A Recipe for Configuring and Operating LDAP Directories

Michael R Gettes
Georgetown University
Georgetown Institute for Information Assurance
http://www.georgetown.edu/giia
Version 2.1 (2002/10/10)
May, 2000 (original version)

1 Tsp. DIT planning
1 Tbsp Schema design
3 oz. configuration
1000 lbs of data
(and a touch of sugar to make it taste good)

This document is intended to be a discussion point toward the development of common directory deployments within the Higher Education community. In particular, a hope is to have institutions configure and populate their directories in similar ways which would allow for two key capabilities:

1. Allow for the deployment of a common set of schema to be defined within the Higher Ed community. In particular, the effort of the eduPerson objectclass is designed to yield a well-defined set of attributes and uses for this common schema. Unfortunately, the history of the schema within LDAP to date has yielded objectclasses and attribute definitions that are rather ill-defined and not well understood in deployments. A common schema design and objectclass deployment will greatly increase the utility of a directory deployment for applications and inter-institutional authorizations, white-pages and application enablement.
2. As alluded to in the previous point, white-pages is an important capability for the Higher Ed community. Imagine that institutions do as suggested in the point above with a common schema and similar configuration deployments... we can then consider a directory of directories service which would search the Higher Ed community without having to store the data centrally, which would mean a loss of control over the data for the individual institutions. A suggestion for such a service is being developed within the Internet2 Middleware and EDUCAUSE communities with principal guidance from MACE.

Below are a set of recommendations and discussions which will hopefully lead us all in the direction of common directory schema and deployments. These recommendations come from the practical deployment of Georgetown University and discussions with other Universities regarding their implementations along with the keen advice and discussion with the members of MACE (Middleware Architectural Committee for Education) which includes Paul Hill (MIT), Keith Hazelton (Wisconsin), Ken Klingenstein (Internet-2/Colorado at Boulder), Steve Carmody (Brown), Mark Poepping (CMU), Bob Morgan (Washington), Michael Gettes (Georgetown). The order of the previous list is arbitrary. Further assistance provided by: David Henry (Maryland), Tom Jordan (Wisconsin), the regular and large crew from Penn State: Jim Leous, Jim Vucollo, Bob Fowles, Steve Kellogg, Renee Shuey. For Version 2 we add acknowledgements for Tom Barton (Memphis), Brendan Bellina (Notre Dame), Rob Banz (Maryland-Baltimore County) and Art Vandenberg (Georgia State University).

It should also be noted that while much of this document references two of the more popular LDAP implementations within Higher Education (Netscape/iPlanet/SunONE and OpenLDAP), that others are using these recommendations for directory servers not mentioned very much in this document. Art Vandenberg of Georgia State University has noted that he has utilized the recommendations in his Novell eDirectory implementation with great success. Art notes:

```
"When we began our directory work, I participated in several BOFs at Internet2, but
as soon as I noted our institution's use of Novell, I ended up in a group of 1 or 2
with a sense that we were going down a "non-standard" path. In fact, this is not so.
Early concerns were whether dc naming would work (it does), whether schema checking
was on (it is by default), whether ACLs were robust (very much so, to attribute
level), or whether replication was provided (yes, multi-master.) All in all, it
would seem advantageous for Internet2 to note that this LDAP Recipe can be relevant
and applicable in Novell environment (certainly, there are other Novell based
institutions in Higher Education!)."
```

A comment regarding the National Science Foundation Middleware Initiative (NMI), MACE and this document: MACE effectively steers the Internet2 Middleware Initiative (I2MI). I2MI plays an important role along with EDUCAUSE and SURA as part of the integration and dissemination pieces of the NMI. The NMI came along after MACE and this document pre-existed the MACE and NMI document naming and format conventions. The style of this document clearly does not match the standards set forth by MACE and the NMI. Additionally, the intent of this document, as described above, is to be a set of recommendations and reasonable or best practices regarding directory environments. The author, when exhibiting personal opinion as opposed to some concensus of MACE-Dir or other bodies, will note such personal opinion or attribute opinions to those contributors.

All comments should be directed to mace-dir-comments@internet2.edu.

Contents

DIT: Directory Information Tree
How to design and structure the directory?

The basic structure of your directory is the DIT. As with any tree, there are roots, branches and leaves. Each node in the tree, the root itself, branch points and leaves, is a Distinguished Name, DN. From the root, you define branches of the tree, a.k.a buckets, containers, orgunits, etc. I prefer to the call them buckets. Keep in mind that in a single LDAP directory, you can have multiple roots with different access lists and structures beneath them. There are many considerations for this type of environment which we will not be covering in this document. We simply wish to point out what is possible.

ROOT: The first problem is to define what your root will be for your organization. LDAP is essentially a "simple" version of X.500. X.500, or DAP, used O=,C= naming assuming that there would be some deity in charge of naming and housed centrally. Using this model, we would have used "O=Georgetown University, C=US" for Georgetown. The "O" is Organization and the "C" is Country. This implies a tree structure as well. Someone would be responsible for the C=US, like the US Government? Gee, that's a good idea! The Internet changed this model a bit. It provides a naming convention which can serve the purpose of the central naming authority. This convention using DC= style naming, the DC stands for **domain component** and is part of the **domain** objectclass, really leverages the existing Domain Name System (DNS) as a method for mapping a DNS name to a directory tree or other object or service. This means that for the DNS domain georgetown.edu we would create a root of dc=georgetown,dc=edu. You simply break down each component of the domain name and prefix it with dc= and separate each component with a comma. Why? Well, let's give a short version of the "DC Rant", from Bob Morgan, by example. Imagine I am joe user and I wish to send e-mail to gettes@georgetown.edu, securely, using certificates. To do so, I must have the cert for gettes@georgetown.edu. How do I find it? Well, with DC naming, my mail client would first look in DNS for georgetown.edu and get back the SRV record (see RFC 2782) for the LDAP service associated with that domain name. With this, I now know who to ask. I then break down the georgetown.edu domain into its components and I now know where to look (inside the dc=georgetown,dc=edu directory tree). I then issue an LDAP query against the LDAP SRV record host of ldap.georgetown.edu searching for "`(|(mail=gettes@georgetown.edu)(uid=gettes))`" in dc=georgetown,dc=edu and I get back certificates in the userCertificate attribute (and maybe other attributes).

Where to put all the people? Try someplace like the ou=People bucket beneath the root. Why? Historically, using the X.500 conventions, the structure of the DIT would be made to reflect the structure of the organization. This ultimately causes additional administrative overhead in maintaining the directory since universities tend to be places where people move around or are multiply affiliated. If you try to pigeon hole people, and they move, there is a cost associated with moving them. If you try to create an organizational structure in the DIT of the directory, you may have to delete and add in order to effect a move (if a modrdn function won't suffice). This may impact help-desk support issues as well as infrastructure needed to effect the change when you consider static groups, access lists and the possible need to change application authorizations. This is the "flat and bushy" model as opposed to "tall and spikey". You will need to pay attention to any special configuration parameters in your directory server product to take into account large buckets. For example, in the Netscape/iPlanet/SunONE directory server, the **allidsthreshold** config option in slapd.conf might need tweaking.

Schema Design
What are the attributes that need to be available in a directory?
What is the structure of this data?

Schema design is critical to the planning process. Depending upon the complexity of the directory, you may need lots of planning or little at all. If all you want to provide is a simple white pages service, then, hopefully, this document will meet your needs. Beyond a white pages, you need to pay keen attention to at least the following:

1. the needs of an application?
2. what your goals are for the application?
3. where the data will reside?
4. what will be the system of record?
5. who can change it?
6. how frequently will it change?
7. should it be visible?

It is believed that there is a special relationship amongst institutions of higher education. We share a common set of problems and have a need for some amount of collaboration and data sharing to provide services to one another and to our vendors and contractors. This implies the need for a common schema definition specific to the needs of higher education. An objectclass has been created called **eduPerson**. You should make sure that **eduPerson** is a defined objectclass for each person entry in your directory to enhance the interoperability of your directory. Every institution has local needs, so create a local objectclass with attributes defined by local requirements. The name of the objectclass should use the DC style naming. Example: georgetown.edu should use an objectclass named **georgetownEduPerson**. Each attribute within that objectclass should be prefixed with the string **georgetownEdu** to ensure the uniqueness of that attribute name (the attribute namespace in LDAP is currently flat and I am not aware of any immediate plans to change this).

Schema checking should be turned **on**. With schema checking turned off, if someone has privs for adding attributes to your directory, then there will be no verification of structure of the attribute within objectclass. This can cause you to have not only bad data in the directory, but bad structure as well. Schema checking on does have a minimal impact when adding objects and attributes to the directory, but since your LDAP directory is a mostly read directory and should not be used for intensive writing, this will not be a problem. An implication of schema checking being turned on means you will want to use standard defined objectclasses for LDAP. For people, this means you will need to include at least the following objectclasses for every person entry in your directory.

```
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: eduPerson
objectclass: localdomainEduPerson
```

and Netscape/iPlanet/SunONE DS users will also want to add objectclass: top which holds the aci attribute. This is good and gives you a lot of flexibility for introducing an aci (access list) at any level in your directory. If you do this, then top tends to be the superior to person.

The order shown for the objectclasses above implies a hierarchy. The top objectclass is at the top of the hierarchy (duh). Most other objectclasses that are not intended to be inferior to another defined class should have top as its superior. Advantages might be as Netscape/iPlanet/SunONE uses top where the aci attribute (access control instruction) is defined which means every entry in the directory will be able to have an aci, as defined in this hierarchy. The person class requires that the CN and SN attributes be defined and allows several others. This may appear to be an annoyance so you will want to turn off schema checking to get around this... it is strongly recommended that you do **not**. The organizationalPerson class inherits from the person class. The inetOrgPerson class inherits from organizationalPerson and Person. The eduPerson class, version 1.0, inherits from inetOrgPerson and finally, your localdomainEduPerson class inherits from eduPerson. All of these objectclasses should be defined (the localdomain is optional, of course) to each entry for a person in your directory.

Please note that as of eduPerson version 1.5, that objectclass is defined as AUXILIARY which means it does not necessarily inherit and gives freedom to use eduPerson in whatever manner you see fit. However, when using the eduPerson objectclass in the context of a person, treating it as a descendant of inetOrgPerson as previously described is not a bad idea. (added for LDAP-Recipe NMI Release 2, 2002-10-25)

Now that we are established for using the standard objectclasses, we should discuss the standard attributes to populate for person entries in your directory. In concert with the **eduPerson** effort, the **Directory of Directories for Higher Education** expects the following attributes to be used and configured as described below. The **DoDHE** is but one application that might make use of the **eduPerson** objectclass, but it also goes one step further to try to define and promote reasonable and best practice for schema design, attribute usage and operational aspects like appropriate indexing of attributes. We hope that you will also find that enabling clients like Netscape Communicator and Microsoft Outlook/IE to utilize your directory for address book lookup is easy using these recommended attributes as well.

Before continuing, we need to take a short detour and mention the importance of access control to data in the directory. This is especially important for institutions of higher ed in the United States because of FERPA, the Family Education Rights to Privacy Act. FERPA essentially allows a student to declare that they do not wish to have their personal data exposed. For a deeper description of FERPA (from an institutional perspective) go visit the University of North Texas web site to learn more.

In the past, most institutions implemented FERPA elections by simply not including the student's data in online directories and/or that student would essentially be electronically "persona non grata", and would not receive electronic services. LDAP has access lists to help us overcome the limitations of former directories. Some vendors do a better job on granularity, performance and flexibility of the access control mechanisms. Whichever vendor LDAP directory you use, become intimate with access control. This way, you can maximize the utility of your directory by installing your entire community in it and making visible that which is necessary and allowing services to still be accessible to the FERPA elections. The importance of access control is cited here in order to help avoid a relatively common misperception leading to a common directory design problem. Because of an assumption that LDAP is the "same old directory stuff just done differently", to overcome the sometimes mis-perceived "limitations" of LDAP one is tempted to install all the desired attributes into a single institutional objectclass and not populate the data elements for the FERPA elections. This is not the LDAP way. If you adhere to the practices suggested within this document, eduPerson intelligence and many of the popular books on LDAP, then, as you deploy other LDAP enabled applications and attempt to have your directory interoperate with others, you will find life to be much easier.

Below are some minimal recommendations for attribute definitions, indexing and usage with respect to the **DoDHE**. A more general set of comments and possible discussion exists with the eduPerson effort found at the eduPerson web site. The attributes below come from the the standard objectclasses defined above as well as the new eduPerson objectclass where noted.

person, organizationalPerson, inetOrgPerson attributes for DoDHE

Attribute	**Indexing Recommendations**	**Comments**
CN, CommonName	pres, eq, sub	Common Name from Person, required. See Name Population for further discussion about populating and using this attribute.
SN, SurName	pres, eq, sub	Last Name from Person, required multi-valued for people with multiple surnames (like Gurley-Brown would be 2 SNs with Gurley and Brown). Please see here for further explanation and Name Population.
givenName	pres, eq, sub	First Name. See Name Population.
displayName	Not Indexed	Display purposes only. Preferred name of the person. Example: where CN shows "Robert L Morgan", displayName may show "Bob Morgan". Likely only 1 value.
O, organizationName	pres, eq, sub	Politically correct name of the institution or University system as appropriate. Likely only 1 value.
OU, organizationalUnitName	pres, eq, sub	Department name (or other org entity within the institution). Multi-value as needed.
title	not indexed since the values are so varied.	Job title or other relevant title.
mail	pres, eq, sub	E-mail address, usually of the form netid@university.edu. This is NOT the final delivery address. Likely only 1 value.
uid	pres, eq	Userid, NetID, login. This is the traditional userid we use for computer access to Unix, mainframe and other systems. Many LDAP enabled products expect this attribute. Likely only 1 value.
telephoneNumber	pres, eq, sub	+CC AAA NNN NNNN for North America the CC is "1". Example: +1 202 687 2202 multi-value as needed. See telephoneNumber Formatting.
facsimileTelephoneNumber	pres, eq, sub	+CC AAA NNN NNNN for North America the CC is "1". Example: +1 202 687 2202. See telephoneNumber Formatting.
mobile	Not Indexed	+CC AAA NNN NNNN for North America the CC is "1". Example: +1 202 687 2202. See telephoneNumber Formatting.
pager	Not Indexed	+CC AAA NNN NNNN for North America the CC is "1". Example: +1 202 687 2202. See telephoneNumber Formatting.
userCertificate	Not Indexed, binary. See eduPerson for further detail.	Base-64 encoded certificate. One attribute per certificate.
jpegPhoto	Not Indexed	Base-64 encoded JPEG - usually a small photo of the person. No motion jpeg please!
description	Not Indexed	free form text describing the object or person. for display purposes only. multi-value as needed.

labeleduri	Not Indexed	is a URL associated with the person. Usually this is the link to the person's "home page". Display purposes only, used as a hot-link.

All of the above attributes belong to one of the standard objectclasses: person, organizationalPerson, inetOrgPerson. As long as you have properly included these objectclasses in your person entry, you will not have any schema violations. The LDIF for the eduPerson schema is provided to allow for easy installation and maintenance in LDAP v3 compliant directories, at least those LDAP implementations that allow for schema maintenance via LDAP.

eduPerson attributes applicable to DoDHE

Attribute	Indexing Recommendations	Comments
eduPersonAffiliation	pres, eq, sub	See eduPerson
eduPersonPrimaryAffiliation	pres, eq, sub	See eduPerson
eduPersonNickname	pres, eq, sub	See eduPerson and Name Population
eduPersonOrgDN	Not Indexed	See eduPerson
eduPersonOrgUnitDN	eq	See eduPerson
eduPersonPrimaryOrgUnitDN	eq	See eduPerson
eduPersonPrincipalName	pres, eq, sub	See eduPerson and below

Schema Design and Application Requirements/Expectations

When designing your schema, you obviously have to take into account how applications intend to use the data and which apps will require special attributes. This gets to be a nightmarish problem but can usually be addressed with some perspective.

1. Use existing "standard" attributes if their definitions and use are clear. You will find an attempt of this task in the eduPerson effort which not only defines the eduPerson objectclass but also attempts to clarify the definitions and, more importantly, the uses of the standard attributes. Please refer to eduPerson for this information.
2. Just because an attribute already exists, doesn't mean you should use it. Hey, isn't this contrary to what was just said? Well, not really. The point is that you need to evaluate your understanding of the definition and use of an attribute along with various applications understanding and use. Example: the mail attribute is intended to be used to show a person's email address. Some use this attribute for routing email. If you treat this attribute as it was intended, something for display only, and create your own local attribute (since it doesn't seem that an actual mail routing address was desirable in any of the standard objectclasses) then you will be safe knowing that some new application won't unexpectedly use your local attribute for mail routing and perform tasks you don't want it to perform. Well, I guess this really demonstrates the point of never re-purpose an existing attribute. If you do create local attributes, make sure their definitions and uses are clear and available to application developers.

Schema - bringing it up to level

The following discussion is what is being carried out at Georgetown as of 9/5/2000. It has not yet been fully implemented but testing shows that this methodology is reasonable.

A fundamental issue is how to bring your schema in line with the recommendations of others. You have already deployed your LDAP directory and there are some applications already using custom attributes that do not follow the naming conventions and/or you have finally obtained your OID arc for your institution and wish to bring that up to snuff on your directory as well. Since a few LDAP implementations were based on the University of Michigan LDAP code, we use it as a reference. Local schema changes are applied to slapd.at.conf and slapd.oc.conf in the config directory. They refer to the local attributes and objectclasses respectively. These files are "`include`"ed in the main slapd.conf file. You should maintain your own schema in additional files that are separate from the base at/oc files distributed with your product. In the case of Netscape/iPlanet

/SunONE, slapd.user_at.conf and slapd.user_oc.conf are where user/local schema are held. Anyway, in the attributes config file, the format of the attributes config directive is:

```
attribute attribute-name [ attribute-aliases ] attribute-oid syntax
```

If you have an attribute called guAffiliations and want to change it to be georgetownEduAffiliations, then re-define the attribute as:

```
attribute georgetownEduAffiliations guAffiliations 1.3.6.1.4.1.6868.1.0.15 cis
```

This config line establishes georgetownEduAffiliations as the true name of the attribute (for indexing) and guAffiliations as an alias and the OID is the real OID for Georgetown University and this attribute. To effect the change, take down the directory. Export your directory. Reconfigure the schema as necessary, similar to the example above. Import the directory. The import will cause a re-indexing of the attributes, especially those that have changed names. Now, here are the gotchas is doing this:

- If you regularly export the directory data to something like LDIF for another application to perform mass operations and verifications of the data, then these other applications will have to be fixed to reference the new attribute names and not the old. When the database exports are performed, they will use the primary attribute name, which has now changed.
- Any application which accesses the directory that does not request specific attributes be returned will need to be modified to use the new names. Example: if a program searches the directory for uid=gettes and does not specify which attributes to return, then ALL attributes will be returned (the found entries will be dumped) in the response. But, if an application is requesting specific attributes, then the names specified on the response will be used. Let's show this by example using LDAP URLs:

  ```
  Searching for uid=gettes returning all attributes:
       ldap://ldap.georgetown.edu/dc=georgetown,dc=edu??sub?(uid=gettes)
  yields
       dn: uid=gettes,ou=People,dc=georgetown,dc=edu
       uid: gettes
       georgetownEduAffiliations: Employee
       cn: Michael R Gettes
       ...
  Searching for uid=gettes returning only the guAffiliations attribute:
       ldap://ldap.georgetown.edu/dc=georgetown,dc=edu?guAffiliations?sub?(uid=gettes)
  yields
       dn: uid=gettes,ou=People,dc=georgetown,dc=edu
       guAffiliations: Employee
  ```

  So, "good" applications that request only the data that they are interested in will continue to work without modifications. Those that do not do so, will need to be fixed to either request the specific attributes or use the new names.

The objectclasses using the old attribute names should use the new names. If you are changing the objectclass name, well, you will need to munge the directory data and any automated processes to "do the right thing" by using the new objectclass name.

Password Management

Should you choose to use the directory for authentication then consider the following suggestions.

- Timebomb new or reset passwords. Georgetown added an attribute to the directory called guPwTimeBomb. This attribute used a datetime value (YYYYMMDDHHMMSSZ) to specify the timestamp of when a password would be timebombed. This is used to force removal of passwords when a new user arrives or someone asks to have their password reset, which we set to some known value that "only the user would know" (and we all know that we would *never* use an SSN for this purpose, right?). When a user has the pw reset by the help desk, then a web page creates a timebomb attribute in the directory for now + 3 days. Every morning, 2am, we inspect the directory for all timebomb attributes with a value < the current time. If any are found, we remove the timebomb and the userPassword attribute. This forces the user to re-authenticate themselves to the help desk to have their password reset again. We control the visibility of the timebomb to only priv'd people as the presence of the timebomb is a clue to a bad guy that the password

for this user may be guessable. We also do not allow anyone to obtain the userPassword attribute.
- Protect authentications with SSL. Using code developed by Michael Gettes at Princeton University and building upon that for Georgetown (development continued by the same person), the directory is protected by SSL for non-anonymous authentications. This means that should someone or some process need to authenticate a user, that process **must** be done over SSL or the directory returns "invalid credentials". This was accomplished using the Netscape/iPlanet/SunONE Directory Server and a pre-op plugin. As with any rule, there are always exceptions. This plugin allows for exceptions at either the IP address level or at the DN level by checking for the existance of an attribute.

Bindings

Binding to the directory is the process of authentication. There are essentially two types of standard LDAP bindings:

1. anonymous bindings where no password is supplied and no identity is attributed to the rest of the connection and associated operations. A binding with a DN and the absence of a password is equivalent to an anonymous binding.
2. non-anonymous bindings are where a DN and a password are supplied, the ldap server performs an authentication phase, and associates the DN with all remaining operations on the connection. The DN is evaluated for access control over attribute access and method (read, search, compare, etc...).

Many applications will assume they have access to the userPassword attribute. This means the userPassword attribute would be made visible, not considered a good practice, and that the application will have to know about the various encryption methods of the userPassword value, this too is not considered a good practice. At Georgetown, we have an ACL that makes the userPassword attribute not visible. This forces applications to perform the standard LDAP binding and the LDAP server has the responsibility of verifying the password supplied. This, combined with the SSL protection noted in the previous sections, gives some reasonable amount of protection of the password. The future of the plugin mentioned above will incorporate Kerberos authentication by the LDAP server which unites the LDAP services with the Kerberos credentials and Kerberos based services (like Windows 2000). If you allow access to the userPassword attribute, then you will have to re-engineer every application to perform alternative authentication techniques.

Some applications assume that the DN is of some pattern, like `uid=<user-name>,ou=People,dc=georgetown,dc=edu` and, given a userid and password, will bind directly using a corrected DN/pw pair. The problem is that not all entries may fit this pattern and could be scattered throughout your DIT. Or, should you choose to alter your DIT, then each application will have to be altered to have this knowledge as well. The more proper method of binding is to first perform a search, get back the DN and then bind. The search can be done on various attributes, depending on data supplied, which gives you the flexibility for someone to authenticate using the name, userid, mail address, id number, etc. You would search the directory on the information supplied using a privileged/service DN so it can map the entries hidden by FERPA and other restrictions, return the DN and then perform a normal binding. This provides the most flexibility.

The previous paragraph has some interesting implications and we wish to be clear about them. Not only is the integrity of the data in a directory important, but so is the technique by which we allow applications (and users) to access the directory. FERPA is a big deal for Higher Education, and given the future of things such as PKI and Shibboleth, we need to insure that access to the data in the directory is carefully controlled. This means that for each application which will be accessing non-public data, that application will have to have a *service DN* allocated to it in the directory. The application will then need to first bind to the directory using that service DN and then search and obtain attributes that it requires. This also implies that there will be an access control specification for the service DN to only search and obtain the data the service DN requires. This may seem like a lot of unnecessary overhead and unnecessarily complex, but, given this is how directories are managed, it is what provides us with the most flexible and powerful means to maintain a central repository of data with integrity. Allocating a service DN for each application also limits exposure should the application become compromised. You don't want to create some uber-DN that all applications would be given the password to access the directory. Upon a compromise of any of the applications in this scenario, you would have to then change the password and **all** the apps that use that same uber-DN at the same time. Good luck! If you are concerned that this technique adds too much overhead to the directory and is too costly, experience by many proves this not to be true.

---

eduPerson objectclass discussions

eduPersonPrincipalName

To provide some perspective on the use of this attribute, Paul Hill (MIT) has contributed the following:

```
"The EPPN is intended to be an expedient attribute, useful for building some inter-institutional
applications. It is intended that this attribute will be useful to create some applications that are
based on currently deployed technologies and code that do not currently use LDAP or require a PKI.
This attribute should help to create a framework to create interesting inter-institutional
collaborations between sites that use different technologies. In short, this attribute provides a
foundation for yet another abstraction layer.

It is also expected that this attribute may become deprecated in the future. This would occur as LDAP
enabled infrastructures and applications become more mature. One metric of this maturity will the
convergence of best practices and their consistent deployments."
```

Access Control

This is arguably the hardest part of instantiating and maintaining an LDAP directory. What information shall I load into the directory? Who should have access to it? What applications will require special access? And the questions begin to flow...

At this point I would like to impart what was done at Georgetown. Not so much that Georgetown did it "right", but maybe that implementation will make you think a little more about the capabilities of LDAP and how to make it best work for you and your institution.

First, you need to become familiar with the access control language for your directory. Thus far, I have found the Netscape/iPlanet/SunONE Directory Server to have a powerful environment and is the one feature that was worth the price of admission... this may change over time. The access control list shows how we structured our applications and general access. The perspective behind the access list shows that the first rule is that all attributes are made public. From this point forward, we concentrate on restricting access to those elements that require control based on institutional policy, privacy and application requirements. Students may opt out of being visible in the directory by selecting the "Buckley" restriction as discussed previously. Each application authenticates to the directory using a special DN. That service/DN may be installed in the *ReadOnlyServiceAdmins* group which is allowed to look at almost every attribute in almost every entry. This allows for hidden entries to still receive services. When developing an access control that requires authentication, base it on a group and not a user. This eliminates unnecessary changes to the ACLs when all you need to do is control the membership to a group. There is a special *group management* ACL for Georgetown that allows all those defined by the owner attribute of an entry to manipulate the uniquemember attribute. This ACL allows for the distributed management of the group objects which may control access to other applications, viewing or modifying the directory. Again, keep in mind that your particular LDAP vendor may or may not provide such abilities.

It would be nice to do as Stanford University has done which affords control of visibility to the individual, down to the attribute level (or groups of attributes). Tom Dopirak (CMU) has provided some guidance on providing this ability in OpenLDAP. Stanford University uses the Netscape/iPlanet/SunONE directory server and implemented it in a similar fashion just using a different method.

```
We use Openldap as our directory server and the object hierarchy is the same
as listed in the recipe. There is actually nothing in "top" , the object
type is "abstract". Since Openldap doesn't have acl's on individual
attributes but rather on objects we have had to use the openldap filter
mechanisms to simulate acls.

Each cmuPerson has an attribute called cmuPrivate. The names of the
attributes that have been designated as hidden get placed in this multivalue
attribute by authenticated web applications. The openldap filters will
stop non-administrators from seeing the values of attributes named.

An example filter is

access to filter=cmuPrivate=telephoneNumber attrs=telephoneNumber
  by group="cn=PhoneReaders,ou=Group,dc=cmu,dc=edu" read
  by * none

access to attrs=telephoneNumber
  by * read

If the person has "opted out" of having their phone number shown,
telephoneNumber is set into cmuPrivate and the first access filter applies -
anonymous readers get no access, but those in the PhoneReaders group get
read access. Otherwise everybody gets read access.

Tom Dopirak
```

Tom and I had an interesting discussion about this approach because access control lists don't come for free. With every directory access, the ACLs are inspected. Too many will have negative impact on performance. How many is too many? When does an ACL become too complex such that it impacts performace? It would be nice if the various directory vendors could provide statistics regarding the amount of time spent in various stages of directory processing like authentication, access control, data retrieval, data transmission, database access (read vs. write) and so on to help us better understand the power and limitations of access control lists.

Replication

When creating a directory environment that involves multiple instances of a directory service that has the same data (same or similar) replicated to multiple hosts, there are many things to consider. First, why replicate? There are largely two major reasons for replication, redundancy (failure tolerance, backups) and application performance. The latter obviously depends upon the needs of the application being deployed and its reliance or expected utilization of the directory. For example, Georgetown handles all of its mail routing based on the directory. There are 2 servers used for mail services, one for IMAP and internal mail routing and another server for external mail and inbound (@georgetown.edu) routing. Both servers have replicated directories given their respective heavy uses of the directory as well as needed reliability. If we need to take down one of our primary directory servers, the mail services should continue to function since we all agree how critical 24x7 mail services are today. Another issue is single vs. multiple master replication. The original LDAP implementation from Umich was a single-master implementation -- only 1 directory could be written and the rest were simply read-only instances of that master. Attempts to write to the read-only replica would generate an error or a referral to the master (depending on which version of the protocol you are using). A multiple-master environment, found in, at least, Active Directory, NDS 8 and above, and iPlanet DS 5 and above, allows for multiple instances that can be written so there is no longer a single write point of failure. Various complex algorithms are used to handle inconsistencies and repair to keep the directories in sync. So, let's exit this rat-hole by pointing out that there are many issues involved in replication and single vs. multiple master environments which may impact your decision to have multiple roots in a single directory or split out each root to be contained by a separate directory instance. When you begin to consider how you intend to manage the data, who will have access to the data and server and how often the data will be written, this should become more obvious. Likely, you will only need 1 tree of data and therefore life will likely be simpler -- we hope.

Groups, Groups, Groups

STOP! Before you go on to read about Groups you really should get the necessary background. Tom Barton of University of Memphis and others have spent a significant amount of time understanding and appreciating the problems presented by Groups and we all believe you will benefit greatly by reading this document first. It wasn't written for the helluvit! So, go read and come back here later.

Okay, now that you have read that document I hope you now appreciate the work of Tom Barton and the other folks involved in the MACE-Dir working group. What you will hopefully realize is that document really doesn't say how to implement groups. Well, you're right. But, it does give you the necessary framework in which to consider, think, discuss and augment your environment in support of Groups. What we can now do is consider some implementations, or those that are soon to be implementations (kinda-sorta vapor-ware, but with good intentions). The following does not necessarily represent a best practice or consensus considering that not much work has been done to group enable directories to the extent considered below (at least none that we are as of yet aware).

Here is what we intend to do at Georgetown University as of this writing. We will implement static group objects in our iPlanet directory **and** forward referencing dynamic groups. iPlanet likes to utilize the *groupOfUniqueNames* objectclass which calls for the *uniqueMember* attribute for static groups, so that is what we will use. Applications that will be static group aware should provide config options to allow you to specify the objectclass name identifying a group object and attribute name used to enumerate the membership of the group. Beat on your vendors if they don't. OpenLDAP likes to use the *groupOfNames* objectclass which calls for the *member* attribute.

Why do both static groups and forward referencing groups? 'Cuz from what we can tell, there will be some apps that want static groups and others that will want dynamic. For now, we will provide the maximal solution and learn what is best. Our hope for the longer term is that we will graduate to only static objects and eliminate the forward referencing when confidence is high enough that the industry has learned how to handle groups properly and it is safe to do so -- some magic 8-ball will predict this for us. We do not intend to use the *isMemberOf* attribute but to hide this functionality in our own attribute called *georgetownEduPersonGroup*. This attribute will not be publicly visible. Since the ultimate goal is to not have this attribute in the future, then controlling access to it is wise. This also prevents applications that "see" this attribute from using it for the app's own purpose, thereby losing control of the behavior of the app. Along the same lines, we will put all the static group objects, the one's maintained by our identity management system for the institutional groups, into the *ou=Groups* branch of the DIT and control access to that branch. Only those apps with sufficient need to know and privs will have access to Group information since this can be juicy information for privacy and security hackers. At Georgetown, we currently guess-timate that we will have approximately 10,000 groups automatically defined and maintained based on data from our core business systems. We are trying to define a configuration specification (language) which will be fed to an engine that will generate groups from the data in the core business systems (more on this later). Users will be able to create personal groups (static only) which will be stored beneath their object in the DIT. So, for a DN of *uid=gettes, ou=People, dc=georgetown, dc=edu* we would create a personal group of *MyFriends* with a DN of *cn=MyFriends, ou=Groups, uid=gettes, ou=People, dc=georgetown, dc=edu*.

Indexing: we will be indexing *georgetownEduPersonGroup* with EQuality only and *uniqueMember* will be indexed EQuality only as well.

Group names presents an interesting problem. The name of the group will be directly related to the data values used to create the group. This is where the configuration language plays an important role in bringing all of this together. Imagine a specification that says HR(field1) gets you the value of field1 from the HR system for the current record. Also imagine being able to specify transformation routines on the data as well as static strings. Putting all this together allows for the creation of groups with names like *faculty-dept-economics-status-emeritus* or *dept-economics-student-class-senior-gender-female*. Hopefully the names are self-evident for this example and hopefully you can see how static strings and data for the core business system will play a role. Transformations are a fact of life here; *gender-female* may be represented as a value of 2 in a gender column in some table. The transformation may convert that value in that context to the string *female*. So, assuming such a language exists and an engine to process it, we have a naming scheme for groups which generates a multiply rooted tree, arbitrarily defined according to the needs of the University or an application. These values will be used as the *cn* value for the *groupOfUniqueNames* objects (requires that this *cn* be the *RDN* value for the *DN* of the group object). This same value would be used to populate the *georgetownEduPersonGroup* attribute for each object that is a member of the group (the forward reference). Hopefully, this now explains how we intend to name our groups and how those names will be used in forming DNs and populating appropriate attributes. For Georgetown there has been a bit of consideration regarding the creation of special attributes for group names and then the associated annoyances of maintaining local attributes that are not well understood or maintained by other software. So, the current feeling is to use the attributes of the *groupOfUnqiueNames* objectclass which essentially includes *cn* and *uniqueMember* which keeps us a relatively vanilla implementation. Given that the *cn* attribute is being used, and has the unfortunate side effect of increasing the index, we need to remember to make sure that groups are not included as part of white page searches in a people context. However, since *cn* is indexed, we can now provide some nice interfaces for searching of group objects in the directory and even limit the scope of the search based on the DIT.

Thus far we have considered personal groups and automatically managed institutional groups. Another class of groups are the ad-hoc institutional groups, the groups that are not defined by data from a core business system and are developed from divine knowledge and result in groups like "the important people" and "people who can make a decision". These are groups that you will likely want to put in a special branch of the DIT labelled something like *ou=Special Groups*. You will undoubtedly not want to be in the business of maintaining these beasties, especially as some may be politically sensitive. This implies you will need some form of delegation for the management of these groups. It is probably a good idea to only allow certain priv'd folk to add/delete group objects from this part of the DIT (yup, this may mean more policy). But, once the priv'd folk can add a static group object the *Special Groups* they should be able to assign membership control to one or more people and/or one or more groups. These groups should not be publicly visible. The iPlanet directory affords an interesting capability in the *aci* specification for access control to help achieve this functionality. They support *userdnattr* and *groupdnattr* as part of the access control spec as shown here:

```
    # group management can be handled by the owners of the group
    #
    aci: (targetattr = "uniquemember")
     (version 3.0; acl "Group Membership by owners";
       allow (write)
        userdnattr = "owner"
       or
        groupdnattr = "owner"
       ;
     )
```

The above stipulates that the owner attribute, which may contain either group or user DN references, defines those who are allowed to write the *uniqueMember* attribute affecting the membership of the group. This is delegation. Georgetown uses the *Directory Services GateWay (DSGW)* product which came with earlier releases of iPlanet directory (version 4) which has a nice web interface for providing this capability. Obviously, you will need to "role" your own or use existing software to perform this function (Software).

A logical question at this point would be "So, if I do all this work, what does it get me?". A gold star. Seriously, as Tom Barton's groups document should help you realize, establishing groups in your enterprise directory is the basis for enabling group controlled access to a plethora of resources. The logical thing to do is to make sure your web servers are LDAP enabled so that you can then include web access control statements that allow/deny access only to specific groups. Have you ever wanted to make web pages available only to the entire campus community and not to the rest of the world? This is just the beginning. Will you necessarily need to know and understand ahead of time all the possible uses of groups? Probably not. Why try to do so since it is so much work? Just provide the enabling technology and see what happens. Ok, enough pontification, let's move on... With groups in the enterprise directory, especially static groups, you should be able to perform a meta-directory function to reflect these same groups into other directory systems such as Active Directory.   This perspective shows that you concentrate on the management of one directory and then translate that information to other systems. The "directory" in this sense could be a person registry, enterprise LDAP directory or what have you. For a discussion of meta-directory techniques, we refer you to a paper by Richard Jones (retired), Brendan Bellina of the University of Notre Dame, and others which attempts to describe the issues and best practices regarding these techniques. Groups can be applied to systems such as the Corporate Time calendar system (when you configure it for LDAP use); for automatic population of electronic mailing lists (see next paragraph on how groups should NOT be used); as simple authorization material for access to business systems that are LDAP enabled such as Cognos and Brio; as access control input to RADIUS+LDAP enabled systems; and insert your LDAP+groups aware application here.

Where groups should not be used; **Warning!** deep personal beliefs of the author here. You can do as you wish. What some are inclined to do is similar to what the iPlanet Messaging Server does, to enable LDAP groups to be used as mailing lists without the use of mailing list software such as LISTSERV, ListProc, majordomo and so on. I think this is a mistake. iPlanet ended up putting in all sorts of additional attributes to include similar behavior as mailing list systems, but, alas, their iMS product is not a Mailing List Manager (MLM). What I would prefer is to have LDAP enabled MLMs. Some products now have this feature such as the freebie www.sympa.org and a company that provides the MLM Puppeteer. ListProc and I believe LISTSERV still have a long way to go in this area. If you have LDAP challenged MLMs, then consider automagically populating the subscribers of a list based on LDAP groups. This implies your writing (or getting from the net if it exists) code that will keep MLM lists in synchronization with LDAP groups on some regular basis for automated administration until such time that MLMs get the LDAP fever. One might also envision the MLM being enabled for personal groups such that email sent to "gettes+MyFriends"@Georgetown.EDU would be directed to the MLM and distributed using some reasonable user-oriented list configuration for all such lists... but this is definitely a future item and falls under the heading of "gee, wouldn't it be nice?".

A technique worth noting regarding groups being visible to certain applications is what Georgetown did to make LDAP groups available to the CorporateTime calendar system. Steltor, the vendor of CorporateTime, is considered to be a good LDAP enabled application because they expose their LDAP filters in the configuration of the service. In particular, the search filter used for groups is exposed and usually looks something like *(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))*. We modified the filter to be *(&(objectclass=guCTgroup)(objectclass=groupOfUniqueNames))*. The *guCTgroup* objectclass has no attributes. This allowed for the addition of just this objectclass attribute to static group objects and make them visible in the directory so that those that should not be visible (and be confusing to our users) are not seen.

Personal groups will likely be used for web services. You create your own personal group and then make your web space available to just that group as you see fit. This probably implies users having to configure Apache htaccess files or something similar and hopefully there is a web interface to help users configure their own access control without having to worry about the ugly details of such configuration options with a nice point and click interface. You could also imagine how centrally managed personal groups propagated to other environments would be useful even if the environment is not LDAP enabled. The directory becomes a reasonable mechanism for centralized distribution of group information to other systems.

Access Control: Groups, the static objects, and attributes that would implement forward reference to groups, for Georgetown this would be the *georgetownEduGroup* attribute, should not be publicly visible. Georgetown intends to keep the institutionally maintained groups private and accessible only to those apps that require access and the same will apply for personal groups and the ad-hoc groups. There will, as with any rule there are exceptions, be those groups you wish to be publicly known and they will likely reside in a special branch of the DIT to allow access. The *ou=Groups* branch of the DIT will likely be secured along with the *ou=Groups* branch that would exist beneath user entries which will be handled by wild-card access control specifications. Personal Groups are intended to remain personal and not to be publicly visible since the simple fact that a publicly visible group will become an ad-hoc institutional group since it will be shared. Groups should be kept private because much can be learned about a person based on their group memberships, from both the personal/privacy and security perspective.

E-Mail Routing

(This section contributed by Tom Barton, University of Memphis)

Many institutions support reception of email addressed to “someID@some.edu” by a variety of means. An LDAP directory can also be used in conjunction with popular SMTP implementations to route such mail towards its final destination. If you’re going to implement an LDAP directory, it is worthwhile to add this to the set of applications managed by a common technology – your directory.

The recommendations in this section stem from experience with sendmail v8.10 and later and iPlanet’s Messaging Server (iMS), and are also influenced by the archived email proceedings of the IETF LASER (LdAp Schema for Email Routing) BOF, even though that group never made it to formal Working Group status.

An LDAP enabled mail transfer agent that receives email for the some.edu domain has two main objectives to fulfill: *Identification*, which is the determination of which LDAP entry is associated with the recipient; and *Routing*, the determination of what next step to take to get the message to the recipient’s mailbox. Identification requires that the directory schema contains attribute(s) whose values are designed to be searched using the entire recipient address or perhaps some part thereof. Once a unique directory entry corresponding to the recipient is identified, the values of a different set of attributes in that entry are used to determine how to route the message.

The *mail* attribute in the IETF’s *inetOrgPerson* objectclass can be used for the Identification task, but it is also commonly used to display a single, preferred, email address. Hence, it’s best to leave the *mail* attribute single-valued and use a different, multivalued, attribute to support the ability to have multiple email addresses identified with the same LDAP entry. Identification is carried out by looking for an exact match in the set of email addresses for each entry. If that search fails to find any matching entries, sendmail necessarily performs and iMS makes available as an option a second LDAP search in which only the domain part of the recipient address (i.e., “some.edu”) is searched for, permitting one LDAP entry to be used to route all email addressed to some.edu for which no specific recipient can be identified.

There is no clear best multivalued attribute to use, in addition to *mail*, to hold an LDAP entry’s multiple email addresses. iMS necessarily uses *mailAlternateAddress*, so if you now run or are planning to run iMS (or plan to never run iMS) it may be reasonable to populate this attribute. On the other hand, if you want to be protected from the possibility that a future version of iMS or another product might use this attribute differently, then you can define and populate an attribute, perhaps called *someEduMailAlternateAddress*, in your *someEduPerson* objectclass. In this case you must not rely on any mail routing product that does not permit you to configure which attributes are used for which purposes.

Having identified an LDAP entry, the next task, routing, is to determine which host to connect to in order to get the message closer to its final destination and what envelope recipient address the message should be given. Two single-valued attributes hold these values. iMS uses *mailHost* and *mailRoutingAddress*, or you can create your own to remain independent of whatever iMS may do with these attributes. The detailed semantics for these two attributes are the same for both iMS and sendmail. The *mailHost* attribute, if present in the LDAP entry, contains the fully-qualified domain name of the host to which the message should be relayed. The *mailRoutingAddress* attribute, if present, contains an email address which is used to rewrite the envelope recipient when the message is relayed. Either one, or both, of these attributes may be present in an LDAP entry. If *mailHost* is present, it determines where the message will be relayed. If *mailRoutingAddress* is also present, the SMTP/RFC821 envelope recipient specified with the relayed message will be the *mailRoutingAddress* value; otherwise the message is relayed to *mailHost* with the original SMTP/RFC821 envelope recipient. If *mailHost* is absent, the domain part of *mailRoutingAddress* is used to determine where the message will be relayed. The relayed message will have its SMTP/RFC821 envelope recipient address rewritten. If *mailHost* is a "local" host, i.e., one for which local delivery may be attempted, and if *mailRoutingAddress* is present, then delivery to *mailRoutingAddress* is attempted directly.

There is an interdependence between DNS, configuration of specific mail transfer agents, and envelope recipient addresses that must be understood in context before determining how to populate *mailHost* and *mailRoutingAddress*. One general guideline can be expressed using the concept of an "LDAP email routing domain". This consists of all mail transfer agents that use the same set of LDAP searches against the same directory for the same set of domains to make email handling decisions. The guideline is:

> LDAP entries for mailboxes residing on a host inside the LDAP email routing domain should only have *mailHost* populated. Other entries should only have *mailRoutingAddress* populated.

With this guideline, it is unnecessary to ever have both attributes populated. Some examples of its use are given at the end of this section.

The indexing required to support LDAP email routing is easy to specify. The types of LDAP search filters to be supported for identification are always of the form (attribute=recipientAddress), so equality indexing is all that's needed. Presence and substring indices are irrelevent to this application, and presumably presence indexing will effectively be a null index for these attributes anyway since a large percentage of your directory entries will have these attributes present. No indexing is needed for routing related attributes since they are not the target of any search.

The following table summarizes the attributes supporting LDAP email routing. With the exception of *mail*, other attribute names might be selected for your implementation.

Attribute	Indexing Recommendations	Comments
mail	eq	Preferred or displayed email address with rfc822mailbox syntax. Populate with a single value.
mailAlternateAddress	eq	Multi-valued set of rfc822mailboxes by which this entry is known, in addition to the mail attribute.
mailHost	none	Hostname where entry's mailbox resides (or hostname of SMTP gateway to the mailbox host), provided that the mailbox host (or gateway) lies within the organization's LDAP routing domain. Single-valued.
mailRoutingAddress	none	Single rfc822mailbox identifying the mailbox for this entry, provided that the mailbox host (or its SMTP gateway) lies outside of the organization's LDAP routing domain.

Following are descriptions of two different implementations of LDAP email routing that follow the recommendations of this section. The first is from the University of Memphis, which uses a mixture of sendmail and iMS based email routing within a single LDAP email routing domain. The second is from Georgetown, which uses sendmail almost exclusively for mail routing with Georgetown-specific attributes. Georgetown also uses iMS, but only to implement mailbox access protocols – they had to do some special construction to avoid use of iMS's smtpd module. (Note: The Georgetown model was originally developed by Princeton University although the same people were involved).

Example 1. The University of Memphis

Memphis uses iMS for mailbox access. The installation process extended the directory schema to include several new mail related objectclasses and attributes, including *mailHost*, *mailAlternateAddress*, and *mailRoutingAddress*. They elected to use these attributes for both its iMS smtpd process and its sendmail v8.11.1 instances. The guideline described above is followed when populating *mailHost* and *mailRoutingAddress*.

The LDAP configuration section of a production sendmail instance is:

```
FEATURE(`ldap_routing',
    ldap –1
      -v mailhost
      -k (|(mailAlternateAddress=%0)(mail=%0)),
    ldap -1
      -v mailroutingaddress
      –k (|(mailAlternateAddress=%0)(mail=%0)),
    `passthru'
  )
```

```
  define(`confLDAP_DEFAULT_SPEC', `
    -b "o=the university of memphis,st=tn,c=us"
    -h ds3.memphis.edu
    -p 389
    -d "uid=svcdn-rmta2,ou=special users,o=the university of
        memphis,st=tn,c=us"
    -P /etc/mail/v8/offtoworkwego/svcdn-rmta2
    -M simple
  ')
```

The two maps defined by the "ldap" clauses of the FEATURE definition implement the identification task. The returned values of *mailHost* and *mailRoutingAddress* then are used by sendmail (and iMS's smtpd) as explained above to determine how to handle the message. Some of the declarations for `confLDAP_DEFAULT_SPEC` are incompletely documented with the sendmail distribution. The `-P` argument specifies the location of a file which contains the authentication credential used to BIND as the user specified by the `–d` argument. The `–M` argument indicates the authentication method to be used.

A sample LDAP entry's mail related attributes:

```
objectclass: mailRecipient
uid: jdoe1
mailhost: postoffice.memphis.edu
mail: jdoe1@memphis.edu
mailalternateaddress: jdoe1@postoffice.memphis.edu
mailalternateaddress: jdoe1@cc.memphis.edu
mailalternateaddress: jdoe1@mocha.memphis.edu
mailalternateaddress: j-doe1@memphis.edu
mailalternateaddress: doej@memphis.edu
```

The *mailRecipient* objectclass was created during iMS installation. An email with recipient equal to any of the addresses above will be relayed to *postoffice.memphis.edu* with its original envelope recipient address. That host is in the same LDAP routing domain as the mail exchangers which will have relayed a message to it, so when it receives the email it will perform the same identification and routing steps, learning that the message should be delivered locally to the mailbox for jdoe1. The *mailAlternateAddress* values represent legacy email addresses that continue to be deliverable as well as an alternate form of address for the mailbox itself.

Another example illustrates delivery to a mailbox not in the LDAP routing domain:

```
objectclass: mailRecipient
uid: jdoe2
mail: yodaddy35@hotmail.com
mailalternateaddress: jdoe2@memphis.edu
mailalternateaddress: j-doe2@memphis.edu
mailroutingaddress: yodaddy35@hotmail.com
```

This entry is typical of individuals who prefer to maintain external mailboxes for their email, and not have to deal with a University-supplied mailbox as well. The *mailAlternateAddress* values permit sending mail to this person with either a standard or a legacy style memphis.edu address.

Example 2. Georgetown University

Georgetown also uses iMS for (some of) its mailbox hosts, but relies almost exclusively on sendmail for mail routing. They also elected to use their own attributes for the identification and routing tasks, together with those needed to support iMS for local delivery. These are:

- *mailHost* for sake of iMS local delivery

- sendmail uses *mail* and *uid* for identification
- sendmail uses *mailHost* and *guEmailbox* (both single valued) for relaying and recipient address rewriting, respectively. *guEmailboxAlternate* is also used to allow for delivery to additional addresses but the primary address *guEmailbox* is always expected to have a value.

The LDAP configuration section of a production sendmail instance is:

```
# Emailbox lookup in LDAP
Kemailbox ldap -q
    -k "(|(mail=%s)(uid=%s))",
    -v "guemailbox,guemailboxalternate",
    -b "dc=georgetown,dc=edu",
    -h "mailhost.georgetown.edu postoffice.georgetown.edu",
    -M LDAP_AUTH_SIMPLE,
    -d "uid=SendmailRoutingAdmin,ou=Specials,dc=georgetown,dc=edu",
    -P "/var/local/somewhereovertherainbow/mail/secret"
```

A sample LDAP entry's mail related attributes:

```
objectclass: mailRecipient
objectclass: guPerson
mail: gettes@georgetown.edu
mailhost: imap.georgetown.edu
guemailbox: gettes@imap.georgetown.edu
guemailboxalternate: gettes@earthlink.net
```

An email for *gettes* will be relayed to imap.georgetown.edu with the recipient address rewritten using the *guEmailbox* value and to earthlink.net rewritten using the *guEmailboxAlternate* value.

Comments from gettes@georgetown.edu on the above specification: Note the -h parameter specifying failover for directory operations. This is the config for mailhost which would failover to postoffice should the local LDAP service be down. The -b argument specifies the entire directory tree to be searched and not just for people. Some sites specify *ou=People,dc=university,dc=edu* and forget to route for non-people addresses. With this configuration, Georgetown does not have to worry about some application that is LDAP aware and also mail aware attempting to utilize any of the standard attributes as described above in ways not intended by Georgetown. This method of attribute firewalling protects the precious mail routing functions from ill or mis-behaving applications and keeps the responsibility for this processing in one place, safe and secure.

Part of the original design of the mail services for Georgetown was to provide services to departments that are currently performing their own mail services. To achieve this, we would need to appear to be their service. So, if Physics, who operates physics.Georgetown.EDU wanted the central IT organization to provide their mail services (routing and receipt), then we would want to do so using our existing infrastructure and preserve their published email addresses for physics.Georgetown.EDU. So, the plan was to support *guEmailRewrite*. This attribute would contain the address of the name that mail could be sent to or from and would be mapped accordingly. Imagine the above LDAP entry also having *guemailrewrite: gettes@physics.georgetown.edu*. We define physics.georgetown.edu in DNS as having an MX record pointing to postoffice.georgetown.edu. E-mail then sent to gettes@georgetown.edu (in the From: or To: fields of the rfc822 headers) would be rewritten with the value of *guEmailRewrite* to gettes@physics.georgetown.edu. Also, any mail sent to gettes@physics.georgetown.edu would be directed to postoffice and it would route gettes@physics.georgetown.edu according to the *guEmailbox** attributes. This is essentially email masquerading based on directory attributes and using local attributes for the above mentioned reasons of firewalling the attributes.

Active Directory Integration

No, sorry, we are not going to get into detail about this topic. Please do not get the idea that this document is anti-AD. Not the case. AD is a complex environment as it tightly integrates with many of the features of a modern Windows world. We recognize that there are sites that will be utilizing AD as their enterprise directory and this notion of integrating into some larger environment is not viable for them. We hope that this document may serve as a reference to such sites as to what those of us who are not using AD as the enterprise directory are doing in the hopes of achieving success in interoperability.

At issue: how to integrate the Windows 2000 (and above) environment into the enterprise infrastructure without having it consume the enterprise infrastructure. We need to provide full support for this environment but also remain independent with respect to enterprise authentication and directory services. MIT has a project called Pismere and CMU has one called Orpheus. Many other schools have achieved similar results in essentially the same way as MIT and CMU. It might also be helpful to see a picture of the Georgetown environment. Hopefully, this enough to get you going. It is a non-trivial, but worthwhile, undertaking.

---

eduPersonOrgDN and eduPerson{Primary}OrgUnitDN

Here we will consider the original intent behind these attributes as defined in the *eduPerson* objectclass specification. As has been discussed elsewhere in this document, a flat DIT structure is recommended for representing your people. You may, of course, utilize branches of the DIT (buckets), for other purposes, but given the multiple affiliation nature of people within Universities and their migratory behavior, mapping people to organizational structure as defined by the DIT does not seem to make sense. Putting people in the engineering school into a branch called *ou=Engineering* and bio-med types into *ou=BioMedical* doesn't make sense when there is a reasonable degree of cross-pollination yielding BioMedical Engineering (I couldn't think of a better example, sorry). If you buy into this line of thinking, then you will likely put everyone into one branch called something like *ou=People*.

"But, but, but... I need to show some form of organizational structure in my directory, how do I do that?" you ask. Good question! In steps *eduPersonOrgDN* and *eduPersonOrgUnitDN*. First and foremost, these are DNs, not LDAP URLs. Just DNs. The intent is to provide a reference (pointer) from a person's entry (likely a person) containing the eduPerson objectclass to an object intended to represent either an Organization, usually the institution itself, or an Organizational Unit (department).

eduPersonOrgDN is intended to reference a single organization as represented by at least the objectclass *organization*. It is conceivable that a single directory can have multiple organizations defined within it. It is likely that the organization object represents the institution. So, for Georgetown University, the *dc=georgetown,dc=edu* object would have the objectclass *organization* defined and most, if not all, people entries would be populated with *eduPersonOrgDN: dc=georgetown,dc=edu*. Web pages and LDAP clients would provide a pointer to the referenced DN and selecting it would yield the referenced object. This attribute is defined as SINGLE-VALUE so there can be only one per entry. Given that the DN is ugly looking, we could always create a new object with a DN of *o=Georgetown University,dc=georgetown,dc=edu* and make that the offical OrgDN object. See the notes associated with this attribute in the eduPerson specification to see what attributes are available and other recommendations.

eduPersonOrgUnitDN is intended to reference objects which include at least the *organizationalUnit* objectclass. These objects will likely represent departmental thingies (yes, that's a technical term). This attribute is defined as multi-valued so having more than one per entry is possible and desirable as it addresses the issue of multiple affiliations for an individual. This attribute should be indexed for equality so that you can provide for searches showing the membership of a department. When you add departmental objects to your directory, you will probably want to have the RDN (the Relative DN, the left-most portion) be something like *cn=Engineering* so that it looks nice when presented by an LDAP client or web interface. See the notes associated with this attribute in the eduPerson specification to see what attributes are available and other recommendations.

eduPersonPrimaryOrgUnitDN was added for eduPerson 1.5 as a suggestion from Mathias Meisfjordskar of the University of Oslo, Norway. For this attribute, consider everything associated with *eduPersonOrgUnitDN* in the way of indexing and usage except that this value will be SINGLE-VALUEd, which should be self-evident by the name. Can your institution politically allow for, and are you able to discern within your registry, the notion of a PrimaryOrgUnit for a person? If so, the value of this attribute should be a DN of an object containing at least the *organizationalUnit* objectclass. Many thanks for suggesting this attribute.

Relative Distinguished Names (RDN) and associated issues

Did you know that DNs are really supposed to be composed of attributes defined by OID rather than by attribute name? Did you know that there are really only a small set of generally accepted attributes (with known OIDs) that you are supposed to use in creating your DNs? Hmmm, well, this seems to be a common problem that not everyone knows this little tid-bit.

So, as an example, one way we can write the DN for *uid=gettes,ou=People,dc=georgetown,dc=edu* is to write it as

0.9.2342.19200300.100.1.1=gettes,2.5.4.11=People,0.9.2342.19200300.100.1.25=georgetown,0.9.2342.19200300.100.1.25=edu. Now, that is very ugly, isn't it! The IETF has a working group called ldap-bis which aims to develop standards that revise and correct ambiguities in the LDAPv3 "core" specification. There is a document dealing with the "*String Representation of Distinguished Names*" which describes how to translate the names used in DNs. While we suggest you read this document and keep abreast of its progress, the up-shot of this issue is that only the following are considered to be "well-known" attribute names associated with "well-known" OIDs (the following table extracted from the cited document above):

```
String  X.500 AttributeType
------  --------------------------------------------
CN      commonName (2.5.4.3)
L       localityName (2.5.4.7)
ST      stateOrProvinceName (2.5.4.8)
O       organizationName (2.5.4.10)
OU      organizationalUnitName (2.5.4.11)
C       countryName (2.5.4.6)
STREET  streetAddress (2.5.4.9)
DC      domainComponent (0.9.2342.19200300.100.1.25)
UID     userId (0.9.2342.19200300.100.1.1)
```

As you can see, the example DN used above is, fortunately, legal according to this Standards Track Internet Draft.

So, did you use only these attribute names when you created your DNs? Well, don't panic yet. As of this writing (note the changelog), very little software, actually enforces this DN component issue. Nonetheless, this may be an issue in the future. So, the intent here is to make you aware of this issue. However, a suggestion to the IETF ldap-bis working group might be to consider the creation of an attribute *dnComp* or *dnComponent* to be added to the "well-known" list, which is defined to be SYNTAX=Directory String and has no limits on the values. These attributes would be used to make up a DN as desired by the DIT architect and the intended use of the attribute is that it explicitly has no use except to create a DN.

So, the above example DN could be rewritten as:
dnComp=23472987348, ou=People, dc=georgetown, dc=edu or
dnComp=gettes, ou=People, dc=georgetown, dc=edu or
dnComp=gettes, dnComp=People, dc=georgetown, dc=edu.

We have no idea if this is going to be accepted by the IETF ldap-bis group but we do intend to work with them on this issue. Our thanks to the MACE-Dir Technical Advisory Board for valuable information and input to this issue.

Name Population

How should the CommonName (CN) attribute (and friends) be populated? Many sites tend to figure that because clients behave in various ways and that is out of central control then it is therefore necesary to populate the CN attribute with every possible contortion of the person's name: "First M Last", "First Last", "Last, First M", "Last First", "Nickname Last", "Last, Nickname", etc. The practical experience seen at both Princeton and Georgetown universities seems to show that this type of population is not necessary. The predominant white-pages capability is handled by a set of web pages centrally developed by an institution. The myriad of clients tend to be E-mail clients which are looking for specific attributes. So as to avoid the population of CN with every possibility, consider using the attributes how they were intended along with the software capabilities that should be available either with the server or client software you have purchased or obtained. Let's take the following example LDAP entry:

```
dn: uid=gettes,ou=people,dc=georgetown,dc=edu
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: eduPerson
uid: gettes
cn: Michael R Gettes
givenName: Michael
sn: Gettes
eduPersonNickname: Mike
```

We will presume the above attributes are indexed as recommended. To provide an insitutional white pages service via the web, I can utilize the OpenLDAP software for performing LDAP searches. To provide an effective white pages service, hopefully your vendor supplied software has a utility that makes use of the ldapfilter configuration file capability similar to that which comes with OpenLDAP. If not, then either you should develop one, find one or let your vendor know that this is a core capability you require. What does this essentially do? It will allow you to develop custom search filters based on the search request to optimize your directory searches and make best use of the data in the directory. We have provided an example of the ldapfilter.conf file taken from Georgetown University's implementation. An example scenario, given the data above, might entail someone entering in "Michael Gettes" as the search terms on some web page. That web page would call a program that searches the directory which utilizes this filter configuration capability. The filter configuration, by pattern matching, detects a two word request and breaks into two components and issues a search on givenName using the first word and surName for the second word. This same configuration also detects "Gettes, Michael" and reverses the words to issue the same search filter. Notice this means you don't have to populate the commonName with every possible permutation of the name. Yes, not all clients behave this way, but you will likely find that how other clients behave is actually quite appropriate for that client. You will want to obtain maximum flexibility, configurability and maintainability for your institutional white pages service.

Further discussion with Bob Fowles (Penn State) has led to the following perspective about nicknames:

```
In ph a standard field is nickname and when ph does a lookup on a
name it actually does a combined lookup on name and nickname.

LDAP of course does not yet have a standard field for a person's
nickname (or nicknames). The attribute eduPersonNickname will help
solve the problem for educational institutions for those institutions
that use eduPerson but it will take a long time for popular
ldap-enabled apps such as Eudora Pro, Outlook (and Outlook Express),
Netscape, and pine to implement search-filters that use
eduPersonNickname. With the exception of Eudora Pro, these apps
cannot be configured to use a different search-filter other than the
one it comes wired with.

If these common apps used givenName consistently and predictably,
that multi-valued attribute could be populated with a person's
nicknames. At this time, the only attribute that is universally used
for name lookup is cn. Therefore at Penn State we are currently
planning to populate cn with multiple values, each additional value
consisting of a nickname (Bob, Mike, etc) with the lastname. E.g., my
cn attribute will have the values "Bob Fowles" and "Robert Fowles".

This is the only solution of the nickname problem that I can think of
that permits any current ldap-enabled app to query our ldap server
and find someone by their nickname.

Bob
```

Add to this the use of the displayName attribute, engineering applications to show the displayName attribute if it is populated or show, maybe, the longest (in bytes) CN attribute if displayName is not populated. At Georgetown, we intend to use this model. So, this perspective combined with the advice above yields a very powerful environment that should solve a multitude of problems. The only real concern becomes what a particular application will display as the name, commonName (CN) or displayName.

The following are notes from RL "Bob" Morgan at University of Washington (formerly of Stanford) giving his thoughts on how the SN attribute should be populated and searched. This description is referenced from the SN attribute comments in the matrix for the DoDHE.

Subject:	some text on eduperson 'sn' attribute recommendation
From:	"RL 'Bob' Morgan" <rlmorgan@washington.edu>

I had started a message on this topic based on our phone discussion from a few days ago and now have sort of lost the proposed content, but I'm sure if I'm in left field someone will correct. I think the idea was a suggestion for the 'sn' (surname) attribute that would deal with multipart surnames, based on how it's done in the Stanford implementation. So here goes.

Part of the design here has to do with how fields are searched. If, with every input of "foo", the search filter is written as sn=*foo*, ie a substring match, then obviously you only have to have a single instance of any surname. IMHO, though, users do often want to search on "morgan" and get only the Morgan entries, not Morganstern and Morganoff; ie, if they want "morgan*" behavior they should ask for it. And in these cases a person with a multipart last name like "Morgan-Perara" very often *does* want to be findable via a "morgan" search. Hence these people in essence have multiple "sn"s for search purposes, not unlike people who have had a change of surname.

Whether to populate these automatically or oblige the user to enter them (those other than the registered surname, that is) is a site choice, but I suggest that it's a common enough desire that doing it automatically is preferable (especially if you don't have a good end-user dir-info maintenance UI). In this case you want to filter out some set of little bitty surname components: "de", "st", "la", "van", etc, so as to avoid confusion.

But the values you end up with for "`Morgan-Perara`" are:

```
Morgan-Perara
Morgan
Perara
```

For "`Ruiz Alvarez Sanchez`" you might end up with:

```
Ruiz
Alvarez
Sanchez
Ruiz Alvarez
Alvarez Sanchez
Ruiz Alvarez Sanchez
```

or you might not bother with the two-component combos and let the user enter those if they wanted them later.

So, what I'd thought might be a one-line recommendation turns into a modest-sized essay on surname best practice.

- RL "Bob"

A LDAP filter config file for white pages

The following is the filter configuration file used by Georgetown University for its white pages service. This filter config file is used by the DSGW (Directory Server GateWay) interface of the Netscape/iPlanet/SunONE directory server. This config file is the same as what is used by OpenLDAP and the original UMich ldap distribution. Please do not consider this propietary to iPlanet, what we do here can be used with OpenLDAP or other products that provide this same functionality. Details on how to code for this config file can be found in the get_ldap_filter and associated routines from the OpenLDAP distribution.

```
################################################################################
# lines like this that start with # or empty lines are ignored
#
# syntax:
#
# <tag>
#   <pattern1>  <delimiters>    <filter1-1>     <desc1-1>       [<scope>]
#                               <filter1-2>     <desc1-2>       [<scope>]
#
#   <pattern2>  <delimiters>    <filter2-1>     <desc2-1>       [<scope>] ...
#
# The "desc" should describe the filter and it should correctly complete
# both of the following phrases:
#
#       One <desc> match was found for...
#       Three <desc> matches were found for...
#
# The scope is optional, and should be one of:
#       "base"
#       "onelevel"
#       "subtree"
# if it is included.
```

```
#

# some of the lines in this file are long, take this into account when viewing.

##################################################################################
# People searches.
#
"dsgw-people"
  "^[a-zA-Z0-9]+="     " "     "(%v))"                 "LDAP filter is"

  "^[+]*[0-9][ 0-9-]*$"  " "     "(telephoneNumber=*%v*))"       "phone number contains"

  "@"   " "     "(mail=%v))"            "email address is"
                "(mail=%v*))"           "email address starts with"

  "^.+,"     ", "    "(cn=%v2* %v1*))"       "last name, first name is"

  "[. _]"  ". _" "(|(cn=%v1* %v2*)(&(ou=*%v*)(objectclass=Person))))"  "first + last or group is"

    ".*"  ". " "(|(cn=*%v*)(sn=%v*)(uid=%v)(&(ou=*%v1*)(objectclass=Person))))"  "name, userid or group contains"


##################################################################################
# Authentication searches, prioritize UID first.
#
"dsgw-auth"
    ".*"                 " "     "(uid=%v))"             "Userid is"
                                 "(cn=*%v1*))"   "user name is"
```

---

telephoneNumber formatting

The following was contributed by Peter Gietz <peter.gietz@directory.dfn.de>:

```
E.123 6.1. states that "Grouping of digits in a telephone number should be
accomplished by means of spaces unless an agreed upon explicit symbol (e.g.
hyphen) is necessary for procedural purposes. Only spaces should be used in an
international number".

This is a bit ambivalent but should mean that hyphens shouldn't be used. This
interpretation is supported by footnote 6 on the word spaces, where it says:
"Administrations using dots or hyphens as separators nationally may require
time to determine the consequences of discontinuing their use." Since this
Standard was published 1993, I think enough time has elapsed by now ;-)
```

Software Reference

In no particular order... (additions accepted -- gratefully???)

http://www.web2ldap.de WWW gateway to LDAP server by Michael Ströder <michael@stroeder.com>
http://www-unix.mcs.anl.gov/~gawor/ldap LDAP Browser/Editor by Jarek Gawor <gawojar@iit.edu>
http://www.openldap.org The OpenLDAP project (OpenLDAP libraries as well)
http://www.metamerge.com Metamerge: $0 license to Higher-Ed world-wide. Metadirectory functionality.

CHANGELOG

2001-01-22 Bring the recipe up to level with respect to the eduPerson 1.0 release.
2001-02-13 telephoneNumber specifications should be groups of numbers space separated as clarified by Peter Gietz referencing E.123 section 6.1. Examples have been corrected and a blurb on this has been included in the recipe.
2002-04-08 Version 1.6: NMI: Discussion about Groups
2002-04-10 Version 1.6: NMI: Software Reference, usually free stuff
2002-04-23 Version 1.7: NMI: A discussion about E-Mail Routing by Tom Barton, Memphis

2002-04-23 Version 1.7: NMI: Active Directory Integration (reference)
2002-04-23 Version 1.7: NMI: eduPersonOrgDN and eduPersonOrgUnitDN discussion
2002-04-29 Version 1.8: NMI: added paragraph at document start regarding this document vs. NMI/MACE.
2002-04-30 Version 1.9: NMI: Various corrections and format changes submitted by MACE-Dir participants.
2002-04-30 Version 1.9: NMI: Added text for *eduPersonPrimaryOrgUnitDN* since MACE-Dir voted to include on 2002-04-29.
2002-05-01 Version 1.9: NMI: Add discussion about RDN issues.
2002-05-03 Version 2.0: NMI: Released for Public Review
2002-10-09 Version 2.1: NMI: Minor edits from "Thomas -Balu- Walter" <tw@itreff.de>
2002-10-09 Version 2.1: NMI: Added comments from Art Vandenberg to the Intro section of the document regarding Novell.
2002-10-09 Version 2.1: NMI: Added note regarding use of eduPerson class (AUX vs a child class).
2002-10-09 Version 2.1: NMI: Added comment to Active Directory Integration section for clearer positioning.
2002-10-09 Version 2.1: NMI: Various minor edits supplied by Jeanette Fielden <fielden@colorado.edu>
2002-10-10 Version 2.1: NMI: Added paragraph to expound upon service DN purpose and usage