IBM – Tech Note

SCIM Adapter support for Extended schema attributes

Table of Contents

SCIM Adapter – Extended Schema attribute support

Create a .txt file to key in the extended schemas attributes of SCIM supported target system. Refer SampleExtendedSchemaAttributeMappingFile.txt comes along with SCIM Adapter Package.

Service form – Enter the .txt filepath (for example : {SDI_HOME}/timsol/ ExtendedSchemaAttributeMappingFile.txt) in the **SCIM Schema Extended File Path.**

Adapter will read the .txt file from service form and segregate the schema and schema related attributes

SampleExtendedSchemaAttributeMappingFile.txt:

This mapping file should follow the below format,

<Schema1>

<attribute1> //with erAttributeMapping

<Schema2>

<attribute2> //with erAttributeMapping

Example:

```
urn:ietf:params:scim:schemas:extension:ibm:2.0:User //schemaExtensionName

attribute1|erattribute1

attribute2|erattribute2
```

Adapter support the following attribute structures, Example patterns of single value & complex values in schema extended attributes which adapter currently supports:

SampleExtendedSchemaAttributeMappingFile.txt

```
schemaExtensionName
attribute1:erattribute
attribute2:erattribute|multivalue:yes
attribute3|NA|value:erattributevalue
attribute4|type:erattributetype|value:erattributevalue
attribute5|type:erattributetype|subattribute1:ersubattribute1|subattribute2:ersubattribute2|subattribute3:ersubattribute3
attribute6|subattribute1:ersubattribute1|subattribute2:ersubattribute2|subattribute3:ersubattribute3
endpoint:/Endpoint|id:erendpointid|displayName:erendpointdisplayName|value:erendpointmember|objectclass:erendpointclass
```

1.Single-value attribute support

Schema:

```
"urn:ietf:params:scim:schemas:extension:ibm:2.0:User": {
        "pwdReset": true,
        "userCategory": "regular",
        "twoFactorAuthentication": false,
        "realm": "cloudIdentityRealm",
        "pwdChangedTime": "2021-07-29T10:48:48Z"
    }
```

Syntax:

realm:erRealm → First data is for target attribute and separated using ":" and followed with source attribute. Schema attribute "realm" value will be stored in "erRealm"

Example:

```
realm:erRealm
```

2.Complex-attributes support

Complex attribute Pattern 1: Root attribute with sub-attributes if the attributes have canonical **type**, by using "**type"** segregate the sub-attributes of root attribute

Schema:

```
"addresses": [
        {
            "country": "IN",
            "type": "work",
            "streetAddress": "100 Universal City Plaza",
            "locality": "Hollywood",
            "region": "CA",
            "postalCode": "91608",
            "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
            "primary": true
        },
       {
            "country": "IN",
            "type": "home",
```

```
            "streetAddress": "100 Universal City Plaza",
            "locality": "Hollywood",
            "region": "CA",
            "postalCode": "91608",
            "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
            "primary": true
        }
    ]
```

Syntax:

addresses → First data is a root attribute.

type:work → Second data is going to be the "Type", depends on the "Type" values will be fetched.

country:erworkcountry → Third data is sub-attibute of root attribute "addresses", take "country" data from response and stored in erworkcountry. Other sub attributes are also stored similarly.

region:erRegion. → Forth data is also a sub attribute of root attribute "addresses", take "region" data from response and stored in erworkRegion

city:erCity → Fifth data is also a sub attribute of root attribute "addresses", take "city" data from response and stored in erworkCity.

Example:

```
addresses|type:work|country:erworkcountry|region:erworkregion|city:erworkcity
addresses|type:home|country:erhomecountry|region:erhomeregion|city:erhomecity
```

Complex attribute Pattern 2: If it is having canonical (type) and value, set the "**value"** as key and data of "**value"** will be set as the value of attribute.

Schema:

```
"phoneNumbers": [
    {
      "type": "work",
      "value": "4435466"
    }
  ]
```

Syntax:

phoneNumbers → First data is a root attribute.

type:work "Type" → Second data is going to be the "Type",depends on the values will be fetched.

value:erPhoneNumbers "value" should be → Third data is for subattributes of root attribute "phoneNumbers". Here "value" is the subattributes, data in stored in erPhoneNumbers.

Example:

```
phoneNumbers|type:work|value:erPhoneNumbers
```

Complex attribute Pattern 3: if no type, get the sub-attributes directly and set the value

Schema:

```
"manager": {
            "displayName": "cUser",
            "value": "6410013TFF",
            "$ref": "https://kaviya02.verify.ibm.com/v2.0/Users/6410013TFF"
        }
```

Syntax:

Manager → First data is a root attribute.

NA → Second data is going to be the "Type", here "Type" is NA.

value:erManager "value" erManager. → Third data is for subattributes of root attribute "manager".Here is the sub-attributes, data in "value" should be stored in

Example:

```
manager|NA|value:ermanager
```

Note:

This kind of supported data attribute will come with **account object Class** because it extended with /Users endpoint, so not required to give separate object class name

Complex attribute Pattern 4: Complex attributes with multiple sub attributes

Schema:

```
"name": {
                "formatted": "givenName surName",
                "familyName": "surName",
                "givenName": "givenName"
            }
```

Syntax:

name → First data is a root attribute.

NA → Second data is going to be the “Type”, here “Type” is NA.

formattedName:erformattedName → Third data is for subattributes of root attribute “formattedName” should be stored in erformattedName.

familyName:erfamilyName → Fouth data is for next subattributes of root attribute “familyName” should be stored in erfamilyName.

Example:

```
name|NA|formattedName:erformattedName|familyName:erfamilyName
```

3.Multivalued attribute support

Schema:

```
"territories" :[
      "Montana",
      "Idaho",
      "Wyoming"
]
```

Syntax:

territories:erTerritories → First data is for target attribute and separated using “:” and followed with source attribute. Schema attribute “territories” value will be stored in “erTerritories”.

multivalue:yes → Second data is for multivalued attribute flag “yes”

Example:

```
territories:erTerritories|mutivalue:yes
```

4.Supported Data attribute support

If extended schema has supported data attribute – supported data will have separate endpoint to recon and the endpoint should be provided in .txt file

Schema:

1.Support data recon for example: /Entitlements


```
    "id": "00e5g0000046UZBAA2",
    "displayName": "Identity User",
```


2. Entitlement membership comes with /Users endpoint for each user accounts


```
"entitlements": [
       {
       "value": "00e5g0000046UYyAAM",
       "$ref": "/Entitlements/00e5g0000046UYyAAM",
       "display": "System Administrator",
       "type": "Profile",
       "primary": true
       }
```


Syntax:

endpoint:/Entitlement -->First data is for endpoint URL.

id:entitlementid -->Second data is for supported data id attribute.

displayName:erEntitlementDisplayName -->Third data is for supported data displayName attribute.

display:erEntitlementDisplay -->Fourth data is for displaying the entitlement membership of users on account form

ObjectClass:erEntitlementClass -->Fifth data is for determining the objectClass.

Example:

```
endpoint:/Entitlements|id:erEntitlementid|displayName:erEntitlementDisplayName|valu
e|erEntitlement|objectclass:erEntitlementclass
```

Customizing the adapter profile

To customize the adapter profile, you must modify the SCIM Adapter JAR file, ScimAdapterProfile.jar.
Files need to be customized to perform the provisioning from ISIM/IGI.

- customLabels.properties
- schema.dsml
- erTDIScimAccount.xml
- service.def
- targetProfile.json (IGI only)

Extract the Adapter profile jar:

- To edit the LdapProfile.jar file, complete these steps:

1. Copy the SCIM profile JAR file into a temporary directory
2. Extract the contents of the JAR file into the temporary directory by running the following command:

   ```
   cd c:\temp
   jar -xvf ScimAdapterProfile.jar
   ```

   The jar command extracts the files into the directory.

3. Edit the file that you want to change.

Modifying the SCIM Adapter Profile files:

1. customLabels.properties:

Add the attributes to custom Labels file.

Example:

```
#entitlements
erscimentitlementmember=Entitlements
erentitlementid=Entitlement ID
erentitlementdisplayname=Entitlement Name
```

2.**schema.dsml:**

Attributes within account object class:

- Attribute type need to added in schema.dsml file and object-identifier must be unique value

Example:

```
<!-- ********************************************************** -->
<!-- eralias                                                    -->
<!-- ********************************************************** -->
<attribute-type single-value = "true" >
    <name>eralias</name>
    <description>Alias</description>
    <object-identifier>1.3.6.1.4.1.6054.3.192.2.41</object-identifier>
    <syntax>1.3.6.1.4.1.1466.115.121.1.15</syntax>
</attribute-type>
```

- Attributes within user object class need to be added inside erTDISCimAccount object class

Example:

```
<!-- ************************************************************ -->
<!-- erTDIScimAccount Class                                       -->
<!-- ************************************************************ -->
<class superior="top">
<name>erTDIScimAccount</name>
<description>SCIM Account class</description>
<object-identifier>1.3.6.1.4.1.6054.3.192.1.2</object-identifier>
<attribute ref = "eralias" required = "false" />
```

Attributes with extended supported data:

- Attribute type need to added in schema.dsml file and object-identifier must be unique value

Example:

```
<!-- ********************************************************** -->
<!-- erScimEntitlementID                                        -->
<!-- ********************************************************** -->
<attribute-type single-value = "true" >
    <name>erentitlementid</name>
    <description>SCIM Entitlement Id</description>
    <object-identifier>1.3.6.1.4.1.6054.3.192.2.42</object-identifier>
    <syntax>1.3.6.1.4.1.1466.115.121.1.15</syntax>
</attribute-type>

<!-- ********************************************************** -->
<!-- erScimEntitlementName                                      -->
<!-- ********************************************************** -->
<attribute-type single-value = "true" >
    <name>erentitlementdisplayname</name>
    <description>SCIM Entitlement Name</description>
    <object-identifier>1.3.6.1.4.1.6054.3.192.2.43</object-identifier>
    <syntax>1.3.6.1.4.1.1466.115.121.1.15</syntax>
</attribute-type>
```

```
<!-- ************************************************************ -->
<!-- erScimEntitlementMember                                      -->
<!-- ************************************************************ -->
<attribute-type single-value = "true" >
    <name>erScimEntitlementMember</name>
    <description>SCIM Entitlement Member</description>
    <object-identifier>1.3.6.1.4.1.6054.3.192.2.45</object-identifier>
    <syntax>1.3.6.1.4.1.1466.115.121.1.15</syntax>
</attribute-type>
```

- Creating a new object class for supported data attributes.

Example:

```
<!-- ************************************************************** -->
<!-- erTDIScimEntitlement Class                                     -->
<!-- ************************************************************** -->
<class superior="top">
<name>erentitlementclass</name>
<description>Scim entitlement class</description>
<object-identifier>1.3.6.1.4.1.6054.3.192.1.5</object-identifier>
<attribute ref = "erentitlementid" required = "false" />
<attribute ref = "erentitlementdisplayname" required = "true" />
</class>
```

3.**erTDIScimAccount.xml:** Modifying account form to display the data in ISIM/IGI.

- Single data form element

```
<formElement name="data.eralias" label="$eralias" isReadOnlyAlways="true">
    <input name="data.eralias" size="50" type="text"/>
</formElement>
```

- Supported data form element should be like,

```
<formElement name="data.erscimentitlementmember" label="$erscimentitlementmember">
    <searchFilter multiple="true" type="select">
    <filter>(objectclass&#61;erentitlementclass)</filter>
    <base>contextual</base>
    <attribute>erentitlementdisplayname</attribute>
    <sourceAttribute>erentitlementid</sourceAttribute>
    <size></size>
    <prepopulate>false</prepopulate>
    </searchFilter>
</formElement>
```

4.**service.def:** These changes only applicable when adding new object class for supported data extension

- Adding new Group Definition for newly added object class

```
<GroupDefinition profileName="SCIMEntitlementProfile"
    className = "erentitlementclass"
    rdnAttribute = "erentitlementdisplayname"
    accountAttribute = "erscimentitlementmember">

    <AttributeMap>
        <Attribute name = "erGroupId" value="erentitlementid"/>
        <Attribute name = "erGroupName"  value= "erentitlementdisplayname"/>
    </AttributeMap>
    <properties>
     <property name = "ExternalType" >
     <value>SCIMEntitlementProfile</value>
     </property>
     <property name="Managed">
     <value>false</value>
     </property>
    </properties>
</GroupDefinition>
```

5.targetProfile.json:

TargetProfile.json can be generated by using **targetprovidercreator** tool or changes can be done manually as follows,

Account object class attributes:

- Attributes within account object class can be included in **userExtension** section in json file.

Example:

```
},
{
  "name" : "eralias",
  "type" : "string",
  "multiValued" : false,
  "description" : "Alias",
  "required" : false,
  "mutability" : "readWrite",
  "caseExact" : false,
  "returned" : "default",
  "specialFlags" : "none",
  "uniqueness" : "none"
},
```

Supported data object class:

- Add the schema inside **resourceTypes,**

Example:

```
{
  "schemas" : [ "urn:ietf:params:scim:schemas:core:2.0:ResourceType" ],
  "id" : "erentitlementclass",
  "name" : "erentitlementclass",
  "endpoint" : "erentitlementclasss",
  "description" : "erentitlementclass Resource",
  "schema" : "urn:ibm:idbrokerage:params:scim:schemas:core:2.0:SupportingData",
  "schemaExtensions" : [ ],
  "meta" : {
    "location" : "/v2/ResourceTypes/erentitlementclasss",
    "resourceType" : "ResourceType"
```

- member attributes need to de included inside **userExtension,**

Example:

```
    },
    {
      "name" : "erScimEntitlementMember",
      "type" : "string",
      "multiValued" : false,
      "description" : "SCIM Entitlement Member",
      "required" : false,
      "mutability" : "readWrite",
      "caseExact" : false,
      "returned" : "default",
      "specialFlags" : "none",
      "uniqueness" : "none"
    }
```

- Objectclass need to be included in **supportingDataAttributeMapping**

Example:

```
  {
    "id" : "erentitlementclass",
    "objectClass" : "erentitlementclass",
    "attributeMapping" : {
      "value" : "erentitlementid",
      "display" : "erentitlementdisplayname"
    }
  }
```

Note: If target creator tool has been used for generating the json file. Verify the output file manually, check the userExtension, supportingDataAttributeMapping and resourceTypes inclusion of supported data. If any mismatch in data do manual change and use the file.

Reconstruct & import the Adapter profile jar:

- After you edit the file, you must import the file into the IBM Security Identity server for the changes to take effect. To import the file, perform these steps:

1. Create a JAR file with the files in the \temp directory by running the following commands:

   ```
   cd c:\temp
   jar -cvf ScimAdapterProfile.jar ScimAdapterProfile
   ```

2. Import the JAR file into the IBM Security Verify Governance Identity Manager server.
3. Stop and start the IBM Security Identity server
4. Restart the adapter service.