

# IDP 4 Attribute resolution

The Shibboleth IDP's [Attribute Resolver](#) collects [data](#) from authoritative sources (systems of record) and transforms and encodes the data as needed, before it is passed on to the [attribute filter](#).

All parts of the resolver are always executed in a default configuration, even though much of the data gathered may be discarded later on in the [attribute release/filtering](#) stage. Because of this **all configured data sources (such as LDAP Directory Services or Relational Database Systems) should be highly available** (and contain up-to-date, correct data, of course) – at least as highly available as you expect your IDP deployment to be. With LDAP directory servers achieving redundancy and replication usually is a much simpler (and cheaper) task than with RDBMSs, so sometimes it's worth the extra effort of synchronising data from other sources into an LDAP directory first and only pointing the IDP to these LDAP data sources. But all of this depends on local Identity Management decisions and processes and no one recipe will fit all. Feel free to [discuss](#) pros and cons of approaches and tools on the [eduD.at community mailing list](#).

We'll essentially replace the default `/opt/shibboleth-idp/conf/attribute-resolver.xml` content with our own definitions below, though keeping all the attribute ids that are now standardised (via the IDP's Attribute Registry feature, more on that below). Backup copies of all configuration files can always be found in `/opt/shibboleth-idp/dist/conf/` for comparison and as source for copying/pasting of more/other definitions.

The attribute resolver contains two kinds of configuration items: `DataConnectors`, which supply input data from data sources as the LDAP or Database servers described above, and `AttributeDefinitions`, which transform the individual data elements (e.g. name, email address) retrieved from those `DataConnectors`. For the proper on-the-wire representation as SAML attributes (or for other protocols) the IDP comes with a default set of transcoding rules referenced in `/opt/shibboleth-idp/conf/attributes/default-rules.xml`. (Deployers of earlier versions of the software will notice how short and clean `AttributeDefinitions` can be, and the "missing" `DisplayName` and `AttributeEncoder` elements can all be found in the referenced transcoding rules, e.g. `conf/attributes/inetOrgPerson.xml`, `conf/attributes/eduPerson.xml` and so on.

- [Attribute definitions](#)
  - [Name attributes](#)
    - [givenName](#)
    - [sn](#)
    - [displayName](#)
  - [Identifiers](#)
    - [mail](#)
    - [eduPersonPrincipalName](#)
    - [SAML SubjectID](#)
    - [SAML PairwiseID](#)
    - [European Student Identifier](#)
  - [Authorization / Org data](#)
    - [eduPersonScopedAffiliation](#)
    - [eduPersonEntitlement](#)
    - [schacHomeOrganization](#)
- [Data Connectors](#)
  - [Static](#)
  - [LDAP](#)

## XML root element

This is the XML "container" element all `AttributeDefinitions` and `DataConnectors` need to be wrapped in. Be sure to also properly close the root element with the final line `</AttributeResolver>` as shown below:

### attribute-resolver.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeResolver
  xmlns="urn:mace:shibboleth:2.0:resolver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mace:shibboleth:2.0:resolver http://shibboleth.net/schema/idp/shibboleth-
attribute-resolver.xsd">

  <!-- *everything* below needs to be included between the opening and closing tags of this AttributeResolver
  element -->

</AttributeResolver>
```

You can always perform minimal sanity checks on your configuration by checking the well-formedness of the XML, in this case with the command (from within your IDP's `conf` directory):

```
xmlwf attribute-resolver.xml
```

## Attribute definitions

Below are examples for all of the attributes commonly used in (inter-)federation today. As such all eduID.at IDPs should at least be able to create all of those attributes, but their specific definitions may vary from organisation to organisation, due to local Identity Management and software choices. For more details on the syntax and semantics of this part of the IDP configuration consult [the relevant upstream documentation](#).



Most `AttributeDefinition` elements below reference an LDAP `DataConnector` by its default name "myLDAP". **Don't change that name** (in your own LDAP `DataConnector` and all `AttributeDefinitions` that reference it), then you can simply use all the examples provided in this wiki as is, without having to adjust the name of the referenced `InputDataConnector` every time you copy something from this documentation.



Also, **don't change the attribute ids used in this configuration** (e.g. `id="givenName"`): While these are only used and meaningful *internally* within the Shibboleth IDP software (and so could be named anything) the [attribute release](#) configuration provided in this documentation as well as the IDPv4's [Attribute Registry](#) reference those internal ids when deciding what attributes to release to what SAML Service Provider and how to encode them into different wire representations for different protocols.

If you have created attribute definitions that might also be relevant for other members of the community – e.g. based on the widely used (within Austria) [Campus Online](#) software – please share them! Contributions to this wiki are very much welcome. You can also just [send them to the eduID.at Operations Team](#), of course, and we will edit and include them here.

## Name attributes

Many SAML Service Providers will need the subject's real name in one form or another, e.g. to allow collaboration between subjects based on trusted, verified names. Some applications will only be able to use names in separate fields (first name, last name) and some are content with a unified field that contains the full name in a format chosen by the institution or the subject herself. So for interoperability every IDP will need to be able to provide both forms. Assuming you have the subject's name available in an LDAP directory and configured a `DataConnector` for that (see further below), these are simple examples to use:

### givenName

```
<AttributeDefinition id="givenName" xsi:type="Simple">
  <InputDataConnector ref="myLDAP" attributeNames="givenName" />
</AttributeDefinition>
```

### sn

```
<AttributeDefinition id="sn" xsi:type="Simple">
  <InputDataConnector ref="myLDAP" attributeNames="sn" />
</AttributeDefinition>
```

### displayName

If you have the `displayName` attribute available in your LDAP directory we'll go with that, as it's not generally being misused to store data other than the subject's real name (like "cn" commonly is, e.g. storing the subject's login name or userid instead of her real name).

#### displayName, alternative 1: direct lookup

```
<AttributeDefinition id="displayName" xsi:type="Simple">
  <InputDataConnector ref="myLDAP" attributeNames="displayName" />
</AttributeDefinition>
```

If you only have the "cn" LDAP attribute available and it contains the subject's full name (*not* her login name / userid) you can simply change the `InputDataConnector`'s `attributeNames` XML attribute to become `attributeNames="cn"` in the second line of the example above. (Don't change anything else!) In this case you'll need to be aware that contrary to "cn" [displayName is defined to be single-valued](#), though, and therefore you should be absolutely certain that your source attribute ("cn" in the modified example here) only ever contains a single value per subject in your LDAP deployment, for all subjects! If that requirement cannot be met you will need to use a slightly more complex `AttributeDefinition` that takes *one* of the values (usually the first one as returned by an LDAP search) and turns that into the `displayName` attribute.

If you don't have the subject's full name available in your LDAP deployment (or your "cn" attribute is unusable because it may contain multiple values for a subject, but your `givenName` and `sn` attributes are single-valued) you can create this SAML attribute from first name and last name dynamically within the Shibboleth IDP:


### displayName, alternative 2: create from givenName and sn

```
<AttributeDefinition id="displayName" xsi:type="Template">
  <InputAttributeDefinition ref="givenName" />
  <InputAttributeDefinition ref="sn" />
  <Template>${givenName} ${sn}</Template>
</AttributeDefinition>
```

Note that `givenName` and `sn` allow for multiple values, just like `cn`! So only use the above method if you're certain that your LDAP directory is only using those attributes with single values.

## Identifiers

With the notable exception of [Library Services](#) almost all (other) SAML Service Providers will need to consistently recognise a returning subject. I.e., they may not need to know *who* you are based on the identifier alone, but they will need to know that you are *the same subject* accessing their services as when you accessed it previously. That's what subjects expect themselves, of course, otherwise all their work/data stored at a service would be unavailable to them the next time they accessed the same service! See also this [comparison of commonly used identifiers and their properties](#) from the Shibboleth wiki.

 There are a few types of identifier attributes in use within the federation community, each with their own unique properties, advantages and disadvantages. These are all explained in the [Attributes](#) section (or rather its child pages), for each attribute. So go read (and possibly re-read) those, as *you will need to understand their definition and usage patterns* in order to be able to decide if/how you can support them in your IDP. As always: [Discuss with the community](#) if anything is unclear!

All eduID.at IDPs should be able to produce all of these identifiers, in order to be able to interoperate with – and make use of – all SAML Services Providers your community's members may need to work with.

## mail

If you have a "mail" attribute with the subject's email address available in your LDAP directory, the example below is all you need. Ideally the LDAP attribute should in practice be single-valued (i.e., never contain more than one attribute value for a subject) as some SAML Service Providers cannot handle multi-valued attributes. This [attribute's specification](#) does allow for multiple values, of course, but you may make your life harder by not being able to supply a single-valued version of what's in your LDAP directory.

```
<AttributeDefinition id="mail" xsi:type="Simple">
  <InputDataConnector ref="myLDAP" attributeNames="mail" />
</AttributeDefinition>
```

Some institutions may need more complex processing than the above, e.g. getting the value from one of several LDAP attribute depending on the role/affiliation of the subject (localPersonStudentMail, localPersonStaffMail). The eduID.at community (or the eduID.at Operations Team) will be able to supply you with other/more complex examples, so please ask.

Our documentation for the creation and usage of [eduPersonPrincipalName](#) also contains some info on using email addresses as identifiers (and why/when best to avoid it), but sometimes services just need an email address for email's sake and will rely on *other* attributes for the unique identification of the subject.

## eduPersonPrincipalName

[eduPersonPrincipalName](#) is commonly produced either from the local login name (`uid`, `sAMAccountName`) or by re-using the institutional email address as its value. Our [documentation for that attribute](#) explains the pros and cons of each approach in detail. Below you'll find examples for both main methods used to create that attribute. First, the variant based on login name (replace `sourceAttributeID="uid"` with `sourceAttributeID="sAMAccountName"` or whatever holds local login names in your LDAP directory):

### eduPersonPrincipalName, alternative 1: from local login name

```
<!-- https://wiki.univie.ac.at/display/federation/eduPersonPrincipalName -->
<AttributeDefinition id="eduPersonPrincipalName" xsi:type="Scoped" scope="{idp.scope}">
  <InputDataConnector ref="myLDAP" attributeNames="uid" />
</AttributeDefinition>
```

And here's a definition when you've chosen to re-use the institutional email address as `eduPersonPrincipalName` attribute value:



Only do this if you are certain that all email address values for all subjects from your institution are within (one of) your own institutional email domain(s). The `eduPersonPrincipalName` is filtered at SAML Service Providers to only allow domain values that have been allowed per each SAML Identity Provider, cf. the "attribute scope" column in the catalog of [eduID.at Identity Providers](#). If needed you can ask the eduID.at operations team to allow more of your domains in your IDP's SAML Metadata, to match the email domains in use within your LDAP email attributes. But **you cannot use email addresses as base for `eduPersonPrincipalName` attribute values if you populate external (arbitrary) email addresses in the referenced LDAP attribute.**

#### eduPersonPrincipalName, alternative 2: from email address

```
<AttributeDefinition id="eduPersonPrincipalName" xsi:type="Prescoped">
  <InputDataConnector ref="myLDAP" attributeNames="mail" />
</AttributeDefinition>
```

A more complex example could use a `ScriptedAttribute` type definition to enforce in code that only email addresses matching local mail domains will be used, but what to do about those addresses that fail the check? So in such cases it's probably best to choose some other strategy to create `eduPersonPrincipalName` values.

## SAML SubjectID

The [SAML SubjectID](#) can be seen as an opaque (not name-based, long "ugly" values), more stable version of [eduPersonPrincipalName](#). It is intended as a replacement for the [eduPersonUniqueID](#) attribute and possibly also for [eduPersonPrincipalName](#) itself. The example provided below re-uses configuration already made to support [persistent NameIDs](#), namely the properties `idp.persistentId.sourceAttribute` (from `/opt/shibboleth-idp/conf/saml-nameid.properties`) and `idp.persistentId.salt` (from `/opt/shibboleth-idp/credentials/secrets.properties`).

Provided you already have a stable, non-recycled (i.e., not reassigned from one subject to another) internal identifier for your subjects you can set that attribute in the `idp.persistentId.sourceAttribute` property of the aforementioned config file and it will also be used as the basis for the SubjectID attribute. The configuration below also re-uses the [salt](#) configured in the property `idp.persistentId.salt` to generate a salted hash of the chosen source attribute as (local part of the) SubjectID attribute value:

#### SubjectID, re-using the definitions for persistentIds

```
<AttributeDefinition id="subjectHash" xsi:type="ScriptedAttribute" dependencyOnly="true">
  <InputDataConnector ref="myLDAP" attributeNames="{idp.persistentId.sourceAttribute}" />
  <Script><![CDATA[
    var digestUtils = Java.type("org.apache.commons.codec.digest.DigestUtils");
    var saltedHash = digestUtils.sha256Hex("{idp.persistentId.sourceAttribute}.getValues().get(0) + "{idp.persistentId.salt}");
    subjectHash.addValue(saltedHash);
  ]]></Script>
</AttributeDefinition>

<AttributeDefinition id="samlSubjectID" xsi:type="Scoped" scope="{idp.scope}">
  <InputAttributeDefinition ref="subjectHash" />
</AttributeDefinition>
```

If you do **not** have such an identifier readily available but you can *fabricate* one based on other/more data that would be an alternative approach. For example, if login names may be reassigned at your organisation – meaning you cannot base SubjectID solely on login names, salted/hashed or not – you could concatenate the login name with something that's *not* going to be the same in the re-assigned account to create an interim attribute within the IDP that then becomes the basis for other attributes and NameIDs:

For example if accounts get a new account creation date after re-activation (i.e., when you first delete and then later re-create accounts) you should also be able to use a combination of `loginname+accountcreationdate` or something along those lines, which would then still be unique even if the same login name would later be reused in a new account for another – or even for the same, which may be an unintended but still acceptable side-effect – person. So first you'd pull the data to combine into an interim attribute definition, e.g.:

#### SubjectID amendment 1: Add interim attribute that combines MS-AD's UPN + whenCreated

```
<AttributeDefinition id="subjectIdBasis" xsi:type="Template" dependencyOnly="true">
  <InputDataConnector ref="myLDAP" attributeNames="userPrincipalName whenCreated" />
  <Template>${userPrincipalName} ${whenCreated}</Template>
</AttributeDefinition>
```

Then you put *this* attribute's id into your `saml-nameid.properties` configuration file so that this property can then be used in the rest of the configuration (same way for everyone, independently of the specifics):

```
idp.persistentId.sourceAttribute = subjectIdBasis
```

Finally you'd have to change *only one line* from the above example of creating SubjectIDs, replacing the `InputDataConnector` on the "subjectHash" attribute definition with the following `InputAttributeDefinition`:

#### SubjectID amendment 2: Change Input

```
<AttributeDefinition id="subjectHash" xsi:type="ScriptedAttribute" dependencyOnly="true">
-   <InputDataConnector ref="myLDAP" attributeNames="{idp.persistentId.sourceAttribute}" />
+   <InputAttributeDefinition ref="subjectIdBasis" />
   <Script><![CDATA[
```

Now the "subjectHash" attribute definition will use the "fabricated" identifier (combined from the `userPrincipalName` and `whenCreated` attributes) as basis and the rest of that example above works the same way for everyone (including the `samlSubjectID` attribute definition which does not need to be changed at all).



This configuration generates `samlSubjectID` attribute values dynamically based on the configured input attribute on each login, and without any way to manually influence or override this in selected cases. So if the configured (or fabricated) internal identifier you derive those attribute values from should ever change for a given person so will all her `samlSubjectID` attribute values for the services that receive them.

## SAML PairwiseID

The [SAML PairwiseID](#) is an opaque, persistent, service-specific pseudonym. It replaces the [eduPersonTargetedID](#) attribute as well as [SAML 2.0 persistent NameIDs](#). The example provided below re-uses configuration already made to support [persistent NameIDs](#), namely the properties `idp.persistentId.sourceAttribute` (from `/opt/shibboleth-idp/conf/saml-nameid.properties`) and `idp.persistentId.salt` (from `/opt/shibboleth-idp/credentials/secrets.properties`).

#### PairwiseID, re-using the definitions for persistentIds

```
<AttributeDefinition id="samlPairwiseID" xsi:type="Scoped" scope="{idp.scope}">
  <InputDataConnector ref="computed" attributeNames="computedId" />
</AttributeDefinition>
```

This references a `DataConnector` with `id="computed"` which you'll create using the next snippet (move it to the end of the `attribute-resolver.xml` file so it ends up next to the other `DataConnector` elements). Provided you already have a stable, non-recycled (not reassigned from one subject to another) identifier for your subjects stored in LDAP you can set that attribute name in the `idp.persistentId.sourceAttribute` property of the referenced config file and it will also be used as the basis for the `PairwiseID` attribute. The configuration below also re-uses the [salt](#) configured in the property `idp.persistentId.salt` to generate a salted hash of the chosen source attribute as (local part of the) `PairwiseID` attribute value:

#### DataConnector for PairwiseID

```
<DataConnector id="computed" xsi:type="ComputedId"
  excludeResolutionPhases="cl4n/attribute"
  generatedAttributeID="computedId"
  salt="{idp.persistentId.salt}"
  algorithm="{idp.persistentId.algorithm:SHA}"
  encoding="{idp.persistentId.encoding:BASE32}">
  <InputDataConnector ref="myLDAP" attributeNames="{idp.persistentId.sourceAttribute}" />
</DataConnector>
```

Similarly to SubjectIDs above: If you don't have a stable, non-reassigned internal identifier in your Systems of Record (LDAP directory, relational database) and decided to fabricate one (as shown in the examples for SubjectID above) you'll need to replace that `DataConnector`'s dependency with the custom one you created earlier, e.g.:

#### PairwiseID amendment: Change Input

```
-   <InputDataConnector ref="myLDAP" attributeNames="{idp.persistentId.sourceAttribute}" />
+   <InputAttributeDefinition ref="subjectIdBasis" />
```



This simple configuration generates `samlPairwiseID` attribute values dynamically based on the configured input attribute on each login, and without any way to manually influence or override this in selected cases. So if the configured (or fabricated) internal identifier you derive those attribute values from should ever change for a given person so will all her `samlPairwiseID` attribute values for the services that receive them.

Using a more complex configuration with [StoredID instead of ComputedID data connectors](#) and a [StorageService](#) it is possible to override or map chosen input attributes to previously generated output attributes. That would allow to keep the generated and released identifiers (`samlPairwiseID` attribute values) the same even in cases where the internal identifier did change for some reason, with a manual change in the database the generated values are persisted to. But having that corrective capability probably only makes sense if you reliably get notified about cases where the underlying internal identifier (that should never be reassigned from one person to another) actually has been reassigned. [Contact us](#) if you'd prefer to configure a storage service with persistent attribute values so that we can provide additional pointers for that. (E.g. an embedded [H2 database](#) does not create an external dependency and/or [SPOF](#) on a database server.)

## European Student Identifier

All Higher Education Institutions will want to make available the [European Student Identifier](#) (ESI) as that's one of the required attributes in order to access [Erasmus+](#) services.



The examples in this section assume use of the Shibboleth IDPv4 software *without* its new [Attribute Registry](#) and so will work as is for IDP systems that have been upgraded from IDPv3. It does so by including `AttributeEncoder` elements in the examples below. Deployers running IDPv4 *with* the Attribute Registry enabled (typically the result of a clean installation of IDPv4) will need to **remove any `AttributeEncoder` elements** from the `AttributeDefinitions` shown below. (They may also remove any `DisplayName` elements used in this section's examples: Like Encoders any `DisplayNames` will be provided by the IDPv4's Attribute Registry.)

(While this whole [documentation set](#) is geared towards fresh installations of IDPv4 it seems more useful to provide instructions that require part of the deployers to *remove* individual lines from the examples rather than expecting them to find out what and where to *add* something. This assessment may change and the direct applicability of the examples "reversed" in the future, though.)

Since the ESI won't be available in the exact format required by the ESI specification we'll provide examples that dynamically generate the ESI based on the `uid` attribute (coming from the `id="myLDAP"` `DataConnector`). In this example the `uid` attribute is expected to have values of the form "`x<MATRIKELNR>`" where `<MATRIKELNR>` is the student's Austrian [immatriculation number](#) so that we can extract a useable form of the immatriculation number from the `uid` attribute using [regular expressions](#). Any other values *not* matching the given pattern will lead to an empty ESI attribute which will ultimately not be released by the IDP:

### schacPersonalUniqueCode for ESI, variant 1

```
<AttributeDefinition id="schacPersonalUniqueCode" xsi:type="Mapped">
  <InputDataConnector ref="myLDAP" attributeNames="uid" />
  <DisplayName xml:lang="de">Europäische Studierendenkennung (ESI)</DisplayName>
  <DisplayName xml:lang="en">European Student Identifier (ESI)</DisplayName>
  <ValueMap>
    <ReturnValue>urn:schac:personalUniqueCode:int:esi:at:$1</ReturnValue>
    <SourceValue>^x([0-9]{8,})$</SourceValue>
  </ValueMap>
  <AttributeEncoder xsi:type="SAML2String" name="urn:oid:1.3.6.1.4.1.25178.1.2.14" friendlyName="schacPersonalUniqueCode" encodeType="false" />
</AttributeDefinition>
```

Don't forget to **adapt the pattern specified in the `SourceValue` element** above since you won't be finding the immatriculation number to be stored within your own systems in exactly the format shown above.

Instead of `uid` any other existing attribute can be used that already contains a form of the student's "Matrikelnummer". That may be the email address, `sAMAccountName`, `userPrincipalName` or [eduPersonPrincipalName](#). Any of those would be fine as long as you can extract the immatriculation number from it using a regular expression in order to transform it into an ESI.

In case you have the immatriculation number available in its own attribute already (i.e., no regular expression matching is needed to extract it from other data; we'll assume use of the fictitious `matrikelnummer` attribute below) you could use a "Template" attribute definition instead of the "Mapped" one above:

### schacPersonalUniqueCode for ESI, variant 2

```
<AttributeDefinition id="schacPersonalUniqueCode" xsi:type="Template">
  <InputDataConnector ref="myLDAP" attributeNames="matrikelnummer" />
  <DisplayName xml:lang="de">Europäische Studierendenkennung (ESI)</DisplayName>
  <DisplayName xml:lang="en">European Student Identifier (ESI)</DisplayName>
  <Template>urn:schac:personalUniqueCode:int:esi:at:${matrikelnummer}</Template>
  <AttributeEncoder xsi:type="SAML2String" name="urn:oid:1.3.6.1.4.1.25178.1.2.14" friendlyName="
schacPersonalUniqueCode" encodeType="false" />
</AttributeDefinition>
```

Finally, institutions that do **not** manage immatriculation numbers for their students but that still do need to provide the ESI attribute can use the following example to dynamically generate ESI values from any other locally available identifier (below again assuming use of the `uid` attribute) combining it with the canonical DNS Domain ("[scope](#)") of the institution:

### schacPersonalUniqueCode for ESI, variant 3

```
<AttributeDefinition id="schacPersonalUniqueCode" xsi:type="Template">
  <InputDataConnector ref="myLDAP" attributeNames="uid" />
  <DisplayName xml:lang="de">Europäische Studierendenkennung (ESI)</DisplayName>
  <DisplayName xml:lang="en">European Student Identifier (ESI)</DisplayName>
  <Template>urn:schac:personalUniqueCode:int:esi:%{idp.scope}:${uid}</Template>
  <AttributeEncoder xsi:type="SAML2String" name="urn:oid:1.3.6.1.4.1.25178.1.2.14" friendlyName="
schacPersonalUniqueCode" encodeType="false" />
</AttributeDefinition>
```

See the local (i.e., Austrian) profile of the [European Student Identifier](#) specification for the abstract requirements.

## Authorization / Org data

### eduPersonScopedAffiliation

[eduPersonScopedAffiliation](#) is sometimes used for simple authorisation cases. [eduPersonAffiliation](#) describes a person's relationship with the IDP's organisation in general terms (from a controlled vocabulary of 8 allowed values), and [eduPersonScopedAffiliation](#) is simply the scoped variant of that (i.e., the applicable affiliation values each suffixed with "@" + the main institutional domain, same as for [eduPersonPrincipalName](#) or [samlSubjectID](#)). Only the scoped version should be used for federated use cases: Even if a receiving Service Provider did not need to differentiate between e.g. `faculty@example.edu` and `faculty@research.example.com` it can always easily throw away the scope with minimal processing, yielding the unscoped version (here: "faculty" in both cases).

In the examples below we'll first create the unscoped version using one of several alternative methods, since this part will need be done differently at most organisations as it depends on local Identity Management choices. Then further below we'll create the scoped variant from that, in a configuration snippet that remains the same for everyone, no matter how the (unscoped) affiliation values was created.

Here's a very simple example that creates a few of the affiliation values based on a [regex](#) match of some other attribute's value, in this case the login name as stored in the "uid" LDAP attribute (cf. `sourceAttributeID`). Replace with "sAMAccountName" or "cn" or whatever as needed in your deployment. This method can be used if you assign login names based on a schema that encodes the role/affiliation of a subject into her login name/userid. While overloading identifiers with semantics (such as role information) is not recommended such practices exist so you might as well make use of them if it makes your IDP configuration easier. Note that identical `SourceValue` elements are used twice below to make sure all students are also members, and all staff are also members, too, as required by the [eduPerson specification](#).

#### eduPersonAffiliation, alternative 1: from login name

```
<AttributeDefinition id="eduPersonAffiliation" xsi:type="Mapped" dependencyOnly="true">
  <InputDataConnector ref="myLDAP" attributeNames="uid" />
  <ValueMap>
    <ReturnValue>student</ReturnValue>
    <SourceValue>m.+</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>staff</ReturnValue>
    <SourceValue>p.+</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>member</ReturnValue>
    <SourceValue>m.+</SourceValue>
    <SourceValue>p.+</SourceValue>
  </ValueMap>
</AttributeDefinition>
```

Here's another example where the correct affiliations are derived from the name of a locally defined group attribute (phoUsergroup in the example below). This example uses partial string matches for the individual affiliations, and a regex that makes all subjects with *any* group value (at least, or also) a member:

#### eduPersonAffiliation, alternative 2: from group names

```
<AttributeDefinition id="eduPersonAffiliation" xsi:type="Mapped" dependencyOnly="true">
  <InputDataConnector ref="myLDAP" attributeNames="phoUsergroup" />
  <DefaultValue>affiliate</DefaultValue>
  <ValueMap>
    <ReturnValue>faculty</ReturnValue>
    <SourceValue partialMatch="true">B</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>student</ReturnValue>
    <SourceValue partialMatch="true">ST</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>alum</ReturnValue>
    <SourceValue partialMatch="true">A</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>member</ReturnValue>
    <SourceValue>[^ ]+</SourceValue>
  </ValueMap>
</AttributeDefinition>
```

And here's another example that derives the applicable affiliations from the subject's LDAP Distinguished Name (DN) or rather its place within the LDAP Directory Information Tree (DIT). The referenced sourceAttributeID distinguishedName here is an operational attribute maintained by the LDAP server. (This may be called entryDN in other LDAP server implemenations.) Everyone within OU=MitarbeiterInnen will be assigned employee, only those within OU=Verwaltung,OU=MitarbeiterInnen will *additionally* become staff, and so on. In the final ValueMap all employees, staff, faculty and students *also* get assigned "member", as required by the [eduPerson specification](#).



### eduPersonAffiliation, alternative 3: from the object's DN or place in the DIT

```
<AttributeDefinition id="eduPersonAffiliation" xsi:type="Mapped" dependencyOnly="true">
  <InputDataConnector ref="myLDAP" attributeNames="distinguishedName" />
  <ValueMap>
    <ReturnValue>employee</ReturnValue>
    <SourceValue partialMatch="true">,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>staff</ReturnValue>
    <SourceValue partialMatch="true">,OU=Verwaltung,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org<
/SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>faculty</ReturnValue>
    <SourceValue partialMatch="true">,OU=Kollegium,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org<
/SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>student</ReturnValue>
    <SourceValue partialMatch="true">,OU=Studierende,OU=Personen,DC=example,DC=org</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>member</ReturnValue>
    <SourceValue partialMatch="true">,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org</SourceValue>
    <SourceValue partialMatch="true">,OU=Verwaltung,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org<
/SourceValue>
    <SourceValue partialMatch="true">,OU=Kollegium,OU=MitarbeiterInnen,OU=Personen,DC=example,DC=org<
/SourceValue>
    <SourceValue partialMatch="true">,OU=Studierende,OU=Personen,DC=example,DC=org</SourceValue>
  </ValueMap>
</AttributeDefinition>
```



There are of course many more ways this could be done, depending on local data available and LDAP deployment decisions (e.g. group implementation). The wiki for the old Shibboleth IDP v2.x software has [more and more complex examples](#), including one to recursively map affiliations from nested groups within Microsoft "Active Directory" deployments.



Note that those old IDPv2 examples will have to be modified for IDPv4, especially those using `Script/ScriptedAttribute` type definitions. The [IDPv3 documentation](#) provides the details for such adaptations, though potential changes to IDPv4 would also have to be considered. If you successfully converted such an example to IDPv3 / IDPv4 format why not share it with the larger community on the Shibboleth wiki!

**Finally**, it's time to turn the (unscoped) `eduPersonAffiliation` values created by one of the methods above into a scoped one:

```
<AttributeDefinition id="eduPersonScopedAffiliation" xsi:type="Scoped" scope="%{idp.scope}">
  <InputAttributeDefinition ref="eduPersonAffiliation" />
</AttributeDefinition>
```

### eduPersonEntitlement

[eduPersonEntitlement](#) is a container for all kinds of values and data, usually only with clearly defined values within specific communities. One notable exception is its global use for location-independent, off-campus authorisation to [Library Services](#). Here's a simple example that should work for most deployments, giving all subjects you have declared to have an `eduPersonAffiliation` of `member` earlier (see above) or of `library-walk-in` (someone physically present in library premises) the `common-lib-terms` entitlement, thereby stating that these all are entitled to access licensed resources on behalf of your organisation, according to the "common library licensing terms" (again, see [Library Services](#) for details):

### eduPersonEntitlement, variant 1: library services for members

```
<AttributeDefinition id="eduPersonEntitlement" xsi:type="Mapped">
  <InputAttributeDefinition ref="eduPersonAffiliation" />
  <ValueMap>
    <ReturnValue>urn:mace:dir:entitlement:common-lib-terms</ReturnValue>
    <SourceValue>member</SourceValue>
    <SourceValue>library-walk-in</SourceValue>
  </ValueMap>
</AttributeDefinition>
```

Another use-case relevant to the AConet and GÉANT communities is the [GÉANT Trusted Certificate Service](#) that relies on a specific eduPersonEntitlement value to signal that a given subject satisfies the criteria to automatically issue them personal X.509 certificates (based on personal data provided in other SAML attributes, such as name and email address).



TCS has very specific requirements when someone should be marked as eligible to request personal certificates. Do not just blindly copy/paste the following configuration into your IDP before having read and understood those requirements. Cf. [TCS Personal Certs](#) and [ACOnet Zertifikats-Service](#) in this wiki, as well as <https://www.aco.net/tcs.html> for more information.

We can amend the previous example above to do both: Assign the common-lib-terms entitlement to all members and library-walk-in users, and also assert that all your subjects with the an affiliation of faculty have had their identity sufficiently verified that they can all request personal certificates via TCS. **You'll need to adapt that second part as needed, depending on what parts of your community you intend to offer the TCS personal service to.** I.e., for TCS we just add the second ValueMap to the above config, resulting in this example:

### eduPersonEntitlement, alternative 2: library services for members, TCS for e.g. faculty

```
<AttributeDefinition id="eduPersonEntitlement" xsi:type="Mapped">
  <InputAttributeDefinition ref="eduPersonAffiliation" />
  <ValueMap>
    <ReturnValue>urn:mace:dir:entitlement:common-lib-terms</ReturnValue>
    <SourceValue>member</SourceValue>
    <SourceValue>library-walk-in</SourceValue>
  </ValueMap>
  <ValueMap>
    <ReturnValue>urn:mace:terena.org:tcs:personal-user</ReturnValue>
    <SourceValue>faculty</SourceValue>
  </ValueMap>
</AttributeDefinition>
```

You can have *several* SourceValue elements in a ValueMap, e.g. to additionally allow everyone with affiliation student to get a personal certificate as well. See the examples above (eduPersonScopedAffiliation) or the Shibboleth wiki for details.



If you're supporting use of your Shibboleth IDP to access [USI](#) services check out [another variant to create eduPersonEntitlement values](#) that specifically includes code for use with (some of) the USI Service Provider.

## schacHomeOrganization

[schacHomeOrganization](#) is sometimes needed by services, usually as an IDP- and entityID-independent identifier for an organization, e.g. to map subjects from an IDP to a contract in the name of the organisation that runs the IDP (without having to hard-code the IDP's entityID into some configurationn file or database). The following will work for anyone, based on the data connector provided below (that's also generic, thanks to its use of Java properties):

```
<!-- https://wiki.univie.ac.at/display/federation/schacHomeOrganization -->
<AttributeDefinition id="schacHomeOrganization" xsi:type="Simple">
  <InputDataConnector ref="staticAttributes" attributeNames="schacHomeOrganization" />
</AttributeDefinition>
```

## Data Connectors

Add these two DataConnector XML elements to the same attribute-resolver.xml file you've been adding the AttributeDefinition XML elements to above, e.g. at the very end of the file (but before the closing tag of the AttributeResolver element).

## Static

First, here's a simple static DataConnector referenced by one of the AttributeDefinitions above. This only produces a single attribute (value), with an id of "schacHomeOrganization" and the IDP's scope as the sole value. "Static" here means its value(s) will unconditionally be produced for everyone using your IDP, no matter who they are or what attributes they have. That's perfectly fine for this SCHAC attribute since that's about identifying the organisation, not the subject.

### Static DataConnector, use as-is (i.e., don't change anything here)

```
<DataConnector id="staticAttributes" xsi:type="Static">
  <Attribute id="schacHomeOrganization">
    <Value>{%idp.scope}</Value>
  </Attribute>
</DataConnector>
```

## LDAP

And finally here's a verbatim copy of the default example of an LDAP DataConnector (taken from the file attribute-resolver-ldap.xml). All the parameters and values in this DataConnector come from the conf/ldap.properties file or from credentials/secrets.properties, so nothing needs to be set/changed below – with the exception of possibly removing some of the XML attributes: E.g. for LDAP directory server deployments *without* any transport-layer security (no TLS and no SSL) you'd need to remove the trustFile XML-attribute (i.e., the whole line starting with trustFile). The rest should still work for everyone, based on the (correct, or default) settings in ldap.properties:

### LDAP DataConnector

```
<DataConnector id="myLDAP" xsi:type="LDAPDirectory"
  ldapURL="{%idp.attribute.resolver.LDAP.ldapURL}"
  baseDN="{%idp.attribute.resolver.LDAP.baseDN}"
  principal="{%idp.attribute.resolver.LDAP.bindDN}"
  principalCredential="{%idp.attribute.resolver.LDAP.bindDNCredential}"
  useStartTLS="{%idp.attribute.resolver.LDAP.useStartTLS:true}"
  connectTimeout="{%idp.attribute.resolver.LDAP.connectTimeout}"
  trustFile="{%idp.attribute.resolver.LDAP.trustCertificates}"
  responseTimeout="{%idp.attribute.resolver.LDAP.responseTimeout}"
  connectionStrategy="{%idp.attribute.resolver.LDAP.connectionStrategy}"
  noResultIsError="true"
  multipleResultsIsError="true"
  excludeResolutionPhases="c14n/attribute">
  <FilterTemplate>
    <![CDATA[
      {%idp.attribute.resolver.LDAP.searchFilter}
    ]]>
  </FilterTemplate>
  <ConnectionPool
    minPoolSize="{%idp.pool.LDAP.minSize:3}"
    maxPoolSize="{%idp.pool.LDAP.maxSize:10}"
    blockWaitTime="{%idp.pool.LDAP.blockWaitTime:PT3S}"
    validatePeriodically="{%idp.pool.LDAP.validatePeriodically:true}"
    validateTimerPeriod="{%idp.pool.LDAP.validatePeriod:PT5M}"
    validateDN="{%idp.pool.LDAP.validateDN:}"
    validateFilter="{%idp.pool.LDAP.validateFilter:(objectClass=*)}"
    expirationTime="{%idp.pool.LDAP.idleTime:PT10M}" />
  </ConnectionPool>
</DataConnector>
```

If you're done with editing activate the changes by restarting Tomcat – assuming you've changed some Java property files (such as saml-nameid.properties) which are only read on startup of the JVM:

```
systemctl restart tomcat9
```

At any later point, once the IDP has all the properties set, you should activate resolver changes in a running IDP by reload only the IDP's attribute resolver sub-system (*not* by restarting the IDP or Tomcat):

```
/opt/shibboleth-idp/bin/reload-service.sh -id shibboleth.AttributeResolverService
```

Check your /opt/shibboleth-idp/logs/idp-process.log for any ERROR or WARN occurrences.

That should cover producing all the common attributes in use in federation (and interfederation) today! Next move up to configuring [attribute release](#).