MegaMol™ is a visualization middleware used to visualize point-based molecular datasets. This software is developed by the MegaMol™ team, at the TU Dresden, Germany, and the University of Stuttgart, Germany. The project was started in the Collaborative Research Center 716, subproject D.3, at the Visualization Research Center (VISUS), University of Stuttgart, Germany. Development involves the Computer Graphics and Visualization Group of the TU Dresden, Germany.

This user's manual contains information about the usage of the MegaMol™ framework for common visualization tasks. It addresses end users of the software and, in parts, developers of module for MegaMol™. This document describes the software version 1.1 available though the project's website:

http://megamol.org
http://www.sfb716.uni-stuttgart.de
http://www.visus.uni-stuttgart.de/institut.html
http://tu-dresden.de/die_tu_dresden/fakultaeten/fakultaet_informatik/smt/cgv

MegaMol™ was partially funded by

- German Research Foundation (DFG), Collaborative Research Center SFB 716,
- European Social Fund (ESF) Project #100098171,
- German Federal Ministry of Education and Research (BMBF) as part of the competence center for Big data ScaDS Dresden/Leipzig (01IS14014), and
- Nvidia GPU Cluster of Excellence, Dresden.

This user's manual is published by the MegaMol™ team for free distribution.

This work is licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.
Contents

1 Overview 5
  1.1 License ......................................................... 5

2 Installation and Setup 7
  2.1 Pre-built Binaries ............................................. 7
  2.2 Building from Source Code – MS Windows ....................... 7
    2.2.1 Prerequisites .................................................. 8
    2.2.2 Vislib & Libs .................................................. 8
    2.2.3 Core ............................................................ 9
    2.2.4 Console Frontend .............................................. 9
    2.2.5 Plugins ........................................................ 10
    2.2.6 Configurator GUI ............................................. 10
  2.3 Building from Source Code – Linux ............................. 11
    2.3.1 Prerequisites .................................................. 11
    2.3.2 Vislib & Libs .................................................. 11
    2.3.3 Core ............................................................ 12
    2.3.4 Console Frontend .............................................. 13
    2.3.5 Plugins ........................................................ 14
  2.4 Configuration ................................................... 15
  2.5 Test ............................................................. 16

3 Viewing Data Sets 19
  3.1 Views, Modules and Calls ...................................... 19
    3.1.1 Modules and Calls ............................................. 20
    3.1.2 Views ............................................................ 20
  3.2 Project Files ..................................................... 21
  3.3 Parameter Files ................................................... 22
  3.4 View Interaction ................................................ 23
  3.5 Making High-Resolution Screenshots ............................ 23
  3.6 Making simple Videos ........................................... 24

4 Jobs: Converting Data 25
  4.1 Jobs .............................................................. 25
  4.2 Converting to MMPLD ............................................. 25

5 Advanced Usage 27
  5.1 Configurator ...................................................... 27
    5.1.1 Class Data Base .............................................. 27
    5.1.2 Editing .......................................................... 28
    5.1.3 Starting MegaMol ............................................. 29
  5.2 Alternative Build for Testing ................................... 29
    5.2.1 VISlib and visglut .......................................... 30
## Contents

5.2.2 MegaMolCore .................................................. 30  
5.2.3 MegaMolConsole ........................................... 31  
5.2.4 Some MegaMol Plugin ................................. 31

### A MegaMol Configuration files  
A.1 MegaMol configuration file megamol.cfg ............... 33  
A.2 Data Viewing Project file dataview.mmprj .......... 34  
A.3 Data Conversion Project file makemimpld.mmprj .... 35
Chapter 1

Overview

MegaMol™ is a visualization middleware used to visualize point-based molecular datasets. This software is developed by the MegaMol™ team, at the TU Dresden, Germany, and the University of Stuttgart, Germany. The project was started in the Collaborative Research Center 716, subproject D.3, at the Visualization Research Center (VISUS), University of Stuttgart, Germany. Development involves the Computer Graphics and Visualization Group of the TU Dresden, Germany.

The goal of the project is to provide a software base for visualization research and to provide a stable environment to deploy newest visualization prototypes to application domain researchers.

MegaMol is not a visualization tool.
MegaMol is a platform for visualization research.

Visit the project website for downloads and more information: http://megamol.org

1.1 License

MegaMol™ is freely and publicly available as open source following the terms of the BSD License.

Copyright (c) 2015, MegaMol Team
TU Dresden, Germany
Visualization Research Center, University of Stuttgart (VISUS), Germany
Alle Rechte vorbehalten.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the MegaMol Team, TU Dresden, University of Stuttgart, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE MEGAMOL TEAM "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE MEGAMOL TEAM BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Chapter 2

Installation and Setup

This chapter discusses installation and setup of MegaMol\(^\text{TM}\), either from the pre-built binary packages or from the source code repository. The later is, however, meant for experience users.

MegaMol\(^\text{TM}\) targets Microsoft Windows (Windows 7 or newer, x86 and x64) and Linux (x64) as supported environments. Currently, Ubuntu is used as Linux distribution for development. Further platforms are not considered during the development. While MegaMol\(^\text{TM}\) might work on further platforms, the development team will currently not grant any support for problems on these environments.

2.1 Pre-built Binaries

The MegaMol\(^\text{TM}\) project provides packages of pre-built binaries for the following platforms:

- MS Windows x64 (built with MSVC 2013)
- MS Windows x86 (built with MSVC 2013)
- Ubuntu Linux 15.4 (built on Gnome Ubuntu 15.4, with GCC 4.9.2 and libc.so.6 v.2.21)

You can download the corresponding package from the project website\(^1\). Note that the pre-built binaries for Ubuntu Linux should work on most other Linux distributions as well, as long as the runtime library versions are compatible. If you use a platform not provided, you still can build MegaMol\(^\text{TM}\) from source, following the instructions in sections 2.2 and 2.3.

To install, just unpack the downloaded MegaMol\(^\text{TM}\). Note that there is a subdirectory \texttt{MegaMol} in all packages.

You can also download pre-built debug versions. These packages are not complete by themselves but only contain the debug versions of the executable and library files. You need to unpack the release version first and then unpack the debug version into the same directory.

We also recommend that you download the \textit{script and examples} package from the website. Unpack this into the same folder you unpacked MegaMol\(^\text{TM}\) to.

After unpacking you can continue with configuring MegaMol\(^\text{TM}\) following the description in section 2.4.

2.2 Building from Source Code – MS Windows

This description is also available online\(^2\) where the description might be updated. MegaMol\(^\text{TM}\) consists of several binaries: a core library (Dll), an executable front end, and several plugins (Dlls).

---

\(^1\)MegaMol\(^\text{TM}\) project website; release v1.1 download page: \url{https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/Release-v1_1}

\(^2\)MegaMol\(^\text{TM}\) project website, building instructions: \url{https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/HowToBuild11}
In addition, several libraries are required for the build process.

### 2.2.1 Prerequisites

To build MegaMol™, you need several programs installed on your system:

- Subversion Client (recommended: [http://tortoisesvn.net/index.html](http://tortoisesvn.net/index.html))
- Git Client (for some plugins)
- Microsoft Visual Studio 2013
- Nuget Extension for Visual Studio (Version 2.8 or higher is recommended)

### 2.2.2 Vislib & Libs

**Vislib**

This is the primary base library used by MegaMol™.

- Check out per SVN: [https://svn.vis.uni-stuttgart.de/utilities/vislib/trunk](https://svn.vis.uni-stuttgart.de/utilities/vislib/trunk)
- Open vislib.sln in Visual Studio
- Build the lib
  
  - Select BUILD > Batch Build
  - Select in the list on the left the four entries for Project vislib by setting the ticks in the right most column (You can simply click on Select All if you want to build all tests too).
  - Click on Build (or Rebuild if you experience problems.)
  - Note: Required dependencies are downloaded via Nuget. If you set up your tool incorrectly, you might experience problems here.

**Visglut**

This is a modified version of the freeglut³.

- Check out per SVN: [https://svn.vis.uni-stuttgart.de/utilities/visglut/trunk](https://svn.vis.uni-stuttgart.de/utilities/visglut/trunk)
- Open Subdir build_windows_msvc2013
- Open visglut.sln in Visual Studio
- Build the lib
  
  - Select BUILD > Batch Build
  - Click on Select All
  - Click on Build (or Rebuild)

**Further Libs**

Download the zip of further precompiled libraries: msvc2013 12.0.21101.00 u4.zip. You find the link at the corresponding website: [https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/MegaMol11WindowsLibs](https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/MegaMol11WindowsLibs) Extract the contained folder.

If you experience problems and need to rebuild some of these libs yourself, use the naming convention used in this package.

2.2 Building from Source Code – MS Windows

2.2.3 Core
This is the central library of MegaMol™.

- Checkout: https://svn.vis.uni-stuttgart.de/projects/megamol/core/trunk
- Run configure.win.pl (or config.bat)
  - Enter your configurations and paths
  - Paths should be found automatically, in which case you simply enter the number of the found path. Paths are searched asynchronously. Thus, if your path was not found, wait a bit and then hit ?. New elements might be found by then. If both **Currently searching** and **Folders still to search** are both empty, the search has finished. If your folder was not found, you then must enter it manually.
- Open Core.sln in Visual Studio
- Build the Core
  - Select BUILD > Batch Build
  - Click on Select All
  - Click on Build (or Rebuild)
- Note: No config file will be created (see section 2.4).

2.2.4 Console Frontend
The console frontend is the most simple and primary front end executable of MegaMol™.

- Checkout: https://svn.vis.uni-stuttgart.de/projects/megamol/frontends/console/trunk
- Run configure.win.pl (or config.bat)
  - Enter your configurations and paths
  - Paths should be found automatically, in which case you simply enter the number of the found path. Paths are searched asynchronously. Thus, if your path was not found, wait a bit and then hit ?. New elements might be found by then. If both **Currently searching** and **Folders still to search** are both empty, the search has finished. If your folder was not found, you then must enter it manually.
- Open MegaMolConsole.sln in Visual Studio
- Build the Console
  - Select BUILD > Batch Build
  - Select in the list on the left the twelve entries for Projects Console, Glut and Viewer, by setting the ticks in the right most column (You can simply click on Select All if you want to build the deprecated WGLViewer too).
  - Click on Build (or Rebuild)
- **KNOWN BUG**: The AntTweakBar.dll will not be copied on the first build run. If you don’t have this Dll in your bin directories, you need to rebuild the Console project once again.
- Note: No config file will be created (see section 2.4).
Chapter 2  Installation and Setup

2.2.5  Plugins

All plugins basically use the similar build process. Differences should only emerge in number and type of settings variables and paths to different libraries. Consult the documentation of plugins, if they do not stick to this process. This description uses mmstd_moldyn as example.

- Checkout:
  [https://svn.vis.uni-stuttgart.de/projects/megamol/plugins/mmstd_moldyn/trunk](https://svn.vis.uni-stuttgart.de/projects/megamol/plugins/mmstd_moldyn/trunk)

- Run configure.win.pl (or config.bat)
  - Enter your configurations and paths
  - Paths should be found automatically, in which case you simply enter the number of the found path. Paths are searched asynchronously. Thus, if your path was not found, wait a bit and then hit ?.. New elements might be found by then. If both Currently searching and Folders still to search are both empty, the search has finished. If your folder was not found, you then must enter it manually.

- Open MegaMolConsole.sln in Visual Studio

- Build the Plugin
  - Select BUILD > Batch Build
  - Click on Select All
  - Click on Build (or Rebuild)

- Note: No config file will be created (see section 2.4).

2.2.6  Configurator GUI

The Configurator GUI is an editor written in C# for MegaMol™ project files (*.mmpld). Further details can be found in section 5.1. The Configurator can be run on Linux using Mono⁴. However, we never tested to compile the Configurator with Mono and do not support this process.

- Checkout: [https://svn.vis.uni-stuttgart.de/projects/megamol/supplement/MegaMolConf/trunk](https://svn.vis.uni-stuttgart.de/projects/megamol/supplement/MegaMolConf/trunk)

- Run configure.pl
  - Enter your configurations and paths
  - Paths should be found automatically, in which case you simply enter the number of the found path. Paths are searched asynchronously. Thus, if your path was not found, wait a bit and then hit ?. New elements might be found by then. If both Currently searching and Folders still to search are both empty, the search has finished. If your folder was not found, you then must enter it manually.

- Open MegaMolConf.sln in Visual Studio

- Build the Configurator
  - Select BUILD > Batch Build
  - Click on Select All
  - Click on Build (or Rebuild)

⁴The Mono project: [http://www.mono-project.com/](http://www.mono-project.com/)
2.3 Building from Source Code – Linux

This description is also available online\(^5\) where the description might be updated. MegaMol\(^\text{TM}\) consists of several binaries: a core library (shared object), an executable front end, and several plugins (shared objects). In addition, several libraries are required for the build process.

2.3.1 Prerequisites

To build MegaMol\(^\text{TM}\), you need several programs installed on your system:

- Subversion client
- cmake in version 2.8.12 or higher
- gcc in version 4.7 or newer (for c++11 features)
- dev headers for several libraries:
  - OpenGL including X11
  - libpng
  - zlib
  - expat
  - Glut
  - perl (see below)

Perl with XML

Currently, perl needs the module 'XML::Parser'. This module is not installed per default, so you need to install it yourself:

```
sudo perl -MCPAN -e "install XML::Parser"
```

2.3.2 Vislib & Libs

Vislib

This is the primary base library used by MegaMol\(^\text{TM}\).

- Checkout: `svn co https://svn.vis.uni-stuttgart.de/utilities/vislib/trunk vislib`
- Build per cmake. For simplicity use the supplied shell script:
  - Recommended: `./cmake_build.sh -dcm`
  - Default behavior: cmake & make for the release version
  - Help on additional arguments: `./cmake_build.sh -h`
  - Add `-d` if you want to also build the debug version
  - `-c` invokes cmake
  - `-m` invokes make
  - If you want to install the vislib, you can specify an extra install prefix (e.g. `-p /_inst`)

\(^5\)MegaMol\(^\text{TM}\) project website, building instructions: [https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/HowToBuild11](https://svn.vis.uni-stuttgart.de/trac/megamol/wiki/HowToBuild11)
Chapter 2  Installation and Setup

* ./cmake_build.sh -dcm configures and makes debug and release versions of the vislib
* sudo ./cmake_build.sh -di installs debug and release versions as superuser

- Usually this process only fails if headers of any other library mentioned in the prerequisites are missing, or if cmake, the compiler or perl do not meet the requirements.

Visglut
This is a modified version of the freeglut\(^6\).
- Checkout: svn co https://svn.vis.uni-stuttgart.de/utilities/visglut/trunk visglut
- cd visglut/build_linux_make
- make
- Note that there is no install target.

AntTweakBar
The AntTweakBar is not required but recommended.
- Download current source package (e.g. v1.16): http://anttweakbar.sourceforge.net/doc/
- Extract files
- cd AntTweakBar/src
- make
- Note that the AntTweakBar does not have an install target.

2.3.3 Core
This is the central library of MegaMol\(^{\text{TM}}\).
- Recommended: create a separate subdirectory for all MegaMol\(^{\text{TM}}\) projects
  - For example, cd
  - mkdir megamol
  - cd megamol
  - This can be also the place for a dedicated install directory.
- Checkout: svn co
  https://svn.vis.uni-stuttgart.de/projects/megamol/core/trunk core
- Build; use the supplied shell script, similar to the one of the vislib.
  - Recommended: ./cmake_build.sh -dcmip /megamol/bin
  - Help on additional arguments: ./cmake_build.sh -h
  - Add -d if you want to also build the debug version
  - -c invokes cmake
  - -m invokes make
  - -i invokes "make install"
  - The install prefix is specified with -p <path>

\(^6\)Freeglut project website: http://freeglut.sourceforge.net/
2.3 Building from Source Code – Linux

Notes on Install

- Copies the compiled core binary to `<prefix>/lib/megamol/
- Copies the core headers files from the `include/mmcore` subdirectory to `<prefix>/include/mmcore. Actually the content of include is copied recursively.
- Copies shader files from the `Shaders` subdirectory to `<prefix>/share/megamol/core/Shaders/
- Copies several cmake-related files to `<prefix>/share/cmake/MegaMolCore/

Notes on alternative cmake commands

- For temporary test builds, which should not change the system in any way:
  `./cmake_build.sh -dcnv ../vislib -C -DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE`
- `-d` also works on the debug version
- `-c` invokes cmake
- `-n` does not register the build tree in the cmake package repository
- `-v` searches for the vislib there
- `-C` cmake commando
  - `-DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE` suppresses searching for MPI. This is helpful if MPI is installed on your system, but you don’t want to build against it.
- After reviewing the configuration output build: `./cmake_build.sh -dm`
  - If you experience compilation issues try to reduce the number of make jobs: `./cmake_build.sh -dmj 1`

2.3.4 Console Frontend

The console frontend is the most simple and primary front end executable of MegaMol™.

- Checkout: svn co
  `https://svn.vis.uni-stuttgart.de/projects/megamol/frontends/console/trunk
  console
  
- Build; use the supplied shell script, similar to the one of the vislib.
  - **Recommended**: `./cmake_build.sh -dcmi`
  - Help on additional arguments: `./cmake_build.sh -h`
  - Add `-d` if you want to also build the debug version
  - `-c` invokes cmake
  - `-m` invokes `make`
  - `-i` invokes "make install"
  - Note that you can omit the install prefix. Cmake will use the same install prefix of the MegaMol™ core component found.
Chapter 2  Installation and Setup

Notes on Install

- Copies the compiled binary MegaMolCon to <prefix>/bin/megamol
- Copies a stub for a MegaMol™ config file (see section 2.4) to <prefix>/bin/megamol
- Copies the compiled viewer glut library to <prefix>/lib/megamol
- Copies the startup shell script megamol.sh to <prefix>/bin
- Also creates several symbolic links for MegaMol™ to start correctly.

You might need to edit the copied config file in order to use MegaMol™ correctly. See section 2.4 for details.

2.3.5  Plugins

All plugins basically use the similar build process. Differences should only emerge in number and type of settings variables and paths to different libraries. Consult the documentation of plugins, if they do not stick to this process. This descriptions uses mmstd_moldyn as example.

- Checkout: svn co 
  https://svn.vis.uni-stuttgart.de/projects/megamol/plugins/mmstd_moldyn/trunk
  mmstd_moldyn

- Build; use the supplied shell script, similar to the one of the vislib.
  - Recommended: ./cmake_build.sh -dcmi (omit -i and don’t install, if you need to
debug your plugin)
  - Help on additional arguments: ./cmake_build.sh -h
  - Add -d if you want to also build the debug version
  - -c invokes cmake
  - -m invokes make
  - -i invokes ’make install’
  - Note that you can omit the install prefix. Cmake will use the same install prefix of the
MegaMol™ core component found.

Notes on Install

- Copies the compiled plugin binary to <prefix>/lib/megamol/
- Copies the plugin headers from the include/mmstd_moldyn subdirectory to
  <prefix>/include/mmstd_moldyn. Actually the content of include is copied recursively.
- Copies shader files from the Shaders subdirectory to
  <prefix>/share/megamol/mmstd_moldyn/Shaders/
- Copies several cmake-related files to <prefix>/share/cmake/mmstd_moldyn/
2.4 Configuration

After successfully installing or compiling MegaMol\textsuperscript{TM}, you should have all executable files inside your bin folder.

Create a file `megamol.cfg` in this `bin` directory, with the content shown in appendix A.1. Locate line 7 containing the tag `<appdir path="." />`. Although the relative path should work here fine, it is recommended to change the path in this line to the global path to the MegaMol\textsuperscript{TM} application bin directory.

Line 5 configures the log mechanism of MegaMol\textsuperscript{TM}. Adjusting the value of `echolevel` changes the amount of log information printed on the console. Specifying a log `file` and the `level` informs MegaMol\textsuperscript{TM} to write a log file and print the messages of the requested level into that file. The log level is a numeric value. All messages with lower numeric values will be printed (or saved). The asterisk stands for the highest numeric value, thus printing all messages.

All MegaMol\textsuperscript{TM} XML files support condition tags (see lines 20 to 27, for example). All tags inside these condition tags are only evaluated if the value of `cond` in the opening condition tag evaluates to `true`. One of the most basic conditions is shown in the example configuration file, i.e. the value of `debug`. This variable is `true` only if the debug version of MegaMol\textsuperscript{TM} is being run. Note the negation operator `!` to test if MegaMol\textsuperscript{TM} is not the debug version. The configuration file uses conditional tags to load the matching release versions or debug versions of all configured plugins. Extend the configuration accordingly if you introduce new plugins into your installation.

Although, there are different ways to specify the plugins to be loaded, the tags in the example configuration file are the most secure way. Each `<plugin>` tag requires three attributes:

- **path** should be the path to find the plugin in. The example configuration file assumes to find the plugins in the same directory as the MegaMol\textsuperscript{TM} executable (which is the case for Windows installations. On Linux systems you need to change this path (e.g. to `../../lib/megamol`).
- **name** is the file name of the plugin.
- **action** refers to an internal parameter of MegaMol\textsuperscript{TM} and should always be `include`.

Rendering modules from plugins require shader codes to function. MegaMol\textsuperscript{TM} searches these codes in all registered shader directories. To register a shader directory, add a corresponding tag to the configuration file, see lines 36 to 41.

The configuration file also specifies global settings variables which can modify the behavior of different modules. Two such variables are set in the example configuration file.

On line 44 the variable `*-window` is set. This variable specifies the default position and size for all rendering windows MegaMol\textsuperscript{TM} will create. The asterisk represents any window name. If you set a variable with a specific name, windows with exactly this name will respect the settings variable. For example, `test-window` will specify the value for the window created by the view instance `test`. The value itself contains five variables:
Chapter 2  Installation and Setup

- The first two variables are prefixed with x and y and specify the location of the window in screen pixel coordinates.

- The second two variables are prefixed with w and h and specify the size of the client area of the window in pixels.

- The last variable nd (stands for no decorations) will remove all window decorations, buttons and border from the created window. This variable allows to create borderless windows filling the complete screen for full screen rendering.

The second settings variable, specified on line 45, activates (or deactivates) the GUI of the console front end, that is the AntTweakBar.

This concludes the building and configuring of MegaMol™. Test your installation following the description in section 2.5.

2.5 Test

The test MegaMol™ simple start the front end executable.

Open a console and change your working directory to the MegaMol™ install directory. Start the MegaMol™ start script:

```
./megamol.sh
```

Alternatively, you can decent into the bin directory and start the front end directly. Doing so, you must ensure that the additional shared objects can be found and loaded. Enter the command:

```
LD_LIBRARY_PATH=. ./MegaMolCon
```

This direct invocation is not recommended. Thus, the remaining examples in this manual will assume that you use the start shell script.

MegaMol™ should start and should print several messages to the console. The leading number of each line, is the log level. There should be no output of warnings (log level of 100 or less) or errors (log level 1). The output should look like this:

```
MegaMol(TM) Console
Copyright (c) 2006 - 2015 by MegaMol Team: VISUS (Universitaet Stuttgart, Germany), TU Dresden (Dresden, Germany)
Alle Rechte vorbehalten.
All rights reserved.
Called: D:\Uni\Vis-Tutorial-Stick\MegaMol\portable\bin\MegaMolCon.exe
250|Path ".\share\megamol\core\Shaders" added as shader search path.
250|Configuration value "*~window" set to "w1280h720".
250|Configuration value "consolegui" set to "on".
350|Configuration sucessfully loaded from "D:\Uni\Vis-Tutorial-Stick\MegaMol\portable\bin\megamol.cfg"
350|Directory "application" is "D:\Uni\Vis-Tutorial-Stick\MegaMol\portable\bin"
350|Plugin mmstd_datatools loaded: 15 Modules, 0 Calls
200|Core Instance destroyed
```
For a better test you should invoke MegaMol\textsuperscript{TM} requesting a simple rendering. Then you can be sure that the graphics drivers, graphics libraries and shader codes are correctly found and are working.

Start:

```bash
./megamol.sh -i testspheres inst
```

![MegaMol running the testspheres instance. The highlighted option in the AntTweak-Bar on the right side of the window adjusts the animation speed.](image)

MegaMol\textsuperscript{TM} should now open a rendering window showing a generated data set with several colored spheres. Hitting the space key starts and stops the animation playback. In the AntTweak-Bar, on the left side of the window, you can adjust all parameters of the running MegaMol\textsuperscript{TM} instance. For example, you can find the parameter `speed` in the group `inst::view::anim` (cf. Figure 2.1). With this parameter you can adjust the playback speed of the animation.
Chapter 3

Viewing Data Sets

In this chapter discusses the principle usage of the prepared project files for data set viewing. This project script files are available in the *script and example* package from the MegaMol™ project website.

3.1 Views, Modules and Calls

The run time functionality of MegaMol™ is constructed by *modules* and *calls*. These two type of objects are instantiated at run time, interconnected and build the *module graph*. Figure 3.1 shows an example module graph containing a rendering content of a window view, a renderer, a data source, and two modules providing additional information for the renderer. The *modules*, shown as blue boxes, are interconnected by *call* objects, shown as gray boxes. The connection end point at the modules are *CallerSlots*, shown as red triangles, and *CalleeSlots* shown as green triangles.

![Module Graph](image)

Figure 3.1: An example module graph. Left-most module *view* of class `View3D` represents the rendering content of a window. The center module *renderer* of class `SimpleSphereRenderer` is called by the window using the corresponding call of type `CallRenderer3D`. The right modules provide data and additional information for the renderer, namely a color map function and a clip plane.

The module graph follows the pull pattern. This means, that modules request function invocation by other modules. For example, the *view* modules needs to update the window content. The *view* modules thus invokes the *renderer* module to provide a new rendering. The *renderer* calls the *data* source if new data is available or to provide the old cached data.
3.1.1 Modules and Calls

Modules are the functional entities of MegaMol™. They provide several programmatic access points, the slots. Two types of these slots are shown in figure 3.1 as colored arrow heads.

CalleeSlots are access points of modules, through which these can be called to perform a function. For example, modules of class SimpleSphereRenderer provide a CalleeSlot rendering through which the rendering function can be invoked.

The counterparts are CallerSlots which are outgoing access points. These allow modules to call other modules. Modules of class View3D provide a corresponding slot rendering to call a connected renderer.

These two types of slots are connected using objects of call classes. These are shown as gray boxes in figure 3.1. Both CalleeSlots and CallerSlots specify types of calls they are compatible with. In the case of the above examples of renderings-relates slots, this is the type CallRender3D.

Calls do not contain significant functionality. Instead they are thin interfaces meant for data transport. For example, data to be visualized is loaded by data source modules. In figure 3.1 the module data of class MMPLDDataSource loads a specified data set into main memory and provides the data trough it’s CalleeSlot. The data is accessed through a MultiParticleDataCall. The call, however, does not copy the data, but provides access to the data in terms of memory pointers, and meta data. This avoidance of copy operations is most important and one of the core design ideas of MegaMol™.

The third type of slots are parameter slots. These are access points to exposed parameters controlling the functionality. Such parameters are automatically included in the front end’s GUI. Examples of such parameters are the setup of the virtual camera and light source in modules of type View3D or the data set file name in data source modules. To store values for these parameters, you can use the project file—see section 3.2—or parameter files—see section 3.3.

The module graph is configured for MegaMol™ using a project file. These files define modules and interconnecting calls for different instance specifications. There are two type of instances: views (see section 3.1.2) and jobs (see section 4.1). The starting command line of the console front end load project files (using -p) and requests instantiation of views and jobs (using -i).

3.1.2 Views

Views are one of the two instance types MegaMol™ can run. They are specified by the corresponding tag in a MegaMol™ project file (see section 3.2).

When a view is instantiated, a correspondingly names namespace will be created and all modules instantiated as part of the view will be create inside this namespace. For example, the project file seen in appendix A.2 defines the module data as part of the view dataview. If this view is instantiated by the command line:

MegaMolCon.exe ... -i dataview inst

then the module will be created with the full name ::inst::data. Correspondingly, it’s parameter slot filename can be globally addressed by ::inst::data::filename. This allows for the instantiation of several independent view instances.

For each view instance a rendering window will be created. To provide the content for the rendering window, each view instance description needs to provide a default view, usually via the viewmod attribute of the view tag in the MegaMol™ project file. The value of this attribute is the name for the view module to be called by the window management code. This module class must be implemented by deriving from ::megamol::core::view::AbstractView. Typically, you use View3D or View2D.

MegaMol™ provides some internal description of views which can be instantiated without loading a project file first. The view description TestSpheres used in section 2.5 is one example of such a built-in description.
3.2 Project Files

Project files are the primary method to start up MegaMol. Appendix A.2 shows the content of the project file `dataview.mmprj` which can be used to simply view a particle data set.

```xml
<view name="dataview" viewmod="view">
  Line 10 opens the view instance description. Although, it is possible to host multiple instance descriptions in a single project file its is recommended to only have one description per file. The view description is named `dataview` and the name for the primary view module is given as `view`.

  <!-- renderer -->
  <if cond="configset!=noao">
    <module class="MDAO2Renderer" name="renderer" />
  </if>
  <if cond="configset==noao">
    <if cond="configset!=usegeoshader">
      <module class="SimpleSphereRenderer" name="renderer" />
    </if>
    <if cond="configset==usegeoshader">
      <module class="SimpleGeoSphereRenderer" name="renderer" />
    </if>
  </if>
  <!-- view & setup -->
  <module class="View3D" name="view" />
  <module class="LinearTransferFunction" name="colors">
    <param name="mincolour" value="forestgreen" />
    <param name="maxcolour" value="lightskyblue" />
  </module>
</view>
```

The view module is specified at line 43, followed by modules for the color transfer function and the clip plane. Starting at the line 30 the renderer module class is selected and instantiated. Here the condition tags are used to select one of three renderers, based on the two config set variables `noao` (standing for `no ambient occlusion`) and `usegeoshader` (to select a renderer using a geometry shader).

Config set variables are flags which can be activated via the starting command line. For example, the start script file `view_laser-klein.bat`, which is part of the samples and scripts package of the release, specifies the config set variable `noao`.

Note: if you experience problems with one of the renderers, for example due to problems with your graphics card or graphics driver, try to select another one by specifying one or both of these config set variables using `-c <name>` in the command line.

In the specific project file provided in appendix A.2 the type of the data source module is also decided based on several config set variables. All used data source modules use mainly slots with the same names, i.e. a parameter slot named `filename` and a CalleeSlot named `getdata`, compatible with `MultiParticleDataCall`, providing access to the loaded data. Specifying the right config set variable thus allows the caller to use data sets from different file formats with this project file. See the online documentation for more information on these file formats. The recommended file format for MegaMol currently is `MMPLD`, and the corresponding data source module is thus the default module.

```xml
<!-- connecting calls -->
<call class="MultiParticleDataCall" from="renderer::getdata" to="data::getdata" />
<call class="CallRender3D" from="view::rendering" to="renderer::rendering" />
<call class="CallGetTransferFunction" from="renderer::gettransferfunction" to="colors::gettransferfunction" />
```
At the lines 55 to 59 the modules are interconnected using call objects. The corresponding tags specify the class of the call, the source CallerSlot to connect from, and the targetted CalleeSlot to connect to. The slot names use only the module names and slot names to form their names. Specifying the full name would require the instance name this view will be instanced as. Searching for the slots does therefor work with relative names.

One important function of project files can be seen, for example, at line 45: specifying parameter values. You can specify values for parameter slots of modules using the <param> tag inside the <module> tag. Use attributes to select the name and value for the corresponding parameter.

3.3 Parameter Files

Figure 3.2: The context menu of the console frontend can be used to write or read the parameter file.

The MegaMol™ console frontend can read and write parameter files. These text files store all value of all parameters of a MegaMol™ instance. This allows to store the exact state, e.g., used to produce a screen shot, and to later restore this exact state, with exception of the data set itself. To enable this feature, add

```
--paramfile <name.txt>
```

to the command line (of MegaMol™ or any of the supplied start script files). The text file will be read on start of MegaMol™, if it exists, and the corresponding parameters will be set to the specified values. Using the console frontend context menu (click with right mouse button) you can write the parameter file or read it in again (cf. Figure. 3.2).

Most attributes shown in the AntTweakBar of MegaMol™ can be directly manipulated and stored, e.g. color settings for the linear transfer function. One exception is the setting of the camera, i.e. the viewing direction and zoom.

To store the camera setting in the parameter file, you must first store the camera setting in one parameter. This is the parameter camsettings in the group inst::view. Click on the button storecam in the same group to create a textual representation of the camera an to store this text string in the parameter. Click on the button restorecam in the same group to set the camera based on this textual representation. The button resetView resets the camera to the default settings. The textual representation of the camera settings in the parameter camsettings can then be stored in the parameter file. If a parameter file is loaded and there is a value in the parameter camsettings, then the camera is set based on this value.

In short, to store the parameter values including the camera settings in a parameter file:

1. Start MegaMol™ with --paramfile filename.txt in the command line.
2. Adjust the values of all parameters the way you want.
3. Adjust the camera view using the mouse or keyboard interaction.
4. Save the camera settings to the parameter by clicking \texttt{storecam}.
5. Right-click the window and select the command \texttt{Write Parameter File} from the context menu.

3.4 View Interaction

The primary interaction with a view is controlling the camera with mouse and keyboard. The keyboard mapping is implemented by button parameters of the view module, also available in the GUI. Most parameters can be found in the sub-namespace \texttt{viewKey} inside the view name, e.g. \texttt{RotLeft}. Hovering with the mouse over the corresponding button reveals the associated hot key in the GUI's status text line. One important hot key is \texttt{Home} (aka \texttt{Pos1}) associated with the button \texttt{resetView}. This function resets the view to default.

Hold the \textit{left mouse button} and move your mouse to rotate the view around the look-at point. The look-at point initially is placed in the center of the bounding box of the data set.

Hold \texttt{shift} while holding and dragging the \textit{left mouse button} rolls the camera around the viewing direction.

Hold \texttt{control} while holding and dragging the \textit{left mouse button} rotates the camera around its center point.

Hold \texttt{alt} while holding and dragging the \textit{left mouse button} moves the camera orthogonally to the viewing direction.

Hold the \textit{middle mouse button} and move your mouse up or down to zoom the view by move the camera forwards or backwards. Note that if you zoom in too much, parts of the data set will be clipped by the near-clipping plane.

Hold \texttt{alt} while holding and dragging the \textit{middle mouse button} zoom the view by changing the opening angle of the camera.

Hold \texttt{control} while holding and dragging the \textit{middle mouse button} moves the look-at point forwards or backwards, changing the center for the corresponding rotation. Use the parameter \texttt{showLookAt} of the view to visualize the look-at point for better adjustment.

3.5 Making High-Resolution Screenshots

\textsc{MegaMol} has special functions to create high-resolution screen shoots of any rendering, namely the \texttt{ScreenShooter} module. The provided starting scripts add this module. If you create a project file of your own, remember to add the ScreenShooter. The corresponding settings can be found in the \texttt{AntTweakBar} in the groups \texttt{inst::screenshooter} and \texttt{inst::screenshooter::anim} (cf. Figure 3.3).

To connect the ScreenShooter with your view, you need to set the instance name of your view instance in the corresponding variable \texttt{inst::screenshooter::anim} (e.g. to \texttt{inst}). When making single screen shots, set the option \texttt{makeAnim} in the group \texttt{inst::screenshooter::anim} to false (disabled, as shown in the figure) and ignore the remaining options in that group. These options will be explained in section 3.6, as they are used to produce videos.

The parameters \texttt{imgWidth} and \texttt{imgHeight} specify the size of the screenshot to be rendered. These values are not limited to the window size and can be, in theory, arbitrarily large. If these values are getting large, the image can be rendered in several \textit{tiles}, i.e. subimages. The size for these tiles is specified by \texttt{tileWidth} and \texttt{tileHeight}. However, many renderers have problems with producing these tiled images. It is, thus, recommended to set \texttt{tileWidth} and \texttt{tileHeight} to be at least as large as \texttt{imgWidth} and \texttt{imgHeight}. The values for \texttt{tileWidth} and \texttt{tileHeight} are limited by the maximum texture size, maximum frame buffer object size and graphics memory size of your graphics card. Thus, these values are often limited, e.g. to 8192.
Chapter 3  Viewing Data Sets

Figure 3.3: The settings of the screen shooter.

The parameter filename specifies the path to the image file to be created. MegaMol\textsuperscript{TM} only creates PNG files.

Hit the button trigger to have MegaMol\textsuperscript{TM} create the requested screenshot.

3.6 Making simple Videos

MegaMol\textsuperscript{TM} cannot create video files directly. However, MegaMol\textsuperscript{TM} can create a sequence of screen shots of a time-dependent data sets showing the different points-in-time. Adjust the parameters in the group inst::screenshooter::anim in addition to the parameters for simple screen shots (cf. Section 3.5).

Enable the option makeAnim to make a screen shot sequence.

The parameters from and to specify the time codes to start and end the animation. The parameter step specifies the time code increment between two screen shots.

For example, if you specify from = 0, to = 100, and step = 10, assuming the data set stores enough time frames, 11 screen shots will be created and stored. These will show the data set at the times 0, 10, 20, ..., 90, and 100. The screen shot file names will be extended by an increasing number: e.g. test.00000.png, test.00001.png, test.00002.png, ...

This sequence of image files can then be merged to produce a video file, e.g. using avconv:

\texttt{avconv -r 30 -i test.\%05d.png test.mp4}

\textbf{KNOWN BUG:} several renderers will request the best data they can get. As usually data is loaded asynchronously, the correct data is often not available yet and the best data is the data from a slightly wrong time. While this is not a big deal for viewing data, it is fatal when rendering images for videos.

Many renderers thus expose a parameter forceTime, or with a similar name. Set this parameter to true and the renderer will always show the correct data. It will need to wait if the correct data is not available, yet, which can reduce the overall performance.
Chapter 4

Jobs: Converting Data

This chapter discusses the job concept available in MegaMol™. Especially, how jobs can be used for data conversion. Examples are based on the project script files available in the script and example package from the MegaMol™ project website.

4.1 Jobs

Jobs are the second type of instances available at the MegaMol™ runtime (compare view instances in section 3.1). The primary difference is the \(<\text{job}>\) tag as primary instance tag, see line 10. Similarly to the viewmod attribute, the \(<\text{job}>\) tag specifies a jobmod module as entry module.

\[
<\text{job name="convjob" jobmod="job">}
\]

One significant limitation of this release, is that the MegaMol™ Configurator is only able to edit view instance descriptions, see section 5.1. If you need or want graphical assistance in creating a job description, the recommended way is to create a view instance description with all required modules and calls. Use an \(\text{DataWriterJob}\) module as entry point. Save the corresponding project file and edit it manually with a text editor. Replace the \(<\text{view}>\) tags with the similarly behaving \(<\text{job}>\) tags and adjust the corresponding attributes.

4.2 Converting to MMPLD

The MegaMol™ Particle List Data file format (MMPLD) is a very fast loading binary memory dump of MegaMol™ meant for small and mid-sized data sets (1-10 mio. particles). MegaMol™ can convert most of it’s supported file formats to MMPLD file format. More precisely, all file formats which are loaded by a MegaMol™ module supporting the \(\text{MultiParticleDataCall}\) can be converted to MMPLD. For this, specify a converter job using a \(\text{DataWriterJob}\) module and a \(\text{MMPLDWriter}\) module. Appendix A.3 shows the content of the project file \(\text{makemmpld.mmprj}\) which can be used to convert data into the MMPLD file format.

The entry module for data conversion is of class \(\text{DataWriterJob}\), see line 30. This job module controls writing several files into a new data set. The output is implemented in corresponding write modules, like the \(\text{MMPLDWriter}\), see line 31.

This writer module is then connected to a module providing the data. In the simplest scenario this is directly a data loader module. The example in appendix A.3 selects one module from several options, in the same way the data viewing project did (see section 3.2 and appendix A.2). The class is selected via multiple config set variables, see lines 12 to 27.

The job is instantiated similarly using the command line:

\[
\text{MegaMolCon.exe -p dataview.mmprj -c load_SIFF -i convjob j}
\]

\[
\text{ -v j::data::filename inputfile.siff -v j::writer::filename outputfile.mmpld}
\]
This command line selects the class SIFFDataSource for the module data. The input file name and output file name are explicitly specified using the `-v` arguments. The job execution starts immediately. After all data is written, MegaMol™ terminates itself. The console output should be similar to this listing:

```plaintext
... 200| Loading project file "makemmpld.mmprj"
550| Created module "class megamol::stdplugin::moldyn::io::SIFFDataSource"
550| Created module "SIFFDataSource" (::conv::data)
550| Created module "class megamol::core::job::DataWriterJob"
550| Created module "DataWriterJob" (::conv::job)
550| Created module "class megamol::core::moldyn::MMPLDWriter"
550| Created module "MMPLDWriter" (::conv::writer)
550| Call "DataWriterCtrlCall" instantiated from "::conv::job::writer" to ":::conv::job::control"
550| Call "MultiParticleDataCall" instantiated from "::conv::writer::data" to "::conv::data::getdata"
200| Setting parameter "::conv::data::filename" to "..\sampledata\exp2mill.00010.siff"
200| Setting parameter "::conv::writer::filename" to "..\sampledata\exp2mill.00010.mmpld"
200| Job "job" started ...
200| Starting DataWriterJob "::conv::job"
200|Started writing data frame 0
200| Completed writing data
200| DataWriterJob "::conv::job" complete
250| Job instance conv terminating ...
550| Released module "class megamol::stdplugin::moldyn::io::SIFFDataSource"
550| Released module "class megamol::core::job::DataWriterJob"
550| Released module "class megamol::core::moldyn::MMPLDWriter"
200| Core Instance destroyed

To convert from other file formats, for which a corresponding loader does exist, you should be able to adjust this project file.
Chapter 5

Advanced Usage

This chapter discusses advanced usage of MegaMol\textsuperscript{TM}.

5.1 Configurator

The Configurator is a utility application for editing MegaMol\textsuperscript{TM} project files. More specifically, it allows to edit the modules, calls and parameters to be instantiated and set for view instances (see sections 3.1 and 3.2). Figure 5.1 shows the main GUI window of the application.

![Figure 5.1: The Configurator main GUI.](image)

The Configurator is written in C#. The Configurator can be run on Linux using Mono\textsuperscript{1}. However, we never tested compiling the Configurator with Mono and do not support this process.

The right editing pane allows you to place modules and interconnect their CallerSlots and CalleeSlots with matching calls. The tabs above that pane allow to open and edit several project files at once. On the left side you have the list of modules available and, below, the parameters of the currently selected module (which is marked in the editing pane by a dashed border). The menu strip contains direct access to all functions.

The Configurator is a stand-alone editor for MegaMol\textsuperscript{TM} project files. There is no way to connect the editor to a running instance of MegaMol\textsuperscript{TM} and, e.g., to edit parameters on the fly.

\begin{footnote}
\footnotesize{\textsuperscript{1}The Mono project: \url{http://www.mono-project.com/}}
\end{footnote}
5.1.1 Class Data Base

To fill the left list of available modules, the Configurator requires a data base containing all classes of the MegaMol™ module graph engine. This file is called the state file. You can load and save the state file using the Settings menu. There, you can also view a summary of the contents of the loaded state file. The MegaMol™ release package contains a state file of all modules and classes available in the released MegaMol™ binary. You can directly load this state file.

On Windows operating systems you can also analyze MegaMol™ binaries to generate a state file containing their classes. You find this function also in the Settings menu. This function relies on system Dlls and direct invoke of native, unmanaged code. So, this function will not work on Linux with Mono.

A dialog window will open in which you can select the folder to analyze. After a short time, the list in this dialog will show you all compatible MegaMol™ binaries, including versions of the core library and all plugins. Select all binaries you want to be included in the state file. Be aware that you can only select binaries which depend on the exactly same core library. Select Ok, and the new state file will be created and loaded. Per default, a dialog will appear, allowing you to save the newly generated state file.

5.1.2 Editing

Project files can be saved and loaded using the corresponding functions for the menu bar. Save will always ask you for a file name, but will use the last file name as default value. New creates a new editing tab. To close a tab use the x button on the right or click a tab with the middle mouse button.

Modules

To add a new module to the current project, double click on the corresponding entry in the module list. The module will be create in the upper left corner and will be automatically selected. To select a module click it with any mouse button. You can move all modules by simple dragging and dropping them with the mouse. This has no influence on the functionality. To delete the selected module hit the Del key.

While a module is selected, the parameter editor on the left shows all parameters of that module. In addition to the parameters exposed by parameter slots, there are some management parameters. You can edit the name of all module instances. For view modules, you can select if this module is the main view module for the whole project.

Click on a free space to deselect any module, call or slot. This will select the project itself. You can now edit project parameters in the parameter editor on the left.

Calls

To select a CallerSlot (red triangles) or a CalleeSlot (green triangles), click the corresponding triangle of the module. The selected slot will be highlighted in yellow. All matching slots, compatible with the selected slot using any call are highlighted by a yellow outline. If the Filter Compatible option is active, the module list will then only show modules which have compatible slots.

To connect two slot, drag with your mouse a line from one onto the other. The drag direction does not matter. If both slots are compatible, a matching call object is created and shown by a gray box. These call boxes are placed automatically. If the connection could be established by different call classes, a dialog window will appear asking to you select the call class to be used. To delete a call, select it by clicking on it’s box and press the Del key.

A (red) CallerSlot can only have a single outgoing call, while (green) CalleeSlots may be connected by multiple calls. Therefore, if you drag from a CallerSlot to a CalleeSlot which already has incoming connections, a new connection is added. If you, however, drag from a CalleeSlot to a CallerSlot which already has an outgoing connection, this connection is replaced.
5.2 Alternative Build for Testing

5.1.3 Starting MegaMol

The Configurator allows you to directly start MegaMol\textsuperscript{TM} with the currently selected project file.

Remember: the Configurator is only an editor for project files and as no online connection to a running MegaMol\textsuperscript{TM} instance. You thus need to remember to save your edited project files before starting MegaMol\textsuperscript{TM}.

![Figure 5.2: The Configurator main GUI.](image)

Selection the menu item MegaMol Start Arguments ... opens the dialog window shown in figure 5.2. Here you can edit all start settings. The menu item Start MegaMol directly starts MegaMol\textsuperscript{TM} with these settings. So does the Start button in this dialog window. The Ok button stores changes to the settings and closes the dialog window without starting MegaMol\textsuperscript{TM}.

The top line of elements control the way of starting MegaMol\textsuperscript{TM}. Tick directly to directly spawn the MegaMol\textsuperscript{TM} process from the configurator. This is the only working option for Linux operating systems. Tick in Cmd or in Windows Powershell and the configurator will first start the corresponding shell and then spawn the MegaMol\textsuperscript{TM} process inside this shell. Tick the keep Open option to start these shells in a way that they will remain open after the MegaMol\textsuperscript{TM} process terminates. This configuration is useful in case of errors which cause MegaMol\textsuperscript{TM} to instantly terminate.

The next text box control holds the full path to the MegaMol\textsuperscript{TM} front end executable. This should be set by the state file. You can then tell to use the directory of this executable as working directory for the MegaMol\textsuperscript{TM} process, or you can specify another one. Specifying another working directory is not recommended, as it influences the way the paths inside configuration file will be interpreted.

The last text box control holds the command line arguments for the MegaMol\textsuperscript{TM} process. The last used command line arguments are available via the drop down list. The start button gives you access to the default command lines for your project.

The whole process start command line can be copied to the clipboard using the remaining menu items in the Start menu. The differences for Cmd shell and Powershell are only the way how special characters and strings are escaped.

5.2 Alternative Build for Testing

We updated MegaMol\textsuperscript{TM} to use cmake to build on Linux OS. This greatly improved the build process on Linux. But this also makes some more uncommon scenarios difficult to realize. For example, cmake usually automatically detects required dependencies. But, in some scenarios you need to override this magic.

In this section we show how to compile a second MegaMol\textsuperscript{TM} on a system on which MegaMol\textsuperscript{TM} already has been compiled and installed. This is useful when working with experimental versions.
Chapter 5  Advanced Usage

This tutorial was originally published on the developers online blog: http://www.sgrottel.de/?p=1405&lang=en

5.2.1 VISlib and visglut

First off you build the visglut the usual way. I assume here, that the installed MegaMol™ uses a different visglut as the one you want to build now:

mkdir megamol_x2
cd megamol_x2
svn co https://svn.vis.uni-stuttgart.de/utilities/visglut/tags/forMegaMol11visglut
cd visglut/build_linux_make
make

If everything worked you can find the following files:
in megamol_x1/visglut/include:
  GL/freeglut_ext.h
  GL/freeglut.h
  GL/freeglut_std.h
  GL/glut.h
  visglut.h
  visglutversion.h
and in megamol_x2/visglut/lib:
  libvisglut64.a
  libvisglut64d.a

If so, let’s continue with the VISlib:

cd megamol_x2
svn co https://svn.vis.uni-stuttgart.de/utilities/vislib/tags/release_2_0 vislib
cd vislib

Now, there is the first action which is different from the default build process. As usual we will use the script cmake_build.sh. This script, however, per default registers the build directories in the cmake package registry. This enables cmake to find this package in its build trees. In this scenario, however, we do not want this special build to be automatically found, because we do not want to get in the way of our system-installed MegaMol™. We thus deactivate the package registry.

This command configures and builds the VISlib, both for debug and release version:

./cmake_build.sh -dcmm

As always, if you encounter build problems due to the multi-job make, reduce the number of compile jobs:

./cmake_build.sh -dcmnj 1

Note that I do not specify an install directory. I do not plan to install this special MegaMol™. I just want to build, for example for a bug hunt.

5.2.2 MegaMolCore

It’s now time for the core.

cd megamol_x2
svn co https://svn.vis.uni-stuttgart.de/projects/megamol/core/branches/v1.1rc core
cd core

We first test the configuration by only configuring release and not building anything:

./cmake_build.sh -cv ../vislib -C -DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE

Note that I also disabled MPI-Support here. The system I am building on has MPI installed, but I don’t want this MegaMol™ to use it.

The output should contain this line:
5.2 Alternative Build for Testing

- Found vislib: /home/sgrottel/megamol_x2/vislib/build.release/libvislib.a

This points to the right vislib, the one we specified. So all is well. We can build MegaMol\textsuperscript{TM}, again without registering it’s build trees in the cmake package repository:

```bash
./cmake_build.sh -dcmnv ../vislib -C -DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE
```

When all worked you got yourself the binaries:

- megamol_x2/core/build.debug/libMegaMolCored.so
- megamol_x2/core/build.release/libMegaMolCore.so

5.2.3 MegaMolConsole

Get yourself a working copy of the console:

```bash
cd megamol_x2
svn co https://svn.vis.uni-stuttgart.de/projects/megamol/frontends/console/branches/v1.1rc console
```

Again, we test if everything works by only configuring release and not building:

```bash
./cmake_build.sh -c -f ../core
```

The Console does not register its build tree per default, since no other project depends on the console. So we are fine here.

The output should contain these lines:

- Looking for MegaMolCore with hints: ../core;../core/build.release;../core/share/cmake/MegaMolCore
- Found MegaMolCore: /home/sgrottel/megamol_x2/core/build.release/libMegaMolCore.so
- MegaMolCore suggests vislib at: /home/sgrottel/megamol_x2/vislib/build.release
- MegaMolCore suggests install prefix: /usr/local
- Using MegaMolCore install prefix
- Found vislib: /home/sgrottel/megamol_x2/vislib/build.release/libvislib.a
- Found AntTweakBar: /home/sgrottel/AntTweakBar/lib/libAntTweakBar.so
- Found visglut: /home/sgrottel/megamol_x2/visglut/lib/libvisglut64.a

If the directories for other libraries are wrong, for example the AntTweakBar or the visglut use the cmake-typical DIR variable to give a search hint. Remember, relative paths might be confusing. Better use absolute paths. I don’t:

```bash
./cmake_build.sh -c -f ../core -Dvisglut_DIR= /megamol_x2/visglut -DAntTweakBar_DIR=../../AntTweakBar
```

But in my case the cmake-magic worked fine in the first place. So, I configure both build types again:

```bash
./cmake_build.sh -dc -f ../core
```

Double check the output. Make sure the core, the vislib and the visglut are found in all the right places. If they are, built it:

```bash
./cmake_build.sh -dm
```

At this point you can quickly test your MegaMol\textsuperscript{TM}. First open the megamol.cfg configuration file in a text editor and adjust the paths in there to yours. Then run MegaMol\textsuperscript{TM}:

```bash
cd build.release
./MegaMolCon
```

If this seems ok, and if you have a local graphics card you can run the demo renderer:

```bash
./MegaMolCon -i demospheres s
```

5.2.4 Some MegaMol Plugin

Finally we need a plugin. I go for the mmstd\_moldyn:

```bash
cd megamol_x2
svn co https://svn.vis.uni-stuttgart.de/projects/megamol/plugins/mmstd_moldyn/
```
branches/v1.1rc mmstd_moldyn

cd mmstd_moldyn

The process is now exactly the same as with the console:

./cmake_build.sh -dcf ../core

The double check the directories for the core and the VISlib. If they are good, build the plugin:

./cmake_build.sh -dm

To test this plugin we go back to the console, and adjust the config file to load the plugin:

cd megamol_x2/console/build.release

Include the following lines in the config file. Obviously adjust the paths to what you need:

<plugin path="/home/sgrottel/megamol_x2/mmstd_moldyn/build.release" name="mmstd_moldyn.mmplg" action="include" />

<shaderdir path="/home/sgrottel/megamol_x2/mmstd_moldyn/Shaders" />

If you now run MegaMol it will try to load your plugin and will report it. The output console should contain something like:

200|Plugin mmstd_moldyn loaded: 11 Modules, 0 Calls
200|Plugin "mmstd_moldyn" (/home/sgrottel/megamol_x2/mmstd_moldyn/build.release/mmstd_moldyn.mmplg) loaded: 11 Modules, 0 Calls registered

And that’s it.
Appendix A

MegaMol Configuration files

These files are also part of the MegaMol\textsuperscript{TM} Release 1.1 download packages.

A.1 MegaMol configuration file megamol.cfg

This is an example configuration file for MegaMol\textsuperscript{TM}. See section 2.4 for details.

```xml
<?xml version="1.0" encoding="utf-8"?>
<MegaMol type="config" version="1.2">

 <!-- default log behavior -->
 <log file="" level="0" echolevel="*" />

 <!-- main configuration -->
 <appdir path="." />
 <shaderdir path="../share/megamol/core/Shaders" />

 <!-- configure plugins like this -->
 <!-- Remember to add the 'd' to the file name if you want to use the debug version

 <if cond="!debug">
  <plugin path="." name="mmstd_moldyn.mmplg" action="include" />
 </if>

 <if cond="debug">
  <plugin path="." name="mmstd_moldynd.mmplg" action="include" />
 </if>

 <shaderdir path="../share/megamol/mmstd_moldyn/Shaders" />

 <!--
  <plugin path="." name="mmstd_datatools.mmplg" action="include" />
  <plugin path="." name="mmstd_moldyn.mmplg" action="include" />
  <plugin path="." name="mdao.mmplg" action="include" />
  <plugin path="." name="mdao2.mmplg" action="include" />
  <plugin path="." name="TriSoup.mmplg" action="include" />
  <plugin path="." name="protein.mmplg" action="include" />

  <if cond="!debug">
   <plugin path="." name="mmstd_datatoolsd.mmplg" action="include" />
   <plugin path="." name="mmstd_moldynd.mmplg" action="include" />
  </if>

  <if cond="debug">
   <plugin path="." name="mmstd_datatoolsd.mmplg" action="include" />
   <plugin path="." name="mmstd_moldynd.mmplg" action="include" />
  </if>

  <plugin path="." name="mmstd_datatoolsd.mmplg" action="include" />
  <plugin path="." name="mmstd_moldynd.mmplg" action="include" />
```
A.2 Data Viewing Project file dataview.mmprj

This is an example project file for MegaMol™. This project file defines a view to visualize particle data sets. See section 3.2 for details. An example start up command line is:

```
MegaMolCon.exe -p dataview.mmprj -i dataview dv
-v ::dv::data::filename cool_random.mmpld
```

```
<?xml version="1.0" encoding="utf-8"?>
<MegaMol type="project" version="1.1">
<!-- Use this command line arguments to start MegaMol
-p mmpld-view.mmprj -i mmpldview inst -v inst::data::filename 1OGZ.mmpld
-->
<view name="dataview" viewmod="view">
  <!-- data -->
  <if cond="configset==load_IMDAtom">
    <module class="IMDAtomData" name="data" />
  </if>
  <if cond="configset==load_MMSPD">
    <module class="MMSPDDataSource" name="data" />
  </if>
  <if cond="configset==load_SIFF">
    <module class="SIFFDataSource" name="data" />
  </if>
  <if cond="configset==load_VIM">
    <module class="VIMDataSource" name="data" />
  </if>
  <if cond="configset==load_VisItt">
    <module class="VisIttDataSource" name="data" />
  </if>
  <if cond="((configset==load_MMPLD)||((configset!=load_IMDAtom)
```
This is an example project file for MegaMol™. This project file defines a job to convert data set into the MMPLD file format. See section 4 for details.

```xml
<?xml version="1.0" encoding="utf-8"?>
<MegaMol type="project" version="1.1">
    <!-- renderer -->
    <if cond="configset!=noao">
        <module class="MDAO2Renderer" name="renderer" />
    </if>
    <if cond="configset==noao">
        <if cond="configset!=usegeoshader">
            <module class="SimpleSphereRenderer" name="renderer" />
        </if>
        <if cond="configset==usegeoshader">
            <module class="SimpleGeoSphereRenderer" name="renderer" />
        </if>
    </if>
    <!-- view & setup -->
    <module class="View3D" name="view" />
    <module class="LinearTransferFunction" name="colors">
        <param name="mincolour" value="forestgreen" />
        <param name="maxcolour" value="lightskyblue" />
    </module>
    <module class="ClipPlane" name="clipplane">
        <param name="colour" value="#80808000" />
    </module>
    <!-- screenshooter -->
    <module class="ScreenShooter" name="screenshooter">
        <param name="view" value="inst" />
    </module>
    <!-- connecting calls -->
    <call class="MultiParticleDataCall" from="renderer::getdata" to="data::getdata" />
    <call class="CallRender3D" from="view::rendering" to="renderer::rendering" />
    <call class="CallGetTransferFunction" from="renderer::gettransferfunction" to="colors::gettransferfunction" />
    <call class="CallClipPlane" from="renderer::getclipplane" to="clipplane::getclipplane" />
</MegaMol>
```
Chapter A  MegaMol Configuration files

7  -p mmpld-view.mmprj -i mmpldview inst -v inst::data::filename 1
   OGZ.mmpld

9  -->
10 <job name="convjob" jobmod="job">
11   <!-- data -->
12   <if cond="configset==load_IMDAtom">
13     <module class="IMDAtomData" name="data" />
14   </if>
15   <if cond="configset==load_MMSPD">
16     <module class="MMSPDDataSource" name="data" />
17   </if>
18   <if cond="configset==load_SIFF">
19     <module class="SIFFDataSource" name="data" />
20   </if>
21   <if cond="configset==load_VIM">
22     <module class="VIMDataSource" name="data" />
23   </if>
24   <if cond="configset==load_VisItt">
25     <module class="VisIttDataSource" name="data" />
26   </if>
27   <!-- writer -->
28   <module class="DataWriterJob" name="job" />
29   <module class="MMPLDWriter" name="writer" />
30   <call class="DataWriterCtrlCall" from="job::writer" to="writer::control" />
31   <call class="MultiParticleDataCall" from="writer::data" to="data::getdata" />
32
33 </job>
34 </MegaMol>