LDAP Integration Technical Specification

Leo Przybylski
przybyls@arizona.edu

January 12, 2010

Contents

1 Technical Description

Integrate KIM with UA NetId LDAP systems for authentication. When users authenticate into KFS, KIM will authenticate via LDAP with UA *EDS*

while roles and permissions will be kept internal to the KIM database.

1.1 Jira Tasks

- KITT-271
- KITT-272
- KITT-474
- KITT-558
- KITT-713

2 Details

2.1 Spring LDAP

Spring LDAP is an adapter layer between Spring and LDAP datasources. The following description is taken from the *Spring LDAP* website:

> Spring LDAP is a Java library for simplifying LDAP operations, based on the pattern of Spring's JdbcTemplate. The framework relieves the user of common chores, such as looking up and closing contexts, looping through results, encoding/decoding values and filters, and more.
>
> The LdapTemplate class encapsulates all the plumbing work involved in traditional LDAP programming, such as creating a DirContext, looping through NamingEnumerations, handling exceptions and cleaning up resources. This leaves the programmer to handle the important stuff - where to find data (DNs and Filters) and what do do with it (map to and from domain objects, bind, modify, unbind, etc.), in the same way that JdbcTemplate relieves the programmer of all but the actual SQL and how the data maps to the domain model.
>
> In addition to this, Spring LDAP provides transaction support, a pooling library, exception translation from NamingExceptions to a mirrored unchecked Exception hierarchy, as well as several utilities for working with filters, LDAP paths and Attributes.

Spring LDAP requires J2SE 1.4 or higher to run, and works with Spring Framework 2.0.x as well as 2.5.x. J2SE 1.4 or higher is required for building the release binaries from sources. Release 1.2.1 also requires an installation of JavaCC 4.0 when building from source. That is not necessary for release 1.3.x, since it uses Maven2, which handles all such dependencies behind the scenes.

To use it:

Listing 1: spring-datasource.xml

```
<beans>
    ...
    ...
    <bean id="contextSource"
        class="org.springframework.ldap.support.
            LdapContextSource">
        <property name="url" value="ldaps://eds.arizona.edu
            :636" />
        <property name="base" value="ou=People,dc=eds,dc=
            arizona,dc=edu" />
        <property name="userName" value="uid=<userid>,ou=
            App Users,dc=eds,dc=arizona,dc=edu" />
        <property name="password" value="secret" />
        <property name="pool" value="true"/>
    </bean>
    <bean id="ldapTemplate" class="org.springframework.ldap
        .LdapTemplate">
        <constructor-arg ref="contextSource" />
    </bean>
    <bean id="ldapContact"
        class="edu.arizona.kim.dao.LdapContactDao">
        <property name="ldapTemplate" ref="ldapTemplate" />
    </bean>
</beans>

\emph{Note that ldaps:// protocol is used.}
```

2.2 KIM

KIM interfaces need to be implemented within *KFS* that communicate over LDAP with *EDS*. *KIM* will delegate to *EDS* over LDAPs with *Spring LDAP* by implementing the following service interfaces.

2.2.1 IdentityService

Below is an description of which methods need to be overwritten to supply *KIM* with access to Person data from *EDS*

getPrincipal
```
/** Get a KimPrincipal object based on the
    principalName. */
KimPrincipalInfo getPrincipal(String principalId);
```

getPrincipalByPrincipalName
```
KimPrincipalInfo
    getPrincipalByPrincipalName(String principalName);
```

lookupEntitys
```
/** Find entity objects based on the
    given criteria. */
List<KimEntity> lookupEntitys(Map<String,String>
    searchCriteria);
```

getEntityDefaultInfo
```
KimEntityDefaultInfo
    getEntityDefaultInfo( String entityId );
```

getEntityDefaultInfoByPrincipalId
```
KimEntityDefaultInfo
    getEntityDefaultInfoByPrincipalId( String
    principalId );
```

getEntityDefaultInfoByPrincipalName
```
KimEntityDefaultInfo
    getEntityDefaultInfoByPrincipalName( String
    principalName );
```

lookupEntityDefaultInfo
```
List<? extends
    KimEntityDefaultInfo> lookupEntityDefaultInfo( Map<
    String,String> searchCriteria, boolean unbounded );
```

getMatchingEntityCount `int getMatchingEntityCount( Map <String,String> searchCriteria );`

getEntityPrivacyPreferences
`KimEntityPrivacyPreferencesInfo getEntityPrivacyPreferences( String entityId );`

getDefaultNamesForPrincipalIds
`Map<String, KimEntityNamePrincipalNameInfo> getDefaultNamesForPrincipalIds(List<String> principalIds);`

getDefaultNamesForEntityIds `Map<String, KimEntityNameInfo> getDefaultNamesForEntityIds(List <String> entityIds);`

2.2.2 UiDocumentServiceImpl

The `IdentityManagementPersonDocument` is still used to save modify role, group, and delegation assignments even though all entity information is coming through EDS. This splits principal and entity information, but the `UiDocumentServiceImpl` makes it possible to accomplish this. The "Modify Entity" permission was removed from all roles because we no longer want entities to be managed through KFS.

Originally, the `UiDocumentServiceImpl` uses `Impl` domain objects couple to a database implementation, so it needs to be modified not to use uncoupled `Info` objects. Below is how `UiDocumentServiceImpl` is modified to do that.

loadEntityToPersonDoc is used to populate the `IdentityManagementPersonDocument` when the page loads from "edit" or "create new". Even though entity information is not being stored in the database, it still needs to be present on persons.

saveEntityPerson is used to store the information and actually update the person. It needed to modified to take into consider the check for the "Modify Entity" permission. Normally, even if the permission isn't present, the document will try to save entity information. By checking for this permission, the desired behavior takes place which is entities

will not be saved. Unlike `loadEntityToPersonDoc`, `Impl` domain objects are desirable here. The domain object that is modified is the `KimPrincipalImpl` which updates the `KRIM_PRNCPL_T` table and the necessary role, group, and delegation tables.

2.3 UA NetId

netid is the UA federated directory and contacts LDAP server. LDAP protocols are secured and tunneled over SSL. This section is just here as an example. NetId is just used by webauth. It actually won't be used for integration with KIM. EDS will be used instead.

Listing 2: An example of search netid.arizona.edu for students and employees using Java.

```
/*
 * Demonstrates how to perform a simple, authenticated bind
     to an LDAP
 * server over SSL, using JNDI
 */
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

/**
 * usage: java LdapExample <username> <password>
 *
 **/
 class LdapExample {
   public static void main(String[] args) throws Exception
      {
     if (args.length != 2) {
       throw new Exception( Usage: LdapExample <username>
          <password>  );
     }
     try {
       // Set up the environment for creating the initial
          context
       Hashtable env = new Hashtable();
       env.put(Context.INITIAL_CONTEXT_FACTORY,
       "com.sun.jndi.ldap.LdapCtxFactory");
```

```
env.put(Context.PROVIDER_URL,
 ldap://netid.arizona.edu:636/ou=Accounts,ou=NetID,
   ou=CCIT,o=University%20of%20Arizona,
c=US");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL,
 uid=  + args[0] +
  ,ou=Accounts,ou=NetID,ou=CCIT,o=University of
   Arizona,c=US );
env.put(Context.SECURITY_CREDENTIALS, args[1]);
env.put(Context.SECURITY_PROTOCOL, "ssl");
// Create initial context
DirContext ctx = new InitialDirContext(env);

/*
 * Initial context has been established and bind
    performed
 */
// Specify the ids of the attributes to return
String[] attrIDs = { dbkey , "activeStudent", "
   activeEmployee"};
// Get the attributes requested for specified entry
Attributes attrs = ctx.getAttributes("uid=" + args
   [0], attrIDs);

/*
 * The attributes for the entry are contained in the
    Attributes object  attrs .
 * Iterate over all attributes and print them out.
 */
 if (attrs == null) {
   System.out.println("No attributes");
 } else {
   /* Print each attribute */
   for (NamingEnumeration ae = attrs.getAll(); ae.
      hasMore();) {
     Attribute attr = (Attribute)ae.next();
     System.out.println("attribute: " + attr.getID()
        );
     /* print each value */
```

```
                for (NamingEnumeration e = attr.getAll();
                e.hasMore();
                System.out.println("\tvalue: " + e.next())) ;
              }
            }
        }
        // Close the JNDI context when we're done
        ctx.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
```

2.4 EDS

EDS is UA's *Enterprise Directory Service*. Communicating with *EDS* is done over LDAP protocol. *EDS* will be used in lieu of netid. Follow the link given below for attributes that are retrievable through *EDS*

`http://iia.arizona.edu/eds_attributes` as a references for EDS attributes.

Below is a mapping of content that can be retrieved from EDS:

Attribute Name	Description	Multi-Valued	Required	ObjectClass	OID
cn	full name (first name middle initial last name), from SIS when the person's primary affiliation is student, PSOS when primary affiliation is employee, or DSV (NetID) when primary affiliation is affiliate			person	2.5.4.3
sn	last name, from SIS when the person's primary affiliation is student, or from PSOS when primary affiliation is employee			person	2.5.4.4
givenName	first name and middle initial, from SIS when the person's primary affiliation is student, PSOS when primary affiliation is employee, or DSV (NetID) when primary affiliation is affiliate			intetOrgPerson	2.5.4.42
eduPersonAffiliation	**Note: Please see the "Inclusion Rules" section for information on populations included in the EDS.** Possible values are student, admit, employee, faculty, staff, affiliate and member. The values are determined by a set of rules/heuristics applied to student, employee and departmental-sponsored visitor data, as represented in UIS. EMPLM_TYPE in the ZPSOS_EMPLOYEES table is consulted and used for employees: EMPLM_TYPE codes of (G, S, W) result in "student" and "employee" values being added to the list of affiliations; (A, C, L, X) codes result in a "staff" value; (E, F) codes require further logic(based on the "PCT" columns in X_FTE_ASSIGN_DISTINCT) to determine if an employee is faculty, staff (the "staff" designation includes all non-faculty–e.g., appointed professionals and administrators), or both. "Staff" and "faculty" affiliations are instances of "employee", thus "employee" will always be included along with these affiliations. Admitted students who have not yet matriculated will have the "admit" affiliation value. Students will have the "student" affiliation value, and departmental-sponsored visitors (DSVs) will have "affiliate". "Member" is added if one or more affiliation values (with the exception of "admit") exist.	y		eduPerson	1.3.6.1.4.1.5923.1.1.1.1

Attribute Name	Description	Multi-Valued	Required	ObjectClass	OID
eduPersonPrimaryAffiliation	**Note: Please see the "Inclusion Rules" section for information on populations included in the EDS.** Possible values are student, admit, employee, faculty, staff, affiliate and member. The determination of "primary affiliation" is based on a set of heuristics closely related to those used in determining the set of affiliations represented in eduPersonAffiliation. In the trivial case of a single value for eduPersonAffiliation, that same value will be used for eduPersonPrimaryAffiliation. When a person is both a student and an employee, ZPSOS_EMPLOYEES.EMPLM_TYPE is consulted; if the employee type is non-student, i.e. not in (G, S, W), the employee classification ("staff" or "faculty") will be the primary affiliation, otherwise "student" will be the primary affiliation. For employees who have both "staff" and e-set, heuristics based on % FTE of various positions from which the employee is funded and the UA OrgMap are used to determine whether "faculty" takes precedence; "staff" includes all non-faculty employment affiliations, including administrators.			eduPerson	1.3.6.1.4.1.5923.1.1.1.5
eduPersonNickName	person's nickname (currently not populated)	y		eduPerson	1.3.6.1.4.1.5923.1.1.1.2
uid	person's UA NetID username			inetOrgPerson	0.9.2342.19200300.100.1.1
uaid	uniquely identifies each UA person. It is currently created in UIS using logic that matches person records from SIS and PSOS and assigns a unique ID to every UA member.	y		arizonaEduPerson	1.3.6.1.4.1.5643.10.0.1
mail	UA email address; if a person is both an employee and a student, the employee email address from PSOS is listed in which case the value of this attribute will be the same as employeeEmail); otherwise the email address listed in the source system reflecting primary affiliation is used			arizonaEduPerson	0.9.2342.19200300.100.1.3
dateOfBirth	date of birth in format YYYYMMDD; from SIS when the person is a student only, or from PSOS when a person is an employee or both an employee and a student			arizonaEduPerson	1.3.6.1.4.1.5643.10.0.49
employeeBldgName	name of the building that corresponds to an employee's primary department			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.13
employeeBldgNum	number of the building that corresponds to an employee's primary department			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.14
employeeEmail	official work email address, as listed in PSOS. This may not be an "@email.arizona.edu" address, but will end in a ".arizona.edu" domain name			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.19
employeeId	9-digit number, currently created by PSOS, that uniquely identifies a UA employee			arizonaEduEmployee	1.3.6.1.4.1.5643.2.0.4
employeeIncumbentPosition	a colon (:) separated list with an employee's title, Position Control Number (PCN) from PSOS (######), start date, and end-date for their position (YYYYMMDD)	y		arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.53

Attribute Name	Description	Multi-Valued	Required	ObjectClass	OID
employeeInfoReleaseCode	"Y" for employees who have elected to publish their UA email address in the campus directory, "N" for people who have chosen not to publish their email address. This value defaults to "Y" for employees who have not explicitly set a preference			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.7
employeeIsFerpaTrained	"Y" for employees who have had FERPA training, and "N" for people who have not			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.42
employeePhone	UA phone number of an employee in the format ##########			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.17
employeePoBox	Post Office Box number of an employee's work-related mailing address			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.12
employeePositionFunding	Colon (:) separated list containing the PSOS position control number (PCN) and the funding department number. For each position an employee occupies (see employeeIncumbentPosition) there will be at least one corresponding value in this attribute if the position is funded; non-funded positions will not appear in the value set of this attribute. Positions funded by multiple departments will have multiple values in this attribute	y		arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.54
employeePrimaryDept	Dept # of employee's primary department (also known as home department); refers to the department where an employee's paycheck is sent			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.8
employeePrimaryDeptName	Textual description corresponding to employeePrimaryDept			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.52
employeeRoomNum	room number associated with an employee's primary office			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.15
employeeRosterDept	Dept # of department to which an employee submits their timesheet			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.10
employeeStatus	PSOS status code for employees; "A" active, "B" retired/back-to-work, "D" deceased, "F" member of affiliated agency, "H" hold (pre-hire), "L" leave of absence w/o pay, "M" away on fellowship, "N" non-salaried "P" leave with pay, "R" retired, "T" terminated, "U" unemployed due to layoff			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.4
employeeStatusDate	date when an employee's current status began, in the format YYYYMMDD			arizonaEduEmployee	1.3.6.1.4.1.5643.10.0.5
employeeType	one letter code from PSOS that identifies the type of an employment. "A" = Ancillary Staff, "C" = Classified Staff, "E" = Appointed - Academic Year, "F" = Appointed, Fiscal Year, "G" = Grad Asst/Assoc, "L" = Federal Appt, "S" = Student, "W" = Work Study, "X" = Flex Staff			2.16.840.1.113730.3.1.4	
studentAcademicProgram	can be multi-valued because students can be enrolled in multiple programs concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 – 4]), Degree (abbreviation), College (see codes below), Major (abbreviation) and Option (abbreviation). This attribute corresponds to programs in which a student is enrolled in the current semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.35

Attribute Name	Description	Multi-Valued	Required	ObjectClass	OID
studentAcademicProgramFuture	can be multi-valued because students can be enrolled in multiple programs concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 − 4]), Degree (abbreviation), College (see codes below), Major (abbreviation) and Option (abbreviation). This attribute corresponds to programs in which a student is enrolled in the future semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.44
studentAcademicProgramPast	can be multi-valued because students can be enrolled in multiple programs concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 − 4]), Degree (abbreviation), College (see codes below), Major (abbreviation) and Option (abbreviation). This attribute corresponds to programs in which a student is enrolled in the past semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.43
studentTermStatus	contains a colon (:) delimited list student status attributes – Term (format from SIS - YY[1 − 4]), Career (G=Graduate,U=Undergraduate,P=Professional), Class Code (see Class Code table below), Full or Part Time ("F"=full time, "P"=part time, "N"=zero enrolled hours) and Residency ("UM"=unclassified, "RM"=resident, "NM"=non-resident, "PM"=pending, "WM"=western graduate exchange, "XX"=not classified if ¡7 units) for current semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.37
studentTermStatusFuture	contains a colon (:) delimited list student status attributes – Term (format from SIS - YY[1 − 4]), Career (G=Graduate,U=Undergraduate, P=Professional), Class Code (see Class Code table below), Full or Part Time ("F"=full time, "P"=part time, "N"=zero enrolled hours) and Residency ("UM"=unclassified, "RM"=resident, "NM"=non-resident, "PM"=pending, "WM"=western graduate exchange, "XX"=not classified if ¡7 units) for future semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.48

Attribute Name	Description	Multi-Valued	Required	ObjectClass	OID
studentTermStatusPast	contains a colon (:) delimited list student status attributes – Term (format from SIS - YY[1 – 4]), Career (G=Graduate,U=Undergraduate, P=Professional), Class Code (see Class Code table below), Full or Part Time ("F"=full time, "P"=part time, "N"=zero enrolled hours) and Residency ("UM"=unclassified, "RM"=resident, "NM"=non-resident, "PM"=pending, "WM"=western graduate exchange, "XX"=not classified if ¡7 units) for past semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.47
studentEmail	UA email address, which concatenates the NetID with "@email.arizona.edu" (or "@u.arizona.edu")			arizonaEduStudent	1.3.6.1.4.1.5643.10.0.32
studentId	"S" + 8 digits, or 9-digit number, that uniquely identifies a UA student			arizonaEduStudent	1.3.6.1.4.1.5643.10.0.39
studentInfoReleaseCode	one-letter code that students can update to restrict who can view address, phone and email and attendence information. "B" indicates blank, which means there are no restrictions. "D" recognizes that a student attends or attended UA, but releases no other information. "L" indicates that no address, phone or email information is released. "M indicates that no address or phone information is released. "N" indicates an outright restriction on the person (no information at all is released). "X" indicates that a student is deceased. "A" indicates that no address information is released. "P" indicates that the permanent address information is not released.			arizonaEduStudent	1.3.6.1.4.1.5643.10.0.31
studentMinor	can be multi-valued because students can have multiple minors concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 – 4]), Degree (abbreviation), College (abbreviation, may not be accurate), Minor (abbreviation). This attribute coresponds to minors for the current semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.36
studentMinorFuture	can be multi-valued because students can have multiple minors concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 – 4]), Degree (abbreviation), College (abbreviation, may not be accurate), Minor (abbreviation). This attribute coresponds to minors for the future semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.46
studentMinorPast	can be multi-valued because students can have multiple minors concurrently. Each value contains a colon (:) delimited list of the program's associated Term Code (format from SIS - YY[1 – 4]), Degree (abbreviation), College (abbreviation, may not be accurate), Minor (abbreviation). This attribute coresponds to minors for the past semesters. (For details on how Terms are determined to be current, past and future, see the note below)	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.45
studentAPDesc	can be multi-valued because students can be enrolled in multiple programs concurrently. text description of a student's academic program (Type of Student - Degree - Major - Term, e.g. "Doctoral Student - Doctor of Philosophy - Spanish - 2008 Fall")	y		arizonaEduStudent	1.3.6.1.4.1.5643.10.0.41
studentAdmitTerm	If student is admitted, but has not yet registered for an orientation session nor enrolled for classes, this attribute will be present and will reflect the term code (format from SIS - YY[1 – 4]) for which the student has been admitted			arizonaEduStudent	1.3.6.1.4.1.5643.10.0.50

Inclusion Rules:

Students admitted for a future term - OR - registered for a future term orientation session - OR - enrolled in a current, past (no more than 1 academic year in the past from current term) or future term

Employees incumbent in a budgeted position - OR - has been incumbent in a budgeted position with an end date less than 100 days in the past

Departmental Sponsored Visitors currently sponsored, non-expired, DSVs

Note: Terms are considered to be "current" if their SIS term status is "R" (registering), and for two weeks after the SIS term status changes from "R" (registering) to "G" (grading). This means that multiple terms may be considered "current" at the same time. This directory contains about one year's worth of information, including the current semester(s), and up to three past and future semesters.

College Abbreviations	**Class Codes**
[AL] Agriculture & Life Sciences	[UNC] Special Undergrad - Unclass
[AF] Family & Consumer Sciences	*Student registering for undergraduate credit who may or may not hold a 4-year college degree and is not presently applying these credits towards a degree.*
[$A0$] Fine Arts	[FR] Freshman
[$A3$] Honors	[SO] Sophomore
[$A4$] Humanities	[JR] Junior
[$A6$] Science	[SR] Senior
[$A8$] Social & Behav Sci	[$1ST$] First Year
[$A9$] University College	[$2ND$] Second Year
[CA] Architecture & Landscape Architecture	[$3RD$] Third Year
[ED] Education	[$4TH$] Fourth Year
[EG] Engineering	[$5TH$] Fifth Year
[MG] Eller College of Management	[SPP] Special Professional - Unclass
[NU] Nursing	Unclassified status to be used for Law, Medicine and Pharmacy.
[OS] College of Optical Sciences	[GM] Masters Student
[PH] Pharmacy	[GMP] Masters Student, Provisional
[PZ] Mel&Enid Zuckerman AZ Col Public Health	[GC] Graduate Certificate
[RG] Graduate College	[GND] Graduate, Non-Degree
[RM] Medicine	Student enrolled in the Graduate College and not presently admitted to a degree program.
[SC] Correspondence	[GP] Graduate Professional
[SG] Guadalajara	[GD] Doctoral Student
[US] University of Arizona South	[GDP] Doctoral Student, Provisional
[XL] James E. Rogers College of Law	[GEX] Graduate Exchange Student
[AG] Agriculture	[GIS] Graduate Foreign Intntl Specl
[AI] Family & Cons	[SPG] Special Graduate - Unclass
[AR] Architecture	*Student holding a 4-Year or Graduate degree registering for credit and not presentlyapplying it toward another degree.*
[AS] Arts & Sciences	[SS] Specialist Student
[AX] CAPLA	[SSP] Specialist Student, Prov.
[BN] Business & Public Admin	[EFR] Enrolling Freshman
[HP] Health Professions	[$SR5$] Senior 5th Year
[PL] Public Health	[SB] Second Bachelors
[RL] Law	[LW] Law
[TG] General	[$GR1$] Graduate 1
[TI] Interdepartmental	[$GR2$] Graduate 2
[XX] Administrative College	[$GR3$] Graduate 3
[AC] Arizona International College	[$GR4$] Graduate 4
[$A2$] A & S - General	[$MD1$] Medical 1
[BX] Eller College of Bus & Public Admin	[$MD2$] Medical 2
[EA] Earth Science	[$MD3$] Medical 3
[EN] Engineering & Mines	[$MD4$] Medical 4
[FA] Fine Arts	[$NU1$] Nursing 1
[HR] Hlth Related Profess	[$NU2$] Nursing 2
[LA] Liberal Arts	[$NU3$] Nursing 3
[MN] Mines	[$NU4$] Nursing 4
[QA] Continuing Education	[$NU5$] Nursing 5
[QI] No Credit	[$NU6$] Nursing 6
[SE] Extension	[$PD1$] Pharmd 1 [$PD2$] Pharmd 2 [$PD3$] Pharmd 3 [$PD4$] Pharmd 4 [$PH6$] Pharmacy Doctoral [SPU] Special Undergrad - Unclass Used by Financial Aid Office. See description for "UNC". [TC] Teacher Certification [CSA] Consortium Agreements [HCA] Host Consortium Agreement Used by Financial Aid Office.

The table below maps KIM Class Attributes to EDS Attributes

KIM Class	Attribute Name	EDS Attribute Name
KimPrincipalInfo	principalId	uaid
KimPrincipalInfo	entityId	uaid
KimPrincipalInfo	principalName	uid
KimEntityDefaultInfo	affiliations	eduPersonAffiliation
KimEntityDefaultInfo	defaultAffiliation	eduPersonPrimaryAffiliati
KimEntityNameInfo	lastName	sn
KimEntityNameInfo	firstName	givenName
KimEntityEmployementInformationInfo	employeeId	employeeId
KimEntityEmployementInformationInfo		employeeEmail
KimEntityEmployementInformationInfo		employeePhone
KimEntityEmployementInformationInfo		employeePoBox
KimEntityEmployementInformationInfo		employeePrimaryDept
KimEntityEmployementInformationInfo		employeePrimaryDeptNam
KimEntityEmployementInformationInfo		employeeType
KimEntityEmployementInformationInfo		employeeStatus

2.4.1 Connecting to EDS

- Hostname: **eds.arizona.edu**
- Port #: 636 (Note: directory may only be accessed via the LDAPS protocol; TLS is not supported)
- Application authentication DN base: **ou=App Users,dc=eds,dc=arizona,dc=edu**
- Authentication attribute: uid (the DN used to authenticate will be of the form **uid=¡appuser¿,ou=App Users,dc=eds,dc=arizona,dc=edu**)
- Search base: **ou=People,dc=eds,dc=arizona,dc=edu**

3 Development Steps

1. Register an EDS Account

3.1 Setup Spring LDAP

3.1.1 Modifications to the edu/arizona/kfs/sys/spring-sys.xml File

The following was added to connect *Spring LDAP* to *EDS*

```
<bean     id="contextSource"
       class="org.springframework.ldap.core.support.
          LdapContextSource">
    <property name="url" value="ldaps://eds.arizona.edu
       :636" />
    <property name="base" value="ou=People,dc=eds,dc=
       arizona,dc=edu" />
    <property name="authenticationSource" ref="
       authenticationSource" />
</bean>

<bean     id="authenticationSource"
       class="org.springframework.ldap.authentication.
          DefaultValuesAuthenticationSourceDecorator">
    <property name="target" ref="
       springSecurityAuthenticationSource" />
    <property name="defaultUser" value="uid=user,ou=App
       Users,dc=eds,dc=arizona,dc=edu" />
    <property name="defaultPassword" value="[secret]" />
</bean>

<bean     id="springSecurityAuthenticationSource"
       class="org.springframework.security.ldap.
          SpringSecurityAuthenticationSource" />

<bean id="ldapTemplate" class="org.springframework.ldap.
    core.LdapTemplate">
     <constructor-arg ref="contextSource" />
</bean>
```

The *Kuali Rice* `ParameterService` is used to store the map between *KIM* and *EDS* attributes. Still, many attribute names are stored in a constants class populated through Spring. See below

```
<bean id="azKimConstants" class="edu.arizona.kim.Constants
   ">
  <property name="uaidEdsProperty"           value="uaid" />
  <property name="uidEdsProperty"            value="uid" />
  <property name="snEdsProperty"             value="sn" />
  <property name="givenNameEdsProperty"      value="
     givenName" />
```

```
  <property name="entityIdKimProperty"        value="entityId
      " />
  <property name="employeeMailEdsProperty"    value="
      employeeMail" />
  <property name="employeePhoneEdsProperty"   value="
      employeePhone" />
  <property name="defaultCountryCode"         value="1" />
  <property name="mappingParameterName"       value="
      KIM_TO_EDS_FIELD_MAPPINGS" />
  <property name="unmappedParameterName"      value="
      KIM_TO_EDS_UNMAPPED_FIELDS" />
  <property name="parameterNamespaceCode"     value="KFS-SYS"
       />
  <property name="parameterDetailTypeCode"    value="Config"
      />
</bean>
```

The constants class as well as the *Spring LDAP* integration and *Kuali Rice* `ParameterService` are injected into the `EdsPrincipalDaoImpl` instance.

```
<bean id="edsPrincipalDao" class="edu.arizona.kim.
    dataaccess.impl.EdsPrincipalDaoImpl">
  <property name="ldapTemplate"       ref="ldapTemplate" />
  <property name="parameterService"   ref="parameterService"
      />
  <property name="kimConstants"       ref="azKimConstants" />

</bean>
```

The `EdsPrincipalDaoImpl` is an implementation of `PrincipalDao` which is delegated by the `EdsIdentityServiceImpl`. The `EdsPrincipalDaoImpl` connects to *EDS* and maps the principal and entity information into *KIM* domain objects.

2. Implement/Override Methods in IdentityService

3. Create PrincipalDao for searching for Principal/Entity information from EDS.

3.1.2 Retrieving EDS Information as KIM Domain Objects

Spring LDAP offers a `ContextMapper` interface for these kinds of mappings; therefore, all of the mappings are in pure java. This is how `KimPrincipal` is mapped from *EDS*.

```
contextMappers.put(KimPrincipalInfo.class, new
    AbstractContextMapper() {
     public Object doMapFromContext(DirContextOperations
         context) {
          final KimPrincipalInfo person = new
              KimPrincipalInfo();
          person.setPrincipalId(context.getStringAttribute(
              getKimConstants().getUaidEdsProperty()));
          person.setEntityId(context.getStringAttribute(
              getKimConstants().getUaidEdsProperty()));
          person.setPrincipalName(context.getStringAttribute(
              getKimConstants().getUidEdsProperty()));
          return person;
     }
});
```

`contextMappers` is an instance map created for holding `ContextMapper` instances. Each DTO type has a mapper associated with it for retrieving the desired information from *EDS*. Notice the use of `getKimConstants()`. This is how constant property names are used in the mapping. Also, notice that here the `ParameterService` is not used. The `ParameterService` is only used for mapping *KIM* criteria in lookup scenarios. When retrieving information from *EDS*, the `ParameterService` is entirely useless. The `ContextMapper` is used instead. It gives more flexibility when mapping attributes of a specific class. Below is how the `ContextMapper` is actually used.

```
public <T> List<T> search(Class<T> type, Map<String, Object
    > criteria) {
     AndFilter filter = new AndFilter();
```

```
    for (Map.Entry<String, Object> entry : criteria.
        entrySet()) {
         if (entry.getValue() instanceof Iterable) {
             OrFilter orFilter = new OrFilter();
             for (String value : (Iterable<String>) entry.
                 getValue()) {
                  orFilter.or(new EqualsFilter(entry.getKey()
                      , value));
             }
             filter.and(orFilter);
         }
         else {
             filter.and(new EqualsFilter(entry.getKey(), (
                 String) entry.getValue()));
         }
    }
    return getLdapTemplate().search(DistinguishedName.
        EMPTY_PATH, filter.encode(), contextMappers.get(type
        ));
}
```

Spring LDAP gives a very flexible API for querying Directory-Based systems. The `search()` method takes advantage of several classes from the API in order to create a fairly generic query of *EDS*. On the last line, the `LdapTemplate` is used with a verb—ContextMapper— retrieved from the `contextMappers` map. It is retrieved by passing through the desired type; therefore, in the case of searching for a `KimPrincipal` we would use something like this:

```
public KimPrincipalInfo getPrincipal(String principalId) {
    Map<String, Object> criteria = new HashMap();
    criteria.put(getKimConstants().getUaidEdsProperty(),
        principalId);
    List<KimPrincipalInfo> results = search(
        KimPrincipalInfo.class, criteria);

    if (results.size() > 0) {
         return results.get(0);
    }

    return null;
```

```
}
```

Again, there isn't any need for the `ParameterService` yet because we know exactly what we want from *EDS*.

3.1.3 Using Mapping KIM Attributes to EDS Attributes for Lookups

KIM has an API method called `lookupEntityDefaultInfo` which is used by Kuali Lookups for querying information. The call will provide a map of information in terms of *KIM* attributes. This means that the map or search criteria is pretty meaningless to *EDS* or any Directory-based service for that matter. The *KIM* attributes need to be mapped to *EDS* attributes in order for the query to be made. For this, the `ParameterService` is used.

```
public List<? extends KimEntityDefaultInfo>
    lookupEntityDefaultInfo(Map<String,String>
    searchCriteria, boolean unbounded) {
     List<KimEntityDefaultInfo> results = new ArrayList();
     Map<String, Object> criteria = new HashMap();

     for (Map.Entry<String, String> criteriaEntry :
        searchCriteria.entrySet()) {
         info(String.format("Searching with criteria %s = %s
            ", criteriaEntry.getKey(), criteriaEntry.
            getValue()));

         if (isMapped(criteriaEntry.getKey())) {
             criteria.put(getEdsAttribute(criteriaEntry.
                getKey()), criteriaEntry.getValue());
         }
     }

     return search(KimEntityDefaultInfo.class, criteria);
}

private Matcher getKimAttributeMatcher(String kimAttribute)
     {
     Parameter mappedParam = getParameterService()
     .retrieveParameter(getKimConstants().
        getParameterNamespaceCode(),
     getKimConstants().getParameterDetailTypeCode(),
```

```
        getKimConstants().getMappedParameterName());

        String regexStr = kimAttribute + "=([^=;]*).*";
        return Pattern.compile(regexStr).matcher(mappedParam.
            getParameterValue());
    }

    private boolean isMapped(String kimAttribute) {
        return getKimAttributeMatcher(kimAttribute).matches();
    }

    private String getEdsAttribute(String kimAttribute) {
        Matcher matcher = getKimAttributeMatcher(kimAttribute);
        matcher.matches();
        return matcher.group(1);
    }
```

By using regular expressions and storing parameters in the database for retrieval by the `ParameterService`, the task of mapping *KIM* attributes to *EDS* attributes is pretty trivial.