TOOLS4EVER
IDENTITY GOVERNANCE & ADMINISTRATION

V1.4 / 07-03-2023

Training Labs

HelloID Provisioning connector training

Index

Preparation - HelloID provisioning Agent 3

LAB 1 – Create a custom PowerShell source system 4

1.1 preparation 4
1.2 create a new source system 4
1.3 configure default source person and contract data mapping 5
1.4 introduction of complex field mappings 5
1.5 create a custom connector UI 6

LAB 2 – Data validation and thresholds for source systems 8

2.1 working with import thresholds 8
2.2 working with data validation thresholds 9

LAB 3 – Create a PowerShell target system to export data 10

LAB 4 – Add DisplayName generation on target system 12

4.1 Preparation 12
4.2 Add the generation to the target system 12
4.3 Check the generated name 13

LAB 5 – Use data from another system 14

5.1 preparation 14
5.2 using account data from other systems 14
5.3 Adding the sAMAccountName to the account object 15

LAB 6 – Correlation on PowerShell target system 16

6.1 preparation 16
6.2 Add correlation to your Account create event 16

LAB 7 – Create PowerShell notification system 17

7.1 preparation 17
7.2 Add a script for Account update lifecycle event 17
7.3 Create a custom event notification 17
7.4 Test the notification 17

Preparation - HelloID provisioning Agent

Before we start using HelloID provisioning to contact the on-premise Active Directory, we need to have the Provisioning Agent running on our local server. Please check if the service is running. If not, please install the agent.

To setup an Agent we first need to configure an Agent Pool in HelloID, please follow the steps from our HelloID docs article: https://docs.helloid.com/hc/en-us/articles/360002682754-How-to-Create-and-Manage-Agent-Pools to create a HelloID Agent pool.

After the Agent pool is created in the HelloID portal, the next step is to install and configure the local HelloID provisioning Agent.

To install the local HelloID Agent please follow the steps from our HelloID docs article: https://docs.helloid.com/hc/en-us/articles/360001597494-How-to-Install-and-Manage-Agents

Reference:

Before we can start... make sure your Active Directory contains the following.

OU structure

- HelloID Training
    - Users
    - Disabled users
    - Groups

AD Groups

- 4 AD groups for demo purpose
- Add groups to AD users (random)

LAB 1 – Create a custom PowerShell source system

Source systems are the start of all provisioning processes. They are where personnel records come from, which are then used by HelloID to create and manage accounts, accesses, and permissions. This exercise will lead you through the basics of setting up and configuring the PowerShell source system in HelloID.

1.1 preparation

- Create a new directory on the local drive of your HelloID server running the HelloID provisioning agent service.
- Download the example source data from the following GitHub repository file into this local directory.

1.2 create a new source system

Please use the following manual to add the new Source connector:
https://docs.helloid.com/hc/en-us/articles/360012557600-Configure-a-custom-PowerShell-source-system

- Add a new source system from the HelloID provisioning portal.
- Select “Source Template” from the template library
- Provide a logical name and description for the system you are working with.
- By turning **“On”** the **Execute on-premises** toggle your Persons and Departments PowerShell scripts will be executed directly on your local (on-premise) HelloID agent, instead of your HelloID cloud instance. Enabling this option is required because in this exercise we are building a source system based on local *.csv source input files.
- Replace the predefined Persons and Departments PowerShell scripts, by using the example **persons.ps1** and **departments.ps1** scripts from this GitHub repository.
- Within both Persons and Departments PowerShell script examples you must replace the **$importSourcePath** variable value to the location of the directory created before.
- After the source system is configured successfully, Click “Import raw data” to import the RAW data.
- To check if the RAW data is loaded successfully, we must open the Raw data tab of the created source system. The Raw data tab lets you see the data HelloID is importing from the source system for every person, without any modifications, filters, or mappings

1.3 configure default source person and contract data mapping

HelloID Provisioning gives you a robust set of default fields to work with in regard to storing Person and Contract data from your source system, you can even add your own custom fields to the Person or Contract object.

With your PowerShell scripts in place, you can configure the field mappings in the Person and Contract tabs.

- Import the predefined **personMapping.json** and **contractMapping.json** from the GitHub repository for this exercise.
- Once the field mappings are in place, you can perform the first import of source data into HelloID.
- After the import is completed successfully, our HelloID provisioning portal should contain the newly created persons from the new source system.
- You can check for persons by opening the **Persons** menu item within the HelloID portal and select a person from the populated list to see the additional information for this person.

1.4 introduction of complex field mappings

HelloID supports JavaScript ECMAScript 5.1 standard in complex field mappings for both source and target systems. Such complex mappings are useful for generating conditional output, or for formatting output in a specific manner. Complex field mappings within HelloID allow you to leverage industry-standard knowledge and best practices to handle the data in your HelloID provisioning processes.

HelloID supports different name patterns for source and target systems. Each is denoted by one of the following abbreviations.

- B: Birth name
- P: Partner name
- BP: Birth name - Partner name
- PB: Partner name - Birth name

In this exercise we create our own complex mapping to translate the imported source data in the correct HelloID mapping format.

- Extend the person mapping we imported before, by adding the default HelloID schema field **Name.Convention**.
- After the field is added we create our own JavaScript function, and use this as a complex field mapping for this field.
- Use the reference table below to translate the data.

Source value	HelloID value (result)
0	B
1	PB
2	P
3	BP

Example complex function to get started:

```
function formatNamingConvention() {
   Write custom logic here to translate the source input value into the default naming parameter
}
formatNamingConvention();
```

Use the value of **source.Naamgebruik_code** as input parameter for your complex function.

- When the complex function is complete, you can check the result by using the live preview window on the right-hand side when a person is selected.
- To finally store this information in HelloID, re-importing of the data is required by preforming a manual import.

More information about complex field mappings can be found here: https://docs.helloid.com/hc/en-us/articles/360013578319

1.5 create a custom connector UI

Both source and target PowerShell systems support the Custom connector configuration option. This feature lets you store general input parameters to be used in your PowerShell scripts, such as API keys and secrets, passwords, URLs, Directory location's, etc. It accomplishes this by letting you define a UI form in which you can enter the parameter values. You then receive them inside a **$configuration** JSON string in your PowerShell scripts.

- To get started, go to the System tab *(for source systems)*.
- Select the wrench icon on the custom connector configuration to open the form JSON UI editor.
- By default, the JSON editor contains a boilerplate example for each type of supported form elements (required text, multi-line text, password, email, toggle, radio button, and drop down menu).
- In this exercise we create an UI text component to replace the **$importSourcePath** variable as previously defined in the **Persons.ps1** and **Departments.ps1**.
- After the UI configuration is in place, we must modify the **$importSourcePath** in the Persons and Departments PowerShell scripting located on the system tab of the connector to read the directory value from the configured UI component.

  Example config to retrieve the input value from the UI configuration into the **$importSourcePath** variable.

  ```
  $connectionSettings = ConvertFrom-Json $configuration
  $importSourcePath = $($connectionSettings.Path)
  ```

- Having the changes in place, we must re-import the data to see if everything is still working as expected.
- When script modification is working as expected we have to modify the **$delimiter** parameter in the same way as we did with the **$importSourcePath** parameter.

More information about input parameters for source and target PowerShell systems can be found here: https://docs.helloid.com/en/provisioning/source-systems/input-forms--provisioning-systems-.html

LAB 2 – Data validation and thresholds for source systems

The Thresholds tab lets you automatically block imports when the amount of added, removed, or blocked Persons exceeds a specified value. When an import occurs for a source system, the pending activity is compared to the last import for the same source system. If the amount of activity exceeds the threshold, the import is blocked until manually approved. In this way, thresholds function as safety nets. They reduce the likelihood of a major mistake in the provisioning process if a source system contains bad or missing data.

2.1 working with import thresholds

You can set separate thresholds for added, removed, and blocked Persons using the respective Enabled toggles. Thresholds can be specified in absolute (count) or relative (percentage) terms and are triggered on an equal-to-or-greater-than basis. If you set both absolute and relative thresholds, they are evaluated with OR logic.

- In this exercise we will configure the threshold for removal of persons in an absolute way to prevent the removal persons in case we don't have any persons available from the latest data feed.
- The first step is to enable the removal threshold by flipping the enabled toggle into the **"ON"** state.
- Secondly we configure the minimum count value for the removal threshold set to 1, please be aware to not use a value set 0 to as threshold limits because this value will be ignored by default.
- After the change is made the settings needs to be saved.
- Edit the **"T4E_HelloID_Persons.csv"** file and remove the last line containing person data for the person **Xaviera Foley**.
- Save and close the file.
- Try to re-import the source data again having the configured import threshold in place and validate the results.
- As you can see the import is blocked and the person **Xaviera Foley** does still exist in the HelloID person list.
- From the source system we now can approve the import, assuming the removal of the person **Xaviera Foley** is correct.
- To manually override the threshold block, select the yellow Approve import button and select the Yes button on the confirmation dialog box. This approves the Blocked import (without performing a new import).
- After the approved import is completed validate if the person **Xaviera Foley** doesn't exist anymore and has been removed from the HelloID person list.

2.2 working with data validation thresholds

Each mapped field on the Person and Contract tabs has a Require this field toggle. When this toggle is turned on, any Persons for whom the field's mappings produce an empty value will not be imported. Persons blocked in this way are listed on the Blocked persons tab.

- In this exercise we will configure the validation of fields required by the source data import.
- The first step is to enable the **Require this field** option on the **Name.NickName** field from the mapping by flipping the enabled toggle into the **“ON”** state.
- After the change is made the mapping needs to be saved.
- Edit the **“T4E_HelloID_Persons.csv”** file and remove the NickName value for the person **Walker Clark**.
- Save and close the file.
- Try to re-import the source data again having the configured field validation in place and validate the results.
- As you can see the import isn’t blocked in general, but we do have a blocked person object in our latest source import.
- To validate the result, we must check the **blocked persons** overview on the connector configuration page.
- Select the wrench icon on the connector configuration and navigate to the **blocked persons** tab.
- To see a side-by-side comparison of raw and mapped data, select the report icon button next to Validation results.

There are no starter scripts available for this lab.

LAB 3 – Create a PowerShell target system to export data

In order for HelloID to provisioning to do anything, it must be connected to a target system. It is within that target system that HelloID will create and manage accounts, accesses, and permissions. Target systems can be added from the HelloID catalog for build in systems like Microsoft Active Directory, Microsoft Azure Active Directory, or a custom PowerShell system that you can configure on your own. The latter option makes HelloID provisioning an extremely powerful and flexible solution for any organization.

In this exercise we create a custom PowerShell target connector, one responsible for creating accounts in a *.csv export file. From the Fields tab within the Target Connector, we can map the full person model. From the Fields tab within the Target Connector, we can map the full person model. The mapped fields will be available in the $actionContext.Data object (as input).

Beside the input, HelloID also requires a result as output which should be storde in the $outputContext object. The objects nested within the $outputContext.Data object can be used for notification purposes when “USE IN NOTIFICATIONS” is toggle on.
Audit logging can be added to the $outputContext.Auditlogs object and contains 4 sub objects: Action, Message and IsError (True/False) and AuditLogs.

There are 4 important objects to send within $outputContext:

- $outputContext.Succes : Result of the script (True/False)
- $outputContext.Data : Objects that can be used for notification
- $outputContext.AccountReference : Reference of the created account
- $outputContext.Auditlogs : information about the action performed (success or failure).

To relate the newly created object we must define a unique attribute, this attribute is used in other processes (enable, disable, update, or delete) to identify the object we are working on. We recommend using a custom identifier by using, as best practice, the ExternalID of the person. This is what we call the AccountReference. This can be stored in the $outputContext.AccountReference object.

After the script was executed successfully, the $outputContext will be sent to HelloID automatically.

In this exercise our goal is to create a *.csv file containing the **$actionContext.Data** information meeting the requirements on the next page.

- Fields needed in the export are listed below. Create a mapping for this fields.

Fieldname	**Person field name**
department	PrimaryContract.Department.DisplayName
displayName	DisplayName
endDate	PrimaryContract.EndDate
externalID	ExternalID
firstName	Name.NickName
lastName	Name.FamilyName
lastNamePartner	Name.FamilyNamePartner
manager	PrimaryManager.DisplayName
middleName	Name.FamilyNamePrefix
middleNamePartner	Name.FamilyNamePartnerPrefix
startDate	PrimaryContract.StartDate
title	PrimaryContract.Title.Name
username	Username

- Export data of the complete $actionContext.Data object need to be in the *.csv file.
- Make sure the connector is configured to act as on-premise system.
- After the account has been successfully created the **$outputContext.Success** must be **$True**. If the create action not completed successfully the **$outputContext.Success** result needs to be **$False**;
- Script variables to used for the export filename and directory need to be configured in a Custom Connector Configuration UI.

A starter script for this lab can be found inside the following GitHub repository

LAB 4 – Add DisplayName generation on target system

In order for the accounts to be created with the desired DisplayName, we need to calculate this field based on the separated name fields available in the person object. Now it’s configured to be mapped with the DisplayName field from the person model. Based on the naming convention from the source system, we need to add logic in our mapping to generate the desired DisplayName.

4.1 Preparation

When you start with the name generation, it is important to determine in advance what the name generation should look like. The name generation should look like this example:

Field	Value
{Name.NickName}	Holly
{Name.FamilyNamePrefix}	van
{Name.FamilyName}	Medina
{Name.FamilyNamePartnerPrefix}	de
{Name.FamilyNamePartner}	Foster

Result:

- **B** **-** Holly van Medina
- **P** **-** Holly de Foster
- **BP** **-** Holly van Medina – de Foster
- **PB** **-** Holly de Foster – van Medina

4.2 Add the generation to the target system

- Edit the DisplayName mapping field so that it becomes a complex mapping. Write a complex mapping in which the DisplayName is generated based on the name preference of the person. Please keep in mind that the DisplayName must also be updated when this is needed (by example, when a person got married or divorced). A starter script for this complex mapping can be found inside the following GitHub repository
- Keep in mind that when the naming convention is not present or empty, we need a correct DisplayName anyway.

4.3 Check the generated name

After the creation of the logic, please check the exported CSV file if the configured DisplayName is present and exported.

A starter script for this lab can be found inside the following GitHub repository

LAB 5 – Use data from another system

In some cases you may want to use data generated in, for example, Active Directory to pass to another system. In this case we want to export the generated sAMAccountName in the CSV. We need to add this to field mapping of our PowerShell target system. We can do this by making a dependency. In this Lab we will extend the mapping in the PowerShell Target connector with the generated sAMAccountName of the Microsoft Active Directory target system.

5.1 preparation

- Verify if the sAMAccountName is stored in the person account data on your Microsoft Active Directory Target system.
- If not, please save it and re-run the provisioning of the accounts.
- Please check on the Accounts tab on the person if the configured data (sAMAccountName) is stored

5.2 using account data from other systems

Before we can use the account data from other systems, we need to configure this on the created PowerShell Target system.

- Go to the settings of the PowerShell Target system and configure the **“Use account data from other systems”** option, select the correct system.
- Check the name of the **“other system”** after configured. Please note that we need that name later within our PowerShell Script.

To be able to use the data from the configured **"other system"**, an additional object is available in the person model within HelloID. The person model now contains an object called **"Accounts"**. This contains all the account information of a person that was saved on other target systems.

5.3 Adding the sAMAccountName to the account object

To add the sAMAccountName to the export CSV, we need to add an additional mapping to extend the **$actionContext.Data** object with an additional field.

- Add an additional field to the mapping and call it ADsAMAccountName
- Get the stored sAMAccountName value from the Microsoft Active Directory target by using a complex mapping.
- Dry-run the PowerShell script and verify if the ADsAMAccountName is present and filled with the correct value.
- Execute the process and check if the value for ADsAMAccountName is present in the exported csv file.

A starter script for this lab can be found inside the following GitHub repository

LAB 6 – Correlation on PowerShell target system

In order to prevent creation of duplicate accounts we need to add correlation to the scripting to be able to check if an account which will be created already exists or not. Here for we need to add some logic to the Account Create event to evaluate this.

First, we need to set the correlation settings in the configuration of the target system. In the Correlation tab we have the option to enable correlation and set which source field (from the person) and which field from the target system we should correlate on. The available fields from the target system are the field we defined on the Fields tab (mapping).

Keep in mind that manual correlation is not possible when using a PowerShell Custom Target Connector. Also no report is available in the UI.

6.1 preparation

- Make sure there is some data present in the current Target System.
- Determine on which field the correlation should be done and enable correlation and set the correct values on the “person correlation field” and the “account correlation field”.

6.2 Add correlation to your Account create event

- Get the data from your previously exported target CSV file.
- Create some logic to correlate the persons from HelloID to the Accounts in your CSV file, when an account already exists don’t create a duplicate account. When an account does not exist, create it.
- Make sure there will be no duplicated accounts created.
- Check if you have no duplicated accounts and check the newly added account(s).

A starter script for this lab can be found inside the following GitHub repository

LAB 7 – Create PowerShell notification system

Within HelloID it is possible to send notifications based on a custom event. For example to sent a notification when a certain value (e.g. displayName) has been changed. In this lab we will create a custom event and create a notification based on that event.

First, we need to add an "Account update" script in our target connector. We need to import the previous account data to use it to check the current fields on the target.

Be aware that this functionality will only be available for the PowerShell V2 target systems. This will **not** be available in the PowerShell V1 target system.

7.1 preparation

- Make sure that a notification is sent when the DisplayName of an account changes.
- Within the notification we need to see the changes that were made.

7.2 Add a script for Account update lifecycle event

- Create a new PowerShell script for the account update lifecycle event. A starter script for this lab can be found inside the following GitHub repository.
- Import the data from the current csv file and use this data to populate the $previousAccount object.

7.3 Create a custom event notification

- Create a new custom event that will trigger when the DisplayName of an account changes.
- Create the custom event based on the PowerShell target connector.
- Create a new notification based on the created custom event.
- Show the differences in the body of the email.

7.4 Test the notification

- Change the lastname of **Brian Schwartz** in the Source Files from the PowerShell Source system.
- Import the data and check if the lastname has been changed on the person record.
- Force HelloID to update the account on the target system and check the email message.

Tools4ever Software BV

Adres	Amalialaan 126c 3743 KJ Baarn Nederland
Telefoon	+31 (0) 35 54 832 55
Website	tools4ever.nl
Informatie	info@tools4ever.com
Sales	sales@tools4ever.com