Date: 22nd Dec. 2014

1. THE GOAL

Main goal of this library is simplifying JavaMail class library following the workflow of common e-mail client like Mozilla Thunderbird or MS Outlook.

If you had an experience of writing codes with JavaMail, you might think that JavaMail is quite complex to understand and use it. I also thought that I need more simple class library to use when I was writing a small application.

So, I did mind few factors which are as below when I wrote DoitFX.

1. The structure of configuration directory has to be human understandable
2. Users of the library can send messages when they finished assembling the message.
3. Users of the library don't need to understand the behind of the codes. They only need to learn about few classes which act as a entry point.
4. Possibly, separate classes and features, also work itself individually.

As I mentioned above, it's okay that the users only need to learn 4 classes.

i. ***Connector*** – Main part of connection to mail servers.
ii. ***MessageReceiver*** – Getting this from Connector and receiving folders and messages
iii. ***MessageSender*** – Getting this also from Connector and sending messages via a designated server which is selected by a user.
iv. ***FolderManipulator*** – Manipulating folder in users' account and doing functions like create, update (rename) and delete the folders.

Also, users know only about 1 variable that distinguishes each identity.

I call it as Service Pair Name, shortly *svcPair*, its format is as follow.

(SERVICE NAME)/(USER NAME)

This is the convenient way to indicate each e-mail account among what I thought.

Date: 22nd Dec. 2014

2. THE STRUCTURE (CONFIGURATION DIRECTORY)

The structure of configuration directory is simple as follow.

(MAIN DIRECTORY)/(SERVICE NAME)/(ACCOUNT ID)

What you need to catch is ***(SERVICE NAME)/(ACCOUNT ID)***.

Why? Don't you remember about svcPair variable?

Actually, the variable ***svcPair*** is plucked out from this structure. This is associated with loading configuration files. (This is why I use '/' as a delimiter)

Each SvcPair has 3 files under its directory.

- ***send.props*** – This file contains SMTP server configuration
- ***recv.props*** – This file contains IMAP4 (or POP3)
- ***pass.code*** – Password byte code file

You can see the extension of 2 files 'props'. The reason is JavaMail uses 'Properties' class for loading and storing its properties. So, I named it to 'props'.

There are 2 more files you need to mind are under ***(MAIN DIRECTORY)***

- ***app.conig*** – Main application configuration file that stores global properties to run the application. It's not related with e-mail accounts.
- ***TheStore.dat*** – KeyStore, JCE (Java Cryptography Extension) has a facility to store passwords as secret as it can. The facility is KeyStore class and this file is the file that exported from KeyStore class.

3. THE FORMAT OF CONFIGURATION FILES

i) ***app.config*** Format

app.config file is global configuration file in application. Currently, however, there is one property in the file

- ***unique_id*** – This property is to indicate application itself and it's UUID format. It is very important for en/decrypting passwords because this is used as KeyStore, itself, password.

ii) ***send.props*** and ***recv.props*** Format

send.props has few properties as below.

- ***unique_id*** – This property works as two roles. The indicator of the acccount and the password for KeyStore's KeyEntry
- ***encryption*** – This means the way of encrytion when establishing connection to the target server. You can use ***SSL/TLS*** or ***STARTTLS***
- ***protocol*** – Which protocol is the protocol among IMAP or POP3
- ***host*** – Target server name
- ***port*** – The port number of the target server
- ***address*** – The email address of the account

There is no difference between two props file, but you need to remember 1 thing. unique_id of 2 files is sharing (in other words, same)

Date: 22nd Dec. 2014

4. PASSWORD MANAGEMENT

There are few cases that I experienced that passwords are stored without encryption. I was so frustrated for that and I decided to implement DoitFX not to store plain password itself.

Regardless of the motivation, the facility of DoitFX password management is based on JCE (Java Cryptography Extension).

Basically, DoitFX uses DES cryptographic method because I want to shrink the time to en/decrypt.

You don't need to think so many things but PasswordManager class. PasswordManager is the representative class for the Password Manament.

The procedure of password en/decryption is as below.

1. Pick up the UUID of the application (global application config file) from ***app.config*** – Maybe already initialised
2. Pick up the UUID of target account from ***recv.props***
3. Put the account UUID into getAccountKey method, you will get the plain password for the connection

As I mentioned above, Application UUID is used for opening KeyStore (***TheStore.dat***) and Acount UUID is to access KeyEntry in the KeyStore.

I'm not sure you guys know about this, but KeyStore doesn't store any password itself as plain text. It only stores Key classes

This is why there is a file which was named as 'pass.code'.

When PasswordManager decrypt passwords, the first thing it does is loading

pass.code to KeyStore. Simple, KeyStore is for storing keys and pass.code file itself is ciphertext (byte type).

Date: 22nd Dec. 2014

ATTENTION!: ***JCE*** was released in the age of ***JavaSE 1.2***, so you should mind that ***Java NIO*** (since ***JavaSE 6***) is not compatible with ***JCE***. If you have an idea to modify or amend DoitFX codes, don't use ***NIO*** (Particularly, ***Files.newBufferedStream*** method). I **WASTED 3 DAYS** to find out this.

5. Config File Management

As I mentioned in 1st Chapter, ***SvcPair*** variable is the core of config file management.

In this part, ***SvcPair*** has 2 meaning, first is that it is unique name to distinguish other accounts and second is the location of the config file for the account.

Of course, ***SvcPair*** is also used a parameter of the method ***getAccountKey***.

By the way, there is a class for this job called to ***AccountConfigManager***. AccountConfigManager has a instance variable which name is ***ConfFileTraverser***.

When you load an application which is using DoitFX, ***ConfFileTraverser*** acts first to read all pre-configured files. In fact, ***ConfFileTraverser*** is an inherited class of ***SimpleFileVisitor*** class. As you think, ***SimpleFileVisitor*** is an implementation of ***Visitor*** design pattern, so it traverse every file under (***MAIN CONFIGURATION DIRECTORY***) to load the configurations.

Unfortunately, however, loading recv/send configuration is split into 2 to reduce its complexity.

6. DoitFX Initialisation

The initialiser of DoitFX is ***AppInitialiser*** class. This is a singleton class and loads all classes (Particularly, *****Manager*** classes). Truth be told, you don't need to invoke every class you need by yourself.

If you want to get *****Manager*** classes, just call ***AppInitialiser.get**Manager*** method. I designed those classes to constructor duty free classes.

Surely, you also don't need to create ***AppInitialiser*** class. It will be invoked when you create ***Connector*** class.