# DIS 2.0 Factory Kit Guidance

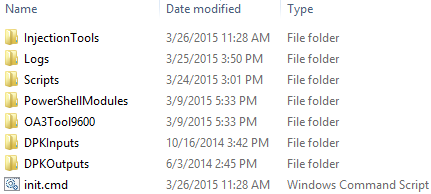
## Overview

DIS 2.0 Factory Kit is a collection of sample scripts, tools and instructions to assist the factories that have adopted DIS 2.0 to complete their OA3.0 process in the production line.

## Working with Production Line Script

### Directory Structure

After extracting the zip package of DIS 2.0 Factory Kit, you will see a sub directory named OA30 in the extracted directory. Navigating this sub directory, you will see the structure like the screen shot below:



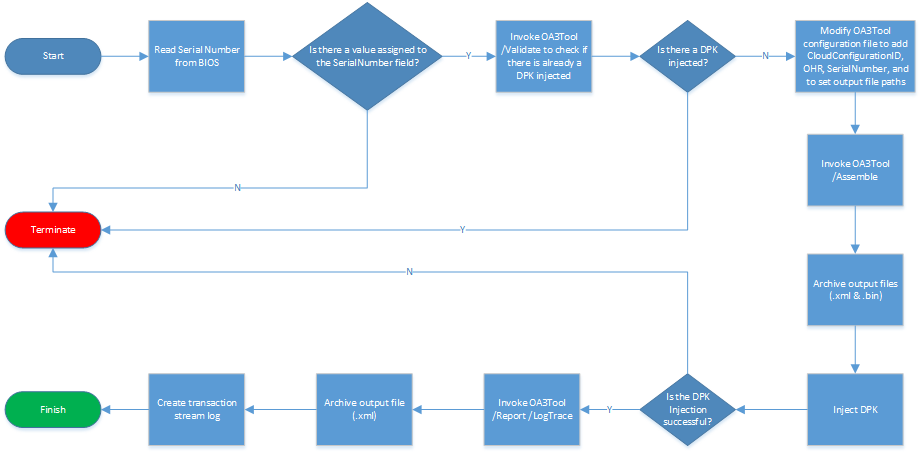
The following table describes each of the directory contents in more detail:

|  |  |  |
| --- | --- | --- |
| **Content Name** | **Content Type** | **Description** |
| InjectionTools | directory | The (suggested) sub directory to place and hold your DPK injection tool |
| Logs | directory | The sub directory for holding all of the production logs |
| Scripts | directory | The sub directory for holding the production line scripts |
| PowerShellModules | directory | The sub directory for holding the PowerShell modules to be used by the production line scripts |
| OA3Tool9600 | directory | The sub directory for holding the 9600 version of OA3Tool, with both x86 and amd64 edition included |
| DPKInputs | directory | The sub directory for holding the key files generated from OA3Tool / Assemble, or exported from DIS |
| DPKOutputs | directory | The sub directory for holding the key files generated from OA3Tool / Report, and to be saved back to DIS |
| init.cmd | file | The Windows command script to create some of the sub directories above in case of missing (optional) |
| startnet.cmd | file | A sample Windows PE initialization script showing how to load up and run a script automatically at start up |
| startnet.cmd.txt | file | A sample Windows PE initialization script showing how to load up and run a script automatically at start up |

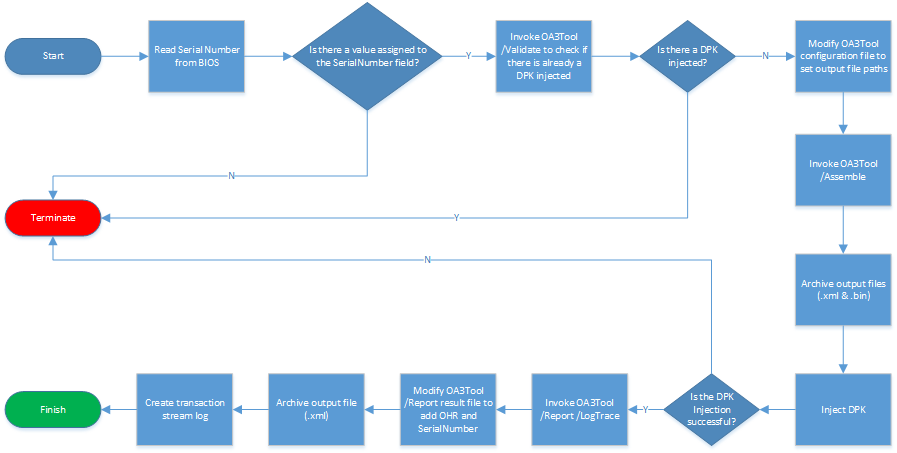
### Types of Sample Scripts

In the **.\Scripts** sub directory, you will see 4 PowerShell scripts in it, each works in OA3Tool’s server-based / file-based mode with 1 of the 2 approaches of unit serial number processing:

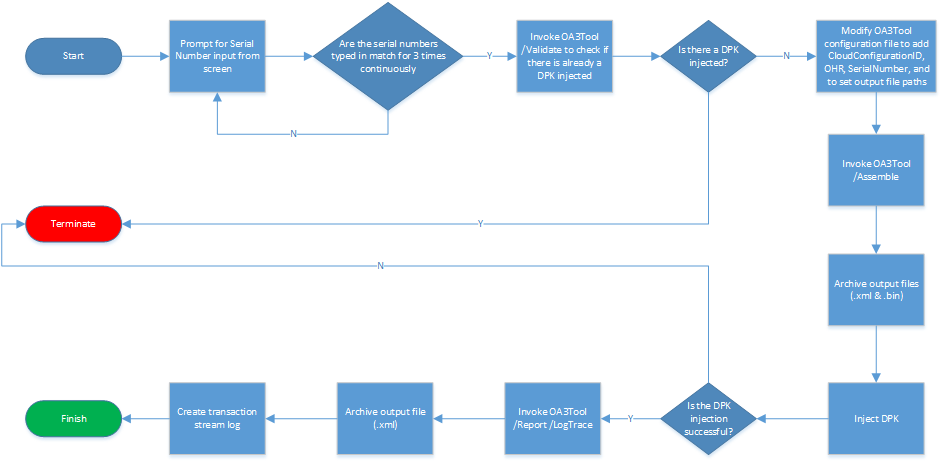
* Reading serial number from the BIOS of each unit:
* For Server-Based mode, the script file **".\Scripts\run-server-based-sn-bios.ps1"** is working following the following flow:



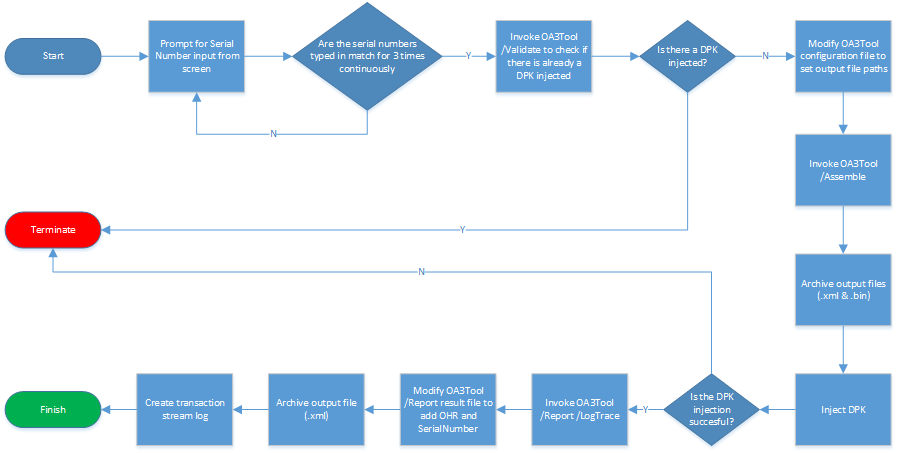
* For File-Based mode, the script file **".\Scripts\run-file-based-sn-bios.ps1"** is working following the following flow:



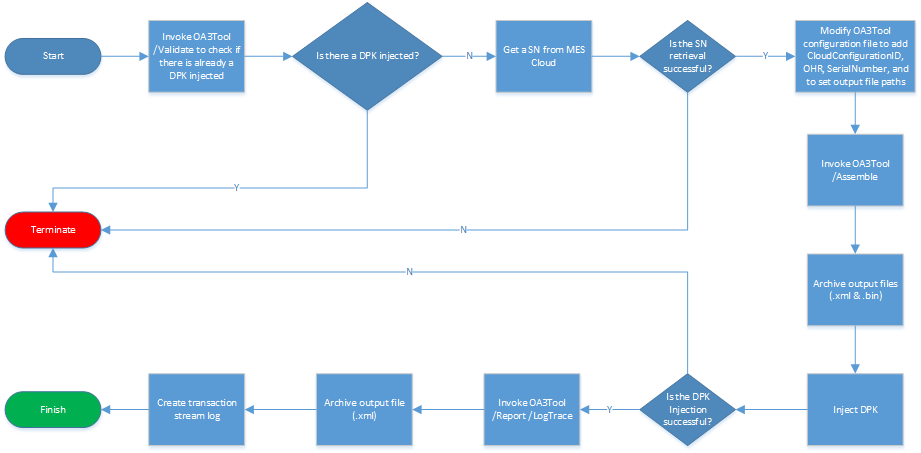
* Reading serial number from the screen input during the boot time of each unit (with a keyboard or barcode reader connected):
* For Server-Based mode, the script file **".\Scripts\run-server-based.ps1"** is working following the following flow:



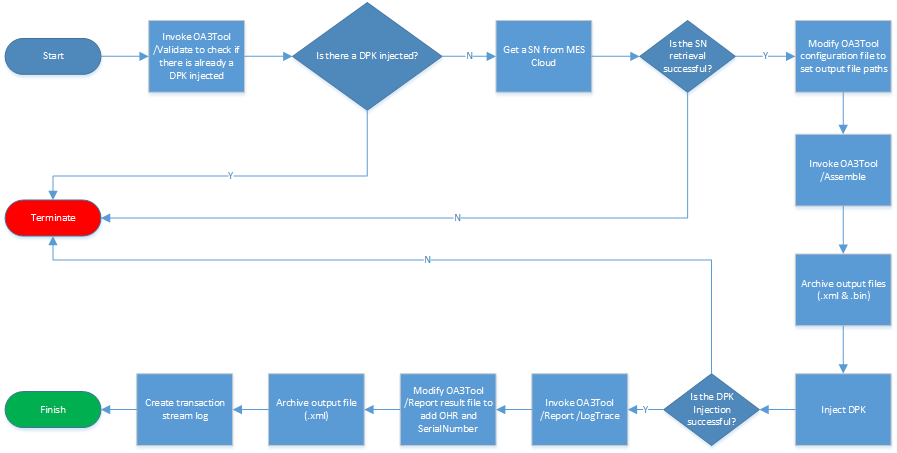
* For File-Based mode, the script file **".\Scripts\run-file-based.ps1"** is working following the following flow:



* Reading serial number centrally from MES Cloud:
* For Server-Based mode, the script file **".\Scripts\run-server-based-sn-cloud.ps1"** is working following the following flow:



* For File-Based mode, the script file **".\Scripts\run-file-based-sn-cloud.ps1"** is working following the following flow:



### Configuring OA3Tool

* For File-Base mode: No extra configuration is needed. The script will generate the oa3tool configuration files dynamically.
* For Server-Based mode: You should modify the oa3tool configuration file of **". \FactoryKit-DIS20\OA30\OA3Tool9600\amd64\OA3Tool-ServerBased.cfg"** (or **". \FactoryKit-DIS20\OA30\OA3Tool9600\x86\OA3Tool-ServerBased.cfg"**) for the following items:
* The IP address / host name of the Key Provider Service;
* Cloud Configuration ID from DIS Configuration Cloud;
* Port number of Key Provider Service;
* OHR Data;

### Modifying Sample Scripts to Integrate Injection Tool

Since the syntaxes for invoking the key injection tools from different vendors (or different version of from a same vendor) differ, the blocks for injection tool invocation in the sample scripts are intentionally left blank. So be sure to modify each (one / some) of the scripts in the **“.\OA30\Scripts\”** sub directory according to the vendor / version of the injection tool you are using. The blank in the following block in each of the scripts will need to be populated accordingly:

##Runs DPK Injection Tool Here

try

{

    $Message = [System.String]::Format("Injecting DPK..., {0}", [System.DateTime]::Now);

    $Message;

    $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

}

catch [System.Exception]

{

    $Message = $Error[0].Exception;

    $Message;

    $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

}

For example, if using a version of injection tool from AMI, you will need to populate this blank similarly as below (highlighted in yellow):

##Runs DPK Injection Tool Here

    try

    {

       $Message = [System.String]::Format("Injecting DPK..., {0}", [System.DateTime]::Now);

       $Message;

       $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

      Start-Process -FilePath ($RootDir + "\InjectionTools\afu\AFUWin.exe")  -NoNewWindow -ArgumentList @([System.String]::Format("/A{0}", $OA3OutputBinFilePath)) -Wait | Out-File -FilePath ($LogPath + "\production-log.log") –Append;

    }

    catch [System.Exception]

    {

       $Message = $Error[0].Exception;

       $Message;

       $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

    }

Or if using a version of injection tool from Intel, you will then need to populate this blank similarly as below (highlighted in yellow):

 ##Runs DPK Injection Tool Here

    try

    {

       $Message = [System.String]::Format("Injecting DPK..., {0}", [System.DateTime]::Now);

       $Message;

       $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

      & ($RootDir + "\InjectionTools\intel\OA3IntelTbtWin.exe") @([System.String]::Format("update 'Intel' 'Intel' '{0}'", $OA3OutputBinFilePath)) | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

    }

    catch [System.Exception]

    {

       $Message = $Error[0].Exception;

       $Message;

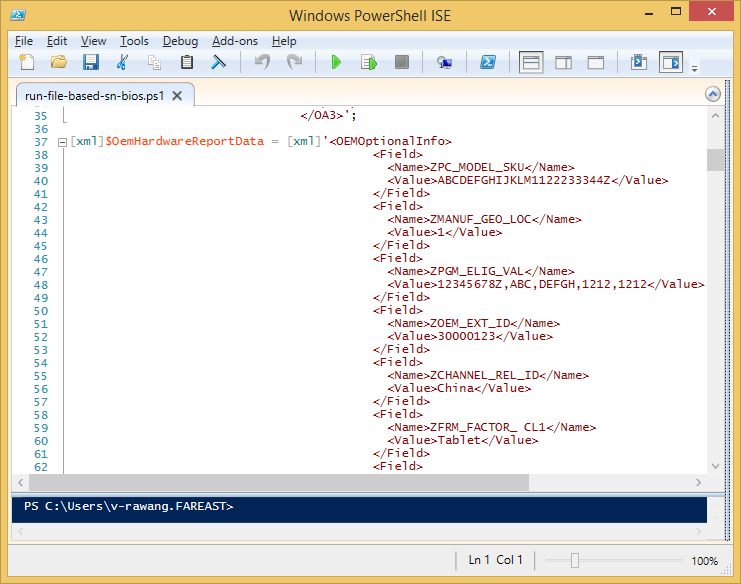
       $Message | Out-File -FilePath ($LogPath + "\production-log.log") -Append;

    }

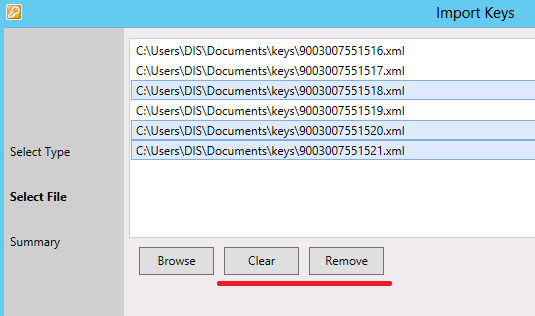
Please also note that, you should make sure that the DPK injection tool is placed in the location where the script can access during runtime. And it’s highly recommended that you place the injection tool in the **“.\FactoryKit-DIS20\OA30\InjectionTools”** sub directory.

### Interacting with DIS 2.0

* For Server-Based mode, if the oa3tool configuration file is configured correctly, the scripts for server based mode will be able to interact with DIS 2.0 directly with oa3tool, and the results can be seen in the FFKI at real time.
* For File-Based mode, the following 2 things should be taken care of:
* **OHR data**: before running the script, you must ensure that the value of the **$OemHardwareReportData** variable declare at the beginning of the script is as specific to the expected value of the OHR data from your OEM customer. And if not, you should modify it accordingly:



* **Data transfer**: in file based mode, the interaction with DIS is through the DPK Import / Export function in FFKI. And from DIS 2.0, you can import more than one oa3tool result files at the same time. All of the oa3tool /report output files are located in the **“.\FactoryKit-DIS20\OA30\DPKOutputs”** sub directory, all in the form of **“[DPK\_ID].xml”**. You can copy and import all of this files into FFKI in a batch:



## Building Boot Image

For the scripts to be loaded up and running automatically in Windows 8.1 PE at startup, you must ensure that at least the following 5 Windows PE packages have been added to the .wim file when customizing the Windows PE boot image:

* WinPE-WMI.cab
* WinPE-NetFx.cab
* WinPE-Scripting.cab
* WinPE-PowerShell.cab
* WinPE-DismCmdlets.cab

The sample DISM PS script file **". \FactoryKit-DIS20\DISM-PrepareWinPEImage.ps1"** has provided you an example on how to add these 5 packages above.

Also, at least 2 lines of command should be appended to the Windows PE initialization script (startnet.cmd) to ensure that PowerShell is loaded up with the script file specified with the execution policy for allowing executing script file turned on. The following example is for reference:

D:\

cd D:\OA30\

PowerShell -ExecutionPolicy ByPass -File .\Scripts\run-server-based-sn-bios.ps1

After the Windows PE image is built, use the **MakeWinPEMedia /ufd** command in Windows 8.1 ADK to create a bootable USB disk with the Windows PE image you just built.

When the USB disk is created, copy the entire OA30 sub directory to the root of the USB disk. And when the unit is booted up with this USB disk, the script can be accessed following the path of **“D:\OA30\Scripts”**.

## Working with Serial Number

If the serial number is read from BIOS, you must ensure that the serial numbers have been burn into the BIOS before running the script **".\Scripts\run-server-based-sn-bios.ps1"** (or **".\Scripts\run-file-based-sn-bios.ps1"**).

If the serial number can be acquired from a barcode, you may use the **".\Scripts\run-server-based.ps1"** (or **".\Scripts\run-file-based.ps1"**) in conjunction with a barcode scanner that supports working in PS/2 keyboard mode. And you may also configure the barcode scanner to append a hard return to the result (if supported) after each scan, thus no keyboard typing will be needed when being prompted for entering the serial number during the script execution.

If the serial number is read from the MES Cloud, you may use the script **".\Scripts\run-server-based-cloud.ps1"** (or **".\Scripts\run-file-based-cloud.ps1"**) to interact with web service from the MES Cloud. There are about two places in each of the two scripts to be edited before running any one of them:

* The server address & http port number of MES Cloud:

$ServerAddress = "http://mes-server-01:8919";

* The authorization header to get authenticated and authorized by MES Cloud:

$RequestAuthHeader = "MES:M(S@OMSG.msft";

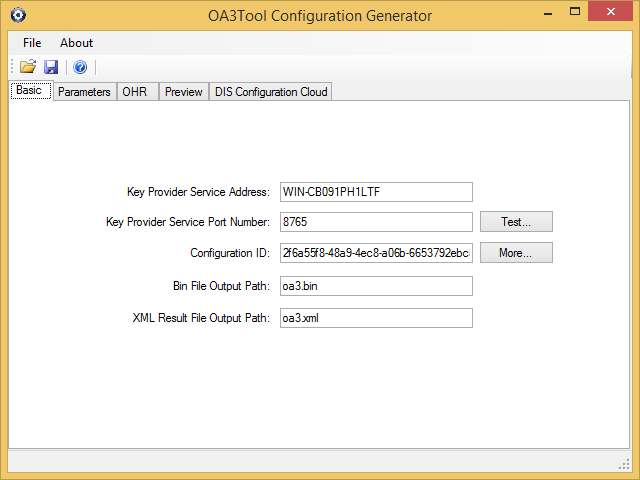
Also, for fool proofing purpose, you may add your own logic in each of the scripts to avoid any accident in your manufacturing process.

## Using OA3Tool Configuration Generator

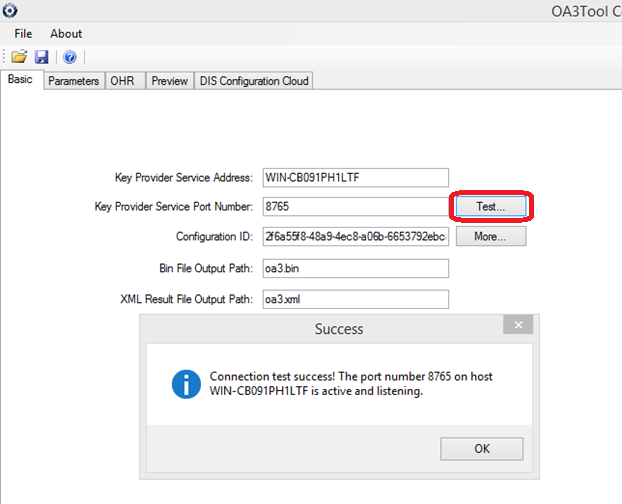
The OA3Tool Configuration Generator is a GUI tool letting you visually create, modify, and generate OA3Tool configuration file in a WYSIWYG way. OA3Tool Configuration Generator is also a good companion of DIS 2.0.

With OA3Tool Configuration Generator, you can:

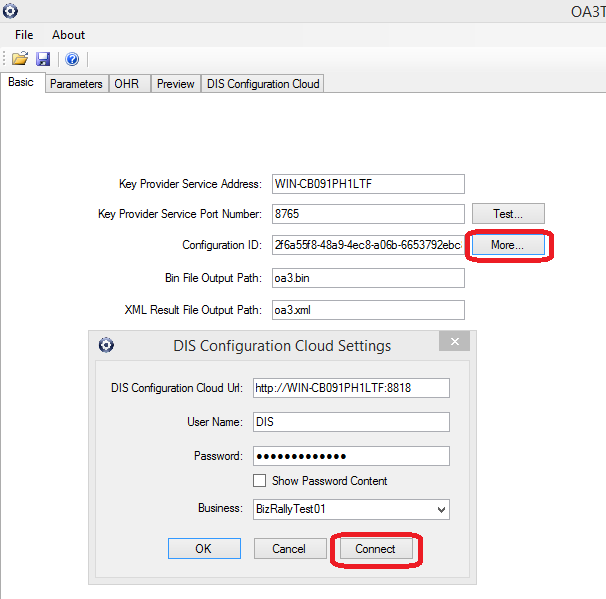
1. Type in the values required for each field in the OA3Tool configuration file into the text input fields without locating each of them in the raw content of the XML file:



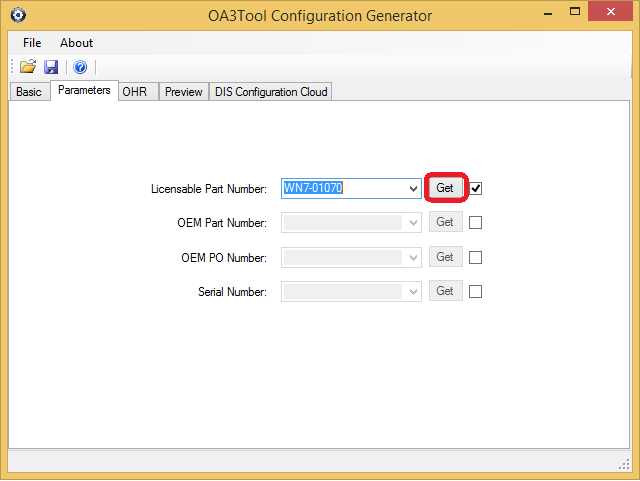
1. Test the running state of the Key Provider Service, thus verify the correctness of the service address and port number before encountering a problem in the production line with OA3Tool:



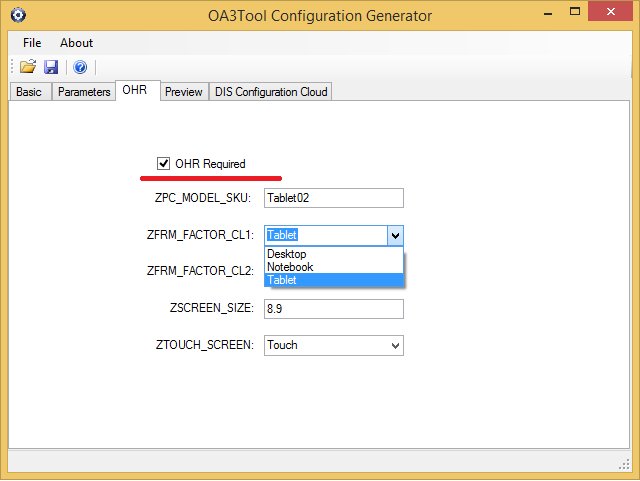
1. Specify the value for CloudConfigurationID needed in the OA3Tool configuration file by connecting to DIS Configuration Cloud and selecting the corresponding business in the drop down list:



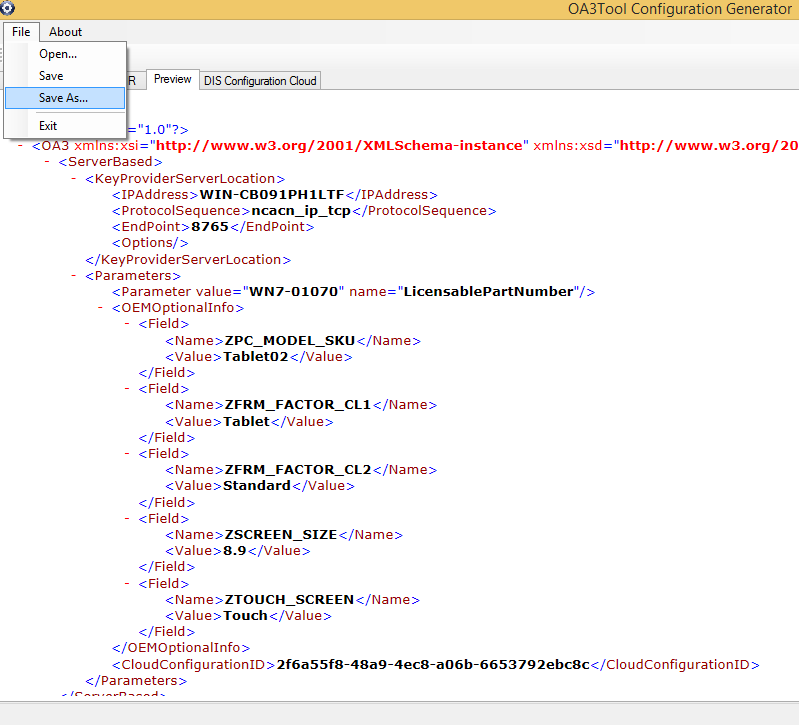
1. Specify the value for each Parameter field by just selecting in the drop down list:



1. Manipulate OHR data visually by selecting from the linkage of drop down lists to ensure domain data integrity:



1. Preview the configuration file contents dynamically before saving it to a file:



1. When the second time you open OA3Tool Configuration Generator, the value you entered in each input filed last time will be automatically remembered.