Oracle® Fusion Middleware

Identity Governance Framework ArisID API Developer's Guide

11*g* (11.1.1)

E16588-02

February 2011

ORACLE®

Oracle Fusion Middleware Identity Governance Framework ArisID API Developer's Guide, 11*g* (11.1.1)

E16588-02

Primary Author: Ellen Desmond

Contributing Author: Venkat Medam, Vasuki Ashok, Trish Fuzesy

Contributors: Amit Sharma, Mark Wilcox

Contents

4 Migrating From the User and Role API to the ArisID API

A Sample Application

List of Figures

Preface

This guide describes how to use the Identity Governance Framework ArisID extension to Oracle JDeveloper to build Oracle Fusion Middleware applications that implement the Identity Governance Framework ArisID API.

Audience

This document is intended for developers who are writing applications that use the Oracle Fusion Middleware Identity Governance Framework ArisID API.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Fusion Middleware Identity Governance Framework UserRole API Reference*
- *Oracle Fusion Middleware Identity Governance Framework IDXUserRole API Reference*
- Online Help for the Identity Governance Framework ArisID JDeveloper Extension
- Javadocs for Project Aristotle - ArisID Attribute Services, at: `http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/`.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Using the Identity Governance Framework ArisID API

This chapter describes the architecture and key functionality of the Identity Governance Framework ArisID API (ArisID API). The ArisID API provides enterprise developers and system architects a library for building identity-enabled applications using multiple identity protocols. The ArisID API enables developers to specify requirements for identity attributes, roles, and search filters by using Client Attribute Requirements Markup Language (CARML).

This chapter contains the following topics:

About the Identity Governance Framework

The Identity Governance Framework (IGF) is an open initiative designed to meet the following goals:

- To simplify the development of identity information regardless of where that information is stored.
- To simplify the management (also known as **governance**) of how applications use identity data, in particular, sensitive data.

As part of this initiative, Oracle has contributed key initial specifications and is making them available to the community. These specifications provide a common framework for defining usage policies, attribute requirements, and developer APIs pertaining to the use of identity related information. These enable businesses to ensure full documentation, control, and auditing regarding the use, storage, and propagation of identity-related data across systems and applications.

Benefits to Organizations

Organizations need to maintain control and integrity of sensitive personal information about their customers, employees, and partners. Data related to social security numbers, credit card numbers, medical history and more are increasingly under scrutiny by regulations seeking to prevent abuse or theft of such information. Privacy conscious organizations frequently have reacted to these requirements by enforcing overly strict controls and processes that hinder business operations and impact productivity, flexibility, and efficiency. At the opposite end of the spectrum, some organizations do not take the care needed to safeguard this information, potentially

putting identity-related data at risk without sufficient oversight and control. The Identity Governance Framework enables a standards-based mechanism for enterprises to establish "contracts" between their applications so that identity related information can be shared securely with confidence that this data will not be abused, compromised, or misplaced. Using this framework, organizations have complete visibility into how identity information is stored, used, and propagated throughout their business. This enables organizations to automate controls to streamline business processes without fear of compromising the confidentiality of sensitive identity related information.

Benefits to Developers

The Identity Governance Framework is an agreed-upon process for specifying how identity-related data is treated when writing applications. This provides developers a standards-based way to easily write applications that use this data so that governing policies can be used to control it. This will result in faster development of privacy aware applications.

IGF enables the decoupling of identity-aware applications from a specific deployment infrastructure. Specifically, using IGF enables developers to defer deciding on how identity related information will be stored and accessed by their application. Developers will not need to worry about whether they should use a SQL database, an LDAP directory, or other system. In the past, developers were forced to write highly specific code, driving technology and vendor lock-in. By using a Client Attribute Requirement Markup Language (CARML) file and declarations, applications will support flexible deployment in a wide range of environments without the need for ongoing specialized developer enhancements. The ArisID API handles the hard work of data retrieval, transformation, and policy-enforcement when it comes to identity-based information.

About the Identity Governance Framework ArisID API

The Identity Governance Framework ArisID API represents a common core service through which all identity information exchange should be passed. While not an official name, the ArisID API is often referred to as Identity Beans by developers.

The 11*g* (11.1.1) release of the ArisID API is a subset of the configuration proposed at:

`http://www.openliberty.org/wiki/index.php/ArisID_Configuration`.

If you have installed Oracle WebLogic Server and Oracle Identity Management, all the necessary jar files for developing applications with this API are already installed on your computer.

See Also:

- *Oracle Fusion Middleware Installation Guide for Oracle Identity Management* for information about installing Oracle Identity Management.
- *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper* for information about installing JDeveloper and its extensions.

The Identity Governance Framework open source API jar files are as follows:

- **openliberty.arisId_1.1.jar** — Provides the core ArisID API with library functions and providers that can be used to retrieve identity subjects that contain collections

of attributes. For more information, see http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/.

- **org.openliberty.arisIdBeans_1.1.jar** — Provides the ArisID beans, which provide Java object abstractions on top of the ArisID API. These convert the transactional approach of the ArisID API to an object or bean approach. For more information, see http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/.

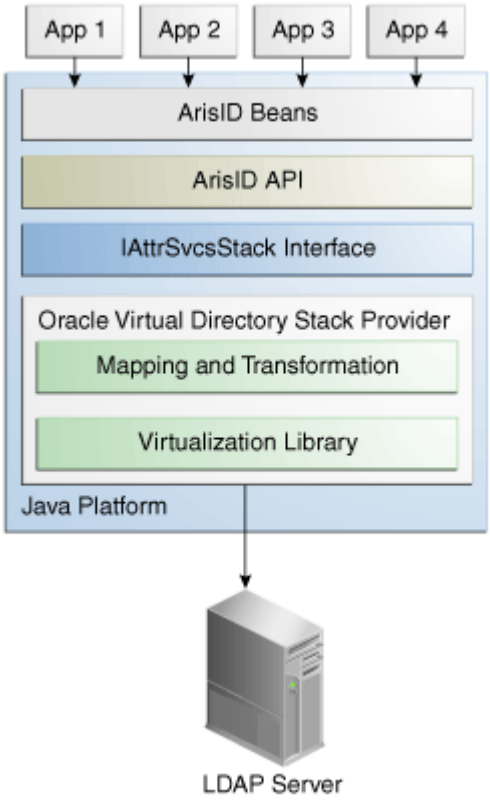
The ArisID API jar files are as follows:

- **idxuserrole.jar** — Provides the Standard User and Role identity read-only operations. This jar is generated from the standard idxuserrole.xml CARML file. For more information, see *Oracle Fusion Middleware Identity Governance Framework IDXUserRole API Reference*.
- **userrole.jar** — Provides the User and Role identity read/write operations for updating identity information. For more information, see *Oracle Fusion Middleware Identity Governance Framework UserRole API Reference*.
- **arisId-stack-ovd.jar** — This jar file is an implementation of the IAttrSvcStack interface with the Oracle Virtualization library to connect to different backends and provide an abstract view of the identity store entities.

The ArisID beans provide the Java APIs required for initialization and accessing CARML interactions. The bean generator generates a set of java files for each entity in the CARML file using Apache Velociy. The CARML file is a declarative document that describes the attribute usage requirements of your application. The ArisID beans are in the jar files idxuserrole.jar and userrole.jar. If the standard ArisID beans do not meet your needs, you can generate new ArisID beans by creating a CARML file and using the bean generator in the Identity Governance Framework ArisID extension to JDeveloper.

The following figure provides a high-level view of the ArisID API architecture.

Figure 1–1 IGF ArisID API Architecture



Developing Applications With the ArisID API

The Identity Governance Framework ArisID extension supports the basic development process Create > Modify > Test > Deploy. Creation requires starting a new JDeveloper project and creating CARML files. Use the CARML editor to modify the CARML XML files to suit your environment. Testing the application can be done in Oracle WebLogic Server embedded LDAP directory server.

Configuring CARML Files

Determine whether the existing ArisID beans meet your application's needs by examining the CARML files idxuserrole.xml (read-only operations) and userrole.xml (read-only and read/write operations). These files are located in DOMAIN_HOME/config/fmwconfig/carml. If you need additional attributes or other customizations, create a new CARML file and generate beans as described in Chapter 3, "Developing Applications".

Configuring the Identity Repository

The identity repository to be used by the ArisID beans must be available. You can use the Oracle WebLogic Server embedded LDAP-based directory server or any LDAP directory supported by 11*g* Oracle Virtual Directory. The ArisID API is integrated with Oracle Platform Security Services. It automatically connects to the LDAP-based identity store configured in Oracle Platform Security Services. The identity stores supported by Oracle Platform Security Services. For more information about system requirements and certification, see "System Requirements and Certification".

For more information about Oracle Platform Security Services, see *Oracle Fusion Middleware Application Security Guide*.

If you must use a different identity store from the Oracle Platform Security Services identity store, then set the following system property:

```
igf.ovd.config.dir=DOMAIN_HOME/config/fmwconfig/arisidprovider/conf
```

Next, edit the adapters.os_xml file to include the `host`, `port` and credentials of the directory to be connected to. The `igf.ovd.config.dir` property can be set to any other directory containing adapaters.os_xml and other configuration files with the right settings.

For OpenLDAP, `Role.MEMBER` is a mandatory attribute for the following APIs:

- `createRole(List<PropertyValue> attrVals, Map<String,Object> appCtxMap)`
- `createRole(List<PropertyValue> attrVals)`

If the `Role.MEMBER` is not included in the input `attrVals` list, role creation will fail.

Configuring the Mapping File

When a CARML file is created a corresponding mapping file is created in the same location. The default mapping file has attribute details specific to Oracle WebLogic Server embedded directory server, which is the Oracle Platform Security Services default identity store. If you are using a default CARML file and the Oracle Platform Security Services identity store, you do not need to configure mapping. The configuration parameters in Oracle Platform Security Services override the parameters in the mapping file.

If you are creating your own CARML file with additional attributes, or if you are using a non-Oracle Platform Security Services identity store, you must edit the mapping file. For more information, see Chapter 3, "Developing Applications".

System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

http://www.oracle.com/technology/software/products/ias/files/fusion_requirements.html

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html

2

Design Recommendations

The default CARML and mapping files make certain assumptions about the deployment scenario. You may need to modify these details depending on your deployment requirements. The configuration parameters that can be modified are discussed in this chapter.

This chapter contains the following topics:

Choose a LoginID

In the default configuration, `email` is used as a unique identifier for identifying user entries. When you are searching for a user, the default attribute expected for search is `email`. For example:

```
SearchUser( String uniqueid,  Map<String, Object>)
```

For performance reasons, the attribute used as a unique identifier must be a searchable attribute in the backend. The mapping between the application's choice of `uniquekey` and the backend attribute is handled at configuration time. This is a configuration in Oracle Virtual Directory mapping. The `HashMap` is used to provide the optional context information to be used while performing the operation. In the current release it supports the following options:

- The Principal user that performs the search - (`ArisIdConstants.APP_CTX_AUTHUSER, (Principal)user`)
- The language constraint if any - (`ArisIdConstants.APP_CTX_LOCALE, "fr"`)
- Pagination support if any - (`ArisIdConstants.APP_CTX_PAGESIZE, 10`)

Choose a UniqueKey

An application occasionally stores the entries accessed from the identity repository's backend in their own application-specific repository. In such cases, you must carefully consider which attribute should be persisted. For instance, if the backend is an LDAP-based repository, you should use the `GUID` attribute as the persisting attribute

because this is the only unique key on the LDAP-based backend. All other LDAP attributes are modifiable.

If the backend is a relational database, choose an attribute on which uniqueness constraint is enforced as the unique key. You can specify this in the ArisID mapping property file. The method to search for a user based on the unique key is:

```
searchUserOnUniqueKey(String UniqueKey, Map<String,Object>)
```

The `HashMap` is used to provide the optional context information to be used while performing the operation. In the current release it supports the following options:

- The Principal user that performs the search - (`ArisIdConstants.APP_CTX_AUTHUSER, (Principal)user`)
- The language constraint if any - (`ArisIdConstants.APP_CTX_LOCALE, "fr"`)
- Pagination support if any - (`ArisIdConstants.APP_CTX_PAGESIZE, "10"`)

Specify Multiple Language Support

Multiple Language Support (MLS) is provided for applications that need locale-specific results. The attributes and the appropriate MLS code are stored in the ArisID properties file in the `multiLanguageAttributes` element.

```
<multiLanguageAttributes>…</multiLanguageAttribute>
```

Because `displayname` is the most commonly used multiple language attribute, it is configured by default as a multi-language attribute. Other attributes can be added as needed in the ArisID mapping file.

Restrictions

Any API to which locale is specified as an argument will return the locale-specific values for all the attributes listed in the ArisID properties file as `<multiLanguageAttributes>` that have locale-specific values. For all other attributes it returns the default values stored.

In the backend system, the data is returned in a form conforming to ISO-3166. For example, if there is a French locale (in addition to English), it is stored as `cn,:fr` for the `cn` attribute. The locale for the client applications should be specified in the properties `HashMap` as `ArisIdConstants.APP_CTX_LOCALE, "fr"` and the ArisID properties file should contain `cn` as `multiLanguageAttribute` and map this attribute.

Handle Large Results

When applications access identity data, the result set for a search is frequently too large to be handled by the application. In such cases you have the option of dividing the result into manageable sized pages. You do this by defining the number of objects to be returned in the page.

The following example shows a typical usage pattern:

```
RoleManager rm = new RoleManager(env);
  List<PropertyFilterValue> attrFilters = new  ArrayList<PropertyFilterValue>();
  attrFilters.add(new PropertyFilterValue(Role.NAME, "admin", AttributeFilter.OP_
CONTAINS));

  HashMap<String,Object> map = new HashMap<String,Object>();
  map.put("ArisIdConstants.APP_CTX_PAGESIZE","2");
```

```
  SearchResults<Role> sr = rm.searchRolesbyPage(attrFilters, map);

  while(sr.hasMore())
  {
     List<Role> roles = sr.getNextSet();

     for (int i=0; i<roles.size(); i++)
       //do the operations with roles.get(i)
}
```

Secure the Application

Two security scenarios are available for executing create, read, update, and delete (CRUD) operations on the target system. They are:

- Domain level credentials
- Application level credentials

Proxy authentication is not supported in this release.

Domain Level Credentials

In this scenario, all applications in a domain use common credentials to connect to the target system and perform operations with those credentials. The application does not maintain a footprint in the target system.

The LDAP Adapter's configuration file, `adapters.os_xml`, contains credentials to connect to the backend directory, along with the host and port details. If you do not provide any other credentials during initialization, the application connects to the target system using the credentials in the LDAP Adapter's configuration file.

If proxy user (logged in user id) is not specified in the API's application context, ArisID operation will be executed with the credentials that are in LDAP Adapter's configuration file.

If your application connects using common credentials, you must build security into the application itself so that it displays or modifies data only for an authorized user.

Code Sample

The LDAP Adapter's configuration file `adapters.os_xml` is configured with domain level userid and encrypted password to connect to backend directory. The following is a snippet of `adapters.os_xml`.

```
<binddn>cn=admin</binddn>
<bindpass>{OMASK}C2QXW1Nmf+s=</bindpass>
```

While initializing the ArisID API do not provide any credentials.

```
Map env = new HashMap();
// Do not set UserManager.SECURITY_PRINCIPAL & SECURITY_CREDENTIALS
UserManager uMgr  = new UserManager(env);
…
…
// Search Operation (with no proxy user in app context)
List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("User.FIRSTNAME", "app1",
AttributeFilter.OP_CONTAINS));
attrFilters.add(new PropertyFilterValue("User.LASTNAME", "user1",
AttributeFilter.OP_BGNSWITH));
Map<String, Object> appCtx = null;
```

```
users = um.searchUsers(attrFilters, appCtx);
```

Application Level Credentials

In this scenario, each application uses application level credentials to connect to the target system and performs CRUD operations with those credentials.

In this case you provide the application's user id and password while initializing the ArisID API. When you do that, the application connects to the target system using those credentials.

If no proxy user is specified in the API's application context then ArisID operation will be executed with the application's credentials.

This scenario has the following features:

- Each application has different privileges to view and update the data in the target system
- You can audit the modifications performed by each application in the target system

Code Sample

The LDAP Adapter's configuration file `adapters.os_xml` is configured with domain level userid and encrypted password to connect to backend directory. The following is a snippet of `adapters.os_xml`.

```
<binddn>cn=admin</binddn>
<bindpass>{OMASK}C2QXW1Nmf+s=</bindpass>
```

While initializing the ArisID API, provide the application user credentials.

```
Map env = new HashMap();
env.put(UserManager.SECURITY_PRINCIPAL, "cn=app1_user,cn=users,dc=oracle,dc=com");
env.put(UserManager.SECURITY_CREDENTIALS, "mypassword");
UserManager uMgr  = new UserManager(env);
…
// Search Operation (with no proxy user in app context)
List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("User.FIRSTNAME", "app1",
AttributeFilter.OP_CONTAINS));
attrFilters.add(new PropertyFilterValue("User.LASTNAME", "user1",
AttributeFilter.OP_BGNSWITH));
Map<String, Object> appCtx = null;
users = um.searchUsers(attrFilters, appCtx);
```

3

Developing Applications

This chapter describes how to use Identity Governance Framework ArisID Extension to Oracle JDeveloper to develop applications.

> **See Also:** *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*

This chapter contains the following topics:

Using the Identity Governance Framework ArisID API

When developing an application with the ArisID API you will typically perform the following tasks:

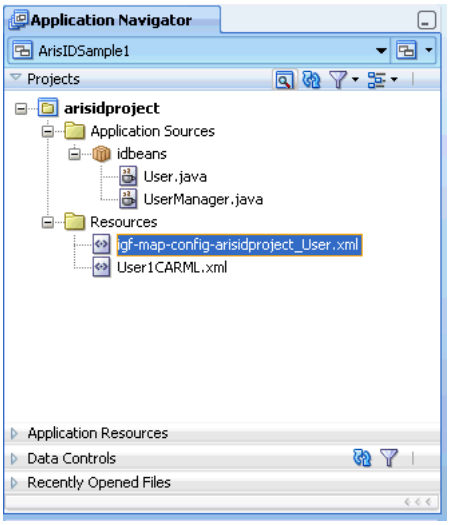
- Create an ArisID project in Oracle JDeveloper.
- Create and configure a CARML file.
- Generate the ArisID beans.
- (Optional) Edit the mapping file.

The Identity Governance Framework ArisID extension for JDeveloper is organized into several different packages. The packages are separated primarily by functionality. At the top level, the packages are for the CARML Overview Editor, the Relationship Editor, the Mapping Editor, Bean Generation, and Project Creation. Project creation contains all the classes required for creating a project structure, managing project properties, and creating CARML files. Common elements shared between many of these packages are kept in .common. This is primarily abstract classes for common Swing components and Parsing/Modeling XML structures.

Creating the Project

The first step in using Identity Governance Framework ArisID is to create a project in Oracle JDeveloper. After the Identity Governance Framework ArisID extension is

installed, **ArisID/IGF Project** is added to the project gallery in JDevelolper. The corresponding project wizard adds the ArisID required libraries, creates a directory structure, and adds the option to test the ArisID configuration. The follow figure shows an example ArisID project and the directory structure as it appears in JDeveloper:

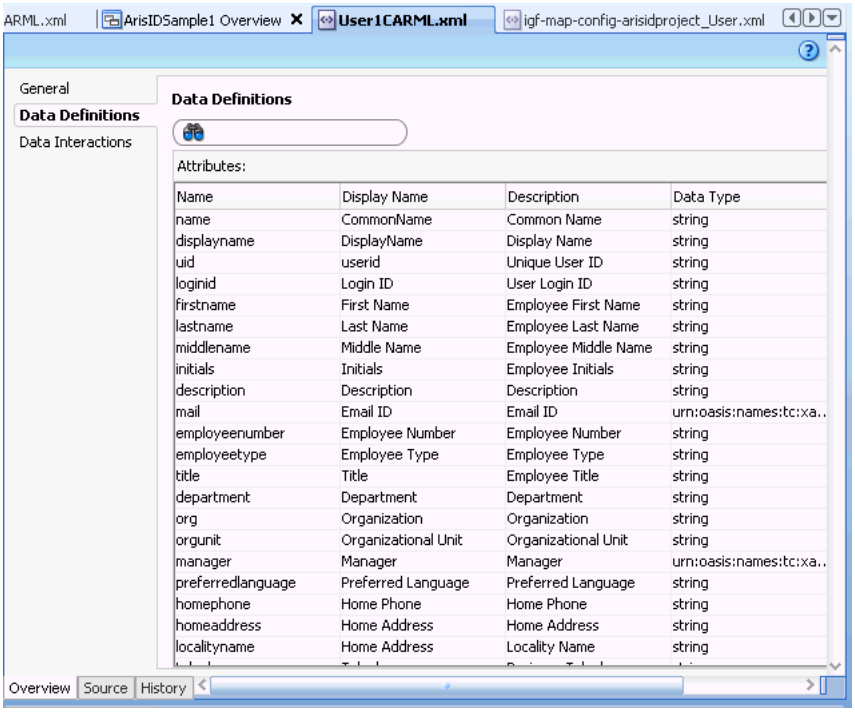


To create a project:

1. Choose **File** then **New** to open the New Gallery.
2. In the Categories tree, expand General and select **Projects**.
3. Choose **IGF/ArisID Project** and click **OK**.
4. Specify a **Project Name** and **Directory** for your project on the Project Name page.
5. Optionally, move additional technologies from the **Available** list to the **Selected** list.
6. Specify a **Default Package**, **Java Source Path**, and **Output Directory** for your project in the Project Java Settings page.
7. Specify **J2SE** or **J2EE** in the Configure IGF/CARML Setting page.
8. Click **Finish**.

Creating and Editing the CARML File

You declare the application requirements in terms of attributes and interactions. These are specified using a Client Attribute Requirements Markup Language (CARML) file. The CARML editor is an XML editor that lets you edit the various fields of a CARML file. The following figure shows an example CARML XML file displayed in the CARML Editor with the Data Definitions section:



To create a new CARML file:

1. Choose **File** then choose **New** to open the New Gallery.
2. In the **Categories** tree, expand Business Tier and choose **Security**.
3. In the Items list, double click **Client Attributes Requirements (CARML)** to open the dialog.
4. In the Create CARML File dialog, specify the name of the file that you would like to create and click **OK**.

   You can use any of the templates provided. The associated description provides the details about each of the template files.

   The General page of the CARML file editor appears.
5. Specify the values for the two fields on the General page: **CARML Unique Indicator** and **CARML description**.
6. Specify your application attribute requirements in the Data Definitions page. Add entities and specify data interactions for entities.
7. Specify the application based interaction requirements in the Data interactions page. During the data interactions step, specify filters for interaction types.

To edit an existing CARML file, double-click the appropriate file in the Projects panel to open it in the overview editor for CARML files.

Generating ArisID Beans

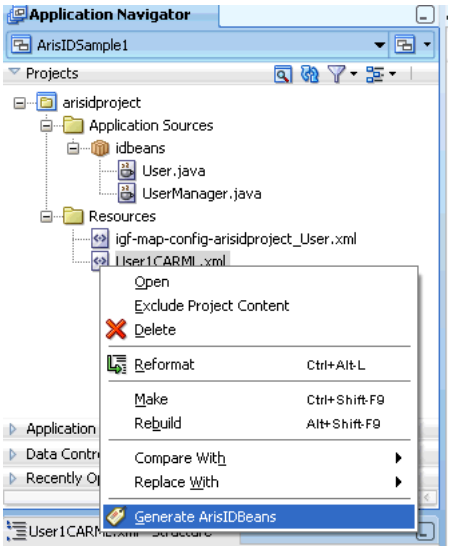
When you have finished editing your CARML file, you can generate the ArisID Beans to use in your application. If you have created a relation file according to the schema definitions bundled with the ArisID API, you can use it when generating the beans.

After you have specified your attribute and interaction declarations in the CARML file, you can generate the corresponding bean classes to use in your application. If you

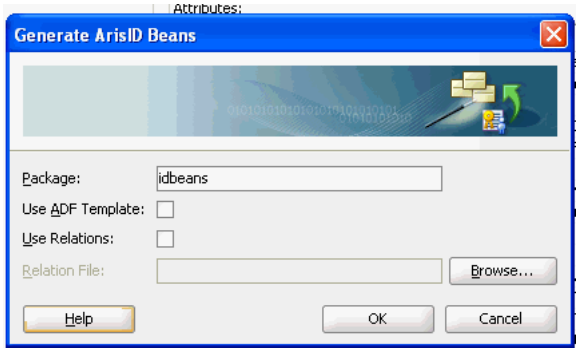
have multiple entities defined in the file, and would like to specify the relationship between the entities, you can do so using the relationship file.

To generate the beans:

1. In the Projects pane, highlight the CARML file you want to use to generate the beans.
2. Right click to display the Application Navigator context menu for a CARML file.
3. Choose **Generate ArisIDBeans**.



4. In the Generate ArisIDBeans dialog, specify the **Package**.



5. If you want to use a relation file, select **Use Relations** and specify or browse to the **Relation File**.
6. Click **OK**.

   You can now incorporate these beans into your application.

How to Use the ArisID Beans in an Application

The Identity Governance Framework ArisID API extension to JDeveloper initially creates the ArisId beans. These beans are from a developer perspective, like any other Java bean. They can be called from any Java application using standard bean semantics. This enables more interesting use cases depending upon the frameworks a developer wants to use.

For example, if building an ADF (Oracle's Java-based Web application framework), the beans can be converted into a Data Control and dragged into an ADF page. This enables developers to quickly wire applications together that utilize the bean - either for searching, updating or displaying on a page - without having to write a single line of code.

Another use case is a developer who wants to expose identity data as a Web Service from Oracle SOA Suite. In this case, the beans could be accessed using a Java call-out instead of using a DSML query against a directory server, such as Oracle Virtual Directory or Oracle Directory Server Enterprise Edition. This can be easier to construct and more efficient because there is less XML parsing being utilized in the SOA process.

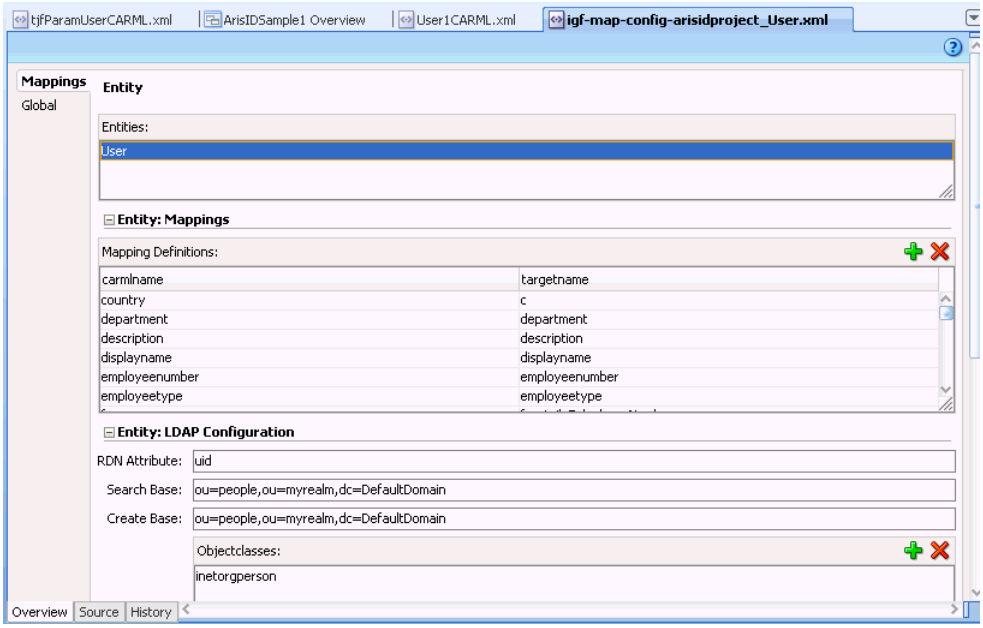
By focusing their efforts on describing objects and attributes and worrying less about specific protocols, developers can be more productive in accessing identity information. Because ArisID has privacy and security built-in, this can be done in a secure, standards compliant way.

Editing the Mapping File

The Oracle implementation maps CARML declarations to an LDAP-based server, such as Oracle Virtual Directory. The mapping editor enables you to map CARML attributes to LDAP attributes, objectclasses and search parameters. You can customize your mapping file for the LDAP-based server at your site. LDAP terms are defined by the Internet Engineering Task Force in RFC's 2251 through 2256. For more information, see http://www.ietf.org.

To edit the mapping file:

1. In the Projects pane, highlight the mapping file.
2. Double-click to open the file in the mapping file overview editor.



3. On the Mappings page, highlight an entity that you want to modify.
4. Highlight a mapping definition you want to modify for that entity.
5. Edit the **RDN Attribute**, **Search Base**, and **Create Base** if necessary.

6. To add **Objectclasses**, **Filter Objectclasses**, **Password Attributes**, and **Multilanguage Attributes**, click the corresponding **Add** icon.
7. Repeat Steps 3 - 6 for other entities that you want to modify.
8. If necessary, change the **Paging** setting on the Global page.

4

Migrating From the User and Role API to the ArisID API

This chapter describes how to migrate applications from the User and Role API to the ArisID API.

This chapter contains the following topics:

Introduction

If you have an application that uses the User and Role API described in *Oracle Fusion Middleware Application Security Guide* and *Oracle Fusion Middleware User and Role Java API Reference for Oracle Platform Security Services*, you can modify it to use ArisID beans instead.

First you must determine whether your application is simple or complex.

An application is simple if it has the following characteristics:

- User attributes belong to the LDAP `inetorgperson` object class.
- Role attributes belong to the LDAP `groupofuniquenames` object class.
- Search filters have only `and` condition on the attribute values.
- The standard userrole jar file meets your requirements.

See Also:

- *Oracle Fusion Middleware Identity Governance Framework UserRole API Reference*
- *Oracle Fusion Middleware Identity Governance Framework IDXUserRole API Reference*

All other applications are identified as complex applications.

Migrate a Simple Application

If you have used the standard User and Role API without adding custom attributes, you can migrate to the ArisID API by following this sequence of steps.

To migrate a simple application, proceed as follows. This sequence applies only to Java EE applications.

Initialize the Application

Initialize your application to use the ArisID APIs.

```
import oracle.igf.userrole.UserManager;
import oracle.igf.userrole.RoleManager;
import org.openliberty.arisidbeans.ArisIdConstants;
import org.openliberty.arisidbeans.PropertyFilterValue;
HashMap env = new HashMap();
```

SECURITY_PRINCIPAL & SECURITY_CREDENTIALS are optional. If they are not used, the application connects to the backend with the credentials configured at domain level.

```
env.put(ArisIdConstants.SECURITY_PRINCIPAL, "cn=orcladmin");
env.put(ArisIdConstants.SECURITY_CREDENTIALS, "mypassword");
```

Create UserManager and Role manager objects.

```
UserManager userMgr = new UserManager(env);
RoleManager roleMgr = new RoleManager(env);
```

If the API will connect to the default identity store configured in Oracle Platform Security Services, the host and port details are automatically obtained from the identity store. If the API connects to a non-default identity store, the following must be configured:

- Set the system property `igf.ovd.config.dir` to point to the directory where the Identity Virtualization library configuration files reside.
- Configure each adapter in the Identity Virtualization library with the host and port details (such as host, port, root dn, plugins, and so on).

For more information about Identity Virtualization library, see *Oracle Fusion Middleware Application Security Guide*.

Perform Search Operations

You have the following search options:

- SearchByGuid
- SearchByName
- SearchUsers
- SearchByPage

SearchByGuid

You can search based on the `GUID` using one of the `searchUserByGuid()` methods. For example:

```
User myObject = userMgr.searchUserByGuid(String guidValue);
```

You can also search specifying more context to the search.

```
User myObject = userMgr.searchUserByGuid(String guidValue, Map appCtx);
```

Where `appCtx` can contain the following details:

- APP_CTX_AUTHUSER - Principal (the Application principal to be used for doing the search)
- APP_CTX_LOCALE - User locale (applicable to all the attributes which have locale specific values)
- APP_CTX_PAGE - Application's page size

SearchByName

You can search based on the loginid using one of the searchUser() methods. For example:

```
User myObject = userMgr.searchUser(String loginid);
```

You can also search specifying more context to the search:

```
User myObject = userMgr.searchUser(String loginid, Map appCtx);
```

Where appCtx can contain the following details:

- APP_CTX_AUTHUSER - Principal (the Application principal to be used for doing the search)
- APP_CTX_LOCALE - User locale (applicable to all the attributes which have locale specific values)
- APP_CTX_PAGE - Application's page size.

SearchUsers

You can search based on the following filter:

```
List myUsers =
userMgr.searchUsers(java.util.List<org.openliberty.arisidbeans.PropertyFilterValue
> attrFiltersList));
```

You can construct the attrFiltersList as follows:

```
attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("firstname", "abc"));
attrFilters.add(new PropertyFilterValue("lastname", "xyz"));
```

SearchByPage

The ArisID API provides pagination support. You can search for users by using searchUsersByPage() as follows:

```
SearchResults<User> sResult = userMgr.searchUsersByPage(java.util.List
 <org.openliberty.arisidbeans.PropertyFilterValue> attrFiltessList));
while (sResult.hasMore()) {
    List<User> users = sResult.getNextSet();
    for (int i = 0; i < users.size(); i++)
// Process each user entry fetched
        Util.printObject(users.get(i));
}
```

Migrate Complex Application

For complex applications, you need to create a custom CARML file and generate ArisID beans. You can migrate search code in the same way as for simple applications. There are some additional preliminary steps, however. Proceed as follows.

Identify the New Attributes

If your application requires custom attributes (that is, attributes which are not supported by `inetorgperson`), you must create an application-specific CARML file. You need to edit the CARML file and add the attribute definitions in the data definitions part of the CARML file.

Identify the Interactions

The default ArisID bean interactions are designed to access all the attributes of the user and role entries. If your application requires custom interactions for performance reasons, you can create the interactions by editing the CARML file.

Generate ArisID Beans by Using the JDeveloper Extension

Use the Identity Governance Framework ArisID JDeveloper extension to create and edit the CARML file and generate the beans. For more information, see Chapter 3, "Developing Applications."

Set Up the Environment

Make the CARML file available to your application. Include the classes generated by the BeanGenerator as part of your application, or make it available in the CLASSPATH environment variable.

Perform Search Operations

Search operations are the same as for simple application migration. See "Perform Search Operations" on page 4-2.

Comparison Between User and Role API and Aris ID API

These APIs are compared to the ArisID API in the following tables:

- User-Related APIs
- Role-Related APIs

User-Related APIs

Table 4–1 provides a comparison between the User-related API method and the corresponding Identity Beans method available in the Identity Governance Framework Aris ID API.

Identity Beans methods marked with a double asterisk (**) have an optional parameter: `Map<String.Object> appCtxMap`. For example, `UserManager.createUser(List<PropertyValue> attrVals, Map<String,Object> appCtxMap).appCtxMap` may contain the following elements:

- `UserManager.APP_CTX_AUTHUSER`: java.security.Principal, the user context to execute under.
- `UserManager.APP_CTX_PAGE`: String value of Page size. This is applicable only for search methods returning `SearchResults` object.
- `UserManager.APP_CTX_LOCALE`: String value of language code.

Table 4–1 Comparison Between User-Related API and ArisID API

Functionality	User/Role API Method	Identity Beans Method
User Creation	User UserManager.createUser (String name, char[] password) User UserManager.createUser (String name, char[] password, PropertySet pset)	** void UserManager.createUser(List<PropertyValue> attrVals)
Delete User	void UserManager.dropUser(UserProfile user) void UserManager.dropUser(User user);	** void UserManager.dropUser(IPrincipalIdentifier ** void UserManager.dropUser(String SubjectId) ** void UserManager.dropUser(User subj)
Authenticate User	User UserManager.authenticateUser(String user_id, char[] passwd) User UserManager.authenticateUser(User user, char[] passwd) User UserManager.authenticateUser(String user_id, String authProperty, char[] passwd)	** User UserManager.authenticateUser(List<PropertyFilterValue> attrFiltersList) ** User UserManager.authenticateUser(String uid, String password)
Check if create User is supported	boolean UserManager.isCreateUserSupported()	boolean UserManager.isCreateUserSupported()
Check if modify User is supported	boolean UserManager.isModifyUserSupported()	boolean UserManager.isModifyUserSupported()
Check if drop User is supported	boolean UserManager.isDropUserSupported()	boolean UserManager.isDropUserSupported()
Search Users by given search criteria	SearchResponse IdentityStore.searchUsers(SearchParameters params)	** List<User> UserManager.searchUsers(List<PropertyFilterValue> attrFiltersList) ** SearchResults<User> UserManager.searchUsersbyPage(List<PropertyFilterValue> attrFiltersList)
Search an User by name /uniquename / guid	User IdentityStore.searchUser (String name)	** User UserManager.searchUser(List<PropertyFilterValue> attrFiltersList) ** User UserManager.searchUser(String loginid)
Check if User exists in the repository for a given User object	boolean IdentityStore.exists (User user)	boolean UserManager.exists(User subj)

Table 4–1 (Cont.) Comparison Between User-Related API and ArisID API

Functionality	User/Role API Method	Identity Beans Method
Simple search filter (search based on a single attribute name, type and value)	SimpleSearchFilter	Filter defined for search interaction in CARML file
Complex Search Filter (search based on more than one attribute with filter conditions and nested filters)	ComplextSearchFilter	Limited Support available where the actual attributes based on which the search will be made is predefined in CARML file.
Getting a property value for a given property name	String User.getPropertyVal(String propName) **Note**: User Role API, fetches the attribute values from cache. If it misses cache, it fetches from repository.	String User.getAttributeValue(String attribute) boolean User.getPredicateValue(String predicate) Object User.getPropertyValue(String property) **Limitation**: Returns attribute values from User object that's already fetched from the repository. This doesn't go to repository again to fetch the latest value. This applies to all variations of get values.
Getting the User property for a given property name	Property User.getProperty(String propName)	IAttributeValue User.getAttribute(String attribute) PredicateValue User.getPredicate(String predicate) Object User.getProperty(String property)
Getting the user properties for a given set of property names	Map User.getProperties()	Map<String,IAttributeValue> User.getAllAttributes() Map<String,PredicateValue> User.getAllPredicates() Map<String,Object> User.getAllProperties()
Get all user property names from the schema	List IdentityStore.getUserPropertyNames() **Note**: Returns the names of all the properties in the schema	List<String> UserManager.getAllAttributeNames() List<String> UserManager.getAllPredicateNames() List<String> UserManager.getAllPropertyNames()
Changing the attribute value in the repository of an user	void User.setProperty(ModProperty mprop)	void User.setAttributeValue(String attrName, String attrValue) void User.setAttribute(ModPropertyValue attr)
Changing the set of attribute values in the repository for an user	void User.setProperties(ModProperty[] modPropObjs) void User.setProperties(LdapContext ctx, ModProperty[] modPropObjs)	void User.setAttributes(List<ModPropertyValue> attrs)

Table 4–1 (Cont.) Comparison Between User-Related API and ArisID API

Functionality	User/Role API Method	Identity Beans Method
Get all the reports of an User either direct or indirect	SearchResponse User.getReportees(boolean direct)	** List<User> UserManager.getReportees(User user, int nLevels)
Get Management chain of an user	List User.getManagementChain(int max, String upToManagerName, String upToTitle)	** List<User> UserManager.getManagementChain(User user, int nLevels, String title, String manager)
Get/Set of Binary Attributes	Available. Property in User/Role API supports binary attributes byte[] user.getJPEGPhoto() void user.setJPEGPhoto(String imgpath)	Available. byte[] User.getJpegphoto() void User.setJpegphoto(byte[] value)
Selecting the Realm	Available. env.put(OIDIdentityStoreFactory.RT_SUBSCRIBER_NAME, "<realm dn>"); IdentityStoreFactory.getIdentityStoreInstance(env);	This is part of Mapping configuration.

Role-Related APIs

Table 4–2 provides a comparison between the Role-related API method and the corresponding Identity Beans method available in the Identity Governance Framework Aris ID API.

Identity Beans methods marked with a double asterisk (**) have an optional parameter: `Map<String.Object> appCtxMap`. For example, `RoleManager.searchRolesbyPage(List<PropertyFilterValue> attrFiltersList, Map<String,Object> appCtxMap).appCtxMap` may contain the following elements:

- `UserManager.APP_CTX_AUTHUSER`: java.security.Principal, the user context to execute under.
- `UserManager.APP_CTX_PAGE`: String value of Page size. This is applicable only for search methods returning `SearchResults` object.
- `UserManager.APP_CTX_LOCALE`: String value of language code.

Table 4–2 Comparison Between Role-Related APIs and ArisID API

Functionality	User/Role API Method	Identity Beans Method
Creating a Role	Role RoleManager.createRole(String name, int scope) Role RoleManager.createRole(String name)	** void RoleManager.createRole(List<PropertyValue> attrVals)

Table 4–2 (Cont.) Comparison Between Role-Related APIs and ArisID API

Functionality	User/Role API Method	Identity Beans Method
Deleting a Role	void RoleManager.dropRole(RoleProfile role) void RoleManager.dropRole(Role role)	** void RoleManager.dropRole(IPrincipalIdentifier principal) ** void RoleManager.dropRole(Role subj)
Check if create role is supported	boolean RoleManager.isCreateRoleSupported()	boolean RoleManager.isCreateRoleSupported()
Check if modify role is supported	boolean RoleManager.isModifyRoleSupported()	boolean RoleManager.isModifyRoleSupported()
Check if delete role is supported	boolean RoleManager.isDropRoleSupported()	boolean RoleManager.isDropRoleSupported()
Is the Group owned by a User	boolean RoleManager.isGranted(Role parent, Principal principal)	** boolean RoleManager.isGranted(Role role, Role member, boolean direct) ** boolean RoleManager.isGranted(Role role, User member, boolean direct)
Is the Group owned by a User	boolean RoleManager.isOwnedBy(Role parent, Principal principal)	** boolean RoleManager.isOwned(Role role, Role owner, boolean direct) ** boolean RoleManager.isOwned(Role role, User owner, boolean direct)
Is the group managed by a User	boolean RoleManager.isManagedBy(Role parent, Principal principal)	** boolean RoleManager.isManaged(Role role, Role manager, boolean direct) ** boolean RoleManager.isManaged(Role role, User manager, boolean direct)
Get all the members of a Role either direct / indirect	SearchResponse Role.getGrantees(SearchFilter filter, boolean direct)	** List<User> RoleManager.getGrantees(Role role, int nLevels, UserManager usermanager)
Add a user as a member to a role	void RoleManager.grantRole(Role parent, Principal principal)	** void Role.addMember(Role role) ** void Role.addMember(String value) ** void Role.addMember(User user)
Remove a user from being member of a role	void RoleManager.revokeRole(Role parent, Principal principal)	** List<User> RoleManager.getOwners(Role role, int nLevels, UserManager usermanager)
Get all the owners of a specific Role either direct / indirect	SearchResponse Role.getOwners(SearchFilter filter, boolean direct) SearchResponse Role.getOwners(SearchFilter filter)	** List<User> RoleManager.getOwners(Role role, int nLevels, UserManager usermanager)
Add a user as a owner of a role	void Role.addOwner(Principal principal)	** void Role.addOwner(Role role) ** void Role.addOwner(String value) ** void Role.addOwner(User user)

Table 4–2 (Cont.) Comparison Between Role-Related APIs and ArisID API

Functionality	User/Role API Method	Identity Beans Method
Remove a user from being a owner of a Role	void Role.removeOwner(Principal principal)	** void Role.deleteOwner(Role role) ** void Role.deleteOwner(String value) ** void Role.deleteOwner(User user)
Get all the managers of a Role either direct / indirect	SearchResponse Role.getManagers(SearchFilter filter, boolean direct) SearchResponse Role.getManagers(SearchFilter filter)	** List<User> RoleManager.getManagers(Role role, int nLevels, UserManager usermanager)
Add an user as a manager of a Role	void Role.addManager(Principal principal)	** void Role.addManager(Role role) ** void Role.addManager(String value) ** void Role.addManager(User user)
Remove an user from being manager of a Role	void Role.removeManager(Principal principal)	** void Role.deleteManager(Role role) ** void Role.deleteManager(String value) ** void Role.deleteManager(User user)
Getting the role property	Property Role.getProperty(String propName) **Note**: User Role API, fetches these attribute values from cache. If it misses cache, it fetches from repository)	IAttributeValue Role.getAttribute(String attribute) PredicateValue Role.getPredicate(String predicate) Object Role.getProperty(String property)
Determine the Role Type	Role.isApplicationRole Role.isEnterpriseRole Role.isSeeded	
Search Roles for a given search criteria	SearchResponse IdentityStore.searchRoles(int scope, SearchParameters params)	** List<Role> RoleManager.searchRoles(List<PropertyFilterValue> attrFiltersList) ** SearchResults<Role> RoleManager.searchRolesbyPage(List<PropertyFilterValue> attrFiltersList)
Search a Role by name / uniquename / guid	Role IdentityStore.searchRole(int searchType, String value)	** Role RoleManager.searchUser(List<PropertyFilterValue> attrFiltersList) ** Role RoleManager.searchUser(String guid)
Search both User and Roles for a given filter	SearchResponse IdentityStore.search(SearchParameters params)	This is available through separate methods: UserManager.searchUsers, RoleManager.searchRoles

A

Sample Application

The following sample application uses IDX User/Role Beans.

SearchUsers.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="org.openliberty.arisid.*"%>
<%@page import="org.openliberty.arisidbeans.*"%>
<%@page import="oracle.igf.userrole.*"%>
<%@page import="java.util.*"%>
<%@page import="java.net.URI"%>
<%!public static UserManager uMgr = null;
{
        try {
                uMgr = new UserManager(null);
        } catch (Exception e) {
                e.printStackTrace();
        }

}
%>
<html>
<head>
<title>Search Users</title>
<%


String firstname = request.getParameter("firstname");
String lastname = request.getParameter("lastname");
String telephone = request.getParameter("telephone");


List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("firstname", firstname,
AttributeFilter.OP_BGNSWITH));
attrFilters.add(new PropertyFilterValue("lastname", lastname, AttributeFilter.OP_
BGNSWITH));
attrFilters.add(new PropertyFilterValue("telephone", telephone,
AttributeFilter.OP_CONTAINS));

List<User> subjs = uMgr.searchUsers(attrFilters);

%>
```

```
</head>
<body>

<a href="SearchUsers.html">Home</a>
<center>List of Users with FirstName starting with "<%=firstname%>", LastName
starting with "<%=lastname%>" and TelephoneNumber containing
"<%=telephone%>"</center>

<%
Iterator<User> sIter = subjs.iterator();
while (sIter.hasNext()) {
        User subj = sIter.next();

        Map<String, IAttributeValue> vals = subj.getAllAttributes();
        Iterator<IAttributeValue> iter = vals.values().iterator();
%>
<table border="0">
        <tr>
                <th>Item</th>
                <th>Value</th>
        </tr>
        <%
                while (iter.hasNext()) {
                        IAttributeValue val = iter.next();
                        String name = val.getNameIdRef();
                        String value = null;
                        if (val.size() > 0)
                                value = val.get(0);
if (value != null)
{
        %>
        <tr>
                <td><%=name%></td>
                <td><%=value%></td>
        </tr>
        <%
}
                }
        %>
</table>
<%
        }
%>
<br>
<br>
<br>
<a href="SearchUsers.html">Home</a>
</body>
</html>
```

SearchUsers.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD><TITLE>Search Users</TITLE></HEAD>
<BODY>
<FORM METHOD=POST ACTION="SearchUsers.jsp">

First Name Starting with <INPUT TYPE=TEXT NAME=firstname SIZE=30><BR><BR>
```

```
Last Name Starting with <INPUT TYPE=TEXT NAME=lastname SIZE=30><BR><BR>
Telephone Number containing <INPUT TYPE=TEXT NAME=telephone SIZE=15><BR><BR>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```