Technical Description

Solution (under construction)

Clear separation between MapTool system and usage data

See User Guide Startup Properties.

Option on startup to start a server

- Add startup property to start the last server on startup
- Add startup property to define how the startup sequence should handle a auto backup file
- Find the correct place to start the server in the startup sequence (tricky, because there are already threats running)

Refactor: UI startup frame

- Separate user from system settings, users first
- Add new values
- Show environment values for memory, language, ...
- Describe priority of places
- Describe cmdOptions
- Describe startup.prefs
- Describe usage of MapTool.cgf
- Describe keys for advanced options

Implementation Decisions

Startup Sequence & Main Classes

The app starts with main in LaunchInstructions. To start the logging as early as possible, at that point the data-dir is already needed.
Most of the system runtime environment is handled in the AppUtil class, incl. the data-dir, the log-dir and OS specific things.
The cmd-options are used in the main method of the MapTool class, which is called near the end of the main of LaunchInstructions. To late for us.

AppUtil holds 2 different kinds of util methods, so it would make sense to separate that class into two, but I'm a newbie in MapTool development, so I will not do that now.

I decided to

- Let return values in the AppUtil class.
- Create App Properties to manage the properties for boolean and String, with (deprecated) references in the AppUtil. (Later easy to refactor in the rest of MapTool)
- Create CmdOptionProperties and StartUpProperties classes to manage there values
- Rearrange the startup methods (& logs), so that the properties are initialized first, second the main directories (data & logs), then the logging and at last call MapTool.main().

UserJvmOptions Class

This class handles the options currently adjustable in the MapTool.cfg file.I guess set all other properties in this file should not be changed by a user.

This options can be seperated into 3 groups

- **Real JVM Parameters**
  Can only be set during the call of the JVM engine and must be set outside the application (e.g. memory)
  This can be handled only inside the cfg file or as java arguments (e.g. -X) during application call.
- **Internal Parameters**
  Parameters a user should only change, if he has problems (e.g. graphic engine). I guess most of them need a restart to make a change active.
  To be on the save side, this should also handled only inside the cfg file or as java arguments (e.g. -X) during application call.

- **User Parameters**
  At least an advanced user want to have control over them (e.g. DATA_DIR, Language). Hard to change them after initialization, so also restart to make a change active.
  This should be handled outside the cfg-file inside a startup.properties file and/or cmdOptions. For downward compatibility an initial value can also be set in the cfg file or by -D java arguments.

The current UserJvmOptions is therefore a mix of JVM, library and application options and should be refactored. Since it has also a lot of methods to handle the cfg file values, I would at least recommend an renaming.
A first closer check shows that this class is mainly used in the PreferenceDialog, so it should be refactored, when the last part (Refactor UI startup frame) will be done.

Implementation Questions

Naming Conventions

- Abbreviations: Where are they ok, where not e.g. usage of properties and props

Testing

Did not found any rules about testing.

Definitions of Done

For the definition of done (a pull request is allowed) it looks like the only request is the correct formatting of the code?

Found Bugs

cmdOption -r -> Exception

To the time the reset method is called, the 'jvmNodeConfiguration' is null. - Found in productive Version 1.12.2

cmdOption -w is double defined

```
cmdOptions.addOption("m", "monitor", true, "sets which monitor to use");
...
cmdOptions.addOption("m", "macros", false, "display defined list of macro functions");
```

Solution

Since the last one works, I change m to g for graphical device (monitor).

```
cmdOptions.addOption("g", "monitor", true, "sets which monitor (graphical device) to use");
...
cmdOptions.addOption("m", "macros", false, "display defined list of macro functions");
```

Start-Properties / JvmOptions not Shown under Windows

At least since 1.12.2 the start properties are not shown in the OptionDialog under Windows.

They are also not editable under Windows, even if the file is in a editable directory.

Under Development the hole tab is not shown. That makes development for the dialog hard.

Recommended Refactoring Issues

Things I feel very obscure, so I would suggest, that say should be refactored, removed or be adjusted a little.

I set to @Deprecated and add a @deprecated JavaDoc comment, where I thing, that they can be removed because say are not used or there exist an alternative implementation.

Some of them are just clean ups, where I let the old method, variable just to hold the first check in small.
I let them also as they are, because I do not know, if the main developers will integrate this issue. If not, I have to hold the changes up to date via merge and then a smaller impact is better.

AppUtil.getAppHome() vs AppUtil.getDataDir()

I do not care about AppHome or DataDir or maybe DataHome, but it should be named consistent. If we thing about that the subdir can be speeded in the future to 3 or more locations the hole thing lost his charm anyway.

But AppHome(subdir) is used very often (~33 times), so this will be hard work anyway.

.

AppUtil.getDataDirAppCfgFile() & UserJvmOptions.copyConfigFile()

This is a helper function to copy the original maptool.cfg from the app directory into the 'DataHome/config' directory as a BACKUP. But it is not renamed and so a user does not now, that the file in the data directory is never used.

Conclusion: Rename the file to backup_from_currentDate&Time and write it into a backup directory.

Notes

Have not clean up the order of methods and variables.

Using MapTool.cfg during development

Its looks like the current cfg file is not generated during the normal build and it is not accessible from IntelliJ run/debug. Does somebody know, how a cfg file can be used during development?

Translations Yes or No

The developer team uses a pragmatic rule: If the information is useful for the user, so he can (re-)act with the information, translate it, otherwise it should be in English, so that it is useful for the developer team.

Examples

CmdLineOption: Description

The description is only in English. Maybe you want to use translation encoding?
I use the description for the help view, so this is also only in English.

Answer: Yes

MapTool.showError()

Should I use a translation here?

MapTool.showError(“Unexpected IOException during load of startup properties.”, catchedIOException);

Answer: No

Log vs. MapTool.show...()

Use the same rule as for translations and thing about it, if the value of the information is high enough for the user to commit it. The log is just written into the file, the dialog needs interaction!