

LDAP Access Using JNDI

Gary Murphy
Hilbert Computing, Inc.
glm@hilbertinc.com



What is JNDI?

- JNDI stands for Java Naming and Directory Interface
- JNDI is not limited to accessing LDAP data stores:
 - CORBA name services
 - RMI registry
 - Filesystem items
 - ...anything for which a service provider interface is written



What is LDAP?

- LDAP stands for Lightweight Directory Access Protocol
- LDAP is the protocol, but is often used to include the datastore.
- It is “lightweight” compared to X.500 directory services
- LDAP was born at the University of Michigan in 1992



Presentation Roadmap

□ LDAP

- Directory services – What they are well-suited and not well-suited for
- LDAP internal structures – schemas, ASN.1, types
- LDIF – export/import

□ JNDI

- Accessing LDAP
- Storing objects in LDAP



What are Directories?

- Directories are data stores that categorize and describe resources that people want access to. Offline directories include:
 - Phone books
 - Library card catalog
 - "TV Guide"
 - etc.



What are Directories? (cont.)

- Online directories have similar characteristics:
 - Tend to be hierarchical in organization.
 - Heavily read-oriented compared to write-oriented.
- Improvements over offline directories:
 - Dynamic
 - Powerful search capabilities



Differences with RDBMS

- Directories are specialized databases. The backing store can be relational. However, they differ:
 - Huge read over write ratio
 - Typically more easily extended
 - Typically more distributed
 - Typically more replication
- Not all databases should be LDAP-accessible datastores



Terminology

- DIT – Directory Information Tree
 - LDAP is a hierarchical datastore with nodes in a tree. The logical DIT may span multiple systems within a network, just as a single logical filesystem in Unix may span multiple system with NFS.
- DN – Distinguished Name
 - The fully-qualified name. Names in LDAP are reversed from filesystem names in that the most granular name is first, not last.



Distinguished Name

- In a filesystem:

`/usr/local/bin/bash`

The root of the tree is listed first and the leaf node is listed last.

- A DN in LDAP:

`cn=Gary Murphy,ou=staff,dc=hilbertinc,
dc=com`

The root of the tree is listed last.



Terminology

- RDN – Relative Distinguished Name
 - Partially-qualified name relative to a node in the DIT
 - Same concept as a relative name in a filesystem
- Schema
 - Describes the content that can be contained in a entry (aka “node”) in the DIT
 - How the data should be treated during search (e.g. collating sequences, binary vs. character, case-sensitive, etc.)



Entry Contents

- An “entry” is a node in the Directory Information Tree (DIT).
- An entry is composed of name/value pairs called “attributes”.
- Any entry can be a leaf node or a container unlike a filesystem which distinguishes containers (i.e. directories) from leaf nodes (i.e. files)



Entry Contents (cont.)

- An entry is defined by its “object class”, which defines the allowable attributes within that entry.
- The object class is defined as an entry in the schema.
- *You should be confused by now. Some examples...*



Distinguished Name Example

□ DN:

uid=glm@hilbertinc.com, ou=Employees, dc=hilbertinc, dc=com

□ In tree form:

dc=hilbertinc,dc=com

ou=Employees

uid=glm@hilbertinc.com

Note that the DN is reversed from filesystem naming, but consistent with other naming schemes such as DNS.



Attributes Example

□ Set of name/value pairs as follows:

```
dn: uid=glm@hilbertinc.com, ou=Employees, dc=hilbertinc, dc=com
objectClass: inetOrgPerson
cn: Gary Murphy
givenName: Gary
sn: Murphy
uid: glm@hilbertinc.com
telephoneNumber: 913-780-5051
facsimileTelephoneNumber: 913-829-2450
street: 13632 S. Sycamore Dr.
postalAddress: 13632 S. Sycamore Dr.$Olathe, KS$66062
postalCode: 66062
homePhone: 913-780-5051
homePostalAddress: 13632 S. Sycamore Dr.$Olathe, KS$66062
initials: glm
mail: glm@hilbertinc.com
```



Objectclass Example

- The objectclass attribute names the schema entry, which defines the valid attributes and the attribute types.
- The object class in the example is **inetOrgPerson**
- The schema entry for inetOrgPerson follows...



inetOrgPerson

```
objectclass ( 2.16.840.1.113730.3.2.2
  NAME 'inetOrgPerson'
  DESC 'RFC2798: Internet Organizational Person'
  SUP organizationalPerson
  STRUCTURAL
  MAY (
    audio $ businessCategory $ carLicense $ departmentNumber $
    displayName $ employeeNumber $ employeeType $ givenName $
    homePhone $ homePostalAddress $ initials $ jpegPhoto $
    labeledURI $ mail $ manager $ mobile $ o $ pager $
    photo $ roomNumber $ secretary $ uid $ userCertificate $
    x500uniqueIdentifier $ preferredLanguage $
    userSMIMECertificate $ userPKCS12 )
)
```



inetOrgPerson – Object ID

objectclass (2.16.840.1.113730.3.2.2

NAME 'inetOrgPerson'

DESC 'RFC2798: Internet Organizational Person'

SUP organizationalPerson

STRUCTURAL

MAY (

audio \$ businessCategory \$ carLicense \$ departmentNumber \$
displayName \$ employeeNumber \$ employeeType \$ givenName \$
homePhone \$ homePostalAddress \$ initials \$ jpegPhoto \$
labeledURI \$ mail \$ manager \$ mobile \$ o \$ pager \$
photo \$ roomNumber \$ secretary \$ uid \$ userCertificate \$
x500uniqueIdentifier \$ preferredLanguage \$
userSMIMECertificate \$ userPKCS12)

)



Object ID Registration

- Object Ids are assigned by IANA and should be globally unique.
- Free and easy to get an enterprise number
- <http://www.iana.org/cgi-bin/enterprise.pl>
- This is the same application for MIB/SNMP numbers, so don't let that confuse you.
- All will start with: 1.3.6.1.4.1
- Hilbert Computing, Inc. is 11993. **Don't use mine!**



inetOrgPerson – MAY & MUST

objectclass (2.16.840.1.113730.3.2.2

NAME 'inetOrgPerson'

DESC 'RFC2798: Internet Organizational Person'

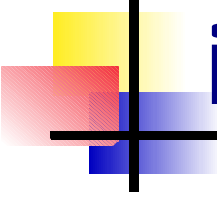
SUP organizationalPerson

STRUCTURAL

MAY (

audio \$ businessCategory \$ carLicense \$ departmentNumber \$
displayName \$ employeeNumber \$ employeeType \$ givenName \$
homePhone \$ homePostalAddress \$ initials \$ jpegPhoto \$
labeledURI \$ mail \$ manager \$ mobile \$ o \$ pager \$
photo \$ roomNumber \$ secretary \$ uid \$ userCertificate \$
x500uniqueIdentifier \$ preferredLanguage \$
userSMIMECertificate \$ userPKCS12)

)



inetOrgPerson – Inheritance

objectclass (2.16.840.1.113730.3.2.2

NAME 'inetOrgPerson'

DESC 'RFC2798: Internet Organizational Person'

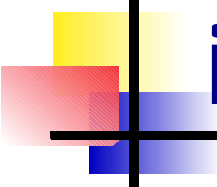
SUP organizationalPerson

STRUCTURAL

MAY (

audio \$ businessCategory \$ carLicense \$ departmentNumber \$
displayName \$ employeeNumber \$ employeeType \$ givenName \$
homePhone \$ homePostalAddress \$ initials \$ jpegPhoto \$
labeledURI \$ mail \$ manager \$ mobile \$ o \$ pager \$
photo \$ roomNumber \$ secretary \$ uid \$ userCertificate \$
x500uniqueIdentifier \$ preferredLanguage \$
userSMIMECertificate \$ userPKCS12)

)



inetOrgPerson – Inheritance

- The inetOrgPerson objectclass has an inheritance hierarchy. Inheritance similar to OO class hierarchies:

```
top
  person
    organizationalPerson
      inetOrgPerson
```

- All MAY and MUST attributes are inherited
- The root class is “top” which is analogous to java.lang.Object
- LDAP supports multiple inheritance



Attribute Schema Entries

- Each objectclass has a schema that defines the attributes that are allowed.
- In addition, each attribute has a schema that describes information about the attribute including its inheritance hierarchy
- One of the attributes of the inetOrgPerson objectclass that was inherited from “person” was the “cn” (common name) attribute. Let's look at the attribute schema for the “cn” attribute.



The 'cn' attribute

- commonName

attributetype (2.5.4.3 NAME ('cn' 'commonName') SUP name)Common

- name – The parent of Common Name

attributetype (2.5.4.41 NAME 'name'

EQUALITY caseIgnoreMatch

SUBSTR caseIgnoreSubstringsMatch

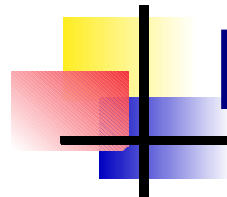
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})

- The “SYNTAX” defines the elementary datatypes. These are typically hardcoded within a LDAP implementation and include things like “Binary”, “Printable String”. The {32768} is a length.



LDAP Review

- LDAP is a hierarchical datastore of entries.
- Each entry has a set of attributes which are name/value pairs according to its objectclass.
- Each objectclass is defined in a schema entry
- Each attribute within each objectclass has a schema entry that defines its metadata, such as type and matching rules
- Each attribute has an inheritance hierarchy



LDAP Questions





Accessing LDAP in Java

- There are (at least) two class libraries for accessing LDAP from Java programs:
 - Netscape Java LDAP SDK – specific to LDAP. Open source, royalty-free.
 - Java Naming and Directory Interface (JNDI) – A generic naming interface that contains an LDAP service provider interface.
- This talk discusses only JNDI.



JNDI Overview

- Java Naming and Directory Interface (JNDI) consists of two architectural components:
 - An object model that serves as the API
 - Service Provider Interface (SPI)
 - LDAP
 - CORBA Common Object Services (COS) name service
 - RMI Registry
 - DNS
 - ... others available via <http://java.sun.com/products/jndi/> or from vendors



JNDI within JDK Releases

- v1.1 – Standard extension
- v1.2 – Standard extension
- v1.3 – Included with: LDAP, CORBA COS Naming, RMI
- v1.4 – Added DNS
- via download: NIS, filesystem, DSML (Directory Services Markup Language)



Key JNDI Interface Overview

- Context – core interface to traverse, searching, etc., objects in the directory
- Name – Abstraction over the namespace of the directory
- Bindings – Name-to-object relationship within the directory. The “object” is often a Context object.
- Reference – Contains information on how to construct a copy of the object in the LDAP store



Context

- A **Context** is the programming interface for an element in the LDAP.
- A **DirContext** extends Context to enable access to the Attribute objects within a Context
- An **InitialContext** is the logical root of the DIT, not necessarily the real root.



Accessing InitialContext

```
protected synchronized void connect(String url)
    throws Exception {

    Hashtable env = new Hashtable();

    env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL,
        "ldap://localhost:389/dc=hibertinc,dc=com");
    env.put(Context.SECURITY_PRINCIPAL, "...");
    env.put(Context.SECURITY_CREDENTIALS, "...");

    DirContext context;
    try {
        context = new InitialDirContext(env);
        setInitialContext(context);
    }
    catch (NamingException original) {
        ...
    }
    return;
}
```



Accessing Attributes

- Attributes are the name/value pairs associated with a particular context.
- The collection of attributes are accessed via the [Attributes](#) interface
- A single attribute is modeled using the [Attribute](#) interface.



Accessing Attributes

```
// Selective subset of attributes
```

```
String[] passwordAttribute = { "userPassword" };  
Attributes passwords =  
    getInitialContext().getAttributes(  
        "ou=staff,uid=glm", passwordAttribute);
```

```
NamingEnumeration values = passwords.getAll();  
while (values.hasMoreElements()) {
```

```
    Attribute attribute = (Attribute) values.nextElement();
```

```
    Object value = attribute.get();
```

```
    if (value instanceof byte[]) {  
        passwordReference.setPassword((byte[]) value);
```

```
    }
```

```
    else
```

```
    if (value instanceof String) {  
        passwordReference.setPassword(((String) value).getBytes());
```

```
    }
```



Accessing Attributes – Notes

- Attributes were accessed for an object relative to the initial context.
- Works better than getting a reference to the DirContext and then getting the attributes from the context
- Attributes may be multi-valued. In that example, there may have been multiple passwords. For single-value attributes:

```
ctx. getAttributes("ou=ts, ui d=gary"). get("password"). get();
```



Accessing Attributes – Notes

- Attribute.get() returns a java.lang.Object. What is it?
- I suspect it is up to the SPI to return a suitable object. It appears to be either a String or a byte [] depending on whether the data is binary (e.g. certificates, encrypted passwords) or character data, which comprises most of the LDAP data
- A definitive reference would be welcome



Listing Contexts

- Context.list() method will return an enumeration of NameClassPair objects which list a name and an associated class name.
- The name is the name of the context, such as "uid=glm". The class is typically javax.naming.DirContext.

Could it be anything else?



Listing Contexts

- `Context.listBindings()` method will return an enumeration of Binding objects, which list a name, the class name and the object itself.
- The object is typically an instance of a `DirContext`.
- This is potentially more expensive since the objects themselves are instantiated.



Creating a Subcontext

```
Attribute obj class = new BasicAttribute("objectclass");  
obj class.add("top");  
obj class.add("something"); // Must be in the LDAP schema
```

```
Attributes attrs = new BasicAttribute(true); // true is case-ignore  
attrs.put(obj class);
```

```
Context subcontext = context.createSubcontext("cn=bi ff", attrs);
```



Delete a subcontext

```
context.destroySubcontext("cn=bi ff");
```



Modifying Attributes

```
// Access the principal as a directory context so we can modify the
// attributes without having to rebind the whole context
```

```
String credentials = "secret";
try {
    // Modify the attributes as appropriate

    ModificationItem[] mods = new ModificationItem[1];
    byte[] password = credentials.getBytes();
    Attribute passwordAttribute
        = new BasicAttribute("userPassword", password);
    mods[0]
        = new ModificationItem(DirContext.REPLACE_ATTRIBUTE,
            passwordAttribute);

    getInitialContext().modifyAttributes("ou=staff,uid=glm", mods);
}
catch(NamingException original) {
    ...
}
```



Searching

- Outstanding Feature of LDAP in general. Robust implementation in JNDI
- Search filters comply with RFC 2254
- Prefix notation expressions like:

`(&(sn=Geisel)(mail=*))`

must have a surname of "Geisel" and a mail attribute with some value



Searching

```
// Given the name, we need to locate any permission assignments
// to any of the principals

SearchControls controls = new SearchControls();
controls.setSearchScope(SearchControls.SUBTREE_SCOPE);
String filter = "(&(objectClass=hfwPermissionMember)(perm="+logicalName+"))";

NamingEnumeration results = getInitialContext().search("", filter, controls);
while (results.hasMoreElements()) {
    // Since this search is from the initial context, we
    // don't need to worry about whether the results have
    // relative names or not.

    SearchResult result = (SearchResult) results.nextElement();
    context.unbind(result.getName());
    ++removed;
} // while
```



Searching

- SearchControls objects can:
 - set attributes returned
 - set number of results returned
 - set time limit for search
 - determine if the objects are returned
- SearchResult contains:
 - the name, attributes as you would expect
 - whether the name is relative



Java Objects

- Java objects can be stored in an LDAP directory in the following ways:
 - Serialized binary stream
 - JNDI References and Referenceable objects
 - Objects with attributes (e.g. DirContext)



Serialization

```
// Create the object to be bound
```

```
Button button = new Button("Push me");
```

```
// Perform the bind (i.e. store it)
```

```
ctx.bind("cn=Button", button);
```

```
// Read the object back
```

```
Button restored = (Button) ctx.lookup("cn=Button");
```



Serialization

- Stores the serialized byte stream in the LDAP context
- Major Advantage
 - Very simple to implement
- Major Disadvantage
 - Stored as a stream of bytes, so only Java applications can see the content



Referenceable

- Serialization stores a copy of an object, Reference objects store a reference or “pointer” to the object in a stateful way.
- A **Reference** is a class name and a list of “addresses” as concrete implementations of the abstract **RefAddr** class.
- To store an object by its reference, the class must implement the **Referenceable** interface.



Referenceable Example

```
public class Fruit implements Referenceable {
    String fruit;

    public Fruit(String f) {
        fruit = f;
    }

    public Reference getReference() throws NamingException {
        return new Reference(
            Fruit.class.getName(),
            new StringRefAddr("fruit", fruit),
            FruitFactory.class.getName(),
            null);           // Factory location
    }

    public String toString() {
        return fruit;
    }
}
```



Referenceable

- Major Disadvantages
 - Classes must implement Referenceable
 - Not appropriate for business objects
 - Objects must know how to bind themselves to LDAP
 - Information useful only to Java objects
- I would be interested in a definitive example of when you would want to use this. I don't get it.



Factories

- Object Factories are used to construct a Java object based on the information in an LDAP context.
- State Factories are used to save the state of a Java object in an LDAP context
- The attributes are available to other applications that query/update the LDAP directory.



Factories

- The Sun LDAP Service Provider implements a factory design pattern:
 - When an object is to be returned via `lookup()`, `search()`, `listBindings()`, etc., the list of available factories have a chance to return an object.
 - If a factory doesn't deal with that context, it returns a null reference.
 - If no factory returns an object, a `DirContext` object is constructed by the service provider.

Yes, it could be something else!



State Factories

- State Factories take an object, and references to the LDAP objects and create attributes for LDAP.
- Method signatures:

```
public Result getStateToBind(  
    Object object,  
    Name name,  
    Context context,  
    Hashtable env,  
    Attributes inAttrs) throws NamingException
```

```
public Result getStateToBind( // Probably not used for LDAP  
    Object object,  
    Name name,  
    Context context,  
    Hashtable env) throws NamingException
```



State Factories

[c]

```
public Result getStateToBind(Object object, ...
    if (!(object instanceof HPermissionMember)) {
        return null; // This will give another state factory a chance
    }

    Attribute attribute = null;
    Attributes outAttrs = new BasicAttributes(true); // Case insensitive
    if (null == outAttrs.get("objectclass")) {
        attribute = new BasicAttribute("objectclass", "hfwPermissionMember");
        attribute.add("top");
        outAttrs.put(attribute);
    }
    HPermissionMember member = (HPermissionMember) object;
    String perm = member.getMemberName();
    if (null == perm) {
        throw new IllegalArgumentException(
            "The member name ... not be a null reference");
    }

    Attribute attribute = new BasicAttribute("hfwPermissionName", perm);
    attributes.remove("hfwPermissionName");
    attributes.put(attribute);
    return new DirStateFactory.Result(null, outAttrs);
}
```



LDAP Schema for Example

```
attributetype ( 1.3.6.1.4.1.11993.1.3.9 NAME (
    'perm' 'permission' 'hfwPermissionName' )
    DESC 'The name of a permission in a collection'
    SUP name )
```

```
objectclass ( 1.3.6.1.4.1.11993.1.4.302 NAME 'hfwPermissionMember'
    DESC 'Contain a reference to a permission name that is ...'
    SUP top
    MUST ( hfwPermissionName )
)
```



Using State Factories

- Add all factory class names to the context as a colon-delimited string
- Call the bind() method on the context



Using State Factories

```
String factoryList =
    "COM.hilbertinc.security.ldap.factory.HPermissionMemberFactory: "
    +"COM.hilbertinc.security.ldap.factory.HPermissionListMemberFactory";
String logicalName = "updateCustomer";
String contextName = "cn=permissions,dc=hilbertinc,dc=com";

context.addToEnvironment(Context.STATE_FACTORIES, factoryList);
DirContext codeContext = (DirContext) context.lookup(contextName);
HPermissionMember member = new HPermissionMember(logicalName);
codeContext.rebind("perm=".concat(logicalName), member);
codeContext.close();
context.close();

// Creates an LDAP context of the name
//
//      perm=updateCustomer, cn=permissions, dc=hilbertinc, dc=com
//
// that matches the schema hfwPermissionMember because that was
// objectclass we specified for this object
```



Object Factories

- Object factories get the LDAP context and attributes and creates an object
- Method signatures:

```
public Object getObjectInstance( // Not used with LDAP – no attrs
    Object object,
    Name name, Context context,
    Hashtable env) throws Exception;
```

```
public Object getObjectInstance(
    Object object, // Possibly null reference/location information
    Name name, Context context, Hashtable env,
    Attributes inAttrs) throws Exception;
```

Object Factories

[c]

```
public Object getObjectInstance(Object object, Name name, ...
    // Take a look at the objectclass attribute and make sure this is of the
    // correct class

    if (null == inAttrs) {
        return null;    // Can't be us...
    }
    Attribute objectclass = inAttrs.get("objectclass");
    if (null == objectclass) {
        return null;
    }
    if (!objectclass.contains("hfwPermi ssi onMember")) {
        return null;
    }

    // Construct an instance of the principal with the name from the
    // LDAP entry attribute

    Attribute attribute = inAttrs.get("perm");
    String memberName = (null == attribute) ? null
        : attribute.get().toString();
    HPermi ssi onMember member = new HPermi ssi onMember(memberName);
    return member;
}
```



LDAP Schema for Example

```
attributetype ( 1.3.6.1.4.1.11993.1.3.9 NAME (
    'perm' 'permission' 'hfwPermissionName' )
    DESC 'The name of a permission in a collection'
    SUP name )
```

```
objectclass ( 1.3.6.1.4.1.11993.1.4.302 NAME 'hfwPermissionMember'
    DESC 'Contain a reference to a permission name that is ...'
    SUP top
    MUST ( hfwPermissionName )
)
```



Using Object Factories

- Add all factory class names to the context as a colon-delimited string
- Use one of the LDAP methods for reading such as `listBindings()`, `search()`, `lookup()`, etc.



Using Object Factories

```
DirContext codeContext = (DirContext) binding.getObject();

// Re-obtain this context as a code source

String factoryList =
    "COM.hilbertinc.security.ldap.factory.HPermissionMemberFactory: "
    +"COM.hilbertinc.security.ldap.factory.HCodeSourceFactory";
context.addToEnvironment(Context.STATE_FACTORIES, factoryList);
CodeSource codeSource = (CodeSource) codeContext.lookup("");

// Iterate through the subcontexts of this code source to
// access the permission references that are associated with
// it

ArrayList permissionNames = new ArrayList();
NamingEnumeration perms = codeContext.listBindings("");
while (perms.hasMoreElements()) {
    Binding permissionBinding = (Binding) perms.nextElement();
    HPermissionMember member
        = (HPermissionMember) permissionBinding.getObject();
    permissionNames.add(member.getMemberName());
} // while more references for a code source
codeContext.close();
```



JNDI Questions



Time Permitting...

- Hilbert Computing implementation decisions
- Programming notes



Using LDAP for Security

- First use of LDAP was to store the Java 2 security policy objects.
- Implementation at:
<http://www.hilbertinc.com/security.html>



Implementation Decisions

- Create a single factory class that combines state and object factory
 - Saving and restoring object states are symmetrical
 - Methods without attributes always return a null reference since LDAP always wants the overload with attributes
 - Convenience methods
- See [COM.hilbertinc.security.Idap.factory.HAbstractFactory](#)



LDAP Programming Problems

- `java.lang.ClassCastException`
 - If the object factory is not properly identified to the environment, you will get **ClassCastException**'s because JNDI SPI will return **DirContext** objects instead of your object or conversely, there was a factory that created a specific object instead of the **DirContext** object that you were expecting.



LDAP Programming Problems

- `javax.naming.directory.SchemaViolationException`
 - Something in your factory or attributes violated the schema. This can also happen if you have factories that unmarshall to different schemas, but check for the same Java object. For example, if you have two factories that unmarshall a `java.lang.String`, and both factories are assigned to the context, the wrong factory may be used for your String since the key to using a factory or returning a null reference is typically the class type of the Java object.



LDAP Programming Problems

- `javax.naming.NameNotFoundException`
 - Typically, this occurs when trying to access an LDAP context that doesn't exist. It can also happen if the state of a Java object is trying to be saved using a JNDI state factory and the factory is not associated with the context.

In OpenLDAP, the error text will have the string:

```
[LDAP: error code 32 - No Such Object]; remaining name  
'name'
```



References

- JNDI API Tutorial and Reference: Building Directory-Enabled Java(TM) Applications
by Rosanna Lee, Scott Seligman
Addison-Wesley Pub Co
ISBN: 0201705028



Thank You

- Thank you for taking the time to attend this session. I hope it has been helpful
- Fill out the session evaluations. They are helpful to Wayne & Peggy and me.
- Feel free to contact me via e-mail at:
glm@hilbertinc.com
- I will be at the conference all week. Feel free to ask any questions that arise after the session.