘FilePath’ field to contain a path for a file that will only exist once the VSCode package has been successfully installed:



1. Then, if required, adjust the ‘InstallExperience’ field. Most things should be installed/run under SYSTEM context, but you can change this to USER if required. If possible, use SYSTEM context as this avoids the need to develop custom AppLocker or Windows Defender Application Control (WDAC) policy exceptions.

In the VSCode example case, we are using SYSTEM:



1. Next, edit the ‘PackageName’ field – this must match the name of the source package *(or script)* that the IntuneWin package tool uses when it creates the .IntuneWin package *(which is what gets uploaded to Intune)*. You must exclude the .exe or .msi *(or .ps1 if it’s a script)* from the name in the PackageName field.

Our VSCode example looks as follows:

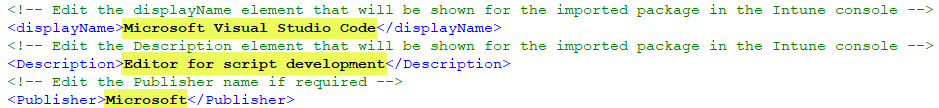


And that is because the name of the file in the \Source folder looks like this:



1. The next 3 fields to edit are related to the way the package is named/displayed in Company Portal *(and also the Intune console)*. ‘displayName’ is how the package name appears, ‘Description’ explains what the package is for and ‘Publisher’ is also displayed for the package accordingly.

For the VSCode example, we used the following:



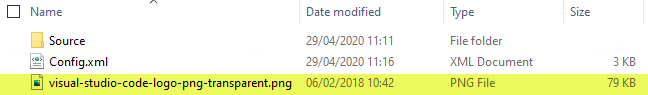
Note – the ‘Category’ field is not yet implemented in the script but must be left in the Config.xml file unchanged.

1. Next, edit the ‘LogoFile’ field to correspond to the (optional) .png logo file to be included in the package. You must store your .png logo file in the same folder as the Config.xml.

Our VSCode example looks like this:

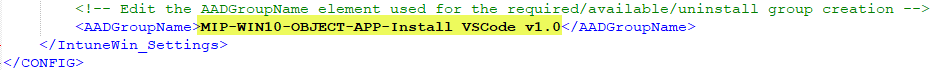


And the folder with the Config.xml and ‘visual-studio-code-logo-png-transparent.png’ logo file looks like this:



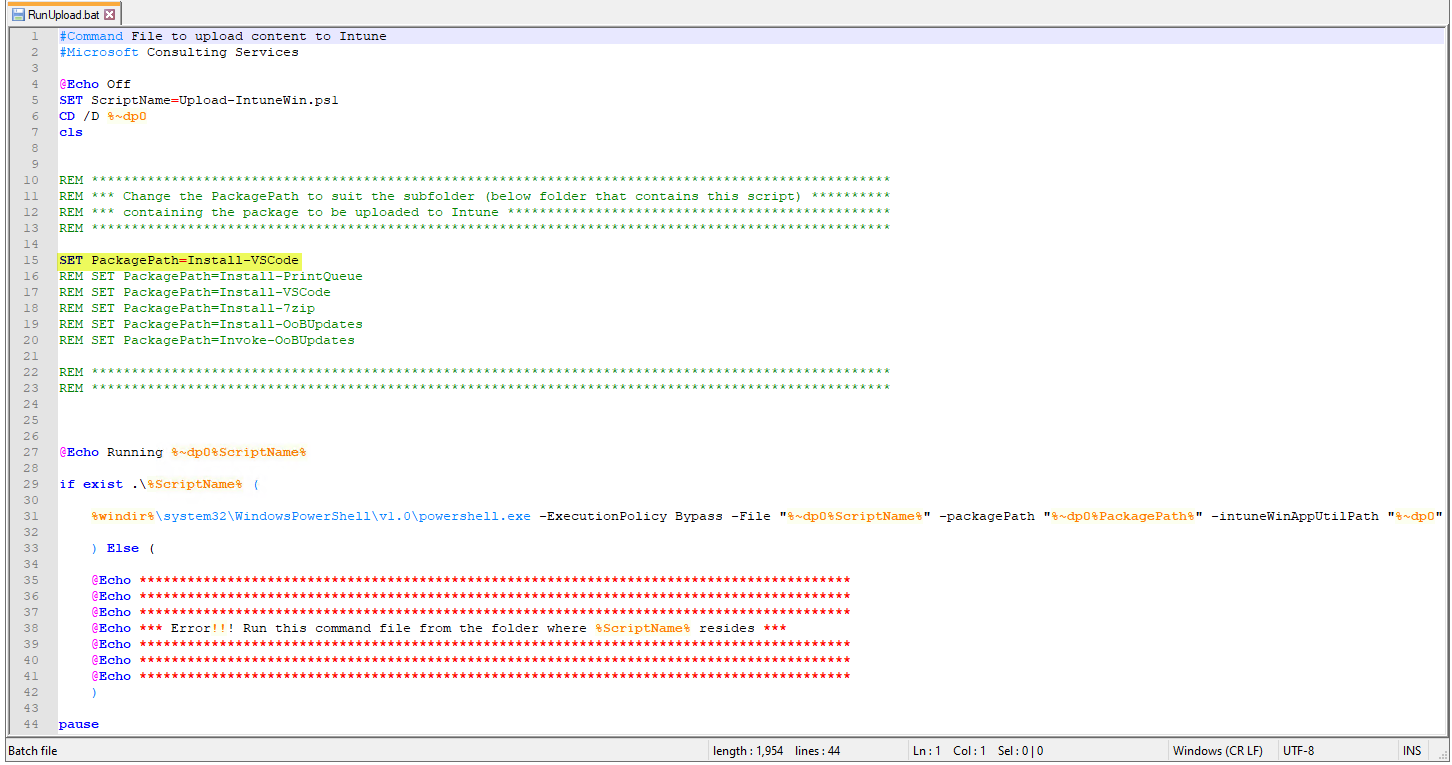
1. The last field to edit is the ‘AADGroupName’ field. This is for the Azure Active Directory group names that will be created. The script creates 3 groups – using the name specified here. The groups use the prefix specified in the ‘AADGroupName’ field – and append ‘-Required’, ‘-Available’ or ‘-Uninstall’ accordingly. These groups control the various deployment targeting groups for the package, once it’s imported into Intune.

Our VSCode example looks like this:

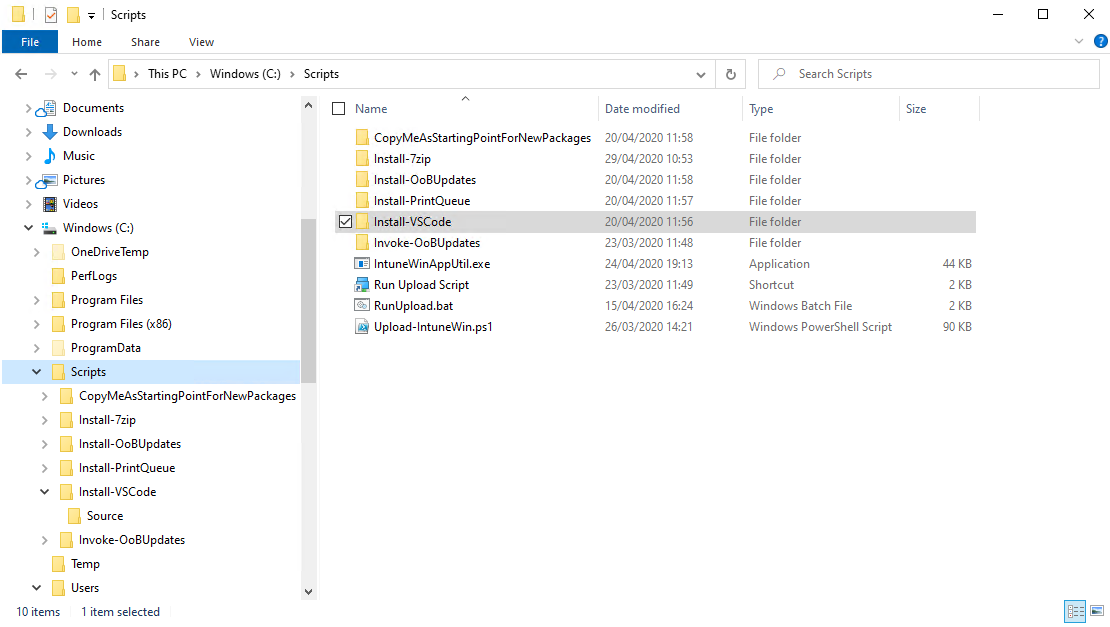


1. Save the changes to the edited Config.xml file
2. Then locate the ‘RunUpload.bat’ file in the C:\Scripts folder and edit the ‘SET PackagePath=’ line (number 15 in the following example) to suit the name of the subfolder containing the package content to be uploaded to Intune.

For the VSCode example, it looks as follows:

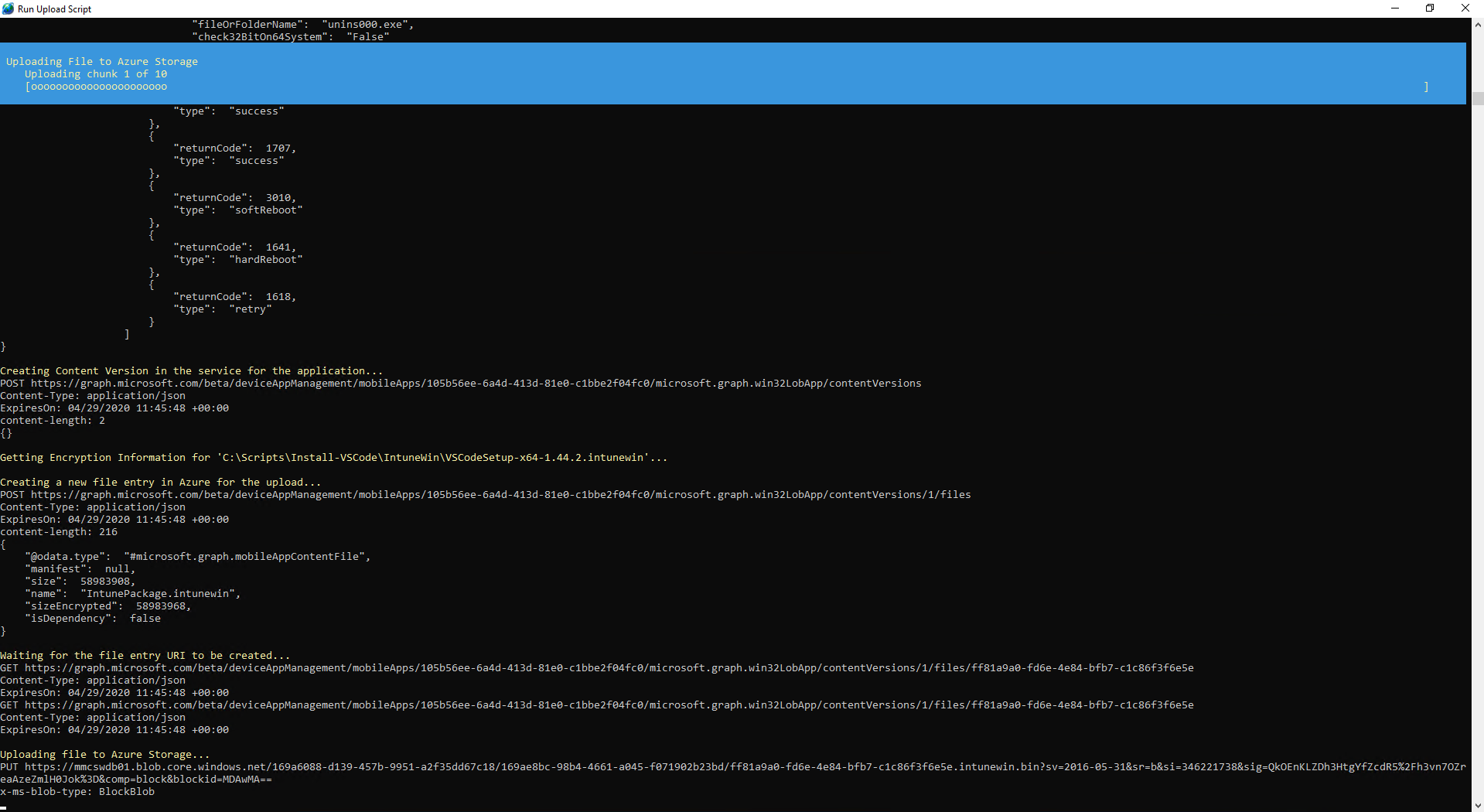


Note – the name used in the ‘SET PackagePath=’ line matches the subfolder name as shown below:



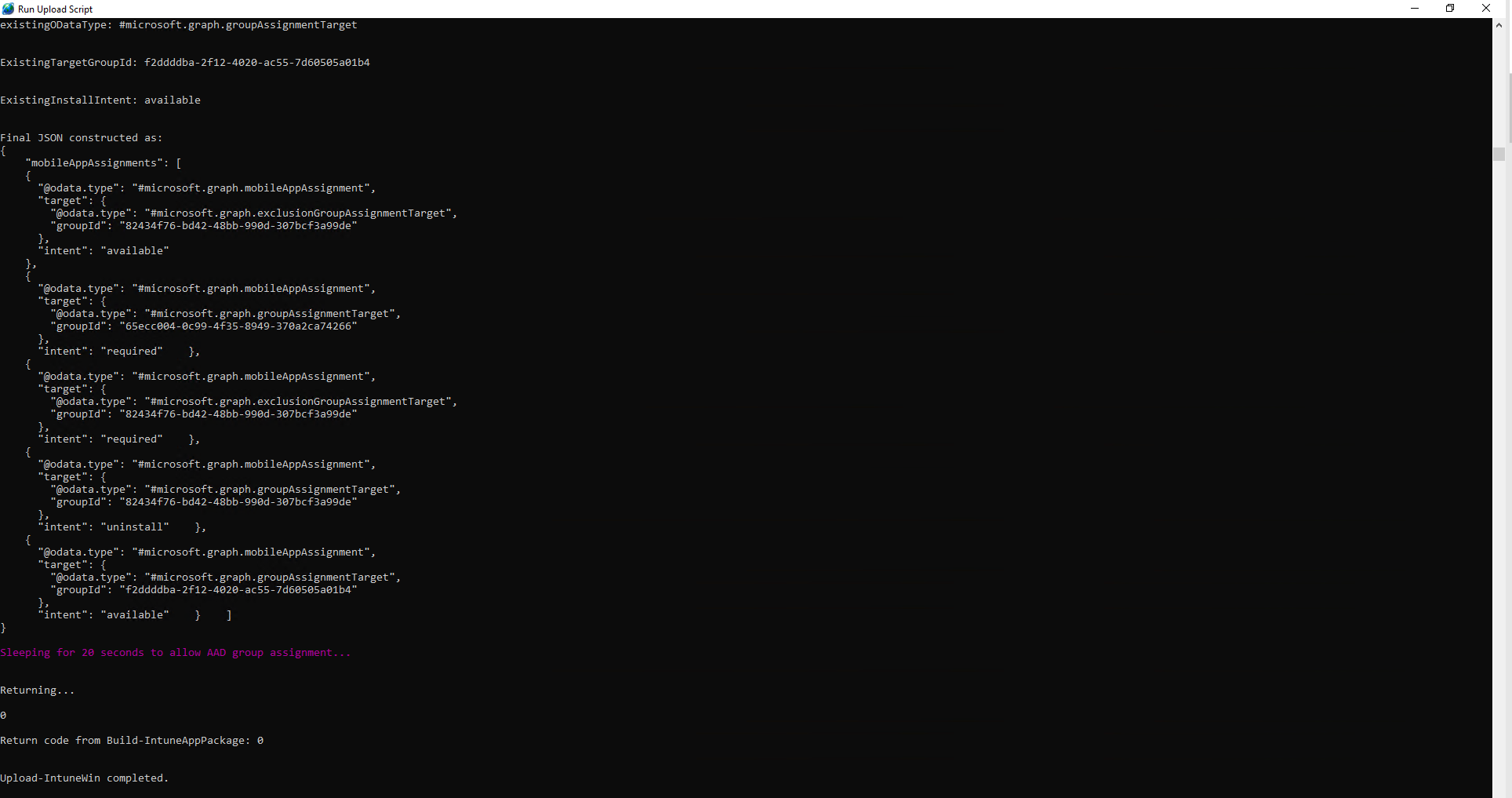
The script automatically appends the full path to the folder, so you only need the subfolder name itself.

1. Double-click on the ‘Run Upload Script’ shortcut file (assuming you extracted to the C:\Scripts path) and this will automatically call the ‘C:\Scripts\RunUpload.bat’ script to run the process:

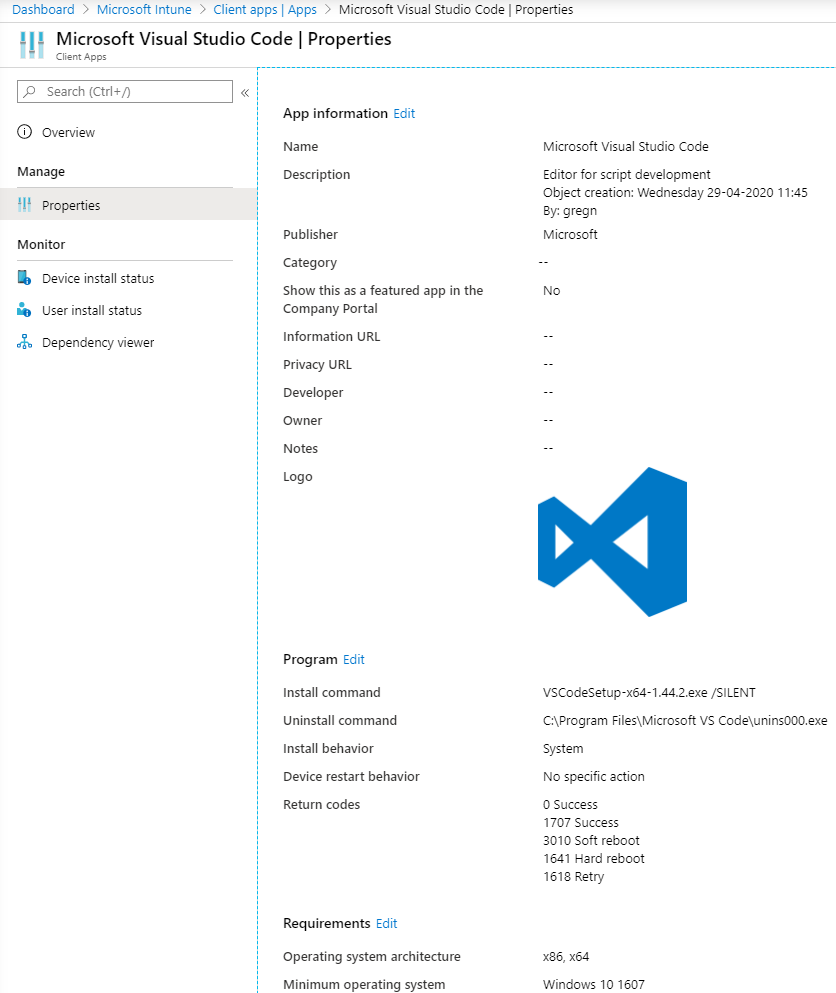


Note you may need to authenticate to Intune if you do not have an active authentication token when you start the upload process.

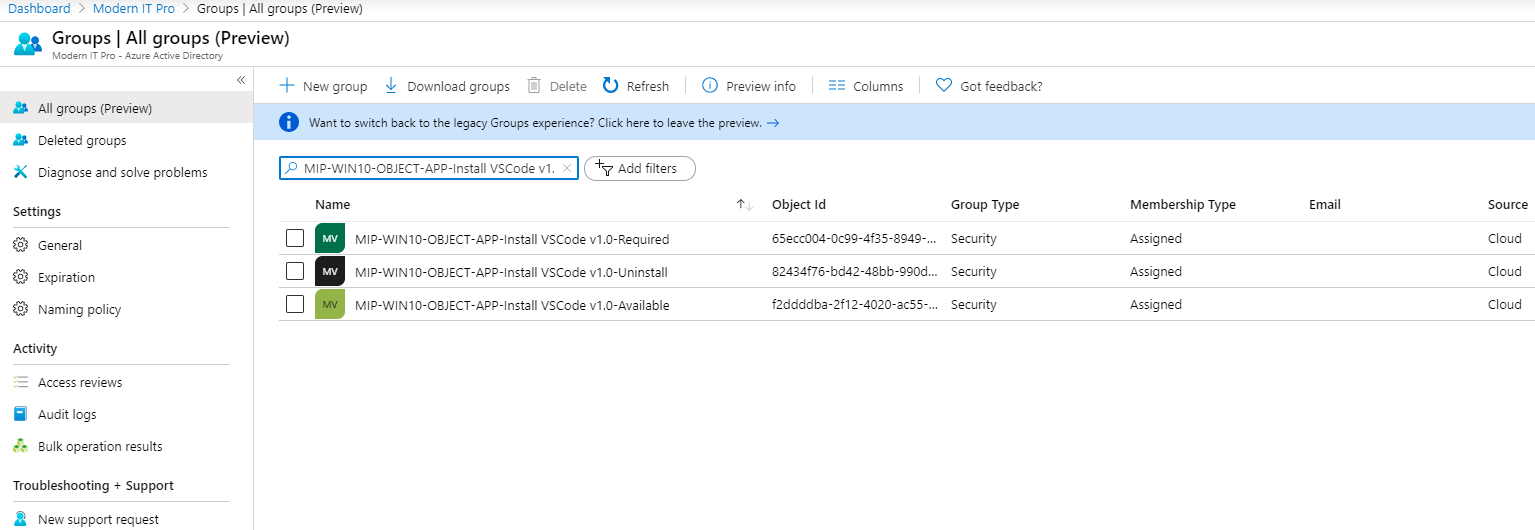
1. The script will take a few minutes to run. Eventually it will look like this:



1. Then you can sign into the Intune portal and locate the package you uploaded.



1. Note the AAD groups that got created too:



1. All you need to do is add users/devices to the relevant AAD groups according to the type of assignment you want (required or available accordingly). You can also use nested (role) groups for assignment too.

# Example walkthrough for an MSI

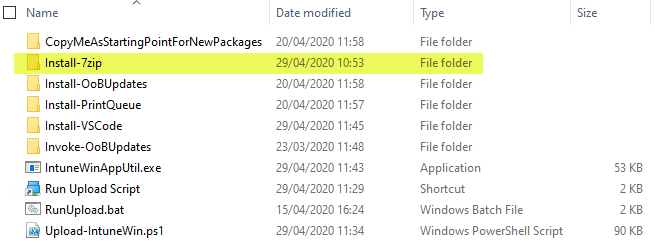
Before starting – ensure you have obtained the following items:

* The ‘Scripts.zip’ package
* The source content you want to package up into the resulting Intune Win32 package
* An optional icon file *(saved in .png format)* that will be shown in Company Portal *(and the Intune console)*
* The relevant silent install/uninstall command switches *(which are required so packages can be installed or uninstalled without any user prompts/wizards, as Intune requires all Win32 package installs to happen without any user interaction)*

Note – using an advanced XML Editor, such as [VSCode](https://code.visualstudio.com/) makes editing the Config.xml much easier.

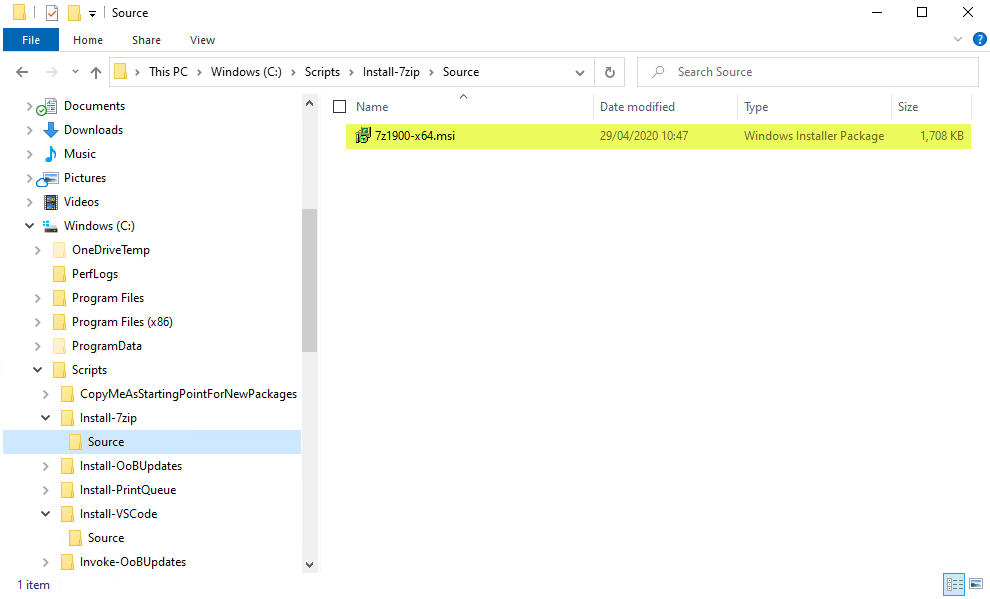
1. Download and extract the Scripts.zip – for example to the root of C:\ drive (this will automatically create the C:\Scripts folder structure that is required for the correct operation of the Upload-IntuneWin.ps1 script). You can extract the file to a path of your choosing – but for the purposes of these instructions we assume you extracted to the root of C:\ drive (so remember to amend the paths to suit where you extracted to on your computer).
2. Copy the ‘CopyMeAsStartingPointForNewPackages’ folder and rename it to suit the package you want to manage.

I’ll use the ‘Install-7zip’ folder as an example:



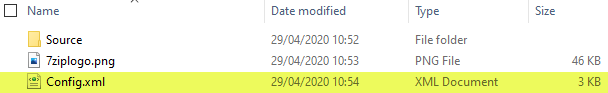
1. Copy the source content for the package into the \Source subfolder.

For the 7zip’ example, it looks like this:

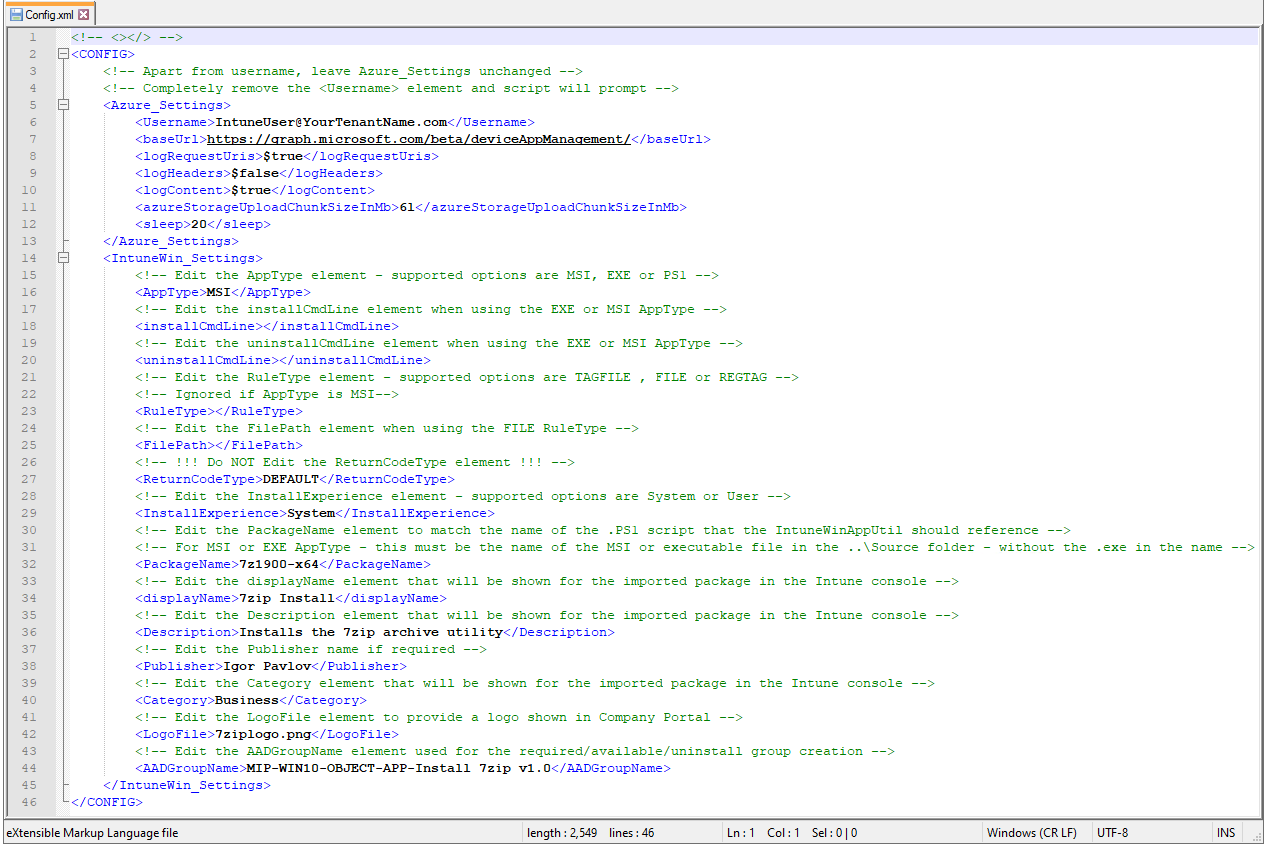


1. Edit the Config.xml file *(at the level above the folder containing the Source)*.

This is what the Install-7zip example looks like:

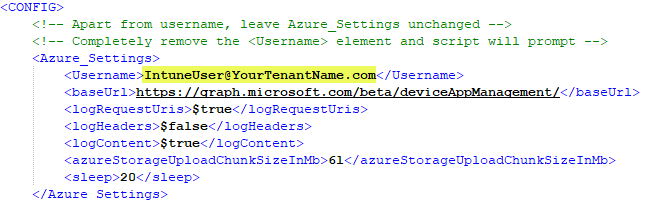


when editing the Config.xml file, it will look similar to this:



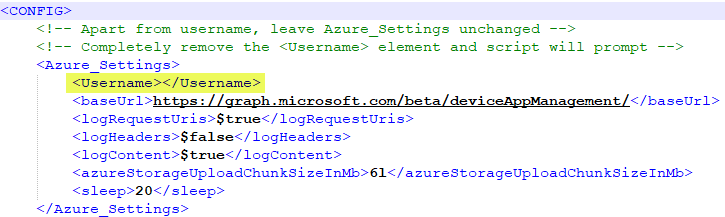
1. First, edit the ‘Username’ field to match an account in your tenant with rights to add new Intune Win32 packages *(you will likely need Intune Administrator rights for this)*.

Note the following highlighted entry confirming what you should edit in your Config.xml:



If you don’t want to specify a username in this Config.xml file – you can leave the field blank and the script will prompt for a username at runtime.

An example without the ‘Username’ field populated, looks like this:



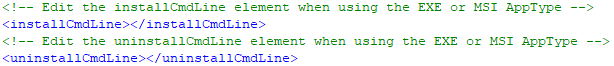
1. Next, edit the ‘AppType’ field to suit the type of package you are installing.

In the 7zip example case, we set the AppType field to MSI *(as the Zscaler installer is an MSI)*:



1. Then, edit the ‘installCmdLine’ field and the ‘uninstallCmdLine’ field to suit the relevant silent install/uninstall command-line switches.

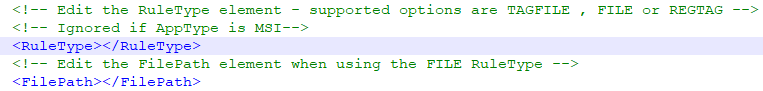
For the 7zip example, it looks like this (note that you don’t need to include the MSIEXEC /I but you may need to add additional switches depending on the package you are installing):



1. Next, note the ‘RuleType’ field. This controls the detection rule for the package in Intune.

See RuleType in the Appendix for more details.

For the 7zip example, we use do not set a RuleType or the ‘FilePath’ fields (as the MSI contains this info already). So just leave both of these fields blank when the package is an MSI – like this:



1. Then, if required, adjust the ‘InstallExperience’ field. Most things should be installed/run under SYSTEM context, but you can change this to USER if required.

In the 7zip example case, we are using SYSTEM:



1. Next, edit the ‘PackageName’ field – this must match the name of the source package *(or script)* that the IntuneWin package tool uses when it creates the .IntuneWin package *(which is what gets uploaded to Intune)*. You must exclude the .exe or .msi *(or .ps1 if it’s a script)* from the name in the PackageName field.

Our 7zip example looks as follows:

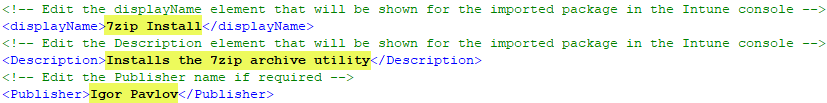


And that is because the name of the file in the \Source folder looks like this:



1. The next 3 fields to edit are related to the way the package is named/displayed in Company Portal *(and also the Intune console)*. ‘displayName’ is how the package name appears, ‘Description’ explains what the package is for and ‘Publisher’ is also displayed for the package accordingly.

For the 7zip example, we used the following:



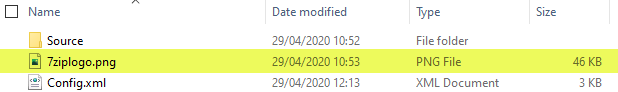
Note – the ‘Category’ field is not yet implemented in the script but must be left in the Config.xml file unchanged.

1. Next, edit the ‘LogoFile’ field to correspond to the (optional) .png logo file to be included in the package. You must store your .png logo file in the same folder as the Config.xml.

Our 7zip example looks like this:

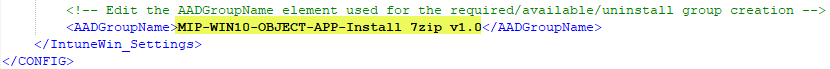


And the folder with the Config.xml and ‘7ziplogo.png’ logo file looks like this:



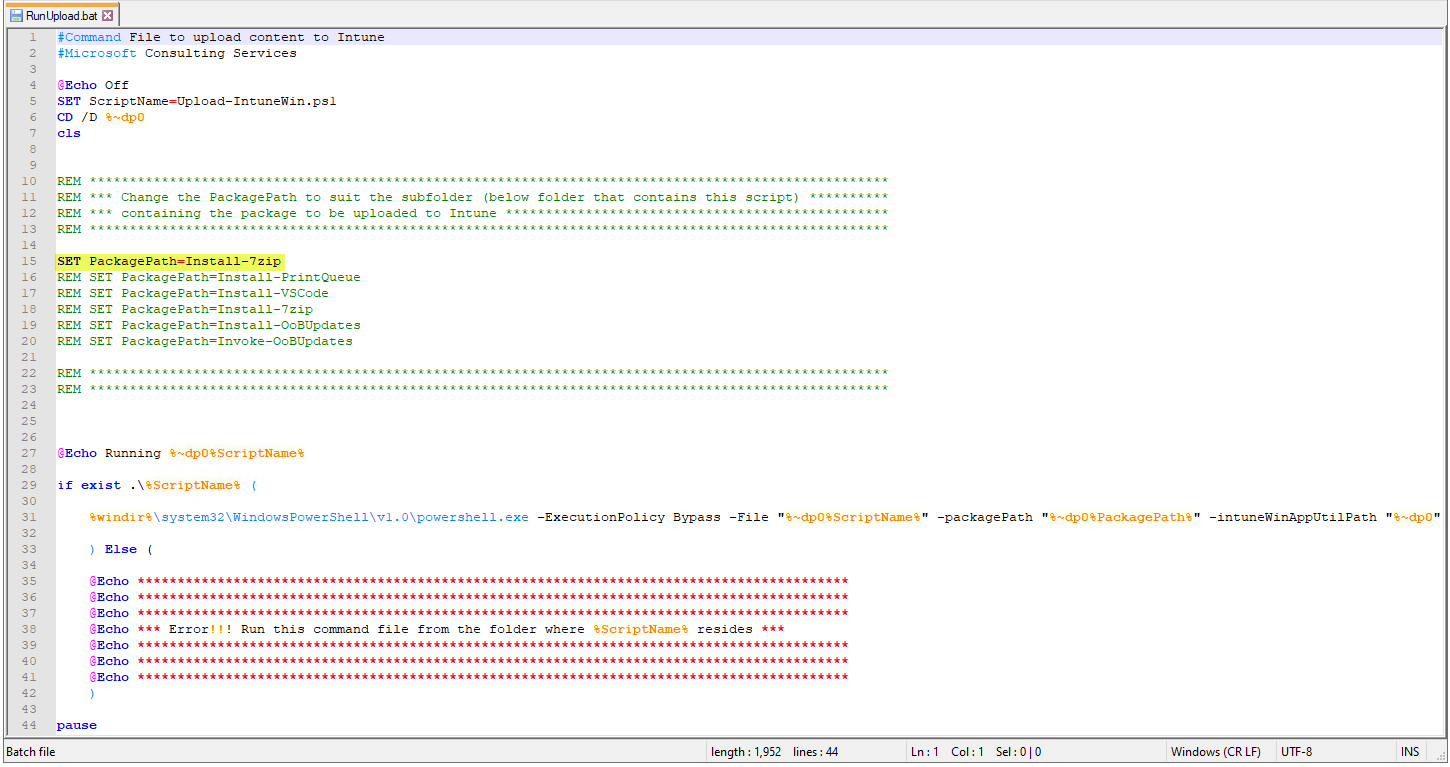
1. The last field to edit is the ‘AADGroupName’ field. This is for the Azure Active Directory group names that will be created. The script creates 3 groups – using the name specified here. The groups use the prefix specified in the ‘AADGroupName’ field – and append ‘-Required’, ‘-Available’ or ‘-Uninstall’ accordingly. These groups control the various deployment targeting groups for the package, once it’s imported into Intune.

Our 7zip example looks like this:

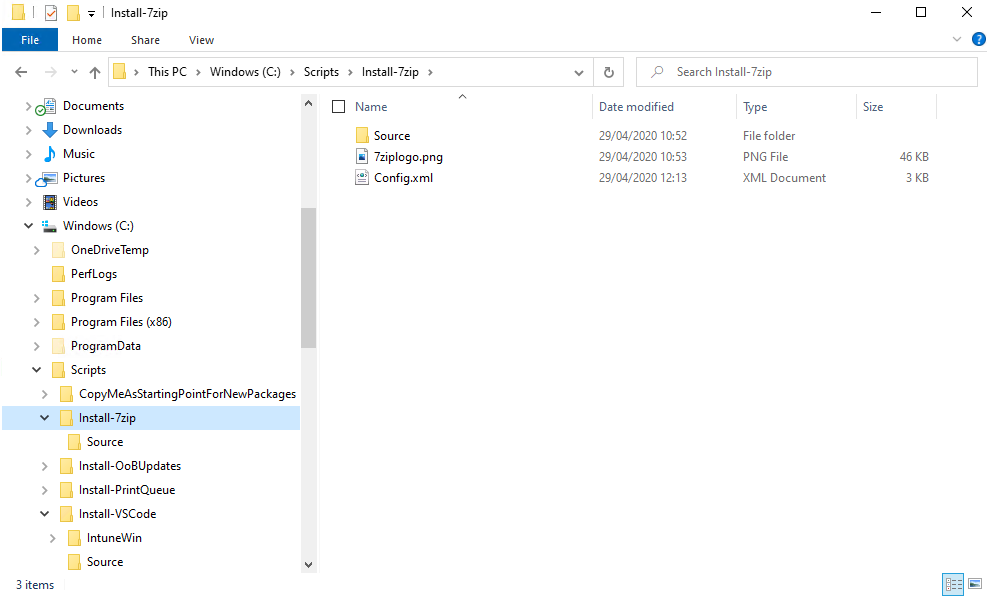


1. Save the changes to the edited Config.xml file
2. Then locate the ‘RunUpload.bat’ file in the C:\Scripts folder and edit the ‘SET PackagePath=’ line (number 15 in the following example) to suit the name of the subfolder containing the package content to be uploaded to Intune.

For the 7zip example, it looks as follows:

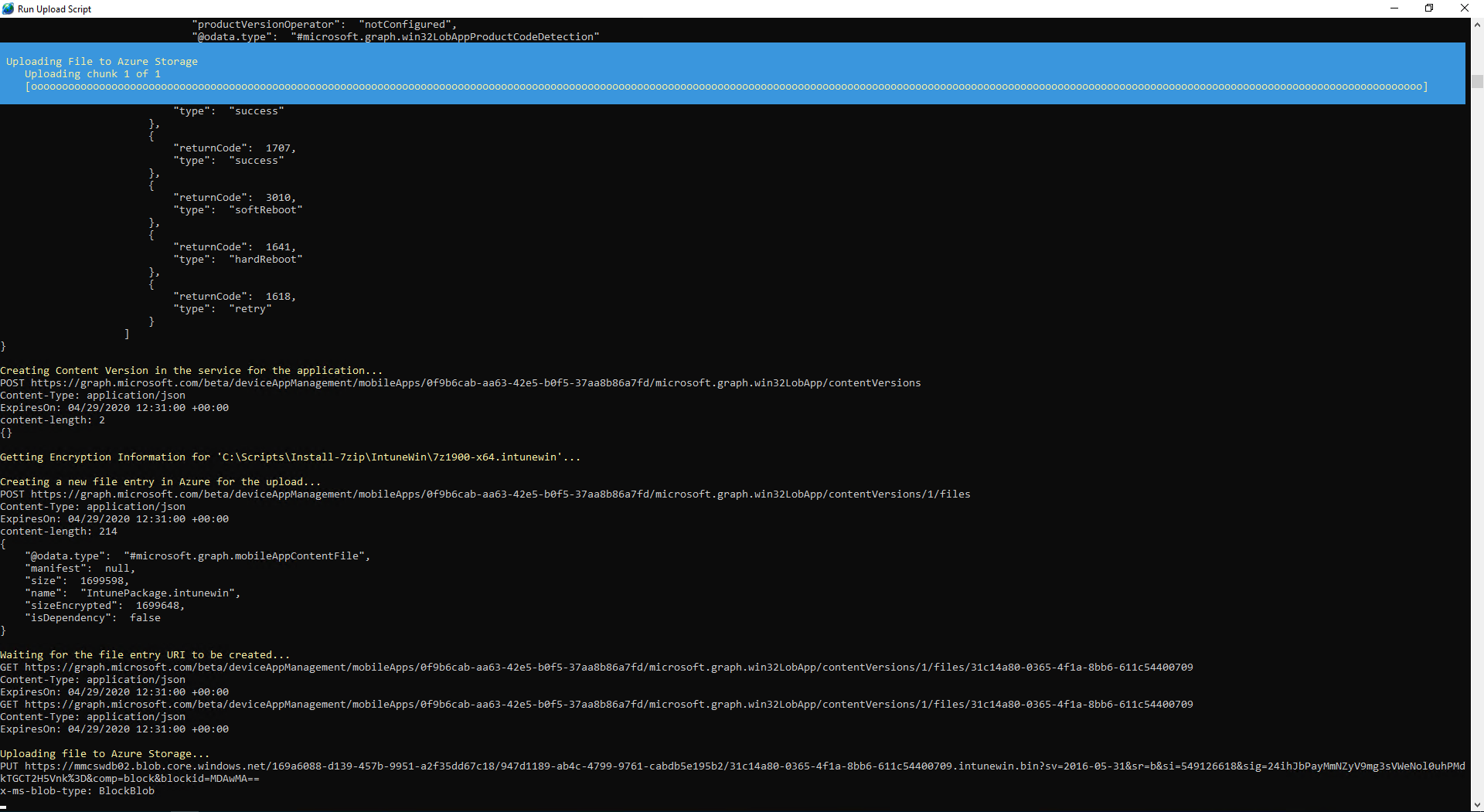


Note – the name used in the ‘SET PackagePath=’ line matches the subfolder name as shown below:



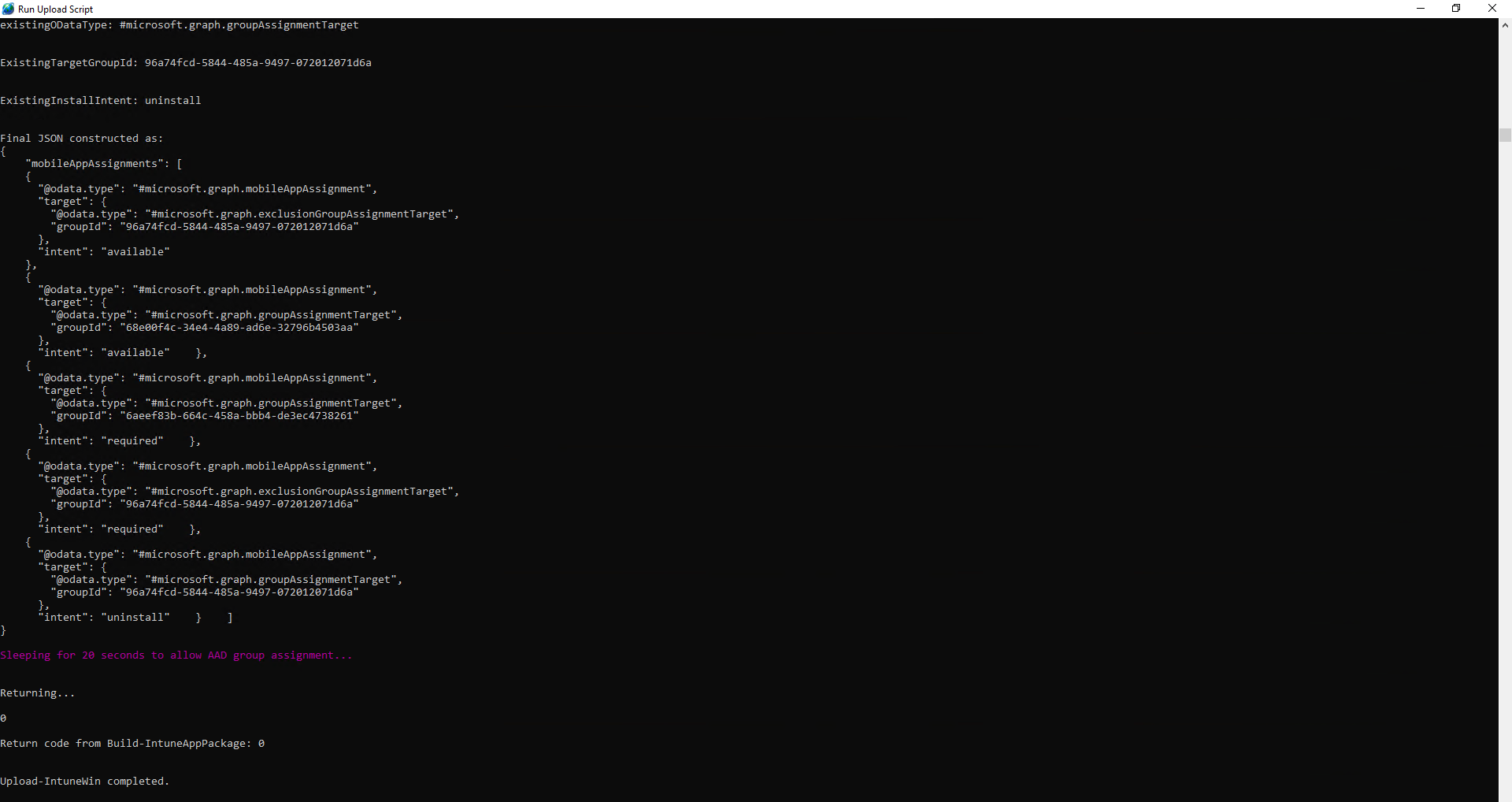
The script automatically appends the full path to the folder, so you only need the subfolder name itself.

1. Double-click on the ‘Run Upload Script’ shortcut file (assuming you extracted to the C:\Scripts path) and this will automatically call the ‘C:\Scripts\RunUpload.bat’ script to run the process:

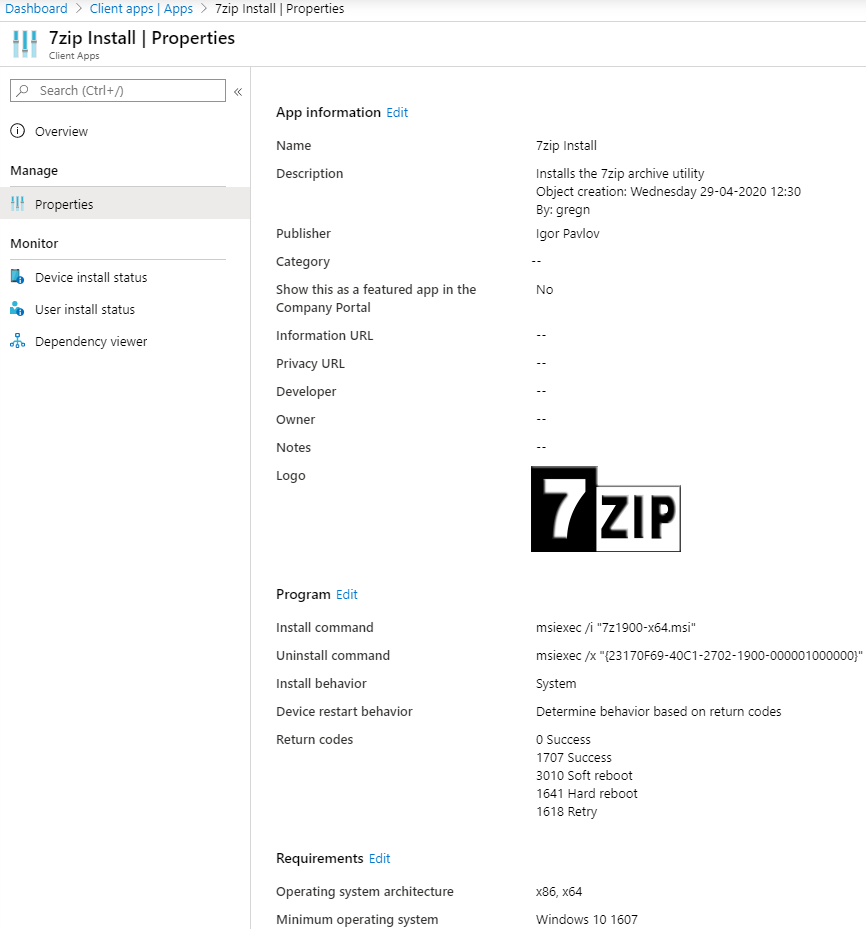


Note you may need to authenticate to Intune if you do not have an active authentication token when you start the upload process.

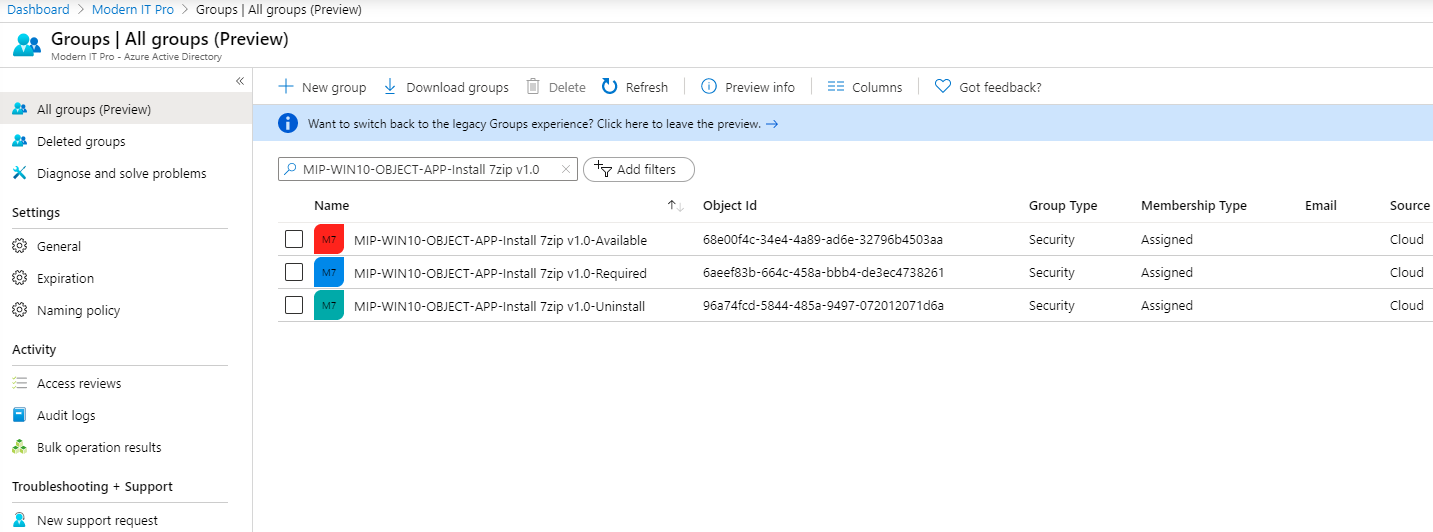
1. The script will take a few minutes to run. Eventually it will look like this:



1. Then you can sign into the Intune portal and locate the package you uploaded.



1. Note the AAD groups that got created too:



All you need to do is add users/devices to the relevant AAD groups according to the type of assignment you want (required or available accordingly). You can also use nested (role) groups for assignment too.

# Example walkthrough for a PS1 (PowerShell script wrapper)

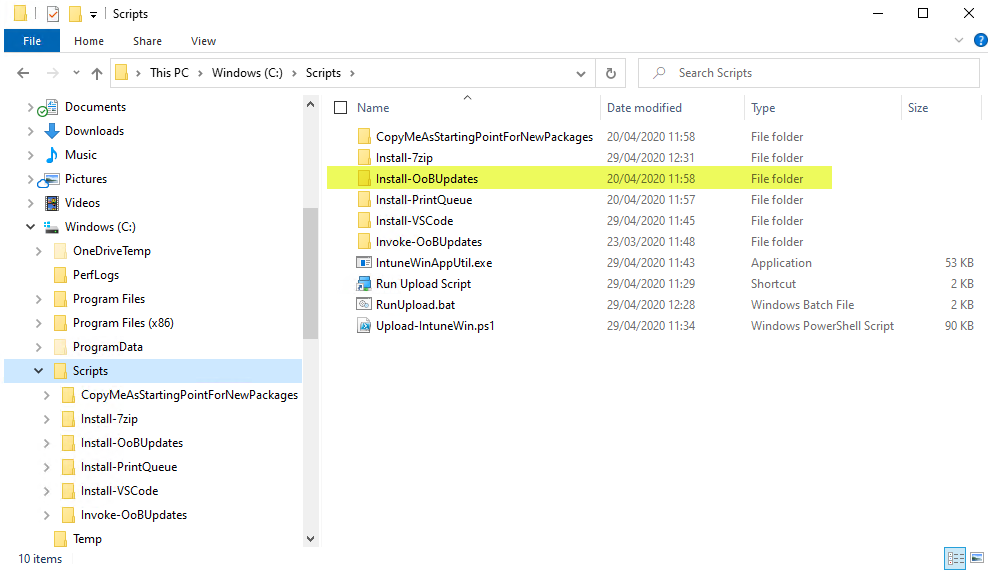
Before starting – ensure you have obtained the following items:

* The ‘Scripts.zip’ package
* The source content you want to package up into the resulting Intune Win32 package
* An optional icon file *(saved in .png format)* that will be shown in Company Portal *(and the Intune console)*
* The relevant silent install/uninstall command switches *(which are required so packages can be installed or uninstalled without any user prompts/wizards, as Intune requires all Win32 package installs to happen without any user interaction)*

Note – using an advanced XML Editor, such as [VSCode](https://code.visualstudio.com/) makes editing the Config.xml much easier.

1. Download and extract the Scripts.zip – for example to the root of C:\ drive (this will automatically create the C:\Scripts folder structure that is required for the correct operation of the Upload-IntuneWin.ps1 script). You can extract the file to a path of your choosing – but for the purposes of these instructions we assume you extracted to the root of C:\ drive (so remember to amend the paths to suit where you extracted to on your computer).
2. Copy the ‘CopyMeAsStartingPointForNewPackages’ folder and rename it to suit the package you want to manage.

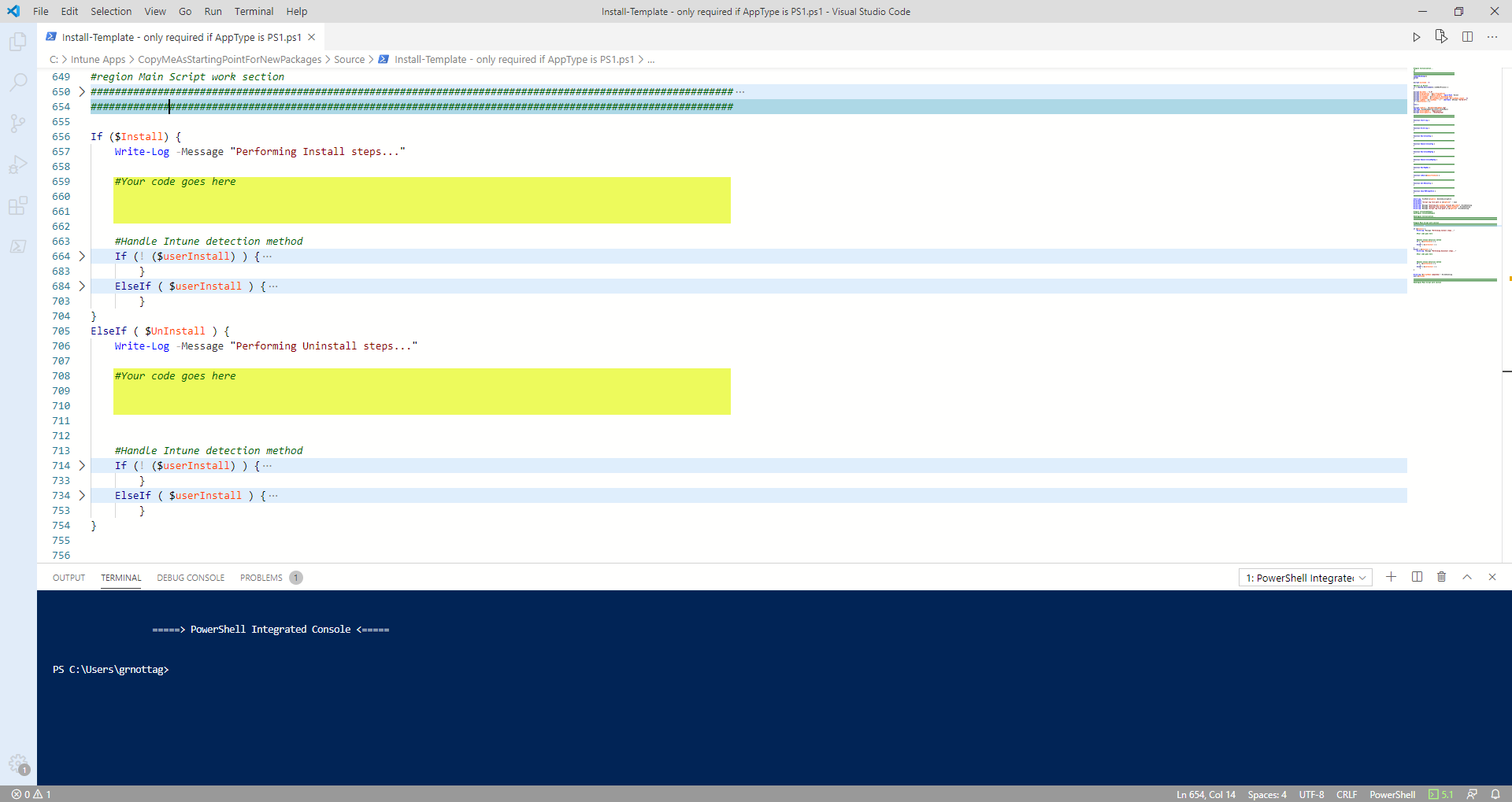
I’ll use the ‘Install-OoBUpdates’ folder as an example:



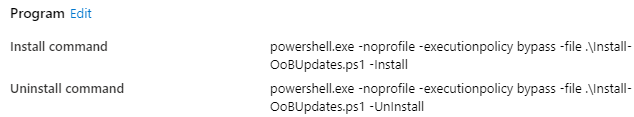
1. Copy the source content for the package into the \Source subfolder.

Note – it’s important to use the ‘\Source\Install-Template - only required if AppType is PS1.ps1’ as the basis for your script. This has the correct code already built in, to enable the detection rules to work properly.

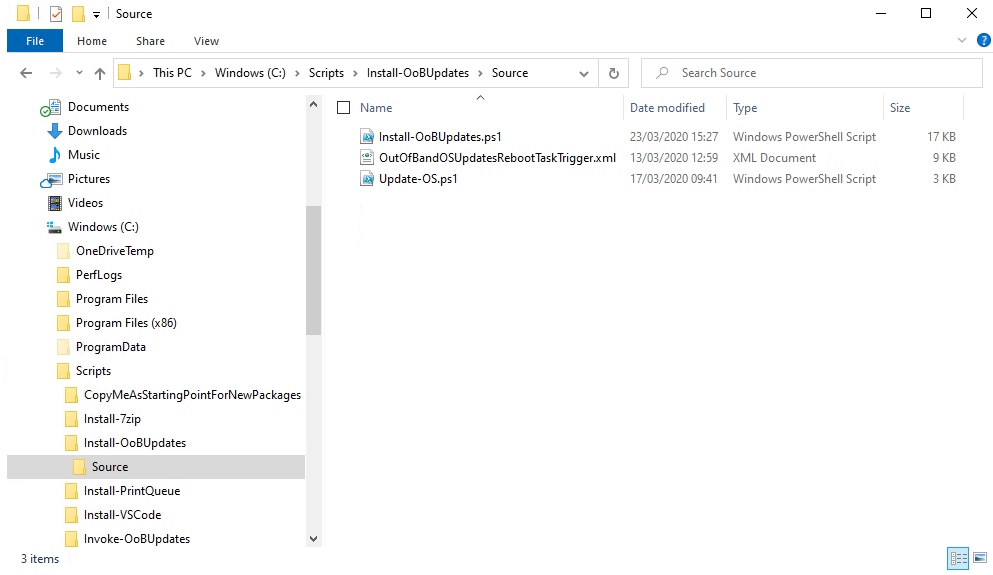
Rename the ‘Install-Template - only required if AppType is PS1.ps1’ script to suit your requirements – then edit the script and paste in your code into the relevant sections highlighted here:



The top highlighted section is run when Intune installs (i.e. deploys) the script and the bottom section is run when Intune uninstalls the script.

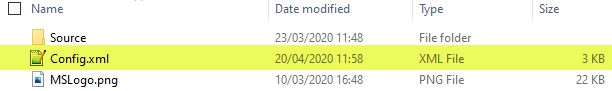


For the Install-OoBUpdates example, the source content folder looks like this:

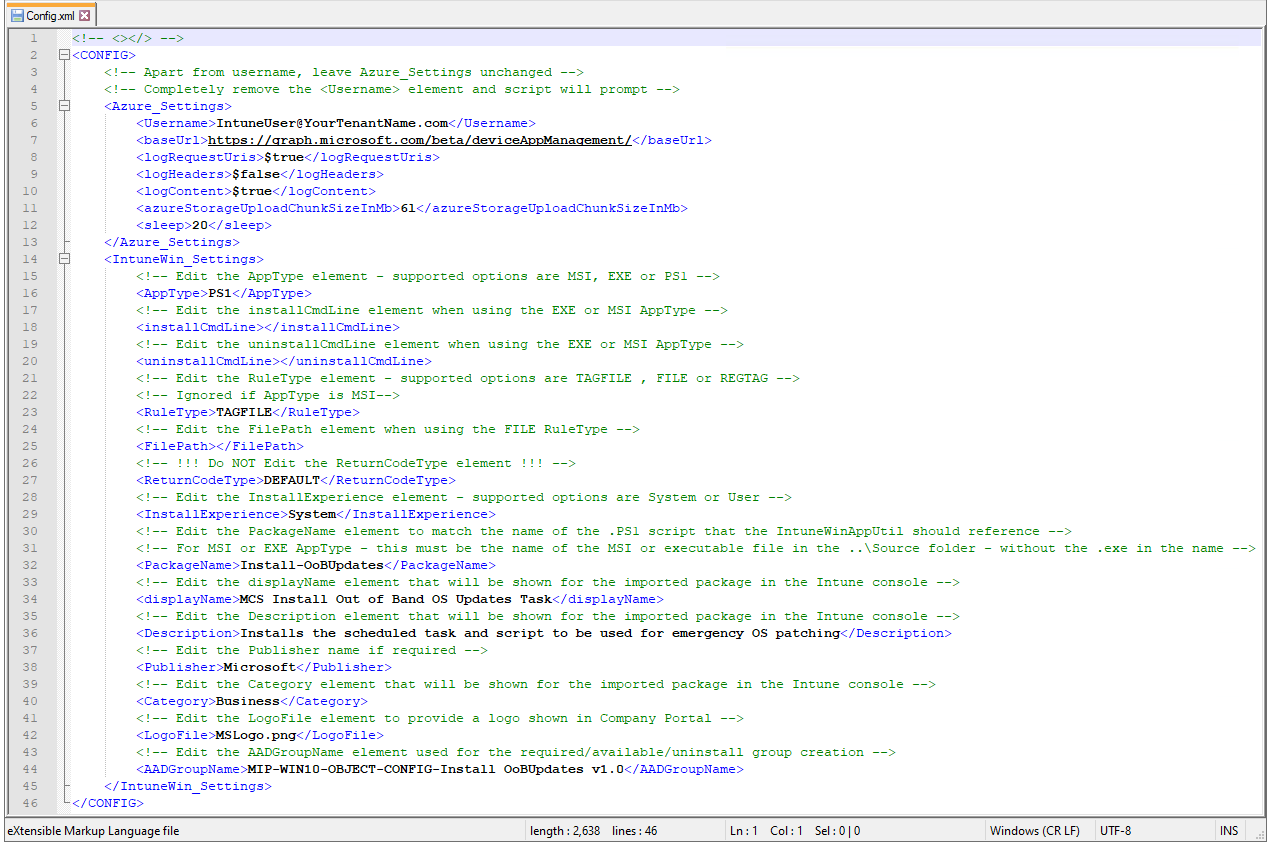


1. Edit the Config.xml file *(at the level above the folder containing the Source)*.

This is what the Install-OoBUpdates example looks like:

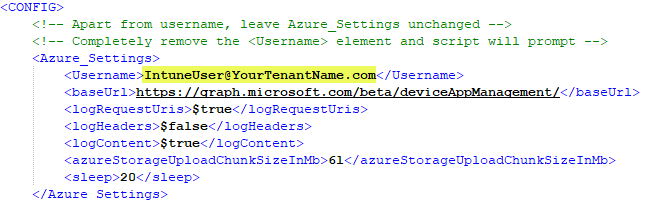


when editing the Config.xml file, it will look similar to this:



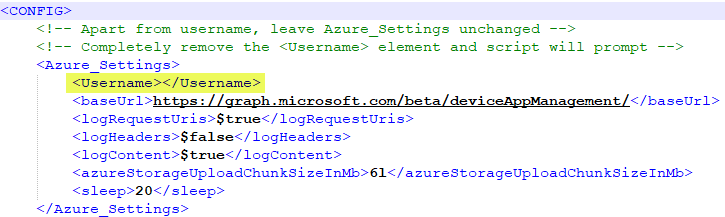
1. First, edit the ‘Username’ field to match an account in your tenant with rights to add new Intune Win32 packages *(you will likely need Intune Administrator rights for this)*.

Note the following highlighted entry confirming what you should edit in your Config.xml:



If you don’t want to specify a username in this Config.xml file – you can leave the field blank and the script will prompt for a username at runtime.

An example without the ‘Username’ field populated, looks like this:



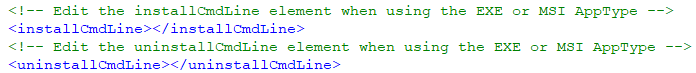
1. Next, edit the ‘AppType’ field to suit the type of package you are installing.

In the Install-OoBUpdates example case, we set the AppType field to PS1 *(as Install-OoBUpdates is a Powershell script)*:



1. leave the ‘installCmdLine’ field and the ‘uninstallCmdLine’ field blank for the PS1 AppType.

For the Install-OoBUpdates example, it looks like this:



1. Next, note the ‘RuleType’ field. This controls the detection rule for the package in Intune.

See RuleType in the Appendix for more details.

For the Install-OoBUpdates example, we are using the TAGFILE method like this:



1. Then, if required, adjust the ‘InstallExperience’ field. Most things should be installed/run under SYSTEM context, but you can change this to USER if required.

In the Install-OoBUpdates example case, we are using SYSTEM:



1. Next, edit the ‘PackageName’ field – this must match the name of the source package *(or script)* that the IntuneWin package tool uses when it creates the .IntuneWin package *(which is what gets uploaded to Intune)*. You must exclude the .exe or .msi *(or .ps1 if it’s a script)* from the name in the PackageName field.

Our Install-OoBUpdates example looks as follows:

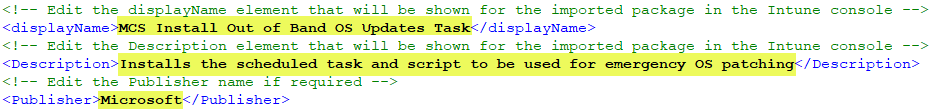


And that is because the name of the file in the \Source folder looks like this:



1. The next 3 fields to edit are related to the way the package is named/displayed in Company Portal *(and also the Intune console)*. ‘displayName’ is how the package name appears, ‘Description’ explains what the package is for and ‘Publisher’ is also displayed for the package accordingly.

For the Install-OoBUpdates example, we used the following:



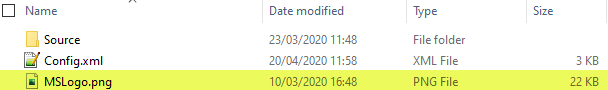
Note – the ‘Category’ field is not yet implemented in the script but must be left in the Config.xml file unchanged.

1. Next, edit the ‘LogoFile’ field to correspond to the (optional) .png logo file to be included in the package. You must store your .png logo file in the same folder as the Config.xml.

Our Install-OoBUpdates example looks like this:

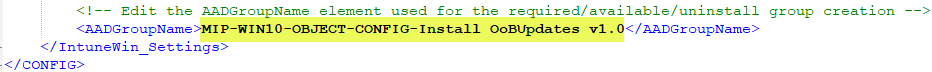


And the folder with the Config.xml and ‘MSLogo.png’ logo file looks like this:



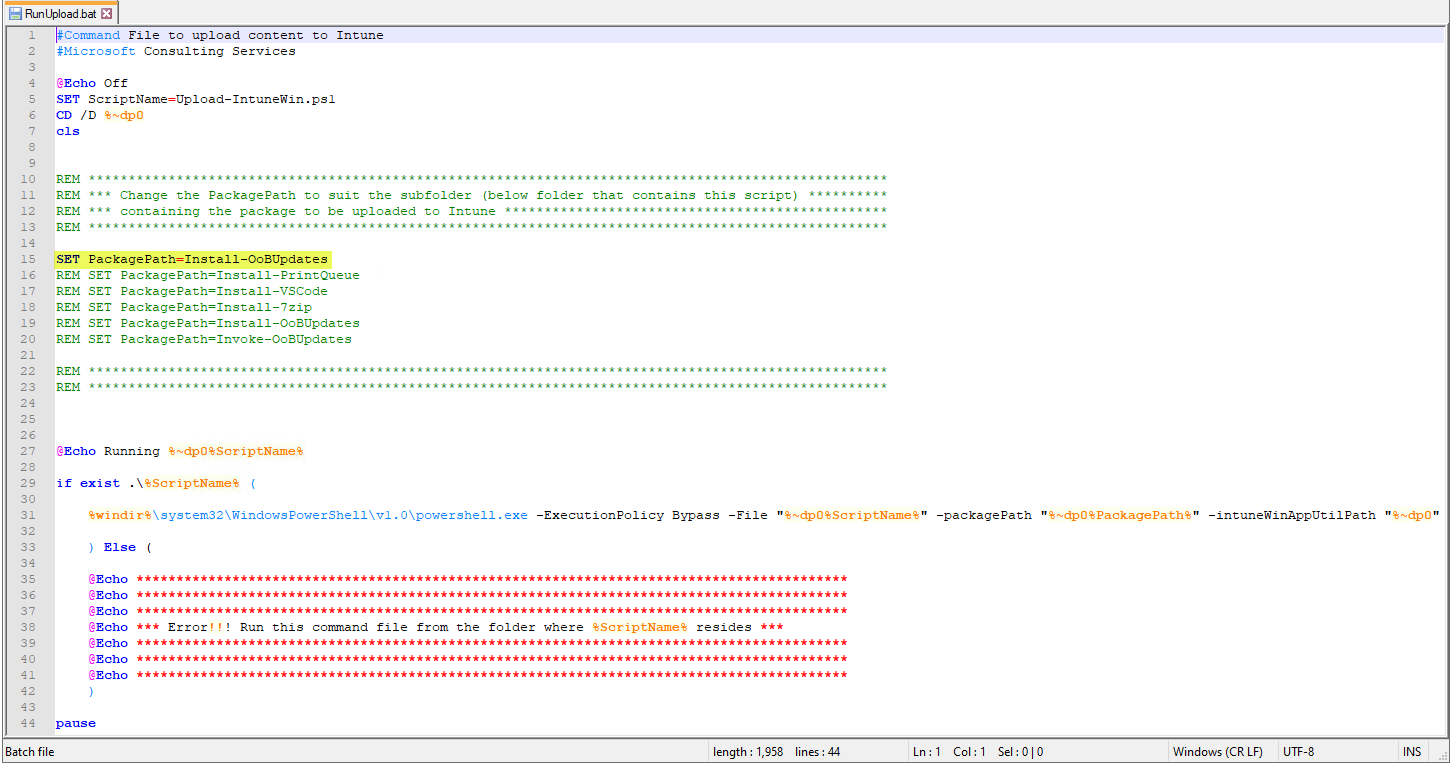
1. The last field to edit is the ‘AADGroupName’ field. This is for the Azure Active Directory group names that will be created. The script creates 3 groups – using the name specified here. The groups use the prefix specified in the ‘AADGroupName’ field – and append ‘-Required’, ‘-Available’ or ‘-Uninstall’ accordingly. These groups control the various deployment targeting groups for the package, once it’s imported into Intune.

Our Install-OoBUpdates example looks like this:

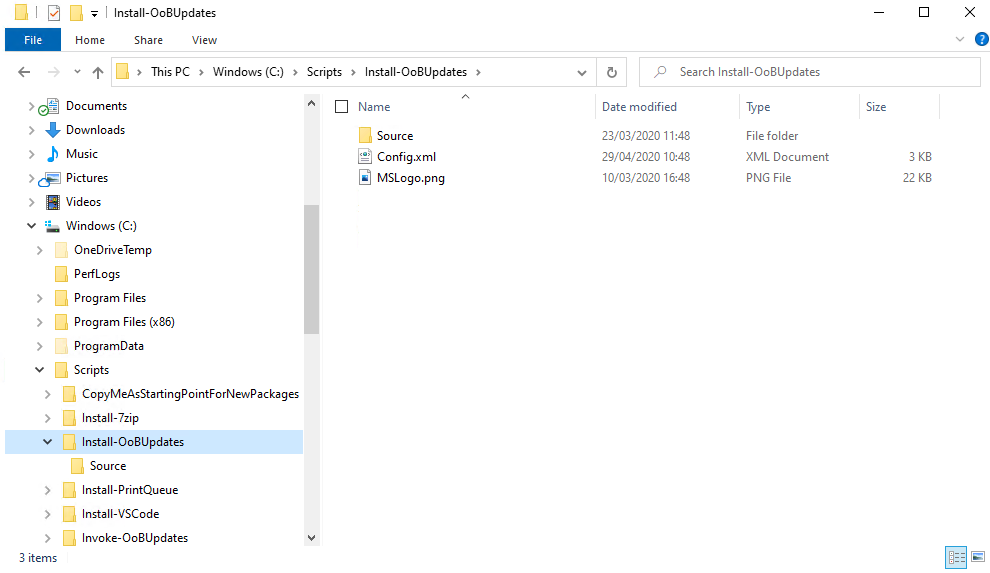


1. Save the changes to the edited Config.xml file
2. Then locate the ‘RunUpload.bat’ file in the C:\Scripts folder and edit the ‘SET PackagePath=’ line (number 15 in the following example) to suit the name of the subfolder containing the package content to be uploaded to Intune.

For the Install-OoBUpdates example, it looks as follows:

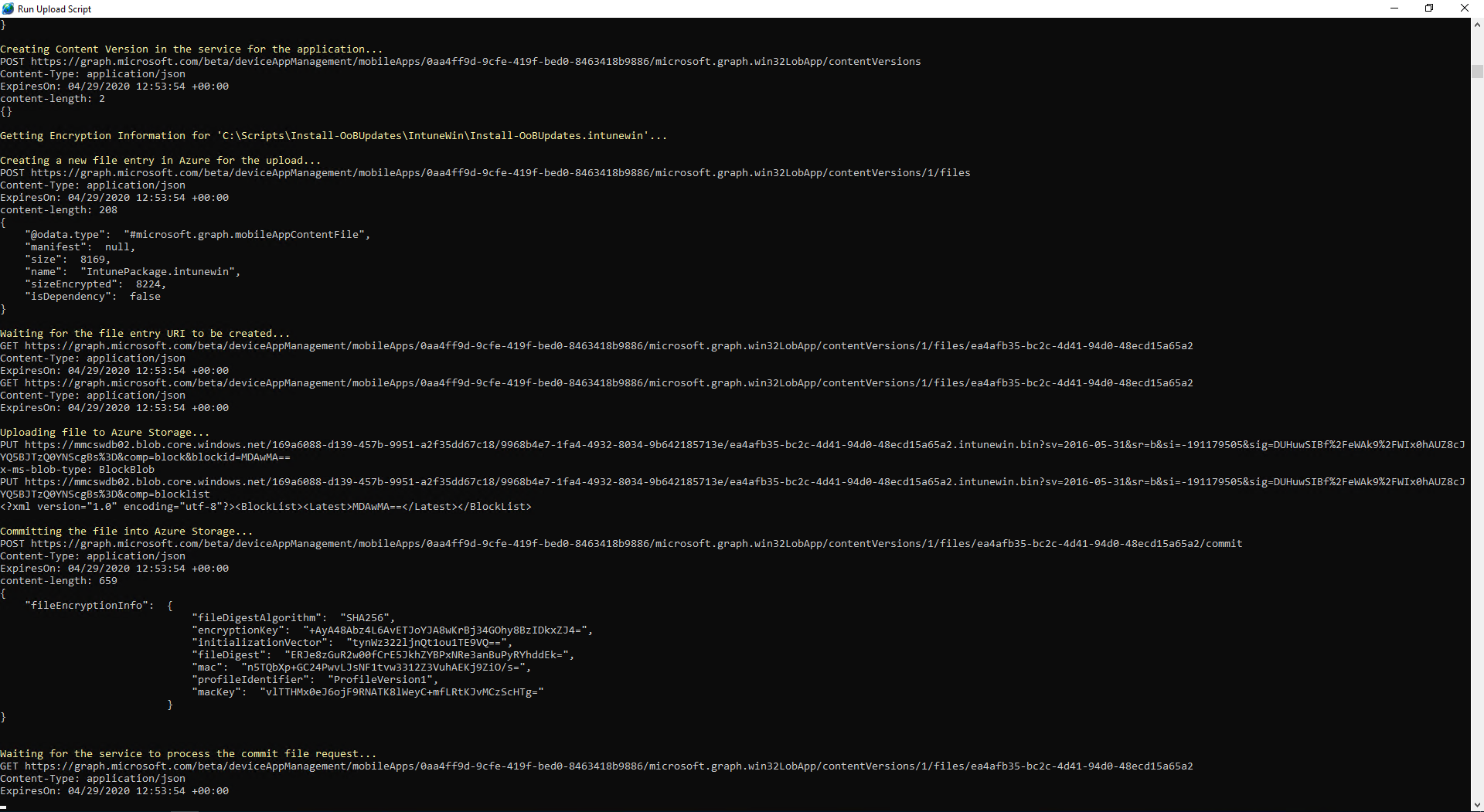


Note – the name used in the ‘SET PackagePath=’ line matches the subfolder name as shown below:



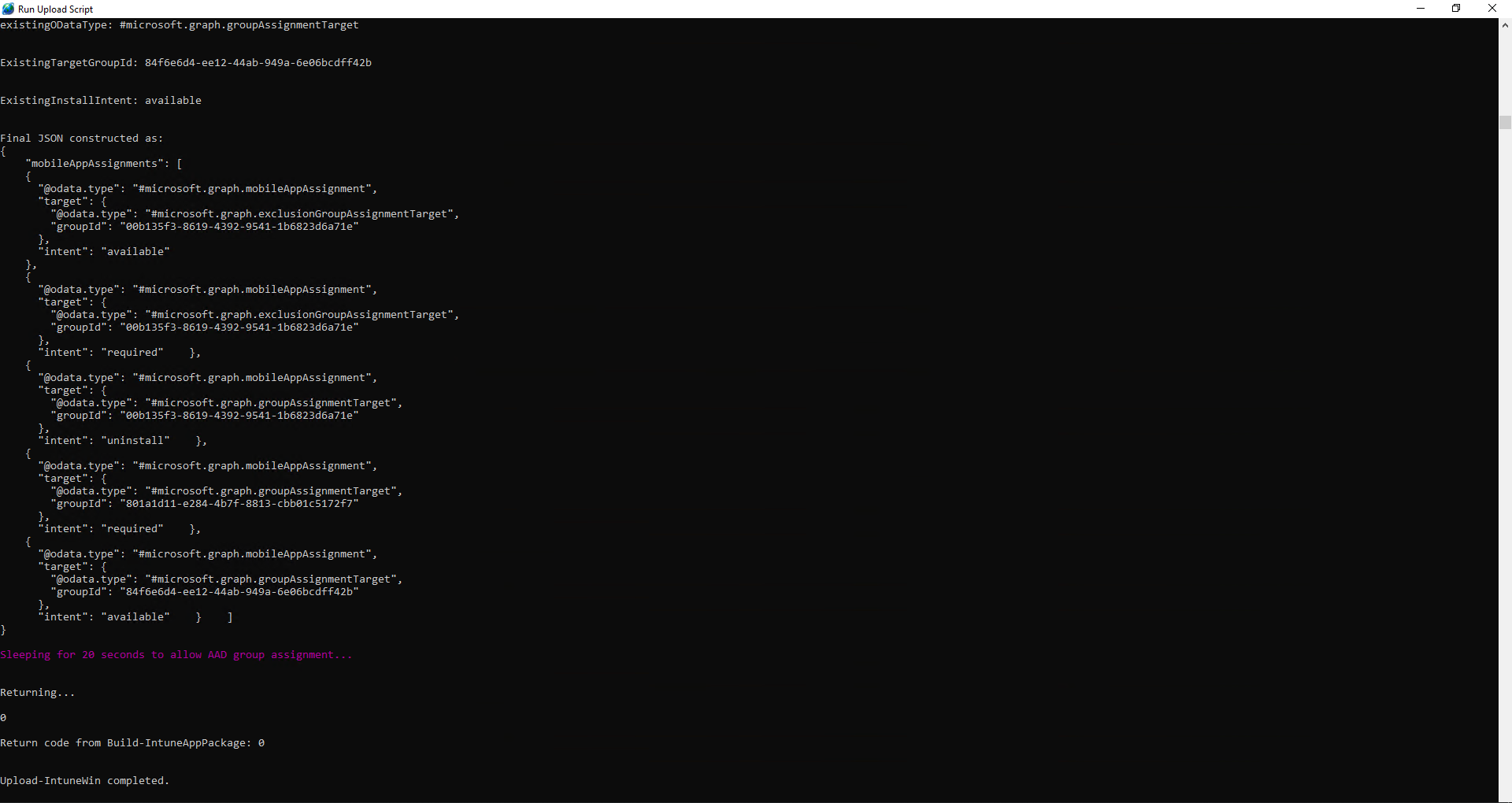
The script automatically appends the full path to the folder, so you only need the subfolder name itself.

1. Double-click on the ‘Run Upload Script’ shortcut file (assuming you extracted to the C:\Scripts path) and this will automatically call the ‘C:\Scripts\RunUpload.bat’ script to run the process:

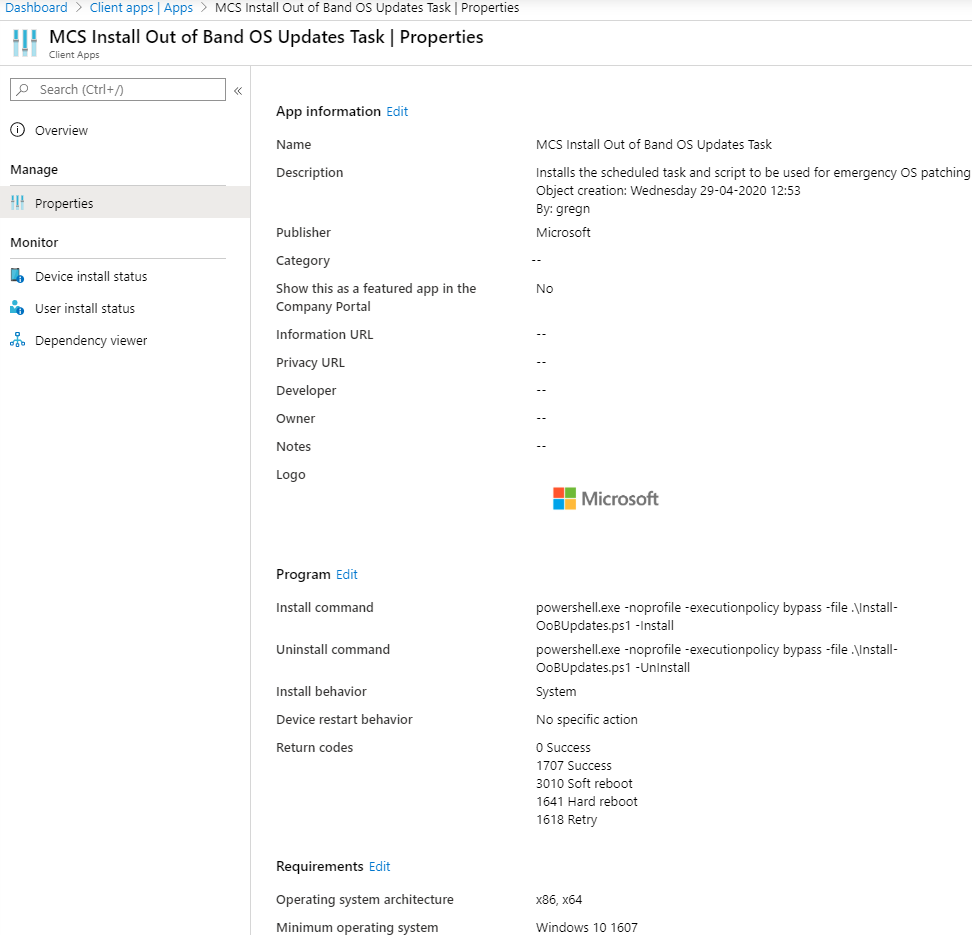


Note you may need to authenticate to Intune if you do not have an active authentication token when you start the upload process.

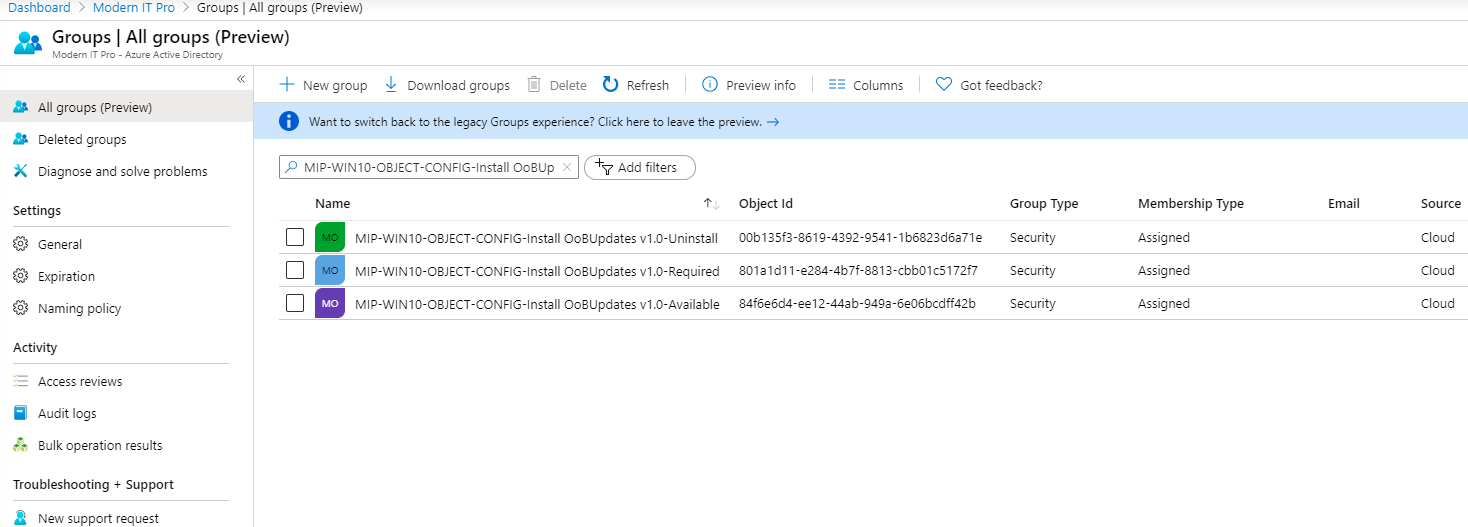
1. The script will take a few minutes to run. Eventually it will look like this:



1. Then you can sign into the Intune portal and locate the package you uploaded.



1. Note the AAD groups that got created too:



All you need to do is add users/devices to the relevant AAD groups according to the type of assignment you want (required or available accordingly). You can also use nested (role) groups for assignment too.

# Appendix – Config.XML Settings explanation

| Section | Setting | Description |
| --- | --- | --- |
| Azure\_Settings | **Username** | This is the only element that should be altered in this section. Adjust to suit your environment. If this element is removed, the script will prompt for a username with appropriate rights to create Intune Win32 packages. |
| IntuneWin\_Settings | **AppType** | This setting defines the type of Win32 package that will be created, prior to upload. Supported options are MSI, EXE or PS1  EXE means a single executable install package (such as VSCode)  PS1 means a PowerShell script – but this script must be developed using the correct template script (as the template script contains code to create the relevant tag-file (or regtag) that is used to determine install/uninstall success with this package type). |
| IntuneWin\_Settings | **installCmdLine** | Only relevant when the AppType is set to MSI or EXE  Put the executable name (without any path) and the appropriate silent install parameters. The upload script expects to find this in the ..\Source subfolder (but not in any folder below that). |
| IntuneWin\_Settings | **uninstallCmdLine** | Only relevant when the AppType is set to MSI or EXE  Put the full path that will be required to run the correct uninstall command. |
| IntuneWin\_Settings | **RuleType** | This setting defines how the Intune client app detection rules are constructed. Supported options are TAGFILE or FILE or REGTAG  TAGFILE means use the tag-file that gets created using the wrapper script that has the right code in to support this (and REGTAG) – create a copy of the ‘C:\Scripts\CopyMeAsStartingPointForNewPackages\Source\Install-Template - only required if AppType is PS1.ps1’ script and use that (as it has the necessary code blocks to support this).  FILE means to use the path specified in the FilePath element (see the next row down in this table for details).  REGTAG is for creating a registry value (either HKCU\Software\Microsoft\IntuneApps and name of the script if it’s a USER based InstallExperience or HKLM\Software\Microsoft\IntuneApps if it’s a SYSTEM InstallExperience). |
| IntuneWin\_Settings | **FilePath** | This setting is only relevant when the AppType is EXE and the RuleType is FILE  The FilePath should be a valid path to a file that will only exist after the executable application gets installed (as this is used as a check by the Intune client app detection rule to confirm successful installation of the application package). |
| IntuneWin\_Settings | **ReturnCodeType** | The ReturnCodeType element should not be edited. |
| IntuneWin\_Settings | **InstallExperience** | The InstallExperience element defines the Intune Install behaviour setting (which configures the context that the package is deployed in). Supported options are System or User |
| IntuneWin\_Settings | **PackageName** | The PackageName element must exactly match the name of either the PS1 script (if the AppType is PS1) or the executable name (if the AppType is EXE), or the MSI filename (if the AppType is MSI).  Note – you must not include the dot extension name – so if your script name is Remove-BDEPin.ps1 – the PackageName that you enter would be Remove-BDEPin |
| IntuneWin\_Settings | **displayName** | The displayName element defines the name of the package that gets displayed in the Intune portal. It can be edited to suit your requirements. |
| IntuneWin\_Settings | **Description** | The Description element defines the description of the package that gets displayed in the Intune portal. It can be edited to suit your requirements. |
| IntuneWin\_Settings | **Publisher** | The Publisher element defines the publisher of the package that gets displayed in the Intune portal. It can be edited to suit your requirements. |
| IntuneWin\_Settings | **Category** | *\*Not yet implemented\**  The Category element defines the category of the package that gets displayed in the Intune portal. It can be edited to suit your requirements. |
| IntuneWin\_Settings | **LogoFile** | The LogoFile element defines the .png graphic file that will be displayed in Company Portal (and the Intune portal) when users browse and locate the package. It can be edited to suit your requirements.  Note – do not put a path to the file (as the upload script automatically configures this, but the requirement is that the logo file exists in the same folder as the config.xml file). |
| IntuneWin\_Settings | **AADGroupName** | The upload script automatically creates 3 Azure AD ‘object’ groups, based on the name you specify in the AADGroupName element. It appends -Required, -Available and -Uninstall to the groups it creates. It automatically assigns these to the relevant Intune install/uninstall commands defined for the package (with excludes on both the Required and Available for the Uninstall group).  The expectation is that you should have AAD ‘role’ groups (so groups containing users or devices, but not both) – then these ‘role’ groups should be nested into the ‘object’ groups for appropriate targeting. |

# Known issues

If you see an error message similar to the following screenshot, you need to install the AzureADPreview PowerShell module by running:

Install-Module -Name AzureADPreview -Scope CurrentUser -Force

