Technical Reference

DaaS Multi-value Transformations

Identity Governance as a Service

MICRO® FOCUS

Contents

Introduction

This technical reference explains how to data can be transformed from multiple data sources into Identity Governance as a Service and introduces the concept of **Multi-field mapping**.

Understanding DaaS

DaaS has long provided the ability to transform data values collected from a target application through the use of ECMAScript and the Java ScriptEngine. These transformations have always been a 1:1 transform of a value from the application namespace into a value in the client namespace. A typical example of this type of transformation would be the conversion of a date format “13-Sep-2014” into the Java Date milliseconds format “1410580800000” expected by Identity Governance product.

Although single value conversion provides a great deal of useful functionality, there are many use cases that have come from analysts, SEs, and customers that require more flexible conversions. For example, while modeling a Data Access Governance scenario using Novell File Reporter data we had a need to combine 2 collected fields (“share” and “rights”) into a single permission Identifier. Since no such capability existed in DaaS, the data had to be massaged to a great degree before it could be used.

Our Sales Engineering team has indicated that this lack of transformation capability may not seem like a hindrance to the customer “business” user, the “technical” users have noted this shortcoming as they observe the SEs massaging data as they build PoCs. As a result, it has been indicated that Identity Governance has thus been adversely affected in competitive situations.

Adding new functionality to DaaS

New functionality can be quickly added to DaaS to allow application namespace field mapping to more than one data field. This is referred to as **Multi-field mapping** in this document.

Before you can collect data using multi-field mappings, you must first create a collector with the appropriate values. To build a client-side “userId” value from the “givenName”, “initials”, and “sn” attributes that are collected from an Identity source, specify the multi-field mapping using a JSONArray as follows:

User ID from Source: [“givenName”, “initials”, “sn”]

First Name: givenName

Last Name: sn

Request Build

DaaS will detect the JSONArray format Multi-field mapping, and the fields of the array elements will be parsed and added to the collector query which is cached for the view:

“read-attrs”: [“givenName”,“initials”,”sn”,”title”…]

It is not necessary for the fields in the Multi-field mapping to be independently mapped to other client attributes. DaaS also ensures that each application attribute is only present one time in the “read-attrs” array passed to the collector

Response Build

After the query is issued and the application collector returns the requested data, DaaS will perform response pre-processing to build a JSONObject containing the Multi-value mapped attributes and the collected values (if any). This JSONObject will be the mapped value. Using our example above, the following value would be mapped to the “userId” attribute:

```
{
   “givenName”: “Bob”,
   “initials”: “J.”,
   “sn”: “Marley”
}
```

This “value” is passed to the transformation for the “userId” attribute.

Transformation

As with all transformations supported by DaaS, the transform script must accept a variable called inputValue and populate a String variable outputValue. Continuing with the example shown above, a sample transformation may be as follows:

```
/* Build a response value with concatenation of first letter of 'givenName',
first letter of 'initials', and 'sn'. All set to lowercase */


/* Parse JSON components */
var jObject = JSON.parse(inputValue);
var fName = jObject.givenName;
var lName = jObject.sn;
var mi = jObject.initials;


/* Do concatenation */
var id = fName.substring(0,1).concat(mi.substring(0,1)).concat(lName);


/* Set outputValue */
outputValue = id.toLowerCase();
```

The result of the transformation is the string “bjmarley”. It is this value that will ultimately be “collected” into Identity Governance as the userId for the object.

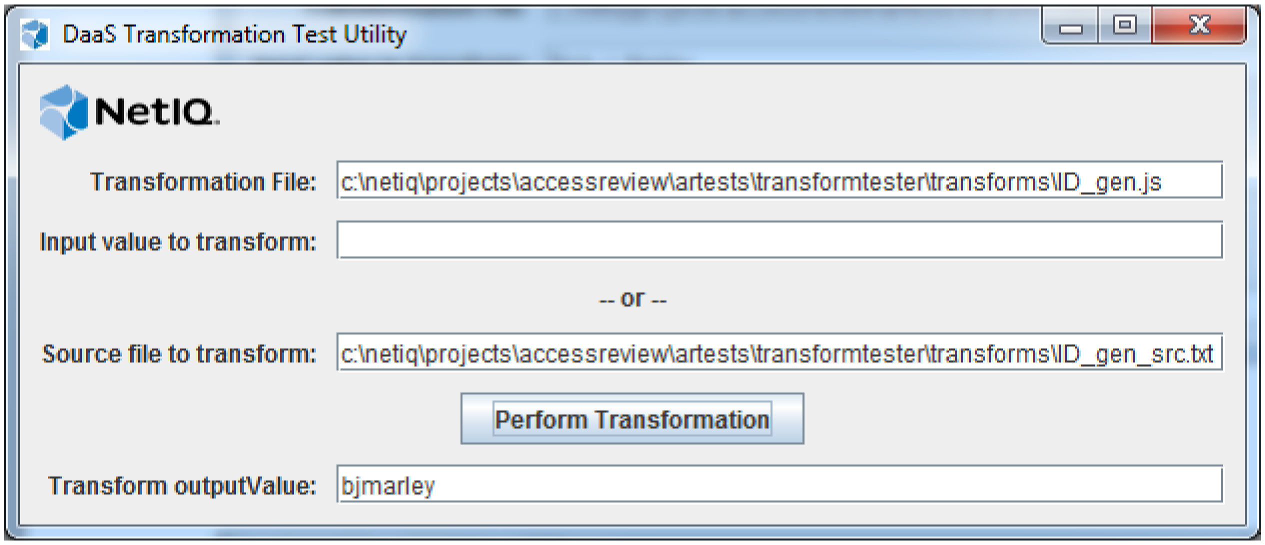
Caution: Only the “C” style /* */ comments are accepted by ScriptEngine.

Testing

To facilitate more rapid development and testing of Transformations, a script testing utility has been developed to ensure any script, either single-field or Multi-field, performs as expected prior to deployment into an Identity Governance Collector Template or Data Source configuration.

The utility accepts the name of a transformation file and either a String value for the script “inputValue” or an input file that is read and passed to the transformation script.

String Input Conversion Example


DaaS Transformation Test Utility

NetIQ

Transformation File: c:\netiq\projects\accessreview\artests\transformtester\transforms\ID_gen.js

Input value to transform:

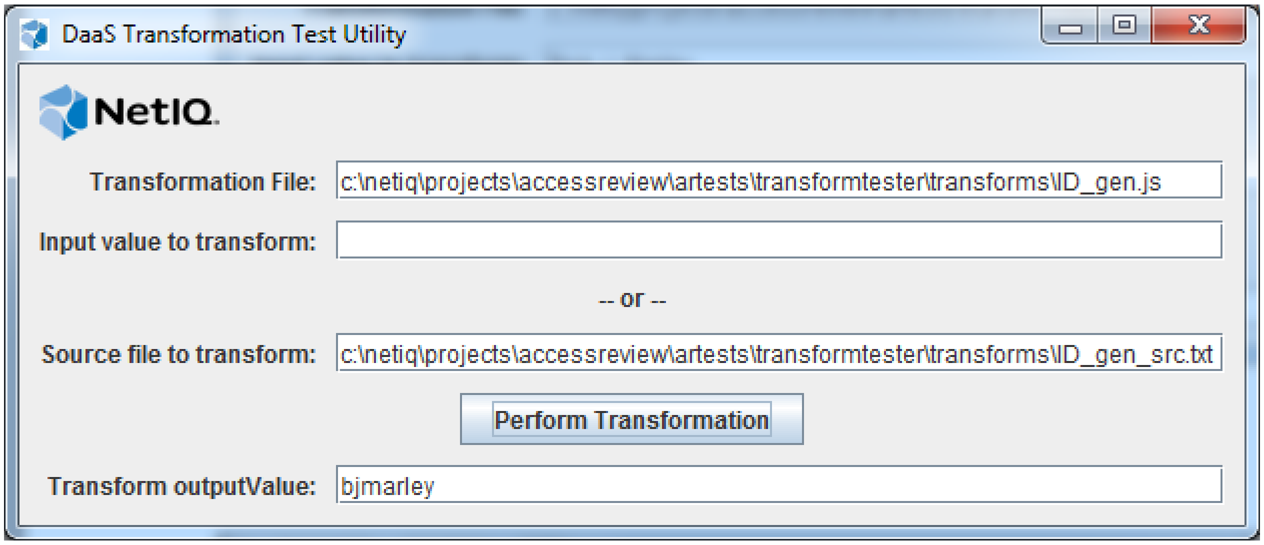
-- or --

Source file to transform: c:\netiq\projects\accessreview\artests\transformtester\transforms\ID_gen_src.txt

Perform Transformation

Transform outputValue: bjmarley

File Input Conversion Example


DaaS Transformation Test Utility

NetIQ

Transformation File: c:\netiq\projects\accessreview\artests\transformtester\transforms\ID_gen.js

Input value to transform:

-- or --

Source file to transform: c:\netiq\projects\accessreview\artests\transformtester\transforms\ID_gen_src.txt

Perform Transformation

Transform outputValue: bjmarley

Supporting Data

The following scenarios, as well as many other anecdotal user cases were used as justification for the effort to develop this feature.

Gartner Questionnaire

Gartner suggests that there will be cases where the account ID of a collected system will be inconsistent with collected identity information making correlation a challenge:

Let's say you have collected identities with attributes such as userid, firstname, lastname, employeeid, department.

Additionally, you have a data feed from an application (EG. RACF) in which the accountid is numeric but there is a field in there called "Programmer" which is formatted "lastname, firstname". We must show that because the accountid is useless for correlation, we can accommodate a matching rule which will nevertheless associate the RACF accounts correctly.

A simpler case (and one that we support already) is where we might have a data feed from an application in which the accountid is unusable but there is an attribute of the account called empid. In this case we would match (application) empid to (identity) employeeid.

Scenario 1 (b): Customer has more than 100 target systems for which they would like to know who has access to what in their environment. They would like to import the entitlements data as quickly as possible into the entitlements repository, so they have identitified LDAP and database interfaces for some of the applications and collected comma- or tab-separated value files for the remaining systems. The collected files do not necessarily conform to any consistent format. They would like to import the entitlements data and then associate accounts with owners based on matching rules. Target system accounts do not necessarily follow a standard naming convention, so matching rules will need to rely on attribute values in some cases.

Solution with new capability

Map the Identity “fullName” to the collected attributes FirstName and LastName:

```
“fullName”  <-> [FirstName, LastName]
```

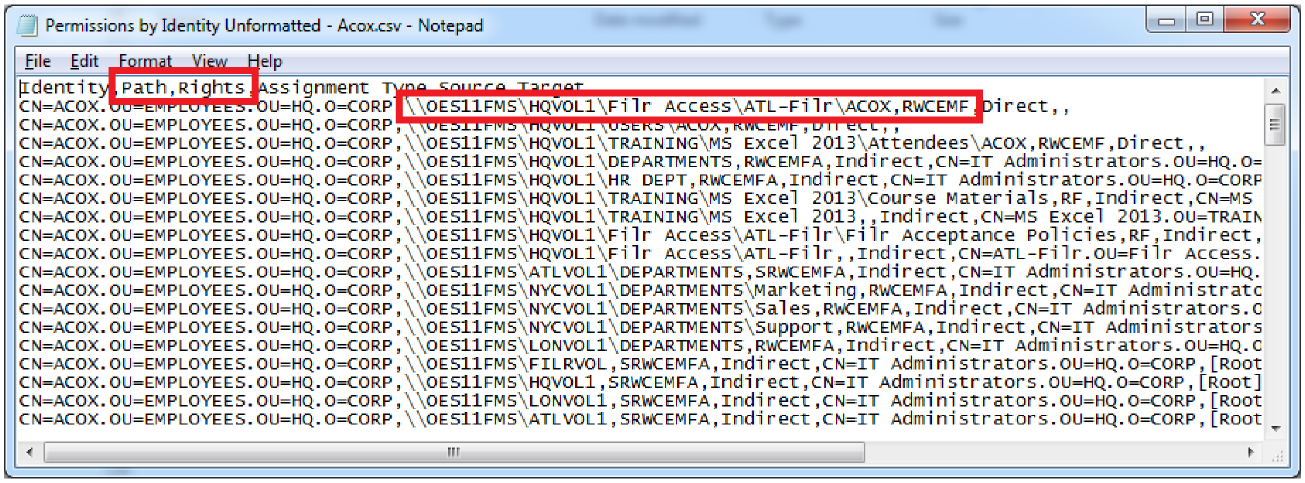
The transformation associated with this mapping will join the two values as desired:

```
“lastname, firstname” into the attribute “fullName”
```

With the new attribute on the identities, the accounts can be easily and natively mapped using the Account `“holderLink”` mapped to “Programmer” and Joined to `“fullName”`

Data Access Governance Scenario

Micro Focus created this Identity Governance scenario that utilized out-of-the-box User Permission report data and collected it with an application collector. The data is shown in the following example:


Permissions by Identity Unformatted - Acox.csv - Notepad
File Edit Format View Help

```
Identity,Path,Rights,Assignment Type,Source,Target
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\Filr Access\ATL-Filr\ACOX,RWCEMF,Direct,,
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\USERS\ACOX,RWCEMF,Direct,,
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\TRAINING\MS Excel 2013\Attendees\ACOX,RWCEMF,Direct,,
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\DEPARTMENTS,RWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\HR DEPT,RWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=CORP
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\TRAINING\MS Excel 2013\Course Materials,RF,Indirect,CN=MS
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\TRAINING\MS Excel 2013,,Indirect,CN=MS Excel 2013.OU=TRAIN
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\Filr Access\ATL-Filr\Filr Acceptance Policies,RF,Indirect,
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1\Filr Access\ATL-Filr,,Indirect,CN=ATL-Filr.OU=Filr Access.
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\ATLVOL1\DEPARTMENTS,SRWCEMFA,Indirect,CN=IT Administrators.OU=HQ.
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\NYCVOL1\DEPARTMENTS\Marketing,RWCEMFA,Indirect,CN=IT Administrato
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\NYCVOL1\DEPARTMENTS\Sales,RWCEMFA,Indirect,CN=IT Administrators.O
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\NYCVOL1\DEPARTMENTS\Support,RWCEMFA,Indirect,CN=IT Administrators
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\LONVOL1\DEPARTMENTS,RWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\FILRVOL,SRWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=CORP,[Root
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\HQVOL1,SRWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=CORP,[Root]
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\LONVOL1,SRWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=CORP,[Root
CN=ACOX.OU=EMPLOYEES.OU=HQ.O=CORP,\\OES11FMS\ATLVOL1,SRWCEMFA,Indirect,CN=IT Administrators.OU=HQ.O=CORP,[Root
```

Since the data contained both a File share “Path” field, and the access “Rights” to that share, we determined that joining those two fields would provide a viable “permissionId” field for Identity Governance. To achieve this, the CSV file had to be manually manipulated to create a new joined ID field. A tedious task for the mere handful of data records we were processing.

Solution with new capability

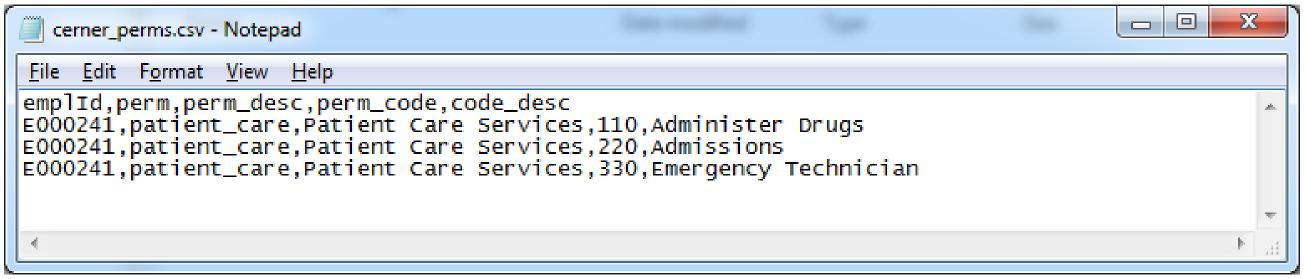
Map the Permission “permissionId” to the collected attributes ‘Path’ and ‘Rights’:

```
“permissionId”  <-> [“Path”, “Rights”]
```

Then provide a transform to join them with a ‘:’ separator.

Similar Scenario

The following CSV file was presented to me with the desire expressed to combine two of the fields into a permission identifier and two other fields joined as the permission description:



```
emplId,perm,perm_desc,perm_code,code_desc
E000241,patient_care,Patient Care Services,110,Administer Drugs
E000241,patient_care,Patient Care Services,220,Admissions
E000241,patient_care,Patient Care Services,330,Emergency Technician
```

The solution is to map the following transformations in the **Permission** collector:

Permission ID from Source: [“perm”, “perm_code”]

Permission Name: [“perm_desc”, code_desc:]

Legal Notices