# User FAQ

## QuickFIX/J User FAQ

We get a lot of repeat questions on the mailing list. Here's a repository of answers.

### How do I set username/password in my Logon message?

To set the username field in a Logon message, put something like this in your toAdmin() callback.

```
final String msgType = msg.getHeader().getString(MsgType.FIELD);
if(MsgType.LOGON.compareTo(msgType) == 0)
{
    msg.setString(quickfix.fields.Username.FIELD, _username);
    msg.setString(quickfix.fields.Password.FIELD, _password);
}
```

### How do I regenerate/rebuild QF/J with a custom data dictionary?

I assume you've already checked out the latest source from the svn repo. You'll need `ant` installed.

QF/J generates the source from the DDs in `core/src/main/resources`. Make a back up of the one you're going to alter, and then alter it however you need to.

Then rebuild as follows:

- `ant jar`
- You will be prompted for a release number; this just determines the suffix given to the jar names. Enter whatever you want.
- Wait for build to finish
- Find your brand-new QF/J jars in `core/target/`

### I altered my data dictionary. *Should* I regenerate/rebuild QF/J?

It depends on how extensive your DD changes are.

If your DD changes aren't very extensive, maybe just a few field changes, then you don't really need to. If you added a whole new custom message type, then you probably should. If you changed field orders inside of repeating groups, then I recommend that you do, especially if those group changes are in outgoing messages.

Want more detail than this? Ok...

INCOMING MSGS: The DD xml file is critical for the engine to validate messages that you receive from the counterparty. When the engine parses messages, it exclusively uses this xml file; the generated java message classes aren't a factor here. When your user code is processing those messages, those message classes become relevant.

- **Note 1:** If you added a new field Foo but didn't rebuild, you won't have getFoo() and setFoo() methods to call, though you can use getString() and setString() instead. No big deal.
- **Note 2:** If you added a new message type but didn't rebuild, I suspect the MessageCracker.crack() method will throw an UnrecognizedMessageType exception, because there is no generated message class corresponding to the new type. This is why I recommend you rebuild in this case. (There are ways to handle this, though.)

OUTGOING MSGS: The DD xml file is irrelevant when you construct outgoing messages. You can pretty much add whatever fields you want to messages using the generic field setters (setString, setInt, etc) and QF will let you. The only trouble is with repeating groups. QF will write repeating group element ordering according to the DD that was used for code generation. If you altered any groups that are part of outgoing messages, you DEFINITELY need to rebuild.

If you wanted to parse a message string using your custom field ordering, then you will need to use your custom generated MessageFactory like so:

```
quickfix.Message message = messageFactory.create(beginString, messageType);
message.fromString(messageString, dataDictionary, validate);
```

If you want to parse arbitrary message strings, then use

```
MessageUtils.parse(MessageFactory, DataDictionary, String)
```

## How do I rebuild QF/J?

You need `ant`.

This will rebuild the jars based on the DataDictionary files in `core\src\main\resources\`. To make DD customizations, change those files. Then run this command:

```
# The `version` argument is just a filename suffix
# The `skip.jalopy` argument is optional and will skip some time-consuming
doc generation.
ant version=SOME_STRING -Dskip.jalopy=true clean jar
```

The new jars will be found in `quickfixj\core\target\`.

## How do I create a repeating group for my custom group?

If you rebuilt QF/J with your custom data dictionary, there should be a class for your new group and you should do it like the documentation shows you:
http://www.quickfixj.org/quickfixj/usermanual/1.5.1/usage/repeating_groups.html

If you **did not** rebuild QF/J and do not want to, you can use generic (non-typesafe) methods.

For example, say you have custom group with counter tag 9610, and delimiter tag 9611:

```
msg = your message of whatever type;
int[] order = [9611,9612,9613]; // syntax probably needs correction
quickfix.Grp grp = new quickfix.Group(9610,9611,order);
grp.setString (9610, "1" );
grp.setString (9611, "SHORT" );
msg.addGroup(grp); // add first group
grp.setString (9612, "3" );
grp.setString (9613, "SNOTE3" );
msg.addGroup(grp); // add second group
```

Compare this with the type-safe methods and you'll see it's pretty similar.