# MekWars Server Startup Howto (MW 0.1.0.0) – by Vertigo

This document explains how to start up a fresh MekWars server. It tries to be as detailed as possible, but for any further help, the sourceforge project page and direct contact with deveopers are the best routes.

## 1. Concepts and assumptions

Throughout this document, I assume you are familiar with basic concepts about computers, internet, Java, and so forth. However, I repeat here the absolute minimum names you should be confident with:

- MegaMek: it's the actual game engine. It's a separate Java program, launched by the clients when two players engage in a campaign battle, and need to solve “on the battlefield” their fight.
- MekWars server: a Java program accepting connections from MekWars clients and holding the data for players, factions, meks, economy, planets, etc. You get the barebone server from SourceForge, and populate it with your own data. You set this program up, which runs on a machine supposedly permanently connected to the internet.
- MekWars client: the Java program used by players to connect to your server, and to launch the MegaMek program to carry on the battles. One of your duties is to prepare and package the client with your own user data, and distribute it to your players. You tie the MekWars client with one MegaMek program (so the former can run the latter) before releasing it.
- MegaMekNet: another client/server environment. It's the project from which MekWars spawned (or “forked”).
- NFC: the base chat system (an IRC derivative) upon which the client/server communication protocol is based. It's an old, crufty, abandoned, contorted and inefficient piece of software. But it's needed. Unless some saint developer gets finally rid of it, writing a better network comm protocol, we must deal with it (hint hint).

All three “Mek” projects (MegaMek, MekWars, MegaMekNet) have separated development teams. Addressing issues specific to one project to another team is considered unpolite and shows lack of understanding of how the whole system works. Remember that actual game issues are MegaMek ones, and campaign and metagame issues are either MekWars or MegaMekNet (hardly both). For sake of brevity, I indicate “MM” for MegaMek, “MW” for MekWars and “MMN” for MegaMekNet.

## 2. Server requirements

Having the hardware needed to run a MW server is probably an easy step. More or less, any machine produced and sold on the market in the last three years is up to the task. Note also that since the programs are coded in Java, you have a rich choice of platforms to run it onto. Furthermore, RAM is the most critical aspect, being Java a memory-intensive environment and the server in particular a large resource user.

The actual minimum system requrements are:

- CPU: 500MHz or more; CPU isn't really a big issue with the server. You need more power, however, if you plan to run also the client on the same machine.
- RAM: 512MB or more; the more the better, really. Java needs lots of memory, and the server even more!
- Disk: 500MB; actually, the disk usage depends on how large the logs are.
- Net: always-on needed; speed can be as low as 1Mb/s sustained. Actual requirements depend on number of concurrent players. It's very important to have a fixed IP, or at least a way to be uniquely identified on the internet (like thru dynamic DNS systems).
- Ports: default listening ports are 2347 and 4867. They must be open and accepting connections. If you're behind a router, firewall or other filtering equipment, make sure that connection on these ports can reach the serving machine.
- I/O: anything recent goes. Fast SCSI obviously preferable over IDE/EIDE/ATA/SATA. You may also want some sort of redundancy (RAID1 or RAID5) and/or backups.
- Java: JRE or JDK 1.5.0+. By design, it won't work anymore with 1.4.x.

In this Howto, I assume a Linux environment. Your mileage may vary in case you're stubborn enough to run the server under Windows, MacIntosh, Solaris or other operating systems.

## 3. Obtain software

As I said above, you can find the stable releases of both MW client and MW server on SourceForge:

```
http://sourceforge.net/project/showfiles.php?group_id=122002
```

The files you're looking for are similar to this:

```
Server-0.1.0.0-STABLE-NODOCS.zip
Client-0.1.0.0-STABLE-NODOCS.zip
```

Remember that you get the old and stable releases over there. If you want bleeding edge features and performance (and don't mind bleeding), you can obtain it from CVS by using anonymous CVS in a standard shell:

```
$ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/mekwars login
$ cvs -z9 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/mekwars co -P mekwars
```

Beware: these instructions won't guide you in the (daunting) task of rebuilding a server from the CVS source code. If you need this level of detail, you have to ask the developers anyway, so I won't bother detailing the procedure.

In any case, the software you get is still in raw format, and unsuitable for direct distribution to your players. You need to add useable data to it. And there's no “default” data available on the SourceForge site. How you populate the server is your own responsibility. MW developers don't distribute any potentially copyright infringing material on purpose.

Keep the zipped packages somewhere on the local disk, like in your home directory. You will need them later.

## 4. First run of the server – creating default config

Once you have the server package, open a shell, create a suitable directory and unzip the package into the directory. The name of the directory doesn't matter:

```
$ mkdir ~/mw-testserver
$ cd ~/mw-testserver
$ unzip ~/Server-0.1.0.0-STABLE-NODOCS.zip
```

Once it's done you start the server for the first time, so to create a default config file:

```
$ cd Server-0.1.0.0-STABLE-NODOCS
$ java -server -jar MekWarsServer.jar >log.txt 2>error.txt
```

(You might want to create a small one-line script that includes the last command. Up to you.)

After a dozen or so seconds, the server should be ready to accept connections. To check if the server is actually running, open another shell and type:

```
$ netstat -tuna | grep 2347
```

If the answer is something similar to:

```
tcp   0   0 0.0.0.0:2347   0.0.0.0:*   LISTEN
```

Then your server is up and running. Congratulations!

## 5. Adding an admin

The server is happily passing its time, all alone. It's time to connect to it and set yourself as the ruler of your own little world. To do this, you need the client you downloaded during step 3). Open yet another shell and do the following:

```
$ mkdir ~/mw-testclient
$ cd ~/mw-testclient
$ unzip ~/Client-0.1.0.0-STABLE-NODOCS.zip
$ cd Client-0.1.0.0-STABLE-NODOCS
$ vi data/mwconfig.txt
```

The default package knows of no servers, and thus you must insert the correct data for the first run of your client. In the mwconfig.txt file add or change the following keyword/value pairs:

- SERVERIP: 127.0.0.1
- SERVERPORT: 2347
- DATAPORT: 4867

Then, finally, start the client with the command:

```
$ java -jar MekWarsClient.jar
```

The client asks you for a name: enter the username of your choice. You can keep a blank password (for now).

Click on “OK” and you should be able to enter your new server. Should anything go wrong, read carefully the error message that pops up, and eventually the client and the server logs.

Once in the server, you're just a new player. You must click on the menu “File” and select item “Register Nickname”. Then insert your username again, and pick a password for the account. It's important to register your nickname, so that your username can be recorded in the players data file. Once you do that, you can enroll, if you want. There's still no data available for you to play with. No units, no pictures, nothing.

Now that you recorded your username on the system, quit the client, and in the shell where the server has been started, interrupt the game server. To interrupt it use “CTRL+C”. You should obtain back the shell prompt from which you started the server.

Now, you must edit the chat password file (I use “vi” as example editor; since the password file is just a text file, feel free to use any editor you like):

```
$ vi conf/nfc.passwd
```

The file should contain just two lines. They look similar (but actually different) to these:

```
imi:200:99abc123abc123:1110123456789:
vertigo:0:99abc123abc123:1110123456789:
```

You can delete the line with “imi” (unless you want to allow full admin access to any player named “Imi” who knows the password for that login). Replace the “0” with a “200” in the line with your name:

```
vertigo:200:99abc123abc123:1110123456789:
```

Save the file, and congratulate with yourself. You  just turned into a full-blown administrator! To double check your newly found divine powers, restart the server, restart the client and reconnect to your own server (just follow the steps you performed before). Your name should sport a nice “@” in the player list. If it does, you're the new admin of the server.

Note: you should never again alter the NFC password file by hand. Once you have at least one administrator for the server, all administrative tasks (including adding other admins, and changing anyone's password) can be performed from within the client, to maintain coherence in the passwords file.

## 6. Second run of the server – creating planets

As you noticed, the server is very empty, as of now. You must add your own “world” to it, in order to provide a reasonable environment for your players. The first step is to create the galactic map. You don't need to do it all at once, because you can add elements to it at any time. But since you have to provide some background data for your server, better start with the worst job.

Restart the server, restart the client, and connect to the server as you did in the previous step. Go to the “Map” tab on the top row, and notice that the map is either totally missing, or still a default map, with less than optimal planets and factories placement. At time of this writing, the default map shipped with the server contains lots of planets, but only for some of the factions; my suggestion is to wipe the map completely and start from scratch.

To create/modify/delete planets, you use the “Admin” menu of your client. That menu is visible only if you are actually an admin of the server. Remember to add factories as well as planets!

## 7. Third run of the server – fresh with planets

In your server you should actually have two modified files: “conf/nfs.passwd” and “planets.xml”. If you modified other files, too bad, I didn't tell you to do it! Save these files somewhere, and delete the entire server directory. Yes, I mean start from scratch a second time:

```
$ cd ~/mw-testserver/Server-0.1.0.0-STABLE-NODOCS
$ cp conf/nfc.passwd ~
$ cp planets.xml ~
$ cd ~
$ rm -rf mw-testserver/*
$ cd mw-testserver
$ unzip ~/Server-0.1.0.0-STABLE-NODOCS.zip
$ cd Server-0.1.0.0-STABLE-NODOCS
$ mv -f ~/nfc.passwd conf/nfc.passwd
$ mv -f ~/planets.xml planets.xml
$ java -server -jar MekWarsServer.jar >log.txt 2>error.txt
```

This should give you a new, working server, already populated with planets, without the “pollution” of base map data, which was stored anyway in the server “live data” dirs. From now on, you can keep running with this installation. This is a good checkpoint to make a backup of your entire server directory.

## 8. Useful directories and files

Once you have a running server, you will face the bulk of the work as administrator. You need to provide data files to your clients, and configure the server itself.

The data objects are created by the server once you configure it and let players populate your world and start battling and influencing the campaign.

There are some server configuration files you must be aware of: even if most of the values written within them can be manipulated through the GUI of the Admin menu, it's important that you know them and their role. All the path and filenames are relative to the server install directory:

| ***Filename(s)*** | ***Subsystem*** | ***Comment*** |
|---|---|---|
| data/campaignconfig.txt | Campaign | Majority of configuration for the campaign itself is stored here. There's a document describing the variables and the values stored there. You **WANT** to make backups of this file, trust me! |
| data/serverconfig.txt | Server | Configuration of the server program, with items like the port where to listen, the MOTD, etc. Roughly, anything not related to the campaign is found here. |
| logs/* | Server | All that needs logging from within the campaign server will be logged in this directory, in the files. (The logging system can be somehow tweaked by modifying the source code of the logging subsystem.) |
| conf/* | NFC Chat | This directory contains configuration files for the chat subsystem “NFC”, included all players' passwords (encrypted). |
| BannedIP.txt | NFC Chat | This file contains a list of IPs you banned from the server. |
| *.htm* | Output | These HTML files contain data about the campaign that can be published easily with an HTTP server (like apache or IIS). |
| News.rdf | Output | This RSS file contains “news” about the ongoing campaign: conquers and faction scores. This can be published like the ones above, and user browsers fetch the updated version automatically. |
| data/iplist.txt<br>data/countrynames.txt | IP/country | In case you want to convert the IP of a player into his (supposed) country name, these are the files you use. The feature can be turned on and off in the main campaign configuration file. |
| data/factions.xml | Campaign | XML list of factions allowed in the campaign, with additional data like long name, short name, symbol, faction color, web page link, etc. |
| data/planets.xml | Campaign | XML list of planets and factories. You already know about that. |
| data/terrain.xml | Campaign | XML list of continent types. Each continent in “planets.xml” is assigned a type, and here those types are defined in terms of terrain features. It only works if the campaign is set to work with maps generated on the fly (RMG). |
| data/*names/* | Campaign | Text files containing lists of pilot and task names. Can be edited at will. The pilot lists are split by faction. |
| data/mails.txt | PM | This file stores all the “undelivered” private messages sent to offline players. The file is plain text. |

## 9. Unit sheets

Another class of data that you must provide are the actual units used in battles. These files MUST be shared between the server and client, exactly as they are.

The MegaMek engine understands many types of unit sheets: you can mix and match them in your distribution, but you must be sure to deliver the same exact copy to clients, or Bad Things (TM) will happen.

| ***Extension*** | ***Created with*** | ***Comment*** |
|---|---|---|
| MTF | - | Originally, it was the official MegaMek data sheet. The file content is smply plain text with some markers. Consider it deprecated. |
| BLK | MekMaker | Another plain text format. Resembles XML, somehow. Too bad, MekMaker isn't supported anymore. Bulk of unit sheets are in BLK format. |

| *Extension* | *Created with* | *Comment* |
|---|---|---|
| HMP | Heavy Metal Pro | Binary only data file, editable only with the commercial program “Heavy Metal Pro”, the de-facto standard in Classic BattleTech universe for unit creation and validation. The official unit sheets are published in HMP format. There's a small utility, provided with MegaMek, to “read” HMP files. |
| XML | The Drawing Board | XML files are exported by TDB using the appropriate export function. They can be hand edited, either with a text editor or an XML editor. The file size is larger compared to others. |

Besides providing the same unit sheets you use with the server to the clients, here are some hints to improve the task of creating an unit sheets set:

- Avoid duplicates. As easy and simple this suggestion might seem, you will save a lot of space (and thus, time to startup, download or update to your players) by choosing only ONE file type for every unit.
- Where possible, use the file types you feel more comfortable with. If you own HMP, go for its datafiles. If you like to be able to quickly edit your units, go for a text file format. If you don't care about file size, XML will be likely be the default file type for MegaMek (and thus MekWars) in the future.
- Remember that you need to include all the units you want to use with the build tables (see below), but also that units that are NOT in the tables will be never generated. So, take extreme care in assembling your set: you risk wasting space if you include unused units, or cripple players if you don't include actually generable ones.
- You can bundle unit datasheets into ZIPped files. Since data files are many and small, they tend to clog the directories they reside in. MegaMek and MekWars can read data files from within ZIP archives. ZIPping all the files into archives has the advantage of using less disk space, and the disadvantage of not being able to add/update single sheets, but force the player to download the entire archive anew if you modify something.

## 10.Build tables

Further down with the files you need for your server (and your server only – you won't distribute them to the clients), here we have the build tabes. These simple text files are lists of units (whose filenames MUST match the unit sheet files described in 9). Besides each unit there are some numbers; these numbers indicate the chance of getting that particular unit while the server generates an appropriate entity for the facton/weight class.

The directory where the build tables are found is:

```
data/buildtables/<build_era>/
```

It is strongly recommended to put all your build tables into a single “era” directory (“3025”, “3050”, “dark-age” or whatever your server time window is are good names). The increased flexibility of build tables split by era might be interesting to explore later. As of now, the added headaches granted by need of balancing multiple tables discourages such approach.

The actual filenames follows this pattern:

```
{<faction>|<rarity>}_<weightclass>[”Infantry”|”Vehicle”].txt
```

For example, we can have these files:

```
Common_Medium.txt
Rare_Assault.txt
Rare_HeavyVehicle.txt
FederatedSuns_Light.txt
FederatedSuns_LightInfantry.txt
FederatedSuns_Medium.txt
FreeWorldsLeague_AssaultVehicle.txt
```

...and so on. You usually have nine files per faction/rarity: four for meks, four for vehicles and one for infantry. Notice that the build tables are used only if your galaxy has actually factories able to produce the units listed within the tables. Having a “MagistracyOfCanopus_HeavyInfantry.txt” file, and actually NO heavy infantry units or factories for that faction, just wastes space on disk, and your time.

There are two types of build tables, the old and the new ones. Which one you are actually using is specified with the campaign variable

```
UseOldStyleBuildTables={TRUE|FALSE}
```

The difference between the old and the new table system is how the chance for each unit is noted in the file. With the old method, you have build table files consisting of lines such as:

```
...
09 11 Marauder MAD-3R.blk
12 20 Archer ARC-2R.blk
21 25 Phoenix Hawk PXH-1.blk
...
```

Each line has three items: the first one and the second ones are the “range” for an hypotetical percentile roll. If the roll falls within the indicated numbers, then the third field shows the actual filename of the unit to be used (beware of using the REAL filename, with appropriate file type and extension). Notice that “12 20” means actually NINE percent chance: both 12 and 20 are values for which the Archer would be built.

This system has various major drawbacks: assigning actual percentage to each unit is somehow complex and counterintuitive; the global percentage range must start with “00” and end with “99”, thus forcing admins to tweak entire tables just to add/remove a single unit; calculating relative chance between units is a test of statistics by itself.

The new method introduces “relative chance”, and ditches percentage rolls:

```
...
3 Marauder MAD-3R.blk
9 Archer ARC-2R.blk
5 Phoenix Hawk PXH-1.blk
...
```

As you can see, to each unit is assigned a single value: that's the relative chance of being built, compared to other units. In the example, roughly every MAD built, the same table should create three ARCs. The total of the relative chances is the actual number rolled: it can be higher or lower than 100, and the server computes the total by itself, without your intervention. With this system, it's easy to add/remove units, check relative “weight” of each unit in the building process and “see” the balance with a quick glance. The drawback is that you cannot say “Archer has a 13% chance of being built”, because, as said, the metric is no longer a percentage.

## 11.Maps

If you want to use static maps (either tiled or picked, as opposed as dynamically generated random maps), you need a place where the map data is stored. You store the single boards/tiles in the directory

```
data/boards/
```

Make sure that clients have a current copy of all the maps you want to use in your server. To edit maps, it's suggested to use the MapEditor shipped with any MegaMek package. Since there's a MegaMek package “within” the server distribution, you can invoke the client (and the map editor included within) with this command:

```
java -jar MegaMek.jar
```

The map editor is somewhat complex to operate correctly. There is some specific documentation about it in the “Megamek Docs” folder. It is beyond the scope of this howto to teach you how to draw good maps. There are good maps around the internet, available for free or with free licensing.

## 12.Camouflage pictures

You can provide camo patterns more or less at will, by adding them in the directory

```
data/camo/
```

in the client package. You don't need them in the server at all. You can nest directories in the “camo” dir, more or less at will. Camo pics are available in a lot of places, and are easy to draw. The available format is GIF. Remember that both players need to have the same camos placed in the same dirs to be able to see each other's camouflages.

## 13.Unit pictures

Another important part of the client package are unit pictures. These files, and the configuration used to use them, are again relevant only for the client side: you don't need any picture on the server. The files must be stored starting with this directory:

```
data/mex/
```

and you can use any subdirectory you want. The files must be GIF ones. Again, there are a lot of pictures freely available around; check the MegaMek project site for a good number of them (all level1 and lots of level2 units are included in the MegaMek data package distribution).

Special treatment must be reserved to the file “data/mex/mechset.txt” because this files links the actual units to the pictures. The file is yet another plain text file, containing lines that start with one of three keywords:

```
exact <string> <image_filename>
```

Assigns the image file to the unit named exactly matches the string;

```
chassis <string> <image_filename>
```

Assigns the image to any unit whose name contains the string, if no “exact” already matches the unit;

```
Include <another_mechset_filename>
```

Process the named file as it was an extension of the original mechset.txt; useful for instance when the bulk of the units are fixed, and the server owner wants to tweak a custom list, or to let players use their own custom lists of matches without overwriting the distributed file.

For the latter, it's always a good idea to write in the last line of your mechset.txt something like:

```
include local_mechset.txt
```

WITHOUT actually providing the file in the client package, so that players can put there their own customizations, and still be able to have them in their games. In such cases, remember to document this feature where your players can know about it!

## 14.Comparision table between client and server files

As we've seen, some files are to be used exclusively on the server, while others are used only on the client, and finally some more needs to be the same in both sides. The following table summarizes what and where must be distributed:

| ***What*** | ***Where*** | ***Who*** | ***Sync?*** |
|---|---|---|---|
| Planets and terrain data | data/planets.xml<br>data/terrain.xml | Server | No |
| Units (BLK, HMP, MTF, XML) | data/mechfiles/* | Both | YES |
| Camos | data/camo/* | Client | No |
| Pictures | data/mex/*<br>data/mex/mechset.txt | Client | No |
| Build tables | data/buildtables/<era>/* | Server | No |
| Maps | data/boards/* | Both | YES |
| Hex pictures | data/hexes/* | Client | No |
| Misc images | data/images/* | Client | No |
| Other data | data/* | Client | No |
| Other server files | (see paragraph 8) | Server | No |

Remember that you must prepare a client package containing enough datafiles to let it connect to your server and play on it. My suggestion is to create a minimal “prototypical” client directory, and then add there the data you want, eventually copying (or better, linking!) from the server directory the files that must be kept in sync, and zipping it for distribution. This client directory must not be used to actually LAUNCH a client from, or else you risk to pollute it with your own configuration files, cache files, live data, etc.

## 15.A word of advice about upgrading

The following text is copied as-is from a mail urgru sent to me. I don't necessarily agree on some of his viewpoints on the matter, but he has good points, and I report it here:

*“Versioning is convoluted. There is a readme in the client's “/MekWars Docs/For Sever Ops/” folder which should be included in the server documentation package, when there is one...*

*Upgrades should be done only when needed. Features may not be useful on your server. If a version changes to add new support for omnimechs, and you run a 3025 server, for the love of god... don't upgrade. Let the people who NEED things take the risks with new software. Same thing for MM versions.*

*If you ARE upgrading, TEST FIRST if you have any doubt whatsoever about a feature. Complex features are more often than not going to have bugs when first introduced. Devs test what they can, but problems naturally get through. Risk/reward for updating is a matter for individual operators to consider.*

*Upgrade in a staggered fashion when possible. Update MM seperately from MekWars, and vice versa, unless a new MM version is required to run a new MekWars version. Staggering upgrades makes tracking problems substantially simpler than dual upgrades.”*