Commands/settings

You can adjust the way CoppeliaSim operates via a plugin or an add-on, but you can also modify its behaviour via:

- command line arguments
- the overall settings file
- environment variables
- the status bar commander

Additionally, you can also interact with CoppeliaSim via script functions and the commander.

Command line

When you start CoppeliaSim via the command line, you have following command line options that are supported:

- **-v<verbosity>**: sets the verbosity level, in the console. Default is **loadinfos**. Other accepted values are **none**, **errors**, **warnings**, loadinfos, **scripterrors**, **scriptwarnings**, **scriptinfos**, **infos**, **debug** , **trace**, **tracelua** and **traceall**. Plugins should output messages via simAddLog , scripts via sim.addLog. Verbosity can change during runtime: from within CoppeliaSim, global verbosity can be adjusted with the consoleVerbosity property, and plugin verbosity with sim.setPluginInfo(pluginName,sim.plugininfo_verbosity,verbosity). By default, plugin verbosity follows global verbosity. Command line verbosity setting can be overridden via the *verbosity* value in *system/usrset.txt*. Additionally, console log messages can be filtered via the *consoleLogFilter* value in *system/usrset.txt*.
- **-w<verbosity>**: similar to the **-v** setting above, but for the verbosity level in the status bar. Default is **scriptinfos**. Status bar verbosity setting can be overridden via the *statusbarVerbosity* value in *system/usrset.txt*.
- **-x<verbosity>**: similar to the **-v** or **-w** setting above, but for the verbosity level for simple dialogs. Default is **infos**. Other accepted valuesare **none**, **errors**, **warnings** and **questions**. Dialog verbosity setting can be overridden via the *dialogVerbosity* value in *system/usrset.txt*.
- **-c<scriptString>**: executes the script string as soon as the sandbox is initialized.
- **-H**: runs CoppeliaSim in *true* headless mode (i.e. without any GUI or GUI dependencies). A display server is however still required. Instead of using library *coppeliaSim*, library *coppeliaSimHeadless* will be used. Keep in mind that in that case, vision sensors won't operate, unless they use the Pov-Ray rendering mode (and the Pov-Ray plugin is installed, binaries available here), and that rendering will be drastically slower.
- **-h**: runs CoppeliaSim in an *emulated* headless mode: this simply suppresses all GUI elements (e.g. doesn't open the main window, etc.), but otherwise runs normally.
- **-s<simulationTimeInMM>**: automatically start the simulation and runs it for a certain amount of milliseconds. Use a value of 0 to disable automatic stopping.
- **-q**: automatically quits CoppeliaSim after the first simulation run ended.
- **-a<addOn.lua>** and/or **-b<addOn.lua>**: loads and runs an additional add-on specified via its filename.
- **-G<key>=<value>**: named parameter, can be queried within CoppeliaSim with named parameter properties. With **-Glicense=licenseKey** you can enable a specific license key string, with **-GpreferredSandboxLang=python** you can specify the sandbox language (lua or python).
- **-g<string>**:represents an optional string argument that can be queried within CoppeliaSim with the appArg* properties.
- **-O<bitCoded>**: disables specific GUI items.
- **-f<scene.ttt>** or **-f<scene.simscene.xml>**: loads a CoppeliaSim scene.
- **-f<model.ttm>** or **-f<model.simmodel.xml>**: loads a CoppeliaSim model.

For example, to start CoppeliaSim in headless mode, load the scene *myScene.ttt*, run the simulation for 5 seconds, then stop the simulation and automatically leave CoppeliaSim again, type from within the CoppeliaSim main folder:

```
Windows:
$ coppeliaSim.exe -h -s5000 -q myScene.ttt

Linux:
$ ./coppeliaSim.sh -h -s5000 -q myScene.ttt

Mac:
$ ./coppeliaSim.app/Contents/MacOS/coppeliaSim -h -s5000 -q ../Resources/myScene.ttt
```

Overall settings file

When CoppeliaSim starts, the file *usrset.txt* is read and values applied. Settings apply to various areas, such as:

- debugging
- rendering/display
- directories
- serialization
- messaging
- compatibility
- floating licence
- etc.

The location of file *usrset.txt* can be queried via the settingsPath property

Environment variables

Following environment variables allow to modify the behaviour of CoppeliaSim:

- COPPELIASIMPLUGIN_IGNORE_MISSING_SYMBOLS. If defined, then plugins will ignore missing

symbols

- COPPELIASIM_USER_SETTINGS_FOLDER_SUFFIX. Will append a suffix to the user settings folder. This can be used to start several CoppeliaSim instances, with different user settings.
- COPPELIASIM_CONSOLE_LOG_FORMAT. The format of the console log, e.g. "[{origin}:{verbosity}] {message}"
- COPPELIASIM_STATUSBAR_LOG_FORMAT. The format of the status bar log, e.g. " [{origin}:{verbosity}] {message}"
- COPPELIASIM_STATUSBAR_LOG_FORMAT_UNDECORATED. The format of the status bar log when *undecorated* is specified, e.g. "{message}"

Status bar commander

The commander, implemented via the simCmd plugin, is a read-eval-print loop, that adds a text input to the CoppeliaSim status bar, which allows entering and executing Python or Lua code on the fly, like in a terminal. The code can be run in the sandbox, or any other active script in CoppeliaSim. The behavior of the plugin can be controlled via [Modules > commander]. It also offers its own API functions. Additionally, *help()* will display usage tips.