

The page features several decorative squares of varying sizes and colors (blue and grey) scattered across the background. The largest blue square is in the top-left corner. Other blue squares are located in the middle-left, bottom-right, and bottom-center areas. Grey squares are positioned in the middle-left and middle-right areas.

# Academus 2.0 Person Directory Configuration Guide

# Contents

---

<b>1.0 Introduction.....</b>	<b>4</b>
1.1 What is the Person Directory? .....	4
1.2 Person Attribute Mapping and Academus Attributes .....	4
<b>2.0 The PersonDirs.xml File .....</b>	<b>5</b>
2.1 Introduction .....	5
2.2 The PersonDirs.xml File .....	5
2.3 The <PersonDirInfo> Structure .....	5
<b>3.0 The &lt;PersonDirInfo&gt; Elements .....</b>	<b>7</b>
3.1 Introduction .....	7
3.2 LDAP Based Data Sources.....	7
3.3 JDBC Based Data Sources.....	10
3.4 Container Defined JDBC Based Data Sources .....	14

# 1.0 Introduction

## 1.1 What is the Person Directory?

The Person Directory is Academus' method of associating portal users with pieces of data about them, such as their first name, last name, or e-mail address. These pieces of data are called person attributes. The Person Directory implementation is designed to be very flexible and to allow person attributes stored in different database and LDAP servers to be mapped to a single user.

## 1.2 Person Attribute Mapping and Academus Attributes

A user's person attributes, which can be collected from multiple data sources, are mapped to Academus attributes that can be used by portal channels. For example, the portal could receive information about the user's first name and last name from an LDAP server, the location of their shared network space from an Active Directory server, and their student ID number from a student information systems database. Academus collects all of these pieces of information and associates them with one user in the portal. The end result is that all of these pieces of information from diverse systems can be collectively associated with particular users and transparently accessed within the Academus framework.

The following Academus attributes are used by the Academus framework:

- username
- password
- lastName
- firstName
- mail
- displayName
- uPortalTemplateUserName
- user.login.id
- user.login.password

Note: the password and user.login.password attributes are mapped based off the input of the user during login time. They are not mapped directly to an attribute in an LDAP or database server. In general, the password attribute is not defined in the PersonDirs.xml file and the user.login.password is mapped to a user's username attribute. These variables are populated by the security contexts and the default mappings should not be modified.

A strong knowledge of the layout and functionality of LDAP and database servers is required to fully take advantage of this flexibility. This guide functions as a reference to the PersonDirs.xml configuration file but does not cover every scenario where this functionality can be leveraged. Please be aware that a complex PersonDirs.xml configuration requires a fairly high level of effort to fully test and implement.

## 2.0 The PersonDirs.xml File

### 2.1 Introduction

The PersonDirs.xml file is located in the following directory:

```
/portal/unicon/Academus/portal-tomcat-a/webapps/portal/WEB-INF/classes/properties/  
PersonDirs.xml
```

This file has been pre-configured by the Academus deployment team for at least one attribute source: the portal database. It may be configured for multiple Active Directory or LDAP sources as well depending on what was requested at deployment time. You are encouraged to add more sources to this file as needed, however, be aware that each additional source will impact the overall performance of the portal, especially when searches are executed.

To minimize performance degradation, be sure that each of your person attribute sources are properly tuned for the best performance. One slow person attribute source can delay the entire process of retrieving person attributes and can cause some portal functions to behave sluggishly.

### 2.2 The PersonDirs.xml File

The basic structure of this file is as follows:

```
<PersonDirs>  
  <PersonDirInfo>  
    ...  
  </PersonDirInfo>  
  <PersonDirInfo>  
    ...  
  </PersonDirInfo>  
  ...  
</PersonDirs>
```

### 2.3 The <PersonDirInfo> Structure

The <PersonDirInfo> element contains a number of configuration parameters depending on the type of datasource:

- 1) LDAP-based data sources:

```
<PersonDirInfo>  
  <url> ... </url>  
  <logonid> ... </logonid>  
  <logonpassword> ... </logonpassword>  
  <uidquery> ... </uidquery>  
  <searchquery> ... </searchquery>  
  <searchquery2> ... </searchquery2>  
  <unionOperator> ... </unionOperator>  
  <intersectionOperator> ... </intersectionOperator>  
  <uidSelect> ... </uidSelect>  
  <wildcard> ... </wildcard>  
  <>trueClause> ... </trueClause>
```

```
<usercontext> ... </usercontext>
<attributes> ... </attributes>
</PersonDirInfo>
```

2) JDBC-based data sources:

```
<PersonDirInfo>
  <driver> ... </driver>
  <url> ... </url>
  <logonid> ... </logonid>
  <logonpassword> ... </logonpassword>
  <uidquery> ... </uidquery>
  <searchquery> ... </searchquery>
  <searchquery2> ... </searchquery2>
  <unionOperator> ... </unionOperator>
  <intersectionOperator> ... </intersectionOperator>
  <uidSelect> ... </uidSelect>
  <wildcard> ... </wildcard>
  <>trueClause> ... </trueClause>
  <attributes> ... </attributes>
</PersonDirInfo>
```

3) Container-defined JDBC-based data sources:

```
<PersonDirInfo>
  <res-ref-name> ... </res-ref-name>
  <uidquery> ... </uidquery>
  <searchquery> ... </searchquery>
  <searchquery2> ... </searchquery2>
  <unionOperator> ... </unionOperator>
  <intersectionOperator> ... </intersectionOperator>
  <uidSelect> ... </uidSelect>
  <wildcard> ... </wildcard>
  <>trueClause> ... </trueClause>
  <attributes> ... </attributes>
</PersonDirInfo>
```

## 3.0 The <PersonDirInfo> Elements

### 3.1 Introduction

The various elements within each <PersonDirInfo> element need to be configured to communicate properly with the desired data source. Some of these elements are used for each of the three types of data sources and some of these elements are only used for one particular type of data source. This section of the guide details the function of each of these elements within the context of the data source type.

### 3.2 LDAP Based Data Sources

An LDAP-based data source is any server that speaks the LDAP protocol, such as Active Directory, OpenLDAP, SunOne, etc.

#### <url>

The <url> element specifies the hostname, port and protocol of this LDAP server.

Standard LDAP example:

```
<url>ldap://ldap.unicon.net:389</url>
```

SSL-enabled LDAP example:

```
<url>ldaps://ldap.unicon.net:636</url><logonid>
```

The <logonid> element specifies the username credential to use when communicating with this LDAP server.

Example:

```
<logonid>queryuser</logonid>
```

Fully qualified DN example:

```
<logonid>uid=queryuser,dc=unicon,dc=net</logonid>
```

#### <logonpassword>

The <logonpassword> element specifies the password credential to use when communicating with this LDAP server.

Example:

```
<logonpassword>secret</logonpassword>
```

### <uidquery>

The <uidquery> element specifies the query to use for retrieving person attributes from the LDAP server. The result of this query will be used to associate attributes mapped by this data source to a portal user.

LDAP example:

```
<uidquery>(uid={0})</uidquery>
```

Active Directory example:

```
<uidquery>(sAMAccountName={0})</uidquery>
```

### <searchquery>

The <searchquery> element is used in Person Directory-based searches within Academus, such as the User Administration channel's user search. The four search criteria fields in this search screen are username, first name, last name, and e-mail address, and they are mapped to LDAP attributes directly. This element needs to be defined only for the data sources that contain these particular attributes, all other data sources can omit this element. This element is generally defined for your primary LDAP server and should not be needed for any additional LDAP servers.

LDAP Example:

```
<searchquery>(~MATCH_ALL_OP~(uid={0}~WILDCARD_0~)(givenName={1}~WILDCARD_1~)(sn={2}~WILDCARD_2~)(mail={3}~WILDCARD_3~))</searchquery>
```

Active Directory Example:

```
<searchquery>(~MATCH_ALL_OP~(sAMAccountName={0}~WILDCARD_0~)(givenName={1}~WILDCARD_1~)(sn={2}~WILDCARD_2~)(userPrincipalName={3}~WILDCARD_3~))</searchquery>
```

### <searchquery2>

The <searchquery2> element is used to construct a search query based on the <searchquery> element, but with an additional username filter. This is used by channels that execute searches on subsets of users or construct lists of users based on username criteria. For example, usernames that begin with the letters A-M could be a potential additional username filter that a channel may require.

Example:

```
<searchquery2>( & ~SEARCH_QUERY ~( | ~USER_FILTER ~ ) )</searchquery2>
```

### <unionOperator>

The <unionOperator> element is used to construct complex queries involving union, or logical OR operations. The value of this element should be the character or string used to specify a union or logical OR when combining LDAP queries. This is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries.

Example:

```
<unionOperator>|</unionOperator>
```

**<intersectionOperator>**

The <intersectionOperator> element is used to construct complex queries involving intersection, or logical AND operations. The value of this element should be the character or string used to specify an intersection or logical AND when combining LDAP queries. This is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries.

Example:

```
<intersectionOperator>&amp;</intersectionOperator>
```

**<uidSelect>**

The <uidSelect> element is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries. The contents of this element are used to construct the additional username filter in these types of queries.

LDAP Example:

```
<uidSelect>(uid={~ARG~})</uidSelect>
```

Active Directory example:

```
<uidSelect>(sAMAccountName={~ARG~})</uidSelect>
```

**<wildcard>**

The <wildcard> element is used in conjunction with the <searchquery> and <searchquery2> elements to specify a wildcard, or 'catch-all' character within the query. The character used to represent a wildcard might vary from LDAP server to LDAP server so it was made configurable. In general, the wildcard character is a \* unless configured otherwise.

Example:

```
<wildcard>*</wildcard>
```

**<>trueClause>**

The <>trueClause> element is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries. The “~USER\_FILTER~” portion of <searchquery2> is replaced with the contents of trueClause in a situation where no additional usernames are supplied. For LDAP, this element should be set to null.

Example:

```
<>trueClause></trueClause>
```

**<usercontext>**

The <usercontext> element specifies the base DN to begin LDAP queries from. For example, if your LDAP begins at “dc=unicon, dc=net” and you only want people in the “ou=students, dc=unicon, dc=net” organizational unit to be accessible by the Person Directory, you can specify the ou as “ou=students, dc=unicon, dc=net” and the Person Directory will ignore users below this level. You can specify multiple contexts by creating an additional <PersonDirInfo> entry for each context.

Example:

```
<usercontext>dc=unicon, dc=net</usercontext>
```

**<attributes>**

The <attributes> element contains all of the person attribute mappings for this data source. Person attribute mappings have the following format:

```
<attribute>
  <name>[LDAP attribute name]</name>
  <alias>[Academus attribute name]</alias>
</attribute>
```

You can specify as many <attribute> elements as needed within the <attributes> element.

Example:

```
<attributes>
  <attribute>
    <name>uid</name>
    <alias>username</alias>
  </attribute>
  <attribute>
    <name>givenname</name>
    <alias>firstName</alias>
  </attribute>
</attributes>
```

### 3.3 JDBC Based Data Sources

JDBC-based data sources are database servers that have not been defined within a JNDI container.

**<driver>**

The <driver> element specifies the JDBC driver to use to communicate with this data source.

PostgreSQL example:

```
<driver>org.postgresql.Driver</driver>
```

Oracle example:

```
<driver>oracle.jdbc.driver.OracleDriver</driver>
```

MS-SQL example:

```
<driver>net.sourceforge.jtds.jdbc.Driver</driver>
```

**<url>**

The <url> element specifies the JDBC driver, hostname, port and database name of this for the target database.

PostgreSQL example:

```
<url>jdbc:postgresql://postgresql.unicon.net:5432/academus</url>
```

Oracle example:

```
<url>jdbc:oracle:thin:@oracle.unicon.net:1521:ACADEMUS</url>
```

MS-SQL example:

```
<url>jdbc:jtds:sqlserver://mssql.unicon.net:1433/academus</url>
```

**<logonid>**

The <logonid> element specifies the username credential to use when communicating with this database server.

Example:

```
<logonid>databaseuser</logonid>
```

**<logonpassword>**

The <logonpassword> element specifies the password credential to use when communicating with this LDAP server.

Example:

```
<logonpassword>secret</logonpassword>
```

**<uidquery>**

The <uidquery> element specifies the query to use for retrieving person attributes from the LDAP server. The result of this query will be used to associate attributes mapped by this data source to a portal user.

PostgreSQL example:

```
<uidquery>SELECT UPD.FIRST_NAME || ' ' || UPD.LAST_NAME AS FIRST_LAST,  
UPD.FIRST_NAME, UPD.LAST_NAME, UPD.EMAIL, UPD.USER_NAME, PDA.prefix,  
PDA.suffix, PDA.address1, PDA.address2, PDA.city, PDA.zip, PDA.phone, PDA.state  
FROM UP_PERSON_DIR UPD LEFT JOIN PERSON_DIR_ATTR PDA on  
PDA.USER_NAME = UPD.USER_NAME WHERE UPD.USER_NAME =?</uidquery>
```

Oracle example:

```
<uidquery>SELECT UPD.FIRST_NAME || ' ' || UPD.LAST_NAME AS FIRST_LAST,  
UPD.FIRST_NAME, UPD.LAST_NAME, UPD.EMAIL, UPD.USER_NAME, PDA.prefix,  
PDA.suffix, PDA.address1, PDA.address2, PDA.city, PDA.zip, PDA.phone, PDA.state  
FROM UP_PERSON_DIR UPD LEFT JOIN PERSON_DIR_ATTR PDA on  
PDA.USER_NAME = UPD.USER_NAME WHERE UPD.USER_NAME =?</uidquery>
```

MS-SQL example:

```
<uidquery>SELECT UPD.FIRST_NAME + ' ' + UPD.LAST_NAME AS FIRST_LAST,  
UPD.FIRST_NAME, UPD.LAST_NAME, UPD.EMAIL, UPD.USER_NAME, PDA.prefix,  
PDA.suffix, PDA.address1, PDA.address2, PDA.city, PDA.zip, PDA.phone, PDA.state  
FROM UP_PERSON_DIR UPD LEFT JOIN PERSON_DIR_ATTR PDA on  
PDA.USER_NAME = UPD.USER_NAME WHERE UPD.USER_NAME =?</uidquery>
```

Simple PostgreSQL example:

```
<uidquery>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME,  
LAST_NAME, EMAIL FROM UP_PERSON_DIR WHERE USER_NAME=?</uidquery>
```

**<searchquery>**

The <searchquery> element is used in Person Directory-based searches within Academus, such as the User Administration channel's user search. The four search criteria fields in this search screen are username, first name, last name, and e-mail address, and they are mapped to database values directly. This element needs to be defined only for the data sources that contain these particular value, all other data sources can omit this element. This element is generally defined for your primary portal database server and should not be needed for any additional database servers.

## PostgreSQL Example:

```
<searchquery>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? = " or lower(USER_NAME) like '%' || lower(?) || '%') AND (? = "
or lower(FIRST_NAME) like '%' || lower(?) || '%') AND (? = " or lower(LAST_NAME) like '%' ||
lower(?) || '%') AND (? = " or lower(EMAIL) like '%' || lower(?) || '%'))</searchquery>
```

## Oracle Example:

```
<searchquery>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? = " or lower(USER_NAME) like '%' || lower(?) || '%') AND (? = "
or lower(FIRST_NAME) like '%' || lower(?) || '%') AND (? = " or lower(LAST_NAME) like '%' ||
lower(?) || '%') AND (? = " or lower(EMAIL) like '%' || lower(?) || '%'))</searchquery>
```

## MS-SQL Example:

```
<searchquery>SELECT FIRST_NAME+' '+LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? = " or lower(USER_NAME) like '%' + lower(?) + '%') AND (? = "
or lower(FIRST_NAME) like '%' + lower(?) + '%') AND (? = " or lower(LAST_NAME) like '%'
+ lower(?) + '%') AND (? = " or lower(EMAIL) like '%' + lower(?) + '%'))</searchquery>
```

**<searchquery2>**

The <searchquery2> element is used to construct a search query based on the <searchquery> element, but with an additional username filter. This is used by channels that execute searches on subsets of users or construct lists of users based on username criteria. For example, usernames that begin with the letters A-M could be a potential additional username filter that a channel may require.

## PostgreSQL Example:

```
<searchquery2>SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? != " AND lower(USER_NAME) like '%' || lower(?) || '%') OR (? !=
" AND lower(FIRST_NAME) like '%' || lower(?) || '%') OR (? != " AND lower(LAST_NAME)
like '%' || lower(?) || '%') OR (? != " AND lower(EMAIL) like '%' || lower(?) ||
'%'))</searchquery2>
```

## MS-SQL Example:

```
<searchquery2> SELECT FIRST_NAME+' '+LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? != " AND lower(USER_NAME) like '%' + lower(?) + '%') OR (?
!= " AND lower(FIRST_NAME) like '%' + lower(?) + '%') OR (? != " AND lower(LAST_NAME)
like '%' + lower(?) + '%') OR (? != " AND lower(EMAIL) like '%' + lower(?) +
'%'))</searchquery2>
```

Oracle Example:

```
<searchquery2> SELECT FIRST_NAME||' '||LAST_NAME AS FIRST_LAST, FIRST_NAME,
LAST_NAME, EMAIL, USER_NAME FROM UP_PERSON_DIR WHERE
(~USER_FILTER~) AND ((? != " AND lower(USER_NAME) like '% ' || lower(?) || '%') OR (? !=
" AND lower(FIRST_NAME) like '% ' || lower(?) || '%') OR (? != " AND lower(LAST_NAME)
like '% ' || lower(?) || '%') OR (? != " AND lower(EMAIL) like '% ' || lower(?) ||
%'))</searchquery2>
```

### <unionOperator>

The <unionOperator> element is used to construct complex queries involving union, or logical OR operations. The value of this element should be the character or string used to specify a union or logical OR when combining database queries. This is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries.

Example:

```
<unionOperator>OR</unionOperator>
```

### <intersectionOperator>

The <intersectionOperator> element is used to construct complex queries involving intersection, or logical AND operations. The value of this element should be the character or string used to specify an intersection or logical AND when combining database queries. This is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries.

Example:

```
<intersectionOperator>AND</intersectionOperator>
```

### <uidSelect>

The <uidSelect> element is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries. The contents of this element are used to construct the additional username filter in these types of queries.

Example:

```
<uidSelect>USER_NAME=?</uidSelect>
```

### <wildcard>

The <wildcard> element is used in conjunction with the <searchquery> and <searchquery2> elements to specify a wildcard, or 'catch-all' character within the query. This setting is not needed for databases.

Example:

```
<wildcard></wildcard>
```

### <>trueClause>

The <trueClause> element is used in conjunction with the <searchquery> and <searchquery2> elements to construct complex queries. The "~USER\_FILTER~" portion of <searchquery2> is replaced with the contents of trueClause in a situation where no additional usernames are supplied. For database servers, this element should be set to 1=1.

Example:

```
<trueClause>1=1</trueClause>
```

**<attributes>**

The <attributes> element contains all of the person attribute mappings for this data source. Person attribute mappings have the following format:

```
<attribute>
  <name>[LDAP attribute name]</name>
  <alias>[Academus attribute name]</alias>
</attribute>
```

You can specify as many <attribute> elements as needed within the <attributes> element.

Example:

```
<attributes>
  <attribute>
    <name>uid</name>
    <alias>username</alias>
  </attribute>
  <attribute>
    <name>givenname</name>
    <alias>firstName</alias>
  </attribute>
</attributes>
```

### 3.4 Container Defined JDBC Based Data Sources

Container-defined JDBC-based data sources are database servers that have been defined within a JNDI container. The key difference between container-defined and non-container-defined JDBC-based data sources is the specification of the database server. In a container-defined resource, also known as a JNDI resource, the database driver, host, port, and login credentials are defined in Tomcat's server.xml file and configured in the unicon-resource-pool.properties file. The configuration of the <PersonDirInfo> elements for this type of data source is the same as for JDBC-based data sources except that the <driver>, <url>, <logonid>, and <logonpassword> elements are replaced with the <res-ref-name> attribute.

**<res-ref-name>**

The <res-ref-name> element specifies JNDI resource to use for this data source.

Example:

```
<res-ref-name>PortalDb</res-ref-name>
```