Max–Planck–Institut für biologische Kybernetik
Max Planck Institute for Biological Cybernetics

Technical Report No. 142

veLib Reference Manual

Library Version 1.2.0

Gerald Franz[1] & Michael Weyel[1]

Aug 2005

[1]Department Bülthoff, Max Planck Institute for Biological Cybernetics, Spemannstr. 38, 72076 Tübingen, Germany.
E-mail: gerald.franz,michael.weyel@tuebingen.mpg.de

Version 1.2.0
This report is available in PDF–format via anonymous ftp at ftp://ftp.kyb.tuebingen.mpg.de/pub/mpi-memos/pdf/TR-142.pdf. The complete series of Technical Reports is documented at: http://www.kyb.tuebingen.mpg.de/bu/techr/

[

veLib Reference Manual

Library Version 1.2.0

Gerald Franz & Michael Weyel

Abstract. The virtual environments library (veLib) is a light-weight yet complete cross-platform software framework for distributed virtual reality (VR) simulations. This document describes its basic design principles, the installation process, and gives a brief introduction in using the veLib for programming VR applications. The introductory texts are complemented by a comprehensive description of the C++ and XML application programmers' interface (API).

Keywords: Virtual reality, distributed systems, C++ library.

]

Contents

Part I

veLib Introduction

Chapter 1

veLib Overview

Welcome to the Virtual Environments Library!

1.1 What is the veLib?

The Virtual Environments Library (veLib) is an extensible framework for the development of distributed realtime high-performance virtual reality applications. It was designed to offer a convenient and unified interface for all sorts of different input and output devices and to hide the hazzle of different system architectures and communication methods from the user under a simple generic abstraction layer. Its design goals are, roughly ordered from most to least important, scalability, extensibility, flexibility, stability, simplicity, ease of use, performance, completeness, fanciness.

More specifically, it contains ready-made interfaces to all sorts of normal input devices (keyboard, mouse, joysticks, game pads, etc.), a basic 3D graphics engine, a nice spatial audio framework, a network communication layer, basic simulation logic such as collision detection and motion models, and various auxiliary functions such as overlay functionality, portable file input/output, and timers. The veLib is a platform independent C++ library, currently it is actively maintained for Linux and MS Windows.

And last but not least, the veLib is free (GPL) software. You can use it for no charge in your own projects and adapt to your particular needs. For licensing details please refer to the licensing documents shipped with the veLib as part of the online documentation or visit the veLib web site (`http://velib.kyb.mpg.de/docu`).

1.2 Design Goals

The veLib project was started in August 2001. The developers have been aware, that numerous libraries with similar goals exist, but miss in all of them certain features. The most important one is, that devices should replace other devices without changing the logic of the program. For example it should be possible to drive a car with a wheel, but also with a joystick or mouse. Furthermore, the logic of the control program should be independent off the connected devices. To achieve these goals, the complete communication between all devices and the central simulation is unified and standardized. It thereby is possible to use any device once implemented inside the veLib in any simulation written for the veLib. The devices have only to be programmed once and can then be used and easily exchanged in simulations. A device in this context is every input,

but also every output device in a VR (Virtual Reality) context. It can be as simple and using the mouse or keyboard, but also more complex like a specialized bike or joystick for the input. Typical output devices are a display (using various graphics libraries) and force feedback devices.

A further consitituent basis of the veLib is a unified data container for all the data flow between devices and the simulation. This data container (classes dataChar, resp. dataContainer) contains standardized fields like position and orientations, but offer also freely definable user data. Therefore they are both general but also at the same time extendable. Data containers are passed to the devices which are expected to provide any user input to the simulation and are also passed to those devices which can give feedback to the user from the simulation. The same data container is used for both purposes. In addition, motion models can implement simple, but also complex user movements in the virtual world. These motion models can be restricted by collision detection implementations based on a graphical or other model.

A further, less technical, but maybe most important design goal of the veLib is to unify efforts that arise from using virtual reality as medium for scientific research. Therefore modularity, extensibility, ease of use, both by well-designed code and clear and complete documentation, are seen as central means to motivate potential contributors to share their own developments within this framework, and so to equally profit themselves by work of others and to let others make use of their accomplishments. To serve this spirit of benevolence, open exchange, and mutual benefit, the terms of License are established.

1.3 License

This license is officially guilty as long as there is no revision. Possible future changes do not have retroactive effects on already published releases.

Copyright 2001-2005 for the project as a whole exclusively by Reinhard Feiler (reinhard.feiler@tuebingen.mpg.de), on behalf of the Max Planck Institute for Biological Cybernetics. This unified reglementation does not affect particular further rights of contributors on parts of the code based on their authorship. However, potential changes of the licensing and/or the code and/or the code does not need explicit agreement by the original contributors. These terms are treated as implicitely accepted, as soon as someone agrees that code or other works of her or his authorship become part of veLib.

The veLib license model is in some aspects similar to QT, which offers different licenses for different purposes. For external users (i. e. people that are not contributors), permission to use, copy, modify, and distribute this software and its documentation under the terms of the GPL GNU General Public License is hereby granted. No representations are made about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. Enlisted contributors have also the right to use it under the terms of the veLicense for Contributors, which basically means a LGPL with minor addendums concerning the explicit permission of static linking, similar to FLTK.

Note that any small change that is submitted and implemented in the main veLib repository by the maintainer does not automatically mean that the submitting person will be granted the additional rights of a contributor. For this the contribution does not need to be large, but has to be to some degree "substantial". If you think that your contribution is substantial and should give you the status of a contributor, you should discuss this with the current holder of the copyright who is the only person who can give these personal and unalienable further rights. If no mutual agreement can be found, you are free to withdraw your current changes or addendums from the veLib.

Chapter 2

Getting started

2.1 About this chapter

This chapter guides you through the first steps when using the veLib. More specifically, it helps you in getting the veLib running and provides an introductory tutorial on its basic programming concepts. The code demonstrated in this document complies with the veLib 1.2 syntax conventions.

The chapter is targeted primarily at beginners that intend to use the veLib for developing (virtual reality) applications. It is presumed that readers already have basic skills in C++ and object-oriented programing, and know how to use the development tools of their system platform. Furthermore, a very basic knowledge on OpenGL and the syntax of XML is generally helpful.

2.2 Getting the veLib

Download as zip archive. The most convenient way to get the veLib is to download it from the official veLib homepage as complete zip archive: `http://www.kyb.mpg.de/prjs/facilities/velib`.

The website has a link to the download section in the left side menu. Before downloading, one has to request a personal password. This registration process is required by German laws, all personal data except of the eMail address are optional and will be handled according to the regulations by law concerning the protection of data privacy. The personal password will then be sent by eMail together with further instructions. It does not expire, hence, for later downloads (e.g., updates) the password can be reused.

After downloading, the veLib package has to be unzipped using platform-specific tools (.e.g., unzip under Linux, winZip under Windows).

Check out from the CVS-tree. Another way to get the veLib is directly via CVS. This method may be used for downloading the most current development version of the veLib that may offer additional features but may not be as well tested as the official package. Thus, this method mainly addresses the experienced programmer. In addition, also the latest stable version is available via CVS. This method may be advantageous if a previous version of the veLib is already installed, so, only the differences have to be transmitted.

CVS (Concurrent Version System) is an (open source) source code management system. For more information on CVS, one may see at `http://cvsbook.red-bean.com/cvsbook.html`. The newest veLib version can always be found on the MPI's CVS-Server though it is not guaranteed to be a stable one. You can either get a CVS-account with write permissions for the CVS-server, but those accounts are only granted to core veLib-developers. Everyone else please use the anonymous checkout. Both methods are described below for Linux and Windows OS.

On Linux:

1. Get an account and password for our CVS-Server (contact Michael Weyel `michael.weyel@tuebingen.mpg.de` for that) OR use the anonymous checkout (see 4.)!
2. If you have an account: login on the CVS-Server (supposed that your user name is "verner"):
   `cvs -d :pserver:verner@cvs.tuebingen.mpg.de:/veLib login`
3. Check out the source code:
   (a) For the development version:
   `cvs -d :pserver:verner@cvs.tuebingen.mpg.de:/veLib co -d veLib private`
   (b) For the latest stable version:
   `cvs -d :pserver:verner@cvs.tuebingen.mpg.de:/veLib co -d veLib public`
4. If you do not have an account, you can checkout the source code as follows:
   (a) For the development version:
   `cvs -d :pserver:anoncvs@cvs.tuebingen.mpg.de:/veLib co -d veLib private`
   (b) For the latest stable version:
   `cvs -d :pserver:anoncvs@cvs.tuebingen.mpg.de:/veLib co -d veLib public`

On Windows:

1. Get CVS for Windows (the following descriptions suppose that you use WinCVS, which can be downloaded for free at `http://sourceforge.net/projects/cvsgui/`)
2. Get an account and password for our CVS-Server (contact Michael Weyel `michael.weyel@tuebingen.mpg.de` for that) OR use the anonymous checkout!
3. In WinCVS, goto Admin->Preferences and enter `:pserver:verner@cvs.tuebingen.mpg.de:/veLib` in the field "Enter CVS root" (supposed that your user name is "verner") OR `:pserver:anoncvs@cvs.tuebingen.mpg.de:/veLib` if you don't have a CVS account.
4. Choose "passwd" file on CVS server for authentification
5. Click on the "Ports" tab, mark "For 'pserver' (password) port:" and enter 2401 for that port
6. Goto Admin->Login... and type your password OR just move on to 7. if you don't have a CVS account.
7. Goto Create->Checkout module..., enter "private" as the module name for the developer version OR enter "public" for the latest stable release. Choose a local directory, where the sources should be checked out to and click OK. If you logged in correctly, the veLib source is now copied to the directory you specified.

2.3 Installation

2.3.1 Hardware requirements

The veLib has no specific requirements regarding hardware. Any reasonably current PC system (2001+) will do provided that you get the required software running (see below). Of course, for getting a decent performance out of the veLib, a modern system having good 3D graphics and sound accelerator cards is advantageous. Furthermore, a joystick or similar input device is required in some demos and is generally a good intuitive interaction device for 3D simulations.

2.3.2 Software requirements

Most basically, the veLib needs a running operating system. Here you can either use a recent Linux, or Windows (when using Visual C++.net only Windows 2000 and XP, when using MinGW also Windows 98 and Me).

Additionally, the veLib is heavily based on two basic libraries that currently are a prerequisite for all veLib programs:

- OpenGL, a platform-independent high performance graphics API, `http://www.opengl.org`.
- SDL, the Simple DirectMedia Layer, v1.2, `http://www.libsdl.org`. SDL provides a portable high performance hardware interface. It is internally used for handling keyboard, mouse, and joystick input, window handling, and multi threading.

Furthermore, the veLib itself makes internally use of a few helper libraries. These library bindings are optional (see Section 2.3.3), but strongly recommended in order to make use of the complete functionality:

- OpenAL (`http://www.openal.org`), a platform-independent spatial audio library.
- libJPEG (`http://www.ijg.org/`), a graphics library for image and texture handling.
- libpng (`http://www.libpng.org/pub/png/`) and zlib (`http://www.gzip.org/zlib/`), a graphics and a compression library for image and texture handling.

Note that on Linux systems most of these libraries normally are already installed. After downloading and compiling the additional packages it is necessary to either install them in your system's standard paths, or to put or link the headers and compiled libraries into the external/include resp. external/lib subdirectories of the veLib root directory.

2.3.3 Customizing veConfig.h

You do not have to provide all the auxiliary libs if you do not need a certain functionality (e.g. if you don't want to load any jpeg-pictures, using the libJPEG is unnecessary). But the veLib needs to know which libs to use and which to avoid. You may need to make some adjustments to `src/include/veConfig.h`. This file is quite self explanatory and well documented, just have a look into it before compiling the veLib. Furthermore, the linker settings of the compiler have to be adjusted correspondingly. Under Linux and MinGW, these adjustments are made in the file `src/makeinclude` in the section "`# libraries to link with`".

2.3.4 Compiling the library

Linux. The use of the gcc compiler version 3.2 or higher is recommended. Compiling the lib may also work with earlier versions, but are not officially supported. Make sure that all different external libraries are compiled with the same compiler that you use for building the veLib.

If all preparatory steps are done correctly, the veLib is compiled by just switching in a shell to the top level veLib directory and running “make”.

MS Windows. Currently, there are two ways to compile the veLib under Windows. One can either the the free MinGW compiler (`http://www.mingw.org`) and follow the Linux preparation and compilation instructions, or use Microsoft’s Visual C++ compiler v7.1+. Here you can load the project file and press the build button for the lib-project, hopefully the veLib will be compiled in a few minutes. Currently, all the sources in the demo and server/client directories should also compile without problems.

2.3.5 Problems and solutions

veLib starts compiling but stops after a while with an error message. It may happen from time to time (especially if you try to compile the developer version), that not all veLib related sources compile properly. If you get any error during compilation, please first check how far your compiler got. Most of the time, it won’t have been the veLib itself that generated the error, but one of the provided example applications. In most cases, you can ignore that, the veLib will work with your application. If the error was caused by the veLib itself, please check again if all your settings are correct and if you have all the necessary libraries installed. If that does not help, please post your problem to the veLib mailing list (see Section 2.3.6).

All sorts of linker errors turn up when trying to compile an application that is linked to the veLib with Microsoft Visual C++. The veLib is build with MS Visual C++ as Multi-threaded (compiler option /MT) if build in Release mode or as Multi-threaded Debug (compiler option /MTd) if build in Debug mode. If you link your own application to the veLib, make sure that the use the exact same compiler option or you may get all sorts of weird linker errors. You can set this option by right-clicking on the project name in the solution-explorer-window and then selecting ”properties” (alternatively go to “Project” in the Menu bar and select “XYZ Properties”, where XYZ stands for your project name). On the Properties page, open the C/C++ folder and go to the “Code-Generation” section. You will find an entry “Runtime library”, which needs to be set to Multi-threaded or to Multi-threaded Debug respectively.

You may also need to set some linker options to build without the corresponding libraries, the errors from the development environment will provide the necessary information. For doing this, again open your projects property page. Open the Linker folder and go to the “Input” section. In the row “Ignore specific library” just enter the name of the lib that you want to have ignored, for example libc.lib;libcd.lib (separated by semi-colons;) .

After updating your veLib repository on Linux from version 0.x to version 1.x the veLib itself compiles fine, but the demos bail out with lots of linker errors. Your linker tries to link with an old shared version of the veLib (`src/lib/libve.so.xyz`). The new veLib is a static library (`src/lib/libve.a`). The easiest solution is to remove the old version by typing `rm -f src/lib/libve.so*` in a shell.

On windows, the library and all demos compile flawlessly, but when trying to run a demo (or a veLib based application), the program returns immediately showing a message similar to "*The application failed to start, because sdl.dll was not found. Re-installing the software may fix this problem.*". The application misses a required shared (dynamically linked) library (in this example SDL). Putting a copy of the requested .dll file into the demo (or your application) directory will fix this.

2.3.6 Getting further information

The veLib comes with a pretty complete html-help and API reference, situated at docs/html/index.html. The same, or ideally an advanced version of this online help is also found on the veLib website (`http://www.kyb.tuebingen.mpg.de/prjs/facilities/velib/docu`). However, this online help certainly does not cover all aspects. So, if you find any bugs or need help on a certain topic, feel free to contact the main maintainer Michael Weyel (`michael.weyel@tuebingen.mpg.de`). You may also send him a mail if anything in this document does not become clear or if something that is described here is not working as it should.

Additionally, there is a veLib mailing list, where all news and changes concerning the veLib are announced. You may also post a veLib related problem there and see if any of the other subscribers can give you a solution for it. If you want to become part of the list, please contact Michael Renner (`michael.renner@tuebingen.mpg.de`)

2.4 Tutorial

Having successfully installed the veLib and its dependencies, it is now time to write a first program!

This tutorial introduces the core classes of the veLib and shows how to structurize programs to make them easily portable on various devices and hardware platforms. This is done by first setting up a small stand-alone program that will be subsequently explained step-by-step and extended to a completely scalable application. This final program does not do much, yet it is a useful core that in similar form underlies most interactive 3D computer simulations from screen savers over ego shooters to full-featured physically-correct professional flight training simulators. In its most basic state it just allows the free navigation through a virtual landscape, the virtual camera is controlled via keyboard and mouse (see Figure 2.1). Yet it may be generically useful as starting point for own projects. While this tutorial of course covers only a small fraction of the veLib functionality, it provides the indispensable survival veLib programing kit and the prerequisites for understanding the further more advanced demos that cover various aspects in more detail (see Section 2.4.4). The basic classes introduced in this tutorial are:

- ve::xmlIni, the veLib interface for initialization and file input/output
- ve::device, the representation of joystick, display, motion platform, audio, network...
- ve::vec6f and ve::flag128, basic data structures
- ve::motion and ve::collision, the veLib approximation of a physical model
- ve::time, a class for all time related purposes
- ve::dataContainer, the communication interface for device states and simulation object states
- ve::deviceContainer, the abstraction layer for (sets of) devices

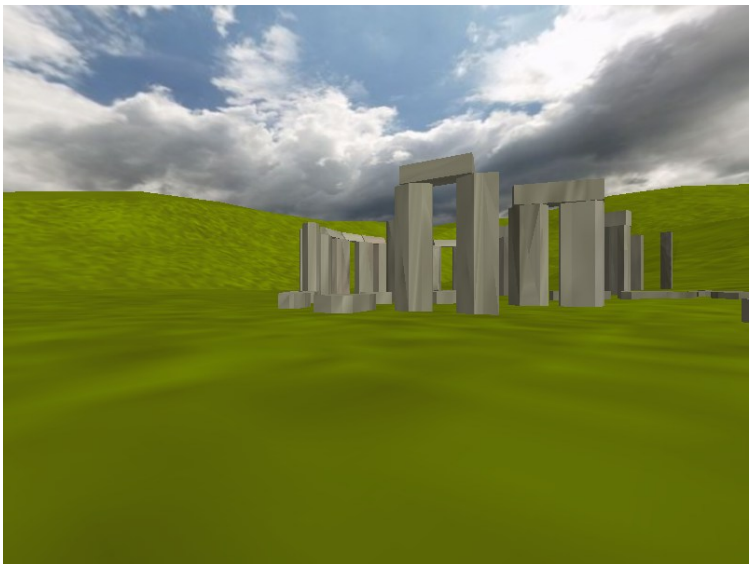


Figure 2.1: Screenshot of the running tutorial program.

2.4.1 A minimal stand-alone program

We directly jump into the code (Note: the source of these examples is located in the demo subdirectory.):

Source code tut01.cpp.

```
// demo/tut01.cpp
#include <veLib.h>

int main( int , char** ) {
    ve::xmlIni ini;
    ini.load("iniTut.xml");
    ve::deviceWindow     devWindow(ini);
    ve::deviceGraphicsGL devVideo(ini);
    ve::vec6f position(0.0f, -110.0f, 1.6f, 310.0f, 0.0f, 0.0f);
    ve::vec6f velocity, acceleration, inputAxes;
    ve::flag128 inputButtons;
    ve::collisionSurface collisionModel(ini);
    collisionModel.addObject(0, position, 1.6f, 1.0f);
    ve::motionSimple motionModel(ini, 0, &collisionModel);
    ve::chrono timer;
    while(!inputButtons[ve::BUTTON_2]&&!inputButtons[ve::KEY_ESCAPE]) {
        timer.update();
        devWindow.getInput(inputAxes,inputButtons);
        motionModel.updateObject(0, inputAxes, position, velocity,
                                 acceleration ,timer.deltaT());
        devVideo.setOutput(position, inputButtons);
        devVideo.update(timer.deltaT());
        devWindow.update(timer.deltaT());
        timer.sleep(0.01);
```

```
    }
    return 0;
}
```

Step by step. Now we will discuss the previous example in more detail:

`#include<veLib.h>`
The veLib classes are accessible via various header files. For convenience purposes, the header veLib.h includes the complete public interface in one command and also includes common C++ standard headers.

A typical simulation consists of at least an initialization part and a main loop. In the first part all necessary simulation objects are created:

```
    ve::xmlIni ini;
    ini.load("iniTut.xml");
```

A central concept of the veLib is the usage of initialization files. These files are written in XML. For parsing XML files, the veLib provides two classes, `ve::xml` and `ve::xmlIni`. While `ve::xml` provides the low level language structure and parsing functionality, `ve::xmlIni` defines a convenient high level interface especially for initialization files. Their content will be briefly explained in the next section. The exact meaning and various options of these files is subject of another document (`veLibXml.pdf`). The most important message for now is that XML objects and initialization files are required for initializing devices. They can be instantiated and initialized using the statements above.

The first line also shows the veLib namespace, `ve::` . To avoid its explicit usage, one can import it completely by the statement `using namespace ve;`. Yet its advantage of an explicit use of the namespace is the better recognizability of veLib commands.

```
    ve::deviceWindow     devWindow(ini);
    ve::deviceGraphicsGL devVideo(ini);
```

The veLib implements the access of hardware using the unified device concept. Device interfaces are one core service of the library. All devices are derived classes from the parent class `ve::device` which defines a common interface. This means, independent from the actual type, all devices can be addressed using exactly the same syntax. In this case, a window object is opened and a 3D OpenGL visualization object is created. The exact parameters for the initialization are provided in the previously loaded `ve::xmlIni ini` object. The window class does not only provide a draw area for the 3D graphics device, but also allows access to keyboard and mouse events that in all contemporary operating systems are bound to the window focus.

```
    ve::vec6f position(0.0f, -110.0f, 1.6f, 310.0f, 0.0f, 0.0f);
    ve::vec6f velocity, acceleration, inputAxes;
    ve::flag128 inputButtons;
```

A 3D simulation normally deals with objects that are somewhere located in space and have certain properties. In this tutorial a few variables are needed to control and influence the camera position and check for user events. For these purposes the veLib makes heavily use of two basic data structures, `ve::vec6f` coordinate objects and `ve::flag128` state containers.

A `ve::vec6f` object (often called a “sixdof”) contains six float values that normally represent the X, Y, Z position as well as the H, P, R (heading, pitch, and roll) orientation coordinates of an object, thus a position and orientation in space is completely described. The veLib coordinate system is similar to OpenGL’s, the only differences are that +Z is the default for the up direction and +Y for forward. The generic class `ve::vec6f` is used as well to store an object’s velocity (linear and angular), acceleration, or the state of the axes of an ve::device. In the later case, the devices always normalize the axis values to the range of -1.0f to 1.0f. The `ve::vec6f` class defines several operators to set, access, or transform its content, most common is the access

`operator[]` which allows to read or change a single ordinate that can be specified via the `ve::X`, `ve::Y`, `ve::Z`, `ve::H`, `ve::P`, and `ve::R` coordinates.

A `ve::flag128` state container is most basically a struct of 4 unsigned int32 variables that contains 128 bool values. It is most often used to store the state of input device buttons. That means, for example, that for each key of a keyboard (normally 101-104 keys) one bit is reserved that can contain its current press state. In `src/include/veTypes.h` a complete set of constants is defined to give all keys and buttons explicit human-readable names (for an example see the while loop later on). Also for this class a similar set of access methods is defined, for more information please refer the veLib API reference manual.

```
ve::collisionSurface collisionModel(ini);
collisionModel.addObject(0, position, 1.6f, 1.0f);
ve::motionSimple motionModel(ini, 0, &collisionModel);
```

Besides access to input/output devices and basic data types the veLib also provides a few classes that are useful to implement some physical logic in simulations. Typical recurrent tasks are collision detection and the transformation of user input into object movements. For these purposes the veLib offers basic ready-made motion and collision classes that are instantiated here. The `ve::collisionSurface` models just keeps an object on a surface geometry that is defined in the ini file. The exact parameters have to be defined for each object separately in the `addObject()` method. Its first parameter is just an id to identify the object later on. The further parameters are the object's position that may be altered by the collision, its preferred altitude over ground and maximum step height. The third command finally initializes a motion model object and binds the collision model to it.

```
ve::time timer;
```

Any realtime computer simulations normally needs high performance timing and timer functionality. For this purpose, the veLib offers the class `ve::time`. `ve::time` objects basically provide timer functionality, that means they provide methods that return the exact number and fraction of seconds that have passed since the timer construction as double value. The actual accuracy of timers differ from platform to platform, but should at least have millisecond precision.

Having set up the timer, all objects required in this simulation are available. The main loop can be started:

```
while(!inputButtons[ve::BUTTON_2]&&!inputButtons[ve::KEY_ESCAPE]) {
```

The while statement initializes the main loop of this tutorial program. Each frame of the simulation the complete loop is processed. Two alternative abort criteria are defined: As soon as the inputButtons container contains the value TRUE at the positions `ve::BUTTON_2` or `ve::KEY_ESCAPE`, the loop is not further executed. The two constants are identifiers for key and button states that are defined in `src/include/veTypes.h`.

```
    timer.update();
```

veLib timers behave a little bit differently from timers of other libs. Instead of single `timestamp()` function, there are two complementary methods for requesting the current time. The `update()` method actually requests the current system time and updates the internal time variable of the timer object correspondingly. It is important to know that this value will stay constant until the next call of `update()` irrespectively from the actually passed time. The current steady internal state of the timer can be read without updating using the `now()` access method. While this interface may seem at first glance a little bit quirky, it is in fact very powerful. It conveniently allows to keep a state of virtual contemporaneity over a defined part of a simulation. In this case, `update()` is only called once per frame, thus all commands in one loop cycle virtually happen at the same simulation time.

```
    devWindow.getInput(inputAxes, inputButtons);
```

This command reads the current state of the input device into the passed data structures.

```
motionModel.updateObject(0, inputAxes, position, velocity,
                         acceleration, timer.deltaT());
```

In this line the position and speed variables are updated by the motion model according to the current input state. The first parameter is the object id previously defined integer the `collisionModel.addObject()` call that will be implicitly called from the motion model. Note that the intended motion does not depend on the state of the input axes alone, but also by the time passed since the last frame that is accessible via the `timer.deltaT()` method. Note that the usage of the motion and collision classes is completely optional. One can try this out, for example, by directly coupling input and output by simply copying the inputAxes `ve::vec6f` into the position `ve::vec6f`.

```
devVideo.setOutput(position, inputButtons);
```

Here the updated position is passed to the visualization. In this case where no other parameters are specified, the device interprets the passed coordinate as its observer position, leading to a movement of the camera position.

```
devVideo.update(timer.deltaT());
devWindow.update(timer.deltaT());
```

Finally, both output devices are updated, that means that the draw commands are actually executed by the visualization device and the window buffers are swapped. Again, the time passed since the last frame is passed via the `timer.deltaT()` method to make the simulation speed independent from the actual frame rate of the devices.

```
timer.sleep(0.01);
```

Beside the base functionality similar to a ticking clock, timer objects also allow access to further time related functionality. For example, they also implement platform independent `sleep()` methods. A sleep basically means that the current thread of an application is interrupted, and control is given back to the operating system. Since realtime simulations depend on various services provided by the operation system, it is generally a good idea not to block the CPU completely by the simulation process. Hence, short regular interruptions of the program flow (here for one-hundredth of a second) may actually help to increase the overall performance of an application.

Twenty-five lines of code, and a first full-featured 3D simulation is done. Almost... Besides the program logic, a complete simulation always consists of some content that is displayed or manipulated. The veLib encourages the division of content and logic by using the initialization files for content definition. They will be subject of the following section.

2.4.2 A minimal XML initialization file

Source code iniTut.xml.

```
<?xml version="1.0"?>
<XperiML version="1.0">
  <deviceWindow id="0" deviceContainer="input">
    <string id="winTitle"> veLib tutorial </string>
    <float id="mouseRelative">  0 </float>
    <float id="mouseNeutral"> 0.1 </float>
    <bool  id="mouseVisible">   0 </bool>
    <bool id="fullScreen">      0 </bool>
    <int  id="zBufferBits">    16 </int>
    <int id="winSizeX">       800 </int>
    <int id="winSizeY">       600 </int>
  </deviceWindow>
```

```
  <deviceGraphics id="0" class="deviceGraphicsGL">
    <float id="nearClipping"> 0.25 </float>
    <float id="farClipping">  1000 </float>
    <float id="frustumLeft">  -1.0 </float>
    <float id="frustumRight">  1.0 </float>
    <float id="frustumBottom">-.75 </float>
    <float id="frustumTop">    .75 </float>
    <float id="frustumScale">  1.0 </float>
    <bool  id="backFaceCulling"> 1 </bool>
  </deviceGraphics>

  <motion id="0" class="motionSimple">
    <float id="translSpeedMax">     10   </float>
    <float id="translAccFactor">    2    </float>
    <float id="translDecFactor">    5    </float>
    <float id="rotSpeedMax">        90   </float>
    <float id="rotAccFactor">       120  </float>
    <bool   id="invertAxisH">       1    </bool>
  </motion>

  <resources basePath="">
    <resource mime="model/vrml" id="1" name="ground"
  url="enviro1.wrl"/>
    <resource mime="model/vrml" id="2" name="background"
  url="hemisphere01.wrl"/>
    <resource mime="model/vrml" id="7" name="temple"
  url="stonehenge.wrl"/>
  </resources>

  <scene>
    <object id="1" shape="1" surface="1" pos="0 0 0"/>
    <object id="2" shape="7"             pos="35 -91 0 -90 0 0"/>
    <light id="0" enabled="1" position="-.5 -1.0 1.0 0.0"
  ambient="0.3 0.3 0.3 1.0"  diffuse="0.7 0.7 0.7 1.0"
  specular="1.0 1.0 1.0 1.0"/>
    <background shape="2" pos="0 0 -2.5"/>
  </scene>
</XperiML>
```

When looking at an initialization file, one sees immediately that their content is structured in several sections. The first three sections initialize various internal parameters of the devices and the motion model (see `veLibXml.pdf`). In this tutorial, the `<resources>` and `<scene>` sections that define the scene content are briefly explained.

The `<resources>` section defines the content that is in principle available to the simulation. Each resource is normally specified by its mime type (i.e., a standardized way to describe file contents, initially defined for eMailing, see, e.g., `http://www.mindspring.com/~mgrand/mime.html`) and a URL defining its (relative) location in the file system. When a device gets initialized, it parses the passed ve::xmlIni object for recognized resources and loads them into memory, so that they are readily available as soon as they are used to instantiate a simulation object. Resources are identified via their id, which can be any arbitrary integer number larger than zero.

The `<scene>` section defines the simulation objects that are to be loaded at startup. Typically it is used to define a static scenery that will not be changed during the simulation. Similar to the

resources section, each devices parses its ini file for recognized keywords. In this example two objects are defined that are recognized by the graphics device `ve::deviceGraphicsGL` by the keyword attribute `shape`. the argument following the shape attribute defines the resource id to be used to instantiate the object. Also scene objects are identified in the simulation by an id for which the same rules apply as for resource ids. The same numbers can be used for resource and object ids, since they address different memory areas. Further attributes either address additional devices, or specify the object further. In this example, the `pos` attribute defines the position and orientation of the objects in 3D space. Further sub-statements of `<scene>` specify general parameters such as lighting that may be interpreted by some devices as well.

2.4.3 A minimal scalable program

Source code tut02.cpp.

```
// demo/tut02.cpp
#include <veLib.h>

int main( int , char** ) {
    ve::xmlIni ini;
    ini.load("iniTut.xml");
    ve::deviceContainer devices(ini);
    ve::dataContainer observer;
    observer.position().set(0.0f, -110.0f, 1.6f, 310.0f, 0.0f, 0.0f);
    ve::collisionSurface collisionModel(ini);
    collisionModel.addObject(observer, 1.6f, 1.0f);
    ve::motionSimple motionModel(ini, 0, &collisionModel);
    ve::chrono timer;
    while(!observer.buttons()[ve::BUTTON_2]
          &&!observer.buttons()[ve::KEY_ESCAPE]) {
        timer.update();
        devices.getInput(observer, ve::DC_MASK_INPUT);
        motionModel.updateObject(observer,timer.deltaT());
        devices.setOutput(observer, ve::CMD_OBJECT_SET);
        devices.update(timer.deltaT());
        timer.sleep(0.01);
    }
    return 0;
}
```

Step by step. Compared to the previous example, this fortunately looks quite familiar, and, surprisingly, the second program is even shorter! We will now have a closer look at the differences.

```
    ve::deviceContainer devices(ini);
```

The first decisive difference is the replacement of the window and graphics device by a single ve::`deviceContainer` object. The ve::`deviceContainer` is not a direct representation of a physical device or library interface, but a virtual placeholder for any standard `ve::device` descendant or even a collection of them. It has the same standard interface as any other `ve::device`. The actual instantiation is defined by the passed `ve::xmlIni` object, for each device definition found there a device object will be created. So, any later changes of the input/output devices do not need adjustments of the source code but only an exchange of the initialization file, and several initialization files describing different hardware combinations can be used parallelly.

```
ve::dataContainer observer;
observer.position().set(0.0f, -110.0f, 1.6f, 310.0f, 0.0f, 0.0f);
```

The second difference is the utilization of the `ve::dataContainer` class instead of using the basic `ve::vec6f` and `ve::flag128` classes explicitly. A `ve::dataContainer` object is essentially a standardized collection of 4 `ve::vec6f` objects, 2 `ve::flag128` objects, and a few further ids and user data variables. The container is designed to completely describe the state of typical simulation entities by just one object. All central veLib classes (e.g., devices, motion model, collision) offer interface methods accepting `ve::dataContainer` objects instead of numerous single parameters. The `ve::dataContainer` class itself offers various access methods to read or change its content in a similar way as changing corresponding basic components directly. The second code line shows such an access to one of the `ve::vec6f` sixdofs. By using the `position()` access method, all `ve::vec6f` methods are available to manipulate or read the position data. Further access methods will be demonstrated below.

```
collisionModel.addObject(observer, 1.6f, 1.0f);
```

Instead of using and tracking an isolated id as in the previous example, now simply the complete observer object is registered at the collision model. The internal object id of the `ve::dataContainer` is accessible via the `objectId()` methods if necessary.

```
while(!observer.buttons()[ve::BUTTON_2]
      &&!observer.buttons()[ve::KEY_ESCAPE])
```

The control statement of the main loop now checks two flags of one `ve::flag128` member of the `ve::dataContainer` by using the `buttons()` method.

```
devices.getInput(observer, ve::DC_MASK_INPUT);
```

The user input is read into the suitable `ve::vec6f` and `ve::flag128` data structures of the `ve::dataContainer`. To guarantee that only these data fields are overwritten, the optional write mask `ve::DC_MASK_INPUT` is defined as second parameter.

```
motionModel.updateObject(observer,timer.deltaT());
```

The motion model reads the user input and velocity information stored in the `ve::dataContainer` and updates all affected data fields of the container accordingly.

```
devices.setOutput(observer, ve::CMD_OBJECT_SET);
```

Subsequently, the observer object containing the updated camera position is passed to the `ve::deviceContainer` that distributes it to all affiliated devices. The optional second parameter specifies the command to be executed. In this example `ve::CMD_OBJECT_SET` brings the corresponding internal representations of the devices in line with the data stored in the observer object.

```
devices.update(timer.deltaT());
```

Finally, all devices within the ve::`deviceContainer` are updated, in this example the scene gets redrawn and the buffers are swapped.

So, the data flow of a typical scalable veLib application can be graphically illustrated as in Figure 2.2.

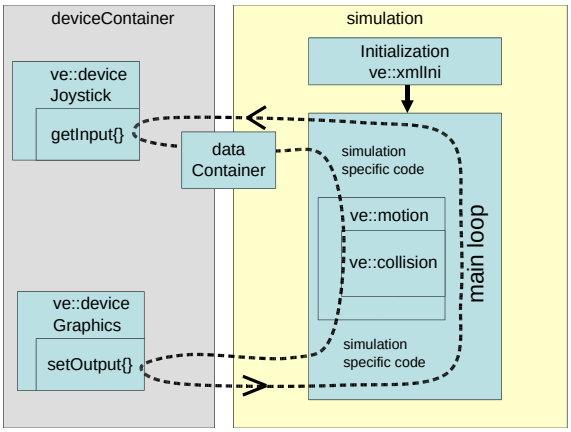


Figure 2.2: Diagram illustrating the basic structure and data flow of normal veLib applications.

Why is this program scalable? Well, there are basically to reasons for that. First, the ve::`dataContainer` combines data structures to completely describe a typical simulation object in one handy C++ object. This makes it much easier to keep simulation code concise that consists of several objects (e.g., multiple observers). More important, the ve::`deviceContainer` completely decouples experiment logic from the instantiation and presentation. A change of the simulation setup (e.g., from desktop during development to distributed VR for the final product) becomes very easy by just adjusting the initialization file. One can drive this even further by using <include> statements in the initialization file to separate device descriptions completely from the resources and scene description (see `veLibXml.pdf` for further information).

2.4.4 Advanced veLib demos - short description

This section provides a brief overview on all demos that are shipped with the veLib. They are located in the demo subdirectory. Note that in contrast to the minimal tutorials presented above, the source code of these demos is heavily annotated, so, they should be widely self-explanatory, and therefore are presented only very briefly in this section.

demo01_helloWorld This is the most simple demo of a veLib based stand-alone application. The program opens a window, loads, and renders a well-known message. The gaze direction can be controlled via the mouse. Press escape or middle mouse button to exit.

demo02_OpenGL This is a small demo application that demonstrates how to use the veLib for low level OpenGL coding. The veLib handles input events and provides a rendering context. The demo renders a spinning colored triangle. Do not bother about various `ve::xml` warnings, here default settings are used, so no XML initialization file is loaded.

demo03_motion This is a simple demo of a veLib based stand-alone application. The program opens a window, loads, and renders a simple scene. The observer position can be controlled via mouse and keyboard. It uses a basic motion model as well as collision to keep the. observer on the ground. Press escape or button 2 to exit.

demo04_animation This is the extended version of demo03. The program opens a window, loads, and renders a scene. An animated object is added that makes a lot of noise. As an additional bonus, the current frame rate is calculated to demonstrate the superior timing qualities of the veLib. The observer position can be controlled via a joystick. Use button 2 to exit.

demo05_dataContainer This demo switches demo04 to the ve::dataContainer. The functionality is the same as in demo04: The observer position can be controlled via mouse and keyboard.

demo06_deviceContainer This demo extends demo05 by using the ve::deviceContainer instead of single input/output devices. This allows to change input/output devices without recompiling the demo by solely adjusting the ini file (iniDemo06.xml). The functionality is the same as in demo04: The observer position can be controlled via mouse and keyboard.

demo07_dynamicScene This demo extends demo06 by showing a scene change triggered by the simulation. The observer position can be controlled via mouse and keyboard.

demo08_multiUser This demo extends demo 06 by introducing a second human-controlled observer. Both are represented by a device- and data container. You can adjust the actual device settings in the user specific initialization files (iniDemo08_1.xml for user 1, iniDemo08_2.xml for user 2). Use button 2 to exit.

demo09_multiPipe This demo does virtually the same as demo06 but uses a different ini file (iniDemo09.xml). Therefore, the visualization is distributed on 3 pipes. You may adjust the ini file to distribute the simulation on different computers. Do not run this demo directly, but use demo09_run.sh, resp. demo09_run.bat to initialize the necessary visualization clients in advance.

demo10_audio3d A test application for spatial audio. Do not forget to unzip demo10Audio3d_resources.zip before running it in order to make the demo work.

demo11_xml A small demo introducing the functionality of the ve::xmlIni class and veLib initialization files. Besides the initialization of variable values, the usage of include files is demonstrated.

Chapter 3

Initialization File Documentation

Introduction

Many parameters of a veLib simulation can be adjusted by using initialization files. This, on one hand, allows users to make changes without having to recompile any code. Also, by using the xml syntax, one can have all different sorts of parameters in a single file and presented in an organized, structured and human-readable way.

All veLib classes that can be initialized by an initialization file try to read specific key values from files. In case the file cannot be found or some values are not specified, standard settings will automatically be used.

All possible values for all veLib classes that read from ini files will be described here. In addition, one may define own xml structures and use the veLibs xml parser to read in the values. Check out the xmlDemo.cpp in the demo directory for an example on how to do that.

For more information on xml in general, you may have a look at `http://www.xml.org/`.

Syntax overview

The veLib inifiles make widely use of a simple xml-based initialization language, called xmlIni (see the file veXml.h and the veLib documentation on the veXml class for details). Defined there are a few basic constructs to read initialization values into standard data types. The major part of the inifiles is based on these constructs. So here are a few examples to get used to the general syntax:

`<float id=”xyz”> 13.4 </float>`
Defines a float with the id xyz and the value 13.4

`<int id=”abc”> 80 </int>`
Defines an integer with the id abc and the value 80

`<string id=”str”> a string </string>`
Defines a string with the id str and the value a string

`<intArray id=”ia”> 23 2 46 </intArray>`
Defines an array of 3 integers with the id ia and the values 23, 2 and 46 (Note that in an int or float array, the values have to be separated by a whitespace)

```
<stringArray id=”sa”>
one string <br/>
another string <br/>
</stringArray>
```
Defines an array of two strings with the id sa and the values a string and another string (Note that in a string array, values have to be newline separated, use the
 tag for that).

As one can see from the examples, the intialization of standard data types follows a common scheme. The xml tag defines the data type (int, float, string, intArray, floatArray, stringArray), and the id attribute is used to specify the (external) variable name. Variable values are read from the content section of the xml statement between the opening and closing tags.

In addition, there are a few special constructs and a predefined framework structure using a tree-like syntax to initialize different veLib classes. These special constructs and the main branches of the overall structure are described in the following sections. Also, you can have your custom constructs to initialize your own application, store experimental data and so on. For these special purposes, please refer to the veXml class documentation for an overview of all functions and the veLibDemos for some examples.

Note: All values that appear in the sub tags descriptions of this document are the default values for the respective sub tag. That is, if you dont specify this tag in your inifile, this value will automatically be set. The default values for attributes are explicitly named were necessary.

Runtime access to variables

A feature that has been added very recently (version 1.2.0) to the veLib is a framework for reading and (sometimes) writing selected internal variables of veLib devices during runtime from the applications. The runtime access is performed using ve::dataContainer classes and the generic getInput()/setOutput() interface of veLib devices. The syntax of a runtime access from the C++ application side may be best explained by a few examples:

// read the horizontal window size from a ve::deviceWindow:
```
dataChar dc(0,”winSizeX”); // the variable name is defined in the data() section
dc.command()=CMD_DEVICE_VAR_GET;
devWindow.getInput(dc);
unsigned int winSizeX=dc.dataUI(0);
```

// change the window title:
```
dataChar dc(0,”winTitle=my new lengthy window title”);
/* in case of string variables, both the variable name and
value have to be passed in the ve::dataChar’s data() section */
devWindow.setOutput(dc, CMD_DEVICE_VAR_SET);
```

// change the right frustum border of a ve::deviceGraphicsGL:
```
dataChar dc(0,”frustumRight”);
dc.dataF(0)=2.0f;
dc.command()=CMD_DEVICE_VAR_SET;
devWindow.setOutput(dc);
```

As one can see from the examples, the variable names are the same as in the xml initialization files. Depending on whether a variable is read or written, either the data container commands CMD_DEVICE_VAR_GET or CMD_DEVICE_VAR_SET is used. Depending on the data type, the actual value is passed in both directions either in the dataUI(0) (integer, unsigned integer, boolean), in the dataF(0) (float, double), or in the data() field (strings, char arrays). In case of a write operation on strings, variable name and value are separated by an equality “=” character.

The sign of an integer variable can be read from the dataChar's dataUI(1), which is 0 in case of positive values and 1 in case of negative numbers.

Note that up to now only a subset of internal device variables is exposed via this framework, an exposure is explicitly mentioned in the following sections. One the one hand, this is due to the relative novelty of the feature, therefore only a few exposures have been implemented. On the other hand, not every initialization variable defined in an ini file is suitable for runtime manipulation, many basic parameters can only be set once during initial class instantiation. Therefore, if your application requires access during runtime to a particular variable which is not yet exposed, please contact the current maintainers, if runtime access is feasible, it can be implemented almost instantly.

3.1 deviceWindow

Set attributes of the application window, the mouse behavior and the overlay plane. For the overlay, there is a special section, where one can define the planes colors, dimensions and add overlay objects, very similar to adding objects to the 3D-scene in the scene-section (see below).

Main tag: `<deviceWindow>`

Possible Attributes:

- id : You may specify more than one window setting in a single xml file (for a multiscreen projection for example). Make them distinguishable by providing a unique id for every deviceWindow section.
- deviceContainer : If you use the deviceContainer class (see the veLib documentation for details), you can choose with this attribute, whether a device should be explicitly excluded from the container or if it should act as the input device. deviceContainer=ignore means that the deviceContainer class will ignore this device when parsing the ini file. deviceContainer=input will make this device the input device of the application. If no device has this attribute, the deviceWindow will be the input device, meaning input via mouse and keyboard. Other possible input devices are currently deviceJoystick and deviceNetwork.

Example: `<deviceWindow id="0" deviceContainer=”input”>`

Sub tags

`<string id="winTitle"> veLib window </string>`
A string that will appear in the window title bar if not in fullscreen mode. *This variable is accessible for read and write operations at runtime.*

`<float id="mouseRelative"> 0.0 </float>`
If zero, the current mouse position is reported in absolute coordinates in the range of -1.0 to 1.0 in x and y direction. If nonzero, the current mouse position is reported relative to the last mouse position, scaled by this value.

`<float id="mouseNeutral"> 0.0 </float>`
Defines a square area in the center of the screen in which mouse coordinates will always be reported as zero. In the above example, is the mouse is in the range of -0.1 to +0.1, the reported position will be 0 (deadzone) (applies for x and y direction).

`<bool id="mouseVisible"> 1 </bool>`
Mouse cursor will be invisible if zero, visible if non-zero. *This variable is accessible for read and write operations at runtime.*

`<bool id="fullScreen"> 0 </bool>`
Application will run in window mode if zero, fullscreen if non-zero

`<int id="zBufferBits"> 16 </int>`
The number of bits of one value in the depth buffer. The more bits, the higher the accuracy, but highest possible value depends on your video card. Internally, the depth buffer size is set to the closest possible value.

`<int id="stencilBufferBits"> 8 </int>`
The number of bits of one value in the stencil buffer.

`<int id="winSizeX"> 640 </int>`
`<int id="winSizeY"> 480 </int>`
Sets the size of the application window in pixels. *These variables are accessible for read operations at runtime.*

`<overlay>`
The overlay plane is a two-dimensional plane in front of the 3D-scene. It can be used for displaying text messages or 2D-images. This tag opens the subsection for the overlay plane. It is a special section, where the dimensions and colors for the overlay area are set and were objects are added to the overlay plane. It has its own sub tags, which are:

`<floatArray id=”bgNormalColor”> 0.0 0.0 0.0 1.0 </floatArray>`
currently unused

`<floatArray id=”fgNormalColor”> 1.0 1.0 1.0 1.0 </floatArray>`
sets the foreground color, meaning the color in which text is drawn.

`<floatArray id=”bgSelectColor”> 0.0 0.0 0.0 1.0 </floatArray>`
currently unused

`<floatArray id=”fgNormalColor”> 0.8 0.8 0.8 1.0 </floatArray>`
currently unused

```
<float id=”minX”> -1.0 </float>
<float id=”minY”> -1.0 </float>
<float id=”maxX”> 1.0 </float>
<float id=”maxY”> 1.0 </float>
```
maps the dimensions of the overlay plane to these values. By doing this, overlay objects will always appear at the same location, no matter what the size of the window is. With the above dimensions, you could, for example, set the position of an overlay object to (0.0, 0.0) and it would always show up in the center of the window, independently of the display windows resolution.

```
<axesInputMapping>
<axesInputScale>
<axesInputShift>
```
Those can be used to map, scale and shift axes of incoming input data. See deviceJoystick for a detailed description.

`<object>`

Possible attributes:

- id : every objects needs an unique id
- resource : the resource id of the 2D object that is to be added to the overlay plane. See the resources section for a list of possible objects.
- position : two float values defining the position of the overlay object. Please note that the position has to be inside the mapped dimensions for the overlay plane (see above). Otherwise, the object will be outside the plane and be partially or fully clipped. Default is (0.0, 0.0)
- size : two float values by which the object will be scaled in x- and y-direction, does NOT apply to text messages! Default is (1.0, 1.0)

- hAlign : horizontal alignment of the object. Possible values are -left -center -right Default is center.
- vAlign : vertical alignment of the object. Possible values are -bottom -center -top Default is center.

3.2 deviceGraphics

Set attributes for the viewing frustum and the camera as well as general drawing properties of the underlying graphics device.

Main tag: `<deviceGraphics>`

Possible attributes:

- id : see deviceWindow
- class : currently unused
- deviceContainer : see deviceWindow

Example: `<deviceGraphics id="0" class="deviceGraphicsGL">`

Sub tags

`<float id="nearClipping"> 1.0 </float>`
Objects which are closer to the camera than this value will be clipped. *This variable is accessible for read and write operations at runtime.*

`<float id="farClipping"> 100.0 </float>`
Objects which are further away from the camera than this value will be clipped. *This variable is accessible for read and write operations at runtime.*

```
<float id="frustumLeft"> -1.0 </float>
<float id="frustumRight"> 1.0 </float>
<float id="frustumBottom">-.75 </float>
<float id="frustumTop"> .75 </float>
```
Use these to set the dimensions of the viewing frustrum. *These variables are accessible for read and write operations at runtime.*

`<bool id=”backFaceCulling”> 0 </bool>`
Polygon backfaces are not culled if zero, culled if non-zero. Culling backfaces can increase performance but polygons viewed from the wrong side will be invisible.

`<bool id=”textureCompression”> 0 </bool>`
If set to non-zero, the veLib texture loader will try to compress textures at load time. This may increase load time significantly but also reduces the amount of texture memory that is used and therefore can speed up rendering for large models on graphic cards with little texture RAM. Please note that the quality and effectiveness of the compression highly depends on your graphics card and driver. It may not work at all if your card does not support OpenGL 1.3 or higher. In that case, this tag will have no effect, compression will always be turned off.

`<float id=”anisotropicFilter”> 1.0 </float>`
This will turn on anisotropic texture filtering, if your graphics card supports it, otherwise this tag will have no effect. Set the value to the desired quality of texture filtering. Common values are 2.0, 4.0 and 8.0. 1.0 means no anisotropic filtering. The higher the value, the better the texture filtering will be but rendering will be slowed down. If you set this to a value higher than supported by your graphics hardware, it will internally be set to the highest possible value automatically. Please note

that the speed and quality of anisotropic filtering is highly dependent on your graphics card and driver.

3.3 deviceAudio

Set attributes for the underlying audio device.

Main tag: `<deviceAudio>`

Possible attributes:

- id : see deviceWindow
- class : currently unused
- deviceContainer : see deviceWindow

Example: `<deviceAudio id=”0” class=”deviceAudioAL” deviceContainer=”ignore”>`

Sub tags

`<int id=”distanceModel”> 1 </int>`
Sets the distance model for the audio device and therefore how sounds are attenuated as they move away from the listener. Possible values are

- 0 None
- 1 Inverse Distance
- 2 Inverse Distance clamped (please refer to the OpenAL documentation for details)

`<float id=”dopplerVelocity”> 330.0 </float>`
Sets the velocity for Doppler effect calculations (please refer to the OpenAL documentation for details).

`<float id=”dopplerFactor”> 1.0 </float>`
Sets a scaling factor for Doppler effect calculations (please refer to the OpenAL documentation for details).

3.4 deviceNetwork

Set attributes for the underlying network device.

Main tag: `<deviceNetwork>`

Possible attributes:

- id : see deviceWindow
- class : currently unused
- deviceContainer : see deviceWindow

Example: `<deviceNetwork id=”0” class=”deviceNetwork”>`

Sub tags

`<intArray id=”connectPorts”> </intArray>`

`<stringArray id=”connectHosts”> </stringArray>`
Network devices, which run in client mode, will read these values and automatically try to connect to all servers specified in connectHosts by using connectPorts. If you want to connect to more that one host, make sure that the order in which you specify hosts and ports is correct. For example, to make your client connect to a host display on port 5010 and a host joystick on port 5020, the two xml arrays have to look like this:

```
<intArray id=”connectPorts”> 5010 5020 </intArray>
<stringArray id=”connectHosts”>
display <br/>
joystick
</stringArray>
```

(Please note that you have to separate strings in a <stringArray> with the
 tag).

`<int id=”serverPort”> 0 </int>`
Network devices, which run in server mode, will open this port and listen to incoming connection requests from clients.

`<bool id=”predictMotion”> 0 </bool>`
If the network device is used to receive motion data from a client (typically for a display server), then it can perform a motion prediction to avoid jerks in the displayed movements. If the device is in client mode, no motion prediction will be performed, no matter what value you set here. If the device is in server mode, motion prediction will be performed on incoming positional data if this value is set to 1.

`<axesInputMapping>`
`<axesInputScale>`
`<axesInputShift>`
Those can be used to map, scale and shift axes of incoming input data. See deviceJoystick for a detailed description.

3.5 deviceJoystick

Set attributes for the underlying joystick device.

Main tag: `<deviceJoystick>`

Possible attributes:

- id : see deviceWindow
- class : currently unused
- deviceContainer : see deviceWindow

Example: `<deviceJoystick id=0 class=”deviceJoystickSDL”>`

Sub tags

`<int id=”joystickNumber”> 0 </int>`
If more that one joystick is attached, choose here which one to use, starting from zero. Note that the order of the joysticks is OSdependent and not even consistent within one OS. It may happen for example that ,even if only one joystick is plugged, it is reported as joystick 1 and not joystick 0.

`<axesInputMapping>0 1 2 3 4 5</axesInputMapping>`
Here you can select which physical axes should be put into which of the six inputAxes float values of the data container. The order of the floats in the dataContainer is X, Y, Z, H(eading), P(itch), R(oll). With a setting like above, the joystick axes 0-5 would be put into that order into the data container, meaning that after a getInput call from the joystick device, inputAxes[X] of the data Container would hold joystick axis 0, inputAxes[Y] would hold axis 1 and so on. With a setting like this, `<axesInputMapping>3 1 2 0 4 5</axesInputMapping>` inputAxes[X] would hold joystick axis 3 and inputAxis[H] would hold joystick axis 0. Please note that as for the joystick number, the order in which joystick axes are reported is also OS and driver dependent. So the mapping probably needs to be adjusted every time you use a different joystick. The window device also has physical axes. The keyboards arrow keys are reported as axes 0 and 1, pageUp/Down is axis 2, and mouse movement is axes 3 and 4, axis 5 is unused, mapping rules same as joystick. Also, inputAxes that are reported from a network device can be mapped, mapping rules same as joystick.

`<axesInputScale>1 1 1 1 1 1</axesInputScale>`
The values from input devices can be scaled. Normally, a veLib input device reports values in the range of -1.0 to +1.0. Those are multiplied by the values that you specify here. Flipping an axis for example is as simple as putting -1 as a scale factor for that axis.

`<inputAxesShift>0 0 0 0 0 0</inputAxesShift>`
Shifts the inputAxes by the given values. An axis shift of 1 for example means that this axis will no longer report values between -1.0 and +1.0, but values between 0.0 and 2.0.

3.6 motion

Set attributes for the motion model. The effects of these values highly depend on how the underlying motion model interprets them. The only motion model that currently ships with the veLib, motionSimple for example only takes into account the max speeds for rotation and translation. The same holds true for the units. If you need a translation speed of 10 m/s for example, you have to build your graphical models in the correct size and program an accurate motion model.

Main tag: `<motion>`

Possible attributes:

- id : see deviceWindow
- class : currently unused

Example: `<motion id="0" class="motionSimple">`

Sub tags

`<float id="translSpeedMax"> 30 </float>`
The maximum translation speed.

`<float id="translAccFactor"> 10 </float>`
The translation acceleration.

`<float id="translDecFactor"> 10 </float>`
The translation deceleration.

`<float id="rotSpeedMax"> 55 </float>`
The maximum rotation speed.

`<float id="rotAccFactor"> 10 </float>`
The rotation acceleration factor

`<bool id="invertAxisH"> 0 </bool>`
currently unused.

3.7 resources

Specify which resources should be loaded for the use in a scene. All resources have a type, an unique id and a name, followed by a filename (except for text/plain, see below). If the file is not located in the same directory as your executable, you have to specify the path. Alternatively, you can use the basePath attribute, to set a global search path for all resources. See the scene and the deviceWindow sections for more information on how to place the loaded resources in your scene or add them to an overlay plane. Also available is a container tag, with which one can put different resources into a group. For example, if you have a car model and an engine sound, you could put them both in the same container and would automatically be moved together.

Main tag: `<resources>`

Possible attributes:

- basePath : Set a path which is added in front of the url-attributes of all resources.

Example: `<resources basePath="/home/3dmodels/">`

Sub tags

`<resource>`

Possible attributes:

- mime : set the type of a resource. It can be one of the following:
  - model/vrml : a 3D model description in the VRML format, scene object
  - model/x3d : a 3D model description in the X3D format, scene object
  - application/x-3ds : a 3D model description in the 3ds format, scene object
  - audio/wav : A wave audio file, scene object
  - image/png : An 2D image in PNG file format, overlay object
  - text/plain : Plain text, doesnt need an url-attribute for loading a file, instead it uses the attribute string for getting the text, overlay object
  - image/svg+xml : A 2D filled rectangle, overlay object
  - font/txf : a font used for drawing overlay or 3D text
- id : each resource needs a unique id
- name : currently unused
- url : set the path on were to find a resource file and the filename itself
- string : set the text (applies only to resources of type text/plain)
- gain : set the gain value for a sound resource, default value is 1.0 (applies only to resources of type audio/wav)
- pitch : set the pitch value for a sound resource, default value is 1.0 (applies only to resources of type audio/wav)

- loop : set to either true or false, the audio file of a sound resource will be played in a loop if true, default value is false (applies only to resources of type audio/wav)
- attenuationDist : sound sources with a higher distance to the listener than this value will get attenuated, default value is 1.0. For details on how the attenuation is calculated, please refer to the OpenAL documentation (applies only to resources of type audio/wav)
- size : the font size (applies only to resources of type font/txf)

Examples:

```
<resource mime=”model/vrml” id=”1” name=”building”
url=”models/house.wrl”/>
<resource mime=”audio/wav” id=”2” name=”ding” url=”sounds/ding.wav”/>
<resource mime=”audio/wav” id=”3” name=”music” loop=”true” gain=”2.0”
pitch=”1.0” attenuationDist=”3.0” url=”song1.wav”/>
<resource mime=”text/plain” id=”100” name=”text” string=”this is a
text resource”/>
<resource mime=”font/txf” id=”200” name=”font” size=”24”
url=”Helvetica.txf”/>
```

```
<container>
```

Possible attributes:

- id : each container needs a unique id
- name : currently unused
- shape : the id of a model resource that is to be placed in the container
- sound : the id of an audio resource that is to be placed in the container

Example:

```
<resource mime=”model/vrml” id=”1” name=”carModel” url=”car.wrl”/>
<resource mime=”audio/wav” id=”2” name=”carSound” url=”motor.wav”/>
<container id=”3” name=”car” shape=”1” sound=”2”/>
```

3.8 scene

Specify in this section which resources should be present in your scene. You do this by defining objects and then linking them with resources. By doing this, you could, for example, have multiple tree objects which all share the same resource, which is an easy way of reducing loading time and memory consumption of your scene.

Main tag: `<scene>`

Possible attributes: none

Sub tags

`<object>`

Possible attributes:

- id : each object needs a unique id.
- shape : the resource id of the 3d model that is to be used by this object, or the id of a resource container that contains a 3d model. All objects with this attribute will be added as visible objects to your scene
- surface : the resource id of the 3d model that is to be added to the surface and hence used by the collision detection. Objects can either have just the shape or the surface attribute or both. Objects with just the surface attribute will not be visible in your scene. This is useful when you want to have the collision model to differ from your visible ground. Just add two objects, one with the shape attribute as the ground that is to be drawn and one (could be a simplified version to save collision detection computation time) with the surface attribute for the collision, which is not visible and thus saving rendering time.
- sound : the resource id of the audio resource that is to be used by this object, or the id of a resource container that contains an audio resource.
- pos : six float values defining the initial position and orientation of the object, will be all 0.0 if not specified.
- speed : six float values defining the initial linear and angular velocity of the object, will be all 0.0 if not specified.
- Examples: A complete example containing a resource- and a scene-description follows at the end of this section.

`<background>`

Possible attributes:

- shape : the resource id of the 3d model that is to be used as a background for the scene. Background objects are not affected by lighting and translations of the camera. Currently only 1 backround is allowed at one time. Use it for a hemisphere model with a sky texture for example.
- pos : six float values defining the initial position and orientation of the background. The position relative to the camera will always be the same.

Examples: A complete example containing a resource- and a scene-description follows at the end of this section.

`<light>`

Possible attributes:

- id : each light needs a unique id. The light ids correspond directly with the OpenGL light numbers. OpenGL supports at least 8 lights, so you can safely specify light ids from 0 to 7. The maximum number of lights depends on your system and OpenGL implementation. Please check an OpenGL documentation for details.
- enabled : specify 0 and the light will be turned off at program start, specify 1 and the light will be turned on at program start.
- pos : four float values. The way in which the first three values are interpreted, depends on the fourth value. If it is zero, the light is directional, and the first three values specify the direction on the light. If the fourth value is non-zero, the light is positional and the first three values specify the position of the light. This is the standard OpenGL way of defining light positions. See any OpenGL documentation for more information on lighting.
- ambient : four float values describing the ambient intensity of each of the lights color components (red, green, blue, alpha).
- diffuse : four float values describing the diffuse intensity of each of the lights color components(red, green, blue, alpha).
- specular : four float values describing the specular intensity of each of the lights color components(red, green, blue, alpha).

If no light is added to the scene, lighting will be disabled and all objects are drawn using their diffuse color.

Examples: A complete example containing a resource- and a scene-description follows at the end of this section.

`<fog>`

Possible attributes:

- start : if objects are further away from the viewpoint that this value, their color will be affected by the fog.
- end : if objects are further away from the viewpoint that this value, they will be totally invisible.
- density : currently unused, since the graphics class veDeviceGraphicsGL only supports fog mode GL_LINEAR.
- color : four float values describing the four color components(red, green, blue, alpha) of the fog.

Example

The following example shows how to define some resources and how to use them in your scene. First, define the resources in the corresponding section:

```
<resources basePath=”/home/gf/”>
<resource mime=”model/vrml” id=”0” name=”visibleFloor”
url=”models/visFloor.wrl”/>
<resource mime=”model/vrml” id=”1” name=”collisionFloor”
url=”models/colFloor.wrl”/>
<resource mime=”model/vrml” id=”2” name=”background”
url=”models/backgrnd.wrl”/>
<resource mime=”model/vrml” id=”3” name=”carModel”
url=”models/car.wrl”/>
<resource mime=”audio/wav” id=”4” name=”carSound” url=”audio/sounds/engine.wav”/>
<container id=”10” name=”car” shape=”3” sound=”4”/>

</resources>
```

next comes the scene definition:

```
<scene>
<object id=”0” shape=”0”/>
```

an object is added, referencing resource id 0 as a shape, so the model visFloor.wrl will be added as a visible object (at position (0,0,0,0,0,0), since no position was specified).

```
<object id=”1” surface=”1”/>
```

this object references resource id 1 as a surface, so the model colFloor.wrl will be used for collision detection and will NOT be visible.

```
<object id=”2” shape=”10” sound=”10” pos=”12.3 4.0 1.0 0 0 0”
speed=”2.0 0 0 0 0 0”/>
```

this object references the container (resource id 10) for shape and sound, so it will be added as a hearable and visible model at the specified position moving with the specified speed.

```
<background shape=”2”/>

<light id=”0” enabled=”1” position="-.5 -1.0 1.0 0.0" ambient="0.3 0.3
0.3 1.0" diffuse="0.7 0.7 0.7 1.0" specular="1.0 1.0 1.0 1.0"/>
```

this will add a directional light (4th component of position is zero) with the specified color intensities.

```
</scene>
```

3.9 deviceContainer

The deviceContainer is a special veLib class that frees users from having to instantiate every single device in the applications source code. Instead, just one deviceContainer object is needed, which will automatically parse the xml file and create all devices accordingly. That allows for easy exchange of devices with no need to recompile any code. You can instruct the container to skip certain devices by using the deviceContainer attribute, which can be attached to any deviceXXX tag (see the section on deviceWindow for details). For a detailed description on how to use the deviceContainer class in your application, please refer to the tutorial section in veLibIntro.pdf and have a look at demo/demo06_deviceContainer.cpp.

In addition to the dynamic initialization and administration of devices, the ve::deviceContainer offers some further services:

Application information

All good applications should provide a minimal documentation and command line help on their function, authorship, basic startup options, and parameters. A convenient way for implementing this is using the applicationInfo capabilities of deviceContainers. They generate a short online help in case that the application is started with the options '-h' or '–help', or if wrong startup parameters are passed. The syntax of the application info xml tags becomes apparent from the following example:

```
.<applicationInfo>
.  <string id="author"> Your name and eMail address </string>
.  <string id="version"> some version number, e.g., 0.9.1 </string>
.  <string id="date"> The release date, e.g., 2005-04-06 </string>
.  <string id="shortDescr"> A brief sentence that explaines what the application does.
</string>
.  <string id="descr"> More detailed information can be provided here. </string>
.  <string id="usage">
.     All required and optional startup parameters, e.g.
.     [-posX,Y,Z(,H,P,R)] [-cCollisionModel.wrl] model.wrl
.  </string>
.</applicationInfo>
```

Log file

The device container is capable of recording or displaying the data container traffic that is passed to it. This feature is particularly convenient during the implementation and debug stage when developing a new application, but may also used for instance to record and replay user input.

Main tag: `<deviceLog>`

Possible attributes:

- url: (optional) The file name which is used as logfile. If the file already exists, new information is appended. If no url attribute is provided, log data is written directly to stdout.

Example: `<deviceLog url="logFile.txt"/>`

3.10 Additional tags

`<?xml version=”1.0”?>`
This is the standard xml version tag. Every xml file should start with this one. Nothing to worry about, just make sure its there. This tag does NOT need to be closed at the end of the file.

`<XperiML version=”1.0”>`
This tag marks the beginning of the veLib related part of a xml file. All xml code that is used for any kind of veLib initialization or data for experiments that make use of the veLib should be enclosed by this tag. It needs to be closed at the end of the section, like so: `</XperiML>`

`<include>`
This tag can be used to include other xml files into the one that contains the include tag. If you have, for example, a desktop experiment with 5 different xml files for 5 different experiment setups, but most of our initializations are the same all the time then you could just put those in a separate file and include it in the other files. Or you could use it for easily switching between different setups (desktop vs. 3 pipe projection,) by including the appropriate ini file. See the xmlDemo for an example.

Possible attributes:

- url : the path and the filename of the xml file that is to be included.

Example: `<include url=”../iniDesktop.xml”>`

Part II

veLib API reference

Chapter 4

veLib Directory Hierarchy

4.1 veLib Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 5

veLib Namespace Index

5.1 veLib Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 6

veLib Hierarchical Index

6.1 veLib Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

veLib Class Index

7.1 veLib Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 8

veLib File Index

8.1 veLib File List

Here is a list of all documented files with brief descriptions:

Chapter 9

veLib Page Index

9.1 veLib Related Pages

Here is a list of all related documentation pages:

Chapter 10

veLib Directory Documentation

10.1 e:/src/veLib/src/include/ Directory Reference

Files

- file veCollision.h
- file veConfig.h
- file veDataContainer.h
- file veDevice.h
- file veDeviceAudioAL.h
- file veDeviceContainer.h
- file veDeviceDirectX.h
- file veDeviceGraphicsGL.h
- file veDeviceNetwork.h
- file veDeviceSDL.h
- file veGeoObj.h
- file veGlUtils.h
- file veImage.h
- file veIo3ds.h
- file veLib.h
- file veMath.h
- file veMotion.h
- file **vePlugins.h**
- file veStd.h
- file veStrUtils.h
- file veTypes.h
- file veUtils.h
- file veXml.h

10.2 e:/src/veLib/src/ Directory Reference

Directories

- directory include

Chapter 11

veLib Namespace Documentation

11.1 std Namespace Reference

Additions to the standard namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const ve::image &img)

11.1.1 Detailed Description

Additions to the standard namespace.
In many classes (p.e. veMath, veXml, ...) the standard output operator<<() is overloaded.

11.1.2 Function Documentation

11.1.2.1 std::ostream& std::operator<< (std::ostream & *os*, const ve::image & *img*)

operator for output in streams

11.2 ve Namespace Reference

veLib standard namespace

Classes

- class collision
  base class for collision models.
- class _collisionSurfaceObj
- class collisionSurface
  collision model based on motion surfaces.
- class dataBaseStruct
  internal struct used as coomon data container header.
- class dataCharStruct
  internal struct underlying the dataChar class
- class dataContainerStruct
  internal struct underlying the dataContainer class
- union dataUnion
  internal union comprising all low level data structures of various data containers
- class dataChar
  veLib basic data class.
- class dataContainer
  veLib data container representing a simulation object
- class _exposedVar
- class device
  The general device class.
- class deviceGraphics
  base class for 3D visualization classes.
- class deviceAudioAL
  a class for 3D audio simulation based on OpenAL
- class deviceLog
  A (pseudo-)device that writes its traffic into log files or to stdout.
- class deviceContainer
  The flexible device container.
- class glBillboard
  a class for rendering billboards
- class glBillbAnim

a class for rendering animated billboards

- class _modelRef
- class deviceGraphicsGL
  OpenGL based graphics device.
- struct networkTime
- struct delayedData
- struct mandatoryData
- struct connectionInfo
  a struct with information on one network connection
- class deviceNetwork
  Network device class using pure UDP.
- class deviceWindow
  class for window handling, keyboard and mouse input.
- class deviceJoystick
  class for joystick input.
- class ioFileHandler
  a small auxilliary class that allows the writing of plugin file handlers
- class ioVrml
  a class for VRML input/output
- class ioX3d
  a class for X3d input/output
- class geoObj
  base class for all derived geometry objects.
- class geoGroup
  base class for organizing ve::geoObjects in a tree-like structure.
- class geoMesh
  a class for static indexedFaceSet mesh objects.
- class geoElevationGrid
  a class for interpreting and displaying elevation grids / terrain models
- class glText
  an abstract base class for OpenGL font renderers.
- class glTextTxf
  a class for rendering txf texture fonts in OpenGL.
- class ovlObj
  parent class for 2D overlay objects.
- class ovlLabel

a single line text overlay widget.

- class ovlRect
  a class for displaying untextextured rectangles in the overlay plane.
- class ovlImage
  a class for displaying images in the overlay plane.
- class image
  ve::image is a class for basic image file input/output.
- class glTexture
  a class for OpenGL image and texture operations.
- class _materialInfo
- class _materialRef
- class _3dsObject
- class io3ds
  this class handles the loading of 3ds files
- class vec2f
  a class for 2d vector and vertex geometry operations.
- class vec3f
  a class for vector and 3D vertex geometry operations.
- class vec4f
  a class for 4D vector geometry operations.
- class sphere
  a class representing a sphere.
- class plane
  a class representing a plane.
- class triangle
  a class for triangle geometry.
- class line
  a class for line mathematics.
- class vec6f
  a class representing a six degree of freedom coordinate.
- class frustum
  class for frustum (clipping) operations.
- class rnd
  an extension of c random number functions.
- class mat4f
  a class for typical 3D geometry 4x4 matrix operations.

- class matStack4f

  a class for typical 3D geometry 4x4 matrix stack operations such as in OpenGL.

- class motion

  Base class for all motion model classes.

- class motionSimple

  A very simple motion model. This class implements a basic generic motion model, all speed and acceleration factors are widely adjustable by an xml initialization file.

- class plugin
- class pluginHandler
- class flag128

  a class for storing a large number of flags.

- class chrono

  time / timer class

- struct fileInfo
- class fileIo

  class facilitating file input/output operations.

- class cmdLine

  a simple static class for preparsing command line arguments and options

- class xml

  basic XML class.

- class xmlIni

  A class for reading variable values from XML inifiles. This class complements the basic xml class with convenient parsing functions for basic data types. It is a good way for reading initialization information and for transfering information between programs in temporary files.

definitions and constants

some handy definitions of bitmasks and bytemasks for packing multiple flags into variables.

- const unsigned int BIT [ ]
- const unsigned int BYTE [ ] = { 0xFF, 0xFF00, 0xFF0000, 0xFF000000 }

Typedefs

- typedef int(∗ pluginInitFunc)()
- typedef void(∗ pluginUpdateFunc)(void ∗, double)
- typedef int(∗ pluginCloseFunc)()
- typedef void(∗ pluginDrawFunc)(void ∗obj)

Enumerations

- enum collisionType { COLLISION_GROUND = 0, COLLISION_FLY }
- enum networkMode { CLIENT, SERVER }
- enum align_t
- enum axis
- enum pluginType
- enum buttonId
- enum dataContainerType
- enum dcAxesType
- enum dcFlagContainerType
- enum dcFlagsType { DC_FLAG_STATIC = 0, DC_NUM_FLAGS }
- enum dcIdType
- enum dcCommandType {

  CMD_OBJECT_SET = 0, CMD_OBJECT_ADD, CMD_OBJECT_DROP, CMD_OBJECT_VAR_GET,

  CMD_OBJECT_VAR_SET, CMD_OBSERVERID_SET, CMD_SCENE_CLEAR, CMD_SCENE_LOAD,

  CMD_SCENE_RESET, CMD_RESOURCE_SET, CMD_RESOURCE_ADD, CMD_RESOURCE_DROP,

  CMD_DEVICE_TERM, CMD_DEVICE_STATUS, CMD_DEVICE_VAR_GET, CMD_DEVICE_VAR_SET ,

  CMD_UNKNOWN }
- enum dcMaskType { DC_MASK_ALL = 0, DC_MASK_INPUT = 1, DC_MASK_STATE = 2 }
- enum dcTransfer
- enum mimeType {

  MIME_NONE = 0, MIME_IMAGE_PNG, MIME_IMAGE_JPEG, MIME_IMAGE_BMP,

  MIME_IMAGE_SVG, MIME_TEXT_PLAIN, MIME_MODEL_VRML, MIME_MODEL_X3D,

  MIME_APPLICATION_3DS, MIME_AUDIO_WAV, MIME_GEOM_RECT, MIME_FONT_TXF }
- enum deviceId {

  DEV_NO_DEVICE = 0x1, DEV_UNKNOWN = 0x2, DEV_INPUT = 0x4, DEV_OUTPUT = 0x8,

  DEV_MOUSE = 0x10, DEV_KEYBOARD = 0x20, DEV_JOYSTICK = 0x40, DEV_TRACKER = 0x80,

  DEV_NETWORK = 0x100, DEV_SIMULATION = 0x200, DEV_SOUND = 0x400, DEV_WINDOW = 0x800,

  DEV_GRAPHICS = 0x1000, DEV_CONTAINER = 0x2000 , DEV_ALL = 0xFFFFFFFF }
- enum errorCodes {

  ERR_OK = 0, ERR_ERROR, ERR_WARNING, ERR_FATAL,

  ERR_DEVICE_BUSY, ERR_DEVICE_NOT_WORKING, ERR_OUT_OF_MEMORY, ERR_IO }
- enum varTypes {

  TYPE_UNKNOWN = 0, TYPE_INT32, TYPE_UINT32, TYPE_FLOAT32,

  TYPE_STRING, TYPE_BOOL }

Functions

- bool hasGlExtension (const std::string &which)
- template<class T> T sqr (T x)
- template<class T> int sgn (T x)
- template<class T> double dsin (T x)
- template<class T> double dcos (T x)
- template<class T> double dtan (T x)
- template<class T> double datan (T x)
- template<class T> double angle (T angle)
- template<class T> double dAngle (T ang1, T ang2)
- template<class T> void swap (T &t1, T &t2)
- template<class T> T min3 (T value0, T value1, T value2)
- template<class T> T max3 (T value0, T value1, T value2)
- float dist (float x1, float y1, float z1, float x2, float y2, float z2)
- float dist (float x1, float y1, float x2, float y2)
- float minAbs (float f1, float f2)
- float distPointSeg (float x1, float y1, float x2, float y2, float x3, float y3)
- float angleXY (const ve::vec3f &p0, const ve::vec3f &p1, const ve::vec3f &p2)
- float angleXY (float x1, float y1, float x2, float y2)
- float angleXY (float x1, float y1, float x2, float y2, float x3, float y3)
- float scalarProduct (float x1, float y1, float x2, float y2)
- float scalarProduct (float x1, float y1, float z1, float x2, float y2, float z2)
- std::ostream & operator<< (std::ostream &os, const ve::vec2f &v)
- std::ostream & operator<< (std::ostream &os, const ve::vec3f &v)
- std::ostream & operator<< (std::ostream &os, const ve::vec4f &v)
- std::ostream & operator<< (std::ostream &os, const ve::plane &pl)
- std::ostream & operator<< (std::ostream &os, const ve::triangle &t)
- std::ostream & operator<< (std::ostream &os, const ve::line &l)
- std::ostream & operator<< (std::ostream &os, const ve::vec6f &sdof)
- std::ostream & operator<< (std::ostream &os, const ve::frustum &fr)
- int veRandi (int max)
- float veRandf (double max)
- std::ostream & operator<< (std::ostream &os, const ve::mat4f &m)
- unsigned int split (const std::string &input, std::vector< std::string > &output, const std::string &separators=" \t\n\015")
- std::string trim (const std::string &s, const std::string &pattern=", \t\n\015")
- bool isWhiteSpace (char ch)
- std::string replaceAll (std::string s, const std::string &search, const std::string &repl)
- std::string replaceChars (const std::string &s, const std::string &pattern=" \t\n\015", char ch=' ')
- void operator-= (std::string &s, unsigned int n)
- std::string i2s (long i)
- std::string f2s (double f)
- std::string f2s (double f, unsigned short nDigits)
- std::string b2s (bool b, bool asText=false)
- std::string c2s (char ch)
- int s2i (const std::string &s)
- unsigned int s2ui (const std::string &s)
- float s2f (const std::string &s)

- bool s2b (const std::string &s)
- unsigned int s2f (const std::string &s, std::vector< float > &vFloat, const std::string &separators=", \t\n\015")
- unsigned int s2i (const std::string &s, std::vector< int > &vInt, const std::string &separators=", \t\n\015")
- unsigned int s2ui (const std::string &s, std::vector< unsigned int > &vUInt, const std::string &separators=", \t\n\015")
- unsigned int s2b (const std::string &s, std::vector< bool > &vBool, const std::string &separators=", \t\n\015")
- std::string toUpper (const std::string &s)
- std::string toLower (const std::string &s)
- unsigned int hex2ui (const std::string &s)
- std::string load (const std::string &filename)
- void save (const std::string &s, const std::string &filename)
- short net2hosts (const char *buffer)
- int host2nets (char *buffer, short number)
- void byteSwap (char *b, int n)

Variables

- const unsigned int totalSize = 256
- const unsigned int headerSize = 64
- const unsigned int NETWORK_MAX_CONNECTIONS = 32
- const unsigned int NETWORK_NUM_CONTAINERS = 5
- const unsigned int NETWORK_BUFSIZE = NETWORK_NUM_CONTAINERS * ve::totalSize
- const unsigned int NETWORK_BUF_CONTAINERS = 10
- const float PI = 3.14159265358979323846f
- const float PI_180 = PI/180.0f
- const float DEG2RAD = PI_180
- const float RAD2DEG = 180.0f/PI
- const double EPSILON = 0.00000001
- const unsigned int DC_NUM_BUTTONS = ve::BUTTON_ID_MAX + 1
- const unsigned int DC_NUM_SIXDOFS = 4
- const unsigned int DC_NUM_AXES = 6 * ve::DC_NUM_SIXDOFS
- const unsigned int ZIP_HEADER_ID = 67324752
- const unsigned int ZIP_DIR_ID = 33639248

11.2.1 Detailed Description

veLib standard namespace

This namespace is distributed across several *.h files. However, the main documentation is written here (in file veStd.h). We try to cover all general enums and defines with this namespace top avoid name clashes with other libraries, tools and compiler defaults.

11.2.2 Enumeration Type Documentation

11.2.2.1 enum ve::collisionType

this enum defines the different working modes of collision models

Enumerator:

COLLISION_GROUND ground mode, object is kept at constant distance to ground

COLLISION_FLY fly mode, collision model tests for positive distance to ground

Definition at line 27 of file veCollision.h.

11.2.2.2 enum ve::networkMode

Enumerator:

CLIENT working in client mode.

SERVER working in server mode.

Definition at line 65 of file veDeviceNetwork.h.

11.2.2.3 enum ve::align_t

symbolic names for the alignment of overlay objects and text

Definition at line 48 of file veGlUtils.h.

11.2.2.4 enum ve::axis

defines obvious names for vec2f, vec3f, vec4f, and vec6f ordinates.

Definition at line 39 of file veMath.h.

11.2.2.5 enum ve::buttonId

large enum for communicating input device states and standard events.

No id's beyond 127 are allowed.

Definition at line 51 of file veTypes.h.

11.2.2.6 enum ve::dataContainerType

this enum defines meaningful names for the kind of information a data container contains

Definition at line 266 of file veTypes.h.

11.2.2.7 enum ve::dcAxesType

this enum gives the 4 sixdofs of a data container meaningful names

Definition at line 278 of file veTypes.h.

11.2.2.8 enum ve::dcFlagContainerType

this enum gives the flag containers of the data container meaningful names

Definition at line 286 of file veTypes.h.

11.2.2.9 enum ve::dcFlagsType

this enum gives some predefined flags of the data container meaningful names

Enumerator:

DC_FLAG_STATIC flag for scene graph objects, object does not change at all during simulation and may therefore be optimized by the device

DC_NUM_FLAGS maximum number of predefined flags, all further flags can be safely used by the user

Definition at line 292 of file veTypes.h.

11.2.2.10 enum ve::dcIdType

this enum gives the ids of the data container meaningful names

Definition at line 301 of file veTypes.h.

11.2.2.11 enum ve::dcCommandType

this enum gives common commands (e.g., in the data container) meaningful names

Enumerator:

CMD_OBJECT_SET sets object data (position, velocity, etc.)

CMD_OBJECT_ADD adds an object to a scene/device

CMD_OBJECT_DROP removes an object from a scene/device

CMD_OBJECT_VAR_GET requests the value of a variable which is part of the object

CMD_OBJECT_VAR_SET sets the value of a variable which is part of the object

CMD_OBSERVERID_SET sets the observer id, the camera will be bound to this object

CMD_SCENE_CLEAR not yet implemented!

CMD_SCENE_LOAD not yet implemented!

CMD_SCENE_RESET not yet implemented!

CMD_RESOURCE_SET updates a resource definition

CMD_RESOURCE_ADD adds a resource definition

CMD_RESOURCE_DROP drops a resource definition

CMD_DEVICE_TERM terminates a device

CMD_DEVICE_STATUS requests the current status of a device

CMD_DEVICE_VAR_GET requests the value of a variable of the device

CMD_DEVICE_VAR_SET sets the value of a variable of the device

CMD_UNKNOWN an unknown/unrecognized command

Definition at line 317 of file veTypes.h.

11.2.2.12 enum ve::dcMaskType

these constants define a bitmask for restricting overwriting a dataContainer

Enumerator:

DC_MASK_ALL the dataContainer might be completely overwritten

DC_MASK_INPUT only input axes and flags are overwritten

DC_MASK_STATE all non-input data fields are overwritten

Definition at line 367 of file veTypes.h.

11.2.2.13 enum ve::dcTransfer

defines names for how the dataContainer should be transfered over the network(e.g. should it arrive for sure?)

Definition at line 377 of file veTypes.h.

11.2.2.14 enum ve::mimeType

constants for the mime types that identify the resources in the xml ini files

Enumerator:

MIME_NONE resource is not identified

MIME_IMAGE_PNG resource is a png image (image/png)

MIME_IMAGE_JPEG resource is a jpeg image (image/jpeg)

MIME_IMAGE_BMP resource is a (windows) bitmap (image/x-bmp)

MIME_IMAGE_SVG resource is a svg image (image/svg+xml)

MIME_TEXT_PLAIN resource is plain ascii text (text/plain)

MIME_MODEL_VRML resource is a vrml model (model/vrml)

MIME_MODEL_X3D resource is a X3D model (model/x3d)

MIME_APPLICATION_3DS resource is a 3D Studio mesh (application/x-3ds)

MIME_AUDIO_WAV resource is a wave audio file (audio/wav)

MIME_GEOM_RECT resource is a rectangle geometry object (geometry/rectangle)
this is not an official mime type, but introduced in order to allow dynamic resource management of overlay rectangles.

MIME_FONT_TXF resource is a txf texture font (font/txf)

Definition at line 384 of file veTypes.h.

11.2.2.15 enum ve::deviceId

device identifiers This enum defines the different veDeviceIds. Those ids are used to identify the class of a device. It is usually combined with a identifying tag as a veDeviceId (see veData.h).

Enumerator:

DEV_NO_DEVICE no device at all

DEV_UNKNOWN no specific device, but at least device

DEV_INPUT some unspecific input device
DEV_OUTPUT some unspecific output device
DEV_MOUSE some unspecific mouse device
DEV_KEYBOARD some unspecific keyboard device
DEV_JOYSTICK some unspecific joystick device
DEV_TRACKER some unspecific tracker device
DEV_NETWORK some unspecific network device
DEV_SIMULATION some unspecific simulation device
DEV_SOUND some unspecific sound device
DEV_WINDOW some unspecific combined window device integrating mouse and keyboard

DEV_GRAPHICS some unspecific graphic device
DEV_CONTAINER some virtual container device
DEV_ALL combines all device ids, e.g., for dataContainer filter masks

Definition at line 421 of file veTypes.h.

11.2.2.16 enum ve::errorCodes

Definition of error codes.

The error codes are globally defined to provide a framework of unique and useful error numbers reporting what was right and wrong in the functions. By the way, many functions return the number of errors which occurred during the execution.

Enumerator:
ERR_OK everything ok.
ERR_ERROR some unspecific error
ERR_WARNING we warn, but everything may work.
ERR_FATAL something unspecific, but fatal
ERR_DEVICE_BUSY current operation failed probably due to overload, but device may work correctly
ERR_DEVICE_NOT_WORKING device does not work at all, and probably will not work again without proper reinitialization
ERR_OUT_OF_MEMORY no dynamic memory available
ERR_IO input/output (e.g., file operation) failed

Definition at line 470 of file veTypes.h.

11.2.2.17 enum ve::varTypes

Definition of variable types.

These identifiers are for example useful for determining the type of an exposed variable.

Enumerator:
TYPE_UNKNOWN unkown type
TYPE_INT32 signed 32 bit integer variable
TYPE_UINT32 unsigned 32 bit integer variable

TYPE_FLOAT32 32 bit float variable
TYPE_STRING string variable
TYPE_BOOL boolean variable

Definition at line 494 of file veTypes.h.

11.2.3 Function Documentation

11.2.3.1 bool ve::hasGlExtension (const std::string & *which*)

checks whether a certain gl extension is available

11.2.3.2 template<class T> T ve::sqr (T *x*)

generic sqrare function template
Definition at line 73 of file veMath.h.

11.2.3.3 template<class T> int ve::sgn (T *x*)

generic signum function template
Definition at line 75 of file veMath.h.

11.2.3.4 template<class T> double ve::dsin (T *x*)

sinus function for degrees
Definition at line 77 of file veMath.h.
References PI_180.
Referenced by ve::vec3f::setPolar().

11.2.3.5 template<class T> double ve::dcos (T *x*)

cosinus function for degrees
Definition at line 79 of file veMath.h.
References PI_180.
Referenced by ve::vec3f::setPolar().

11.2.3.6 template<class T> double ve::dtan (T *x*)

tangens function for degrees
Definition at line 81 of file veMath.h.
References PI_180.

11.2.3.7 template<class T> double ve::datan (T *x*)

arcustangens function for degrees
Definition at line 83 of file veMath.h.
References PI_180.

11.2.3.8 template<class T> double ve::angle (T *angle*)

an "abs" and "mod" function for degree angles: restricts float values to 0 <= angle < 360.0
Definition at line 85 of file veMath.h.
Referenced by dAngle().

11.2.3.9 template<class T> double ve::dAngle (T *ang1*, T *ang2*)

returns the positive angular difference between ang2 and ang1 in degrees
Definition at line 87 of file veMath.h.
References angle().

11.2.3.10 template<class T> void ve::swap (T & *t1*, T & *t2*)

swaps two values
Definition at line 89 of file veMath.h.

11.2.3.11 template<class T> T ve::min3 (T *value0*, T *value1*, T *value2*)

returns minimum of 3 values
Definition at line 111 of file veMath.h.
References min.

11.2.3.12 template<class T> T ve::max3 (T *value0*, T *value1*, T *value2*)

returns maximum of 3 values
Definition at line 113 of file veMath.h.
References max.

11.2.3.13 float ve::dist (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*) [inline]

returns distance between P1(x1|y1|z1) and P2(x2|y2|z2)
Definition at line 118 of file veMath.h.

11.2.3.14 float ve::dist (float *x1*, float *y1*, float *x2*, float *y2*) [inline]

returns distance between P1(x1|y1) and P2(x2|y2)

Definition at line 121 of file veMath.h.

11.2.3.15 float ve::minAbs (float *f1*, float *f2*) [inline]

returns minimum absolute value

Definition at line 124 of file veMath.h.

11.2.3.16 float ve::distPointSeg (float *x1*, float *y1*, float *x2*, float *y2*, float *x3*, float *y3*)

returns minimum distance between P(x1|y1) and line segment((x2|y2)|(x3|y3))

11.2.3.17 float ve::angleXY (const ve::vec3f & *p0*, const ve::vec3f & *p1*, const ve::vec3f & *p2*)

returns angle between line p0|p1 and line p0|p2

Referenced by ve::vec3f::angleToXY().

11.2.3.18 float ve::angleXY (float *x1*, float *y1*, float *x2*, float *y2*)

returns angle between point xy1 and point xy2

11.2.3.19 float ve::angleXY (float *x1*, float *y1*, float *x2*, float *y2*, float *x3*, float *y3*)

returns angle between line xy1|xy2 and line xy1|xy3

11.2.3.20 float ve::scalarProduct (float *x1*, float *y1*, float *x2*, float *y2*) [inline]

returns scalar product between vectors x1|y1 and x2|y2

Definition at line 136 of file veMath.h.

11.2.3.21 float ve::scalarProduct (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*) [inline]

returns scalar product between vectors x1|y1|z1 and x2|y2|z2

Definition at line 139 of file veMath.h.

11.2.3.22 std::ostream& ve::operator<< (std::ostream & *os*, const ve::vec2f & *v*)

operator for output of vec2f objects in streams

11.2.3.23 std::ostream& ve::operator<< (std::ostream & *os*, const ve::vec3f & *v*)

operator for output in streams

11.2.3.24 std::ostream& ve::operator<< (std::ostream & *os*, const ve::vec4f & *v*)

operator for output in streams

11.2.3.25 std::ostream& ve::operator<< (std::ostream & *os*, const ve::plane & *pl*)

operator for output in streams

11.2.3.26 std::ostream& ve::operator<< (std::ostream & *os*, const ve::triangle & *t*)

operator for output in streams

11.2.3.27 std::ostream& ve::operator<< (std::ostream & *os*, const ve::line & *l*)

operator for output in streams

11.2.3.28 std::ostream& ve::operator<< (std::ostream & *os*, const ve::vec6f & *sdof*)

operator for output in streams

11.2.3.29 std::ostream& ve::operator<< (std::ostream & *os*, const ve::frustum & *fr*)

operator for output in streams

11.2.3.30 int ve::veRandi (int *max*)

provide a random int number between 0 and max-1

11.2.3.31 float ve::veRandf (double *max*)

provide a random float number between 0 and max

11.2.3.32 std::ostream& ve::operator<< (std::ostream & *os*, const ve::mat4f & *m*) [inline]

operator for output in streams

Definition at line 1080 of file veMath.h.

References ve::mat4f::str().

11.2.3.33 unsigned int ve::split (const std::string & *input*, std::vector< std::string > & *output*, const std::string & *separators* = " \t\n\015")

general tool for splitting strings into pieces.

11.2.3.34 std::string ve::trim (const std::string & *s*, const std::string & *pattern* = ", \t\n\015")

general tool for stripping definable characters from both ends

11.2.3.35 bool ve::isWhiteSpace (char *ch*)

returns true if char is whitespace otherwise false

11.2.3.36 std::string ve::replaceAll (std::string *s*, const std::string & *search*, const std::string & *repl*)

replaces all occurences of search with repl in s

11.2.3.37 std::string ve::replaceChars (const std::string & *s*, const std::string & *pattern* = " \t\n\015", char *ch* = ' ')

replaces occurences of pattern in string s by a single ch

11.2.3.38 void ve::operator-= (std::string & *s*, unsigned int *n*)

trims n characters from the end of string s

11.2.3.39 std::string ve::i2s (long *i*)

converts integer value to string

Referenced by ve::xmlIni::setAttribute(), ve::xml::setAttribute(), and ve::flag128::str().

11.2.3.40 std::string ve::f2s (double *f*)

converts float value to string

Referenced by ve::xmlIni::setAttribute(), and ve::xml::setAttribute().

11.2.3.41 std::string ve::f2s (double *f*, unsigned short *nDigits*)

converts float value to string, specify number of digits

11.2.3.42 std::string ve::b2s (bool *b*, bool *asText* = `false`)

converts bool value b to string, optionally specify mode (0 1, true false)

Referenced by ve::xmlIni::setAttribute(), and ve::xml::setAttribute().

11.2.3.43 std::string ve::c2s (char *ch*)

converts character to string

11.2.3.44 int ve::s2i (const std::string & *s*)

converts string to integer value

11.2.3.45 unsigned int ve::s2ui (const std::string & *s*)

converts string to unsigned integer value

Referenced by ve::flag128::set().

11.2.3.46 float ve::s2f (const std::string & *s*)

converts string to float value

11.2.3.47 bool ve::s2b (const std::string & *s*)

converts string to bool value

11.2.3.48 unsigned int ve::s2f (const std::string & *s*, std::vector< float > & *vFloat*, const std::string & *separators* = ", \t\n\015")

converts string to float vector.

The string is splitted according to optional argument separators.

Returns:
the number of generated floats.

11.2.3.49 unsigned int ve::s2i (const std::string & *s*, std::vector< int > & *vInt*, const std::string & *separators* = ", \t\n\015")

converts string to int vector.

The string is splitted according to optional argument separators.

Returns:
the number of generated ints.

11.2.3.50 unsigned int ve::s2ui (const std::string & *s*, std::vector< unsigned int > & *vUInt*, const std::string & *separators* = ", \t\n\015")

converts string to unsigned int vector.

The string is splitted according to optional argument separators.

Returns:
the number of generated unsigned ints.

11.2.3.51 unsigned int ve::s2b (const std::string & *s*, std::vector< bool > & *vBool*, const std::string & *separators* = ", \t\n\015")

converts string to bool vector.

The string is splitted according to optional argument separators.

Returns:
the number of generated bools.

11.2.3.52 std::string ve::toUpper (const std::string & *s*)

converts a string to upper case, if possible.

11.2.3.53 std::string ve::toLower (const std::string & *s*)

converts a string to lower case, if possible.

11.2.3.54 unsigned int ve::hex2ui (const std::string & *s*)

converts a hexadecimal string into an unsigned int

11.2.3.55 std::string ve::load (const std::string & *filename*)

loads a string from a file

11.2.3.56 void ve::save (const std::string & *s*, const std::string & *filename*)

saves a string to a file

11.2.3.57 short ve::net2hosts (const char * *buffer*)

These functions take in data with swapped byte order and swap it back. WORKS ONLY IF data size between client and server match

11.2.3.58 int ve::host2nets (char * *buffer*, short *number*)

These function take in data, change the byte order and store the result to the memory area pointed to by the buffer pointer. It returns the size of the data written out to the buffer, in bytes. WORKS ONLY IF data size between client and server match

11.2.3.59 void ve::byteSwap (char * *b*, int *n*)

changes the byte order of a given value.

Parameters:

- ***b*** pointer to the value
- ***n*** size of the value in bytes

11.2.4 Variable Documentation

11.2.4.1 const unsigned int ve::totalSize = 256

The physical size of all data containers.

Definition at line 30 of file veDataContainer.h.

Referenced by ve::dataContainer::nData(), ve::dataChar::nData(), and ve::dataChar::size().

11.2.4.2 const unsigned int ve::headerSize = 64

sets dataContainer header size (size of this class) in bytes

it's just a sum of the sizeof's of the shared variables defined above. Be aware of 32/64 bit issues!

Definition at line 35 of file veDataContainer.h.

Referenced by ve::dataContainer::nData(), and ve::dataChar::nData().

11.2.4.3 const unsigned int ve::NETWORK_MAX_CONNECTIONS = 32

number of maximal parallel client/servers on one network device

Definition at line 53 of file veDeviceNetwork.h.

11.2.4.4 const unsigned int ve::NETWORK_NUM_CONTAINERS = 5

defines the max number of data containers that can be send in one TCP/IP-package

Definition at line 58 of file veDeviceNetwork.h.

11.2.4.5 const unsigned int ve::NETWORK_BUFSIZE = NETWORK_NUM_CONTAINERS * ve::totalSize

definition of the buffer size, which is used for socket communication

Definition at line 60 of file veDeviceNetwork.h.

11.2.4.6 const unsigned int ve::NETWORK_BUF_CONTAINERS = 10

the max number of data containers that can be buffered before overflow occurs

Definition at line 62 of file veDeviceNetwork.h.

11.2.4.7 const float ve::PI = 3.14159265358979323846f

defines PI

Definition at line 30 of file veMath.h.

11.2.4.8 const float ve::PI_180 = PI/180.0f

defines PI/180, for angle conversions from deg to rad.

Definition at line 33 of file veMath.h.

Referenced by datan(), dcos(), dsin(), and dtan().

11.2.4.9 const float ve::DEG2RAD = PI_180

defines PI/180, for angle conversions from deg to rad.

Definition at line 35 of file veMath.h.

11.2.4.10 const float ve::RAD2DEG = 180.0f/PI

defines 180/PI, for angle conversions from rad to deg.

Definition at line 37 of file veMath.h.

11.2.4.11 const double ve::EPSILON = 0.00000001

just a small value

Definition at line 67 of file veMath.h.

11.2.4.12 const unsigned int ve::BIT[ ]

Initial value:

```
{
    0x1,0x2,0x4,0x8, 0x10,0x20,0x40,0x80, 0x100,0x200,0x400,0x800,
    0x1000,0x2000,0x4000,0x8000, 0x10000,0x20000,0x40000,0x80000,
    0x100000,0x200000,0x400000,0x800000, 0x1000000,0x2000000,
    0x4000000,0x8000000, 0x10000000,0x20000000,0x40000000,0x80000000 }
```

this array defines names for all 32 bits of an unsigned int

Definition at line 37 of file veTypes.h.

Referenced by ve::flag128::off(), ve::flag128::on(), and ve::flag128::operator[ ]().

11.2.4.13 const unsigned int ve::BYTE[ ] = { 0xFF, 0xFF00, 0xFF0000, 0xFF000000 }

this array defines names for all 4 bytes of an unsigned int
Definition at line 44 of file veTypes.h.

11.2.4.14 const unsigned int ve::DC_NUM_BUTTONS = ve::BUTTON_ID_MAX + 1

the number of buttons in the veData class.
Definition at line 261 of file veTypes.h.

11.2.4.15 const unsigned int ve::DC_NUM_SIXDOFS = 4

number of sixdofs in a data container (currently 3 : position, velocity, acceleration)
Definition at line 272 of file veTypes.h.
Referenced by ve::dataContainer::nData().

11.2.4.16 const unsigned int ve::DC_NUM_AXES = 6 * ve::DC_NUM_SIXDOFS

The number of axes in the veData class.
Definition at line 275 of file veTypes.h.

11.2.4.17 const unsigned int ve::ZIP_HEADER_ID = 67324752

constant identifying normal zip file members
Definition at line 129 of file veUtils.h.

11.2.4.18 const unsigned int ve::ZIP_DIR_ID = 33639248

constant identifying central zip directory
Definition at line 131 of file veUtils.h.

Chapter 12

veLib Class Documentation

12.1 ve::_3dsObject Class Reference

Public Member Functions

- _3dsObject ()

Public Attributes

- char objName [255]
- std::vector< ve::vec3f > vCoords
- std::vector< ve::vec2f > vTexCoords
- std::vector< unsigned int > vIndices
- std::vector< unsigned int > vFaceEnds
- std::vector< _materialRef > vMaterial

12.1.1 Detailed Description

Definition at line 62 of file veIo3ds.h.

12.1.2 Constructor & Destructor Documentation

12.1.2.1 ve::_3dsObject::_3dsObject () [inline]

default constructor

Definition at line 65 of file veIo3ds.h.

12.1.3 Member Data Documentation

12.1.3.1 char ve::_3dsObject::objName[255]

The name of the object.

Definition at line 65 of file veIo3ds.h.

12.1.3.2 std::vector<ve::vec3f> ve::_3dsObject::vCoords

The object's vertices.

Definition at line 70 of file velo3ds.h.

12.1.3.3 std::vector<ve::vec2f> ve::_3dsObject::vTexCoords

The texture's UV coordinates.

Definition at line 72 of file velo3ds.h.

12.1.3.4 std::vector<unsigned int> ve::_3dsObject::vIndices

stores indices

Definition at line 75 of file velo3ds.h.

12.1.3.5 std::vector<unsigned int> ve::_3dsObject::vFaceEnds

stores indices of face ends

Definition at line 77 of file velo3ds.h.

12.1.3.6 std::vector<_materialRef> ve::_3dsObject::vMaterial

stores assigned materials

Definition at line 79 of file velo3ds.h.

The documentation for this class was generated from the following file:

- velo3ds.h

12.2 ve::_collisionSurfaceObj Class Reference

Public Member Functions

- _collisionSurfaceObj (const ve::vec6f &pos, float offsetZ=0.0f, float deltaZMax=1.0f, unsigned int mode=ve::COLLISION_GROUND, const ve::triangle *pCurrSurface=0)
- const ve::vec6f & prevPos () const
- ve::vec6f & prevPos ()
- float offsetZ () const
- float deltaZMax () const
- unsigned int mode () const
- const ve::triangle * currSurface () const
- const ve::triangle *& currSurface ()

Protected Attributes

- ve::vec6f m_prevPos
- float m_offsetZ
- float m_deltaZMax
- unsigned int m_mode
- const ve::triangle * m_pCurrSurface

12.2.1 Detailed Description

Definition at line 96 of file veCollision.h.

12.2.2 Constructor & Destructor Documentation

12.2.2.1 ve::_collisionSurfaceObj::_collisionSurfaceObj (const ve::vec6f & *pos*, float *offsetZ* = `0.0f`, float *deltaZMax* = `1.0f`, unsigned int *mode* = `ve::COLLISION_GROUND`, const ve::triangle * *pCurrSurface* = `0`)

constructor

Parameters:

pos current (allowed) position

offsetZ (optional) the fixed height over ground when in ground mode

deltaZMax (optional) the maximum climb height when in ground mode

mode (optional) mode one of the collisionMode ids, defining whether collision model works in ground or fly mode

pCurrSurface (optional) the surface triangle the object is currently on.

12.2.3 Member Function Documentation

12.2.3.1 const ve::vec6f& ve::_collisionSurfaceObj::prevPos () const `[inline]`

returns previous position

Definition at line 110 of file veCollision.h.

References m_prevPos.

12.2.3.2 ve::vec6f& ve::_collisionSurfaceObj::prevPos () [inline]

allows access to previous position

Definition at line 112 of file veCollision.h.

References m_prevPos.

12.2.3.3 float ve::_collisionSurfaceObj::offsetZ () const [inline]

returns fixed object offsetZ

Definition at line 114 of file veCollision.h.

References m_offsetZ.

12.2.3.4 float ve::_collisionSurfaceObj::deltaZMax () const [inline]

returns maximum climb height

Definition at line 116 of file veCollision.h.

References m_deltaZMax.

12.2.3.5 unsigned int ve::_collisionSurfaceObj::mode () const [inline]

returns collision mode

Definition at line 118 of file veCollision.h.

References m_mode.

12.2.3.6 const ve::triangle* ve::_collisionSurfaceObj::currSurface () const [inline]

returns pointer to current surface

Definition at line 120 of file veCollision.h.

References m_pCurrSurface.

12.2.3.7 const ve::triangle*& ve::_collisionSurfaceObj::currSurface () [inline]

allows access to current surface pointer

Definition at line 122 of file veCollision.h.

12.2.4 Member Data Documentation

12.2.4.1 ve::vec6f ve::_collisionSurfaceObj::m_prevPos [protected]

stores the last allowed position

Definition at line 122 of file veCollision.h.

Referenced by prevPos().

12.2.4.2 float ve::_collisionSurfaceObj::m_offsetZ [protected]

stores fixed object height

Definition at line 127 of file veCollision.h.

Referenced by offsetZ().

12.2.4.3 float ve::_collisionSurfaceObj::m_deltaZMax [protected]

stores maximum climb height

Definition at line 129 of file veCollision.h.

Referenced by deltaZMax().

12.2.4.4 unsigned int ve::_collisionSurfaceObj::m_mode [protected]

stores collision mode

Definition at line 131 of file veCollision.h.

Referenced by mode().

12.2.4.5 const ve::triangle* ve::_collisionSurfaceObj::m_pCurrSurface [protected]

stores pointer to current surface triangle

Definition at line 133 of file veCollision.h.

Referenced by currSurface().

The documentation for this class was generated from the following file:

- veCollision.h

12.3 ve::_exposedVar Class Reference

Public Member Functions

- _exposedVar (void *pVar, unsigned int type, bool readOnly)

Public Attributes

- void * m_pVar
- unsigned int m_type
- bool m_readOnly

12.3.1 Detailed Description

Definition at line 29 of file veDevice.h.

12.3.2 Constructor & Destructor Documentation

12.3.2.1 ve::_exposedVar::_exposedVar (void * *pVar*, unsigned int *type*, bool *readOnly*) [inline]

constructor

Definition at line 32 of file veDevice.h.

12.3.3 Member Data Documentation

12.3.3.1 void* ve::_exposedVar::m_pVar

pointer to the variable

Definition at line 33 of file veDevice.h.

12.3.3.2 unsigned int ve::_exposedVar::m_type

type of the variable, a varType constant defined in veTypes.h

Definition at line 37 of file veDevice.h.

12.3.3.3 bool ve::_exposedVar::m_readOnly

stores whether variable is read only

Definition at line 39 of file veDevice.h.

The documentation for this class was generated from the following file:

- veDevice.h

12.4 ve::_materialInfo Class Reference

Public Member Functions

- _materialInfo ()

Public Attributes

- char matName [255]
- char fileName [255]
- unsigned char color [3]
- float opacity

12.4.1 Detailed Description

Definition at line 36 of file veIo3ds.h.

12.4.2 Constructor & Destructor Documentation

12.4.2.1 ve::_materialInfo::_materialInfo ()

default constructor

12.4.3 Member Data Documentation

12.4.3.1 char ve::_materialInfo::matName[255]

the texture name

Definition at line 41 of file veIo3ds.h.

12.4.3.2 char ve::_materialInfo::fileName[255]

the texture file name (If this is set it's a texture map)

Definition at line 43 of file veIo3ds.h.

12.4.3.3 unsigned char ve::_materialInfo::color[3]

the color of the object (R, G, B)

Definition at line 45 of file veIo3ds.h.

12.4.3.4 float ve::_materialInfo::opacity

the opacity of the object (A)

Definition at line 47 of file veIo3ds.h.

The documentation for this class was generated from the following file:

- veIo3ds.h

12.5 ve::_materialRef Class Reference

Public Member Functions

- _materialRef ()

Public Attributes

- char matName [255]
- std::vector< unsigned short > vFace

12.5.1 Detailed Description

Definition at line 51 of file veIo3ds.h.

12.5.2 Constructor & Destructor Documentation

12.5.2.1 ve::_materialRef::_materialRef () [inline]

default constructor

Definition at line 54 of file veIo3ds.h.

12.5.3 Member Data Documentation

12.5.3.1 char ve::_materialRef::matName[255]

The material name of the object.

Definition at line 54 of file veIo3ds.h.

12.5.3.2 std::vector<unsigned short> ve::_materialRef::vFace

stores face indices

Definition at line 58 of file veIo3ds.h.

The documentation for this class was generated from the following file:

- veIo3ds.h

12.6 ve::_modelRef Class Reference

Public Member Functions

- _modelRef (ve::geoObj *obj, const ve::vec6f &newpos, float sqrDist)
- const vec6f & pos () const
- geoObj & model ()
- bool operator< (const _modelRef &r) const

Protected Attributes

- ve::geoObj * pObj
- const ve::vec6f * pPos
- float distSqr

12.6.1 Detailed Description

Definition at line 90 of file veDeviceGraphicsGL.h.

12.6.2 Constructor & Destructor Documentation

12.6.2.1 ve::_modelRef::_modelRef (ve::geoObj * *obj*, const ve::vec6f & *newpos*, float *sqrDist*) [inline]

constructor

Definition at line 93 of file veDeviceGraphicsGL.h.

References distSqr, pObj, and pPos.

12.6.3 Member Function Documentation

12.6.3.1 const vec6f& ve::_modelRef::pos () const [inline]

returns model pos reference

Definition at line 96 of file veDeviceGraphicsGL.h.

References pPos.

12.6.3.2 geoObj& ve::_modelRef::model () [inline]

returns model geometry reference

Definition at line 98 of file veDeviceGraphicsGL.h.

References pObj.

12.6.3.3 bool ve::_modelRef::operator< (const _modelRef & *r*) const [inline]

auxiliary method for sorting by camera distance

Definition at line 100 of file veDeviceGraphicsGL.h.

12.6.4 Member Data Documentation

12.6.4.1 ve::geoObj* ve::_modelRef::pObj [protected]

pointer to model geometry

Definition at line 101 of file veDeviceGraphicsGL.h.

Referenced by _modelRef(), and model().

12.6.4.2 const ve::vec6f* ve::_modelRef::pPos [protected]

pointer to model pos

Definition at line 106 of file veDeviceGraphicsGL.h.

Referenced by _modelRef(), and pos().

12.6.4.3 float ve::_modelRef::distSqr [protected]

squared distance to camera

Definition at line 108 of file veDeviceGraphicsGL.h.

Referenced by _modelRef().

The documentation for this class was generated from the following file:

- veDeviceGraphicsGL.h

12.7 ve::chrono Class Reference

time / timer class

`#include <veUtils.h>`

Public Member Functions

- chrono (double initTime=0.0)
- void set (double time)
- double update ()
- double now ()
- double deltaT ()

Static Public Member Functions

- static double stamp ()
- static void sleep (double sec)
- static std::string date ()

Protected Attributes

- double m_currentTime
- double m_lastUpdate

12.7.1 Detailed Description

time / timer class

Definition at line 78 of file veUtils.h.

12.7.2 Constructor & Destructor Documentation

12.7.2.1 ve::chrono::chrono (double *initTime* = `0.0`) `[inline]`

constructor

Parameters:
initTime (optional) sets the start time for the timer.

Definition at line 83 of file veUtils.h.

References m_currentTime, and m_lastUpdate.

12.7.3 Member Function Documentation

12.7.3.1 void ve::chrono::set (double *time*) `[inline]`

sets the timer time to the given value

Definition at line 85 of file veUtils.h.

References m_currentTime.

12.7.3.2 double ve::chrono::update () [inline]

updates the timer time according to the system clock and returns the updated time

Definition at line 87 of file veUtils.h.

References m_currentTime, m_lastUpdate, and stamp().

12.7.3.3 double ve::chrono::now () [inline]

returns the current timer time (will always be the same between two update calls)

Definition at line 89 of file veUtils.h.

References m_currentTime.

12.7.3.4 double ve::chrono::deltaT () [inline]

returns the time that passed between the last two consecutive update calls

Definition at line 91 of file veUtils.h.

References m_currentTime, and m_lastUpdate.

12.7.3.5 static double ve::chrono::stamp () [static]

returns a time stamp with double precision

Referenced by update().

12.7.3.6 static void ve::chrono::sleep (double *sec*) [static]

portable sleep function

Lets the currently running process sleep for the given amount of time in seconds. For example a ve::sleep(0.010) would let the process sleep for about 10ms and allows a general update rate of 100Hz. Be aware that due to operating system restrictions the finest sleep granularity is often only 0.01 seconds.

Parameters:

sec the timespan the process should sleep

12.7.3.7 static std::string ve::chrono::date () [static]

returns the current date as string

the date is returned at a precision of seconds. This is mainly useful for naming temporary files.

12.7.4 Member Data Documentation

12.7.4.1 double ve::chrono::m_currentTime [protected]

stores current time, time of latest update call

Definition at line 112 of file veUtils.h.

Referenced by chrono(), deltaT(), now(), set(), and update().

12.7.4.2 double ve::chrono::m_lastUpdate [protected]

stores time stamp of previous update

Definition at line 114 of file veUtils.h.

Referenced by chrono(), deltaT(), and update().

The documentation for this class was generated from the following file:

- veUtils.h

12.8 ve::cmdLine Class Reference

a simple static class for preparsing command line arguments and options

`#include <veUtils.h>`

Static Public Member Functions

- static void interpret (int argc, char **argv, bool optsAsArgs=false)
- static bool parsed ()
- static char opt (unsigned int i)
- static bool opt (const char c)
- static std::string optArg (const char c)
- static const std::string & arg (unsigned int i)
- static unsigned int nArg ()
- static unsigned int nOpt ()
- static std::string dir ()
- static std::string cmd ()
- static void name (const std::string &s)
- static const std::string & name ()
- static void author (const std::string &s)
- static const std::string & author ()
- static void version (const std::string &s)
- static const std::string & version ()
- static void date (const std::string &s)
- static const std::string & date ()
- static void shortDescr (const std::string &s)
- static const std::string & shortDescr ()
- static void descr (const std::string &s)
- static const std::string & descr ()
- static void usage (const std::string &s)
- static const std::string & usage ()
- static const std::string help ()

Static Protected Attributes

- static std::vector< std::string > vArg
- static std::vector< std::string > vOpt
- static std::string ownDir
- static std::string ownCmd
- static bool isParsed
- static std::string name_
- static std::string author_
- static std::string version_
- static std::string date_
- static std::string shortDescr_
- static std::string descr_
- static std::string usage_

12.8.1 Detailed Description

a simple static class for preparsing command line arguments and options

Definition at line 213 of file veUtils.h.

12.8.2 Member Function Documentation

12.8.2.1 static void ve::cmdLine::interpret (int *argc*, char ** *argv*, bool *optsAsArgs* = false) [static]

parses command line and command line options.

Parameters:

argc standard C main() number of arguments.

argv standard C main() pointer to arguments char array.

optsAsArgs optional parameter that causes all command line tokens to be interpreted as arguments.

12.8.2.2 static bool ve::cmdLine::parsed () [inline, static]

returns whether command line has already been parsed

Definition at line 222 of file veUtils.h.

References isParsed.

12.8.2.3 static char ve::cmdLine::opt (unsigned int *i*) [inline, static]

returns command line option number i

Definition at line 224 of file veUtils.h.

References vOpt.

12.8.2.4 static bool ve::cmdLine::opt (const char *c*) [static]

returns true if a single char command line option exists

12.8.2.5 static std::string ve::cmdLine::optArg (const char *c*) [static]

returns optional argument of a single char command line option, or ""

12.8.2.6 static const std::string& ve::cmdLine::arg (unsigned int *i*) [inline, static]

returns a reference of command line argument i

Definition at line 230 of file veUtils.h.

References vArg.

12.8.2.7 static unsigned int ve::cmdLine::nArg () [inline, static]

returns number of command line arguments

Definition at line 232 of file veUtils.h.

References vArg.

12.8.2.8 static unsigned int ve::cmdLine::nOpt () [inline, static]

returns number of command line options
Definition at line 234 of file veUtils.h.
References vOpt.

12.8.2.9 static std::string ve::cmdLine::dir () [inline, static]

returns path to main's directory
Definition at line 236 of file veUtils.h.
References ownDir.

12.8.2.10 static std::string ve::cmdLine::cmd () [inline, static]

returns main's command name
Definition at line 238 of file veUtils.h.
References ownCmd.

12.8.2.11 static void ve::cmdLine::name (const std::string & *s*) [inline, static]

sets program's name.
Definition at line 240 of file veUtils.h.
References name_.

12.8.2.12 static const std::string& ve::cmdLine::name () [inline, static]

returns program's name.
Definition at line 242 of file veUtils.h.
References name_.

12.8.2.13 static void ve::cmdLine::author (const std::string & *s*) [inline, static]

sets program's author.
Definition at line 244 of file veUtils.h.
References author_.

12.8.2.14 static const std::string& ve::cmdLine::author () [inline, static]

returns program's author.
Definition at line 246 of file veUtils.h.
References author_.

12.8.2.15 static void ve::cmdLine::version (const std::string & *s*) [inline, static]

sets program's version.

Definition at line 248 of file veUtils.h.

References version_.

12.8.2.16 static const std::string& ve::cmdLine::version () [inline, static]

returns program's version.

Definition at line 250 of file veUtils.h.

References version_.

12.8.2.17 static void ve::cmdLine::date (const std::string & *s*) [inline, static]

sets program's date.

Definition at line 252 of file veUtils.h.

References date_.

12.8.2.18 static const std::string& ve::cmdLine::date () [inline, static]

returns program's date.

Definition at line 254 of file veUtils.h.

References date_.

12.8.2.19 static void ve::cmdLine::shortDescr (const std::string & *s*) [inline, static]

sets program's short description.

Definition at line 256 of file veUtils.h.

References shortDescr_.

12.8.2.20 static const std::string& ve::cmdLine::shortDescr () [inline, static]

returns program's short description.

Definition at line 258 of file veUtils.h.

References shortDescr_.

12.8.2.21 static void ve::cmdLine::descr (const std::string & *s*) [inline, static]

sets program's long description.

Definition at line 260 of file veUtils.h.

References descr_.

12.8.2.22 static const std::string& ve::cmdLine::descr () [inline, static]

returns program's long description.

Definition at line 262 of file veUtils.h.

References descr_.

12.8.2.23 static void ve::cmdLine::usage (const std::string & s) [inline, static]

sets program's long description.

Definition at line 264 of file veUtils.h.

References usage_.

12.8.2.24 static const std::string& ve::cmdLine::usage () [inline, static]

returns program's long description.

Definition at line 266 of file veUtils.h.

References usage_.

12.8.2.25 static const std::string ve::cmdLine::help () [static]

returns a help string containing previously setted data.

12.8.3 Member Data Documentation

12.8.3.1 std::vector<std::string> ve::cmdLine::vArg [static, protected]

stores command line arguments

Definition at line 272 of file veUtils.h.

Referenced by arg(), and nArg().

12.8.3.2 std::vector<std::string> ve::cmdLine::vOpt [static, protected]

stores command line options

Definition at line 274 of file veUtils.h.

Referenced by nOpt(), and opt().

12.8.3.3 std::string ve::cmdLine::ownDir [static, protected]

stores path to main's directory

Definition at line 276 of file veUtils.h.

Referenced by dir().

12.8.3.4 std::string ve::cmdLine::ownCmd [static, protected]

stores main's command name

Definition at line 278 of file veUtils.h.

Referenced by cmd().

12.8.3.5 bool ve::cmdLine::isParsed [static, protected]

stores whether class is already initialized

Definition at line 280 of file veUtils.h.

Referenced by parsed().

12.8.3.6 std::string ve::cmdLine::name_ [static, protected]

stores program's name.

If no name is provided by the user, the program's command line call is taken as default.

Definition at line 285 of file veUtils.h.

Referenced by name().

12.8.3.7 std::string ve::cmdLine::author_ [static, protected]

stores program's author.

Definition at line 287 of file veUtils.h.

Referenced by author().

12.8.3.8 std::string ve::cmdLine::version_ [static, protected]

stores program's version.

Definition at line 289 of file veUtils.h.

Referenced by version().

12.8.3.9 std::string ve::cmdLine::date_ [static, protected]

stores program's date.

Definition at line 291 of file veUtils.h.

Referenced by date().

12.8.3.10 std::string ve::cmdLine::shortDescr_ [static, protected]

stores program's short description.

Definition at line 293 of file veUtils.h.

Referenced by shortDescr().

12.8.3.11 std::string ve::cmdLine::descr_ [static, protected]

stores program's long description.

Definition at line 295 of file veUtils.h.

Referenced by descr().

12.8.3.12 std::string ve::cmdLine::usage_ [static, protected]

stores program's usage.

Definition at line 297 of file veUtils.h.

Referenced by usage().

The documentation for this class was generated from the following file:

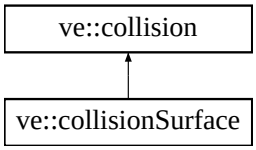
- veUtils.h

12.9 ve::collision Class Reference

base class for collision models.

`#include <veCollision.h>`

Inheritance diagram for ve::collision::



Public Member Functions

- collision ()
- virtual ~collision ()
- virtual int addObject (unsigned int objectId, const ve::vec6f &pos, float offsetZ=0.0f, float deltaZMax=1.0f, unsigned int mode=ve::COLLISION_GROUND)
- virtual int addObject (ve::dataContainer object, float offsetZ=0.0f, float deltaZMax=1.0f, unsigned int mode=ve::COLLISION_GROUND)
- virtual float updateObject (unsigned int objectId, ve::vec6f &pos)
- virtual int dropObject (unsigned int objectId)

12.9.1 Detailed Description

base class for collision models.

This base class provides the interface for collision model implementations. A motion model class calls automatically collision::updateObject() in its update() method. Note that the general interface may likely change as soon as additional collision models are implemented and when it becomes clearer what functions are actually shared.

Author:
MvdH & gf
Revision
2.1

Definition at line 48 of file veCollision.h.

12.9.2 Constructor & Destructor Documentation

12.9.2.1 ve::collision::collision () `[inline]`

default constructor. The general constructor just initializes the private variables of the base class.

Definition at line 54 of file veCollision.h.

12.9.2.2 virtual ve::collision::~collision () `[inline, virtual]`

destructor

Definition at line 56 of file veCollision.h.

12.9.3 Member Function Documentation

12.9.3.1 virtual int ve::collision::addObject (unsigned int *objectId*, const ve::vec6f & *pos*, float *offsetZ* = `0.0f`, float *deltaZMax* = `1.0f`, unsigned int *mode* = `ve::COLLISION_GROUND`) [virtual]

registers an object and sets its characteristic values

(implementation specific) local data are allocated. This is a non-functional interface definition!

Parameters:

objectId the object id to be registered
pos current (allowed) position
offsetZ (optional) the fixed minimal altitude over ground
deltaZMax (optional) the maximum climb height when in ground mode
mode (optional) one of the collisionMode ids, defining whether collision model works in ground or fly mode

Returns:

0 in case of success.

Reimplemented in ve::collisionSurface.

Referenced by addObject().

12.9.3.2 virtual int ve::collision::addObject (ve::dataContainer *object*, float *offsetZ* = `0.0f`, float *deltaZMax* = `1.0f`, unsigned int *mode* = `ve::COLLISION_GROUND`) [inline, virtual]

registers an object and sets its characteristic values

(implementation specific) local data are allocated. This is a non-functional interface definition!

Parameters:

object the dataContainer to be registered
offsetZ (optional) the fixed minimal altitude over ground
deltaZMax (optional) the maximum climb height when in ground mode
mode (optional) one of the collisionMode ids, defining whether collision model works in ground or fly mode

Returns:

0 in case of success.

Definition at line 76 of file veCollision.h.

References addObject(), ve::dataChar::objectId(), and ve::dataContainer::position().

12.9.3.3 virtual float ve::collision::updateObject (unsigned int *objectId*, ve::vec6f & *pos*) [virtual]

applies the collision detection method.

This is a non-functional interface definition!

Parameters:

objectId the object's id that is subjected to collision detection
pos the position to be tested, it might be changed by this routine.

Returns:

altitude over ground or NaN if no ground plane has been found

Reimplemented in ve::collisionSurface.

12.9.3.4 virtual int ve::collision::dropObject (unsigned int *objectId*) [inline, virtual]

unregisters an object

(implementation specific) local data are deallocated. This is a non-functional interface definition!

Parameters:
objectId the object id to be unregistered

Returns:
0 in case of success.

Reimplemented in ve::collisionSurface.

Definition at line 91 of file veCollision.h.

The documentation for this class was generated from the following file:

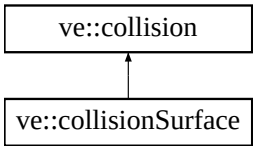
- veCollision.h

12.10 ve::collisionSurface Class Reference

collision model based on motion surfaces.

`#include <veCollision.h>`

Inheritance diagram for ve::collisionSurface::



Public Member Functions

- collisionSurface ()
- collisionSurface (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~collisionSurface ()
- int load (const std::string &filename)
- void clear ()
- virtual int addObject (unsigned int objectId, const ve::vec6f &pos, float offsetZ=0.0f, float deltaZMax=1.0f, unsigned int mode=ve::COLLISION_GROUND)
- virtual int addObject (const ve::dataContainer &object, float offsetZ=0.0f, float deltaZMax=1.0f, unsigned int mode=ve::COLLISION_GROUND)
- virtual float updateObject (unsigned int objectId, ve::vec6f &pos)
- virtual float updateObject (ve::dataContainer &object)
- virtual int dropObject (unsigned int objectId)
- virtual int dropObject (const ve::dataContainer &object)
- bool verbose () const
- void verbose (bool newVerbose)
- std::vector< ve::triangle > & triangles ()
- const ve::vec3f & minCoord () const
- const ve::vec3f & maxCoord () const

Protected Member Functions

- int loadModel (const std::string &filename)
- int loadAscii (const std::string &filename)
- float keepOnSurface (ve::vec6f &pos, ve::_collisionSurfaceObj &obj) const
- float keepOverSurface (ve::vec6f &pos, ve::_collisionSurfaceObj &obj) const

Protected Attributes

- std::vector< ve::triangle > vTriangle
- ve::vec3f minVtx
- ve::vec3f maxVtx
- std::map< unsigned int, ve::_collisionSurfaceObj > m_mObj
- bool verb_

12.10.1 Detailed Description

collision model based on motion surfaces.

This class is a reference implementation of a basic 3D collision detection by testing the height over ground planes defined in an array of triangles. The triangle geometry can be loaded using the geometry loaders from ve::geoGroup. in ground mode (=COLLISION_GROUND) collisionSurface takes a ground model and keeps the observer on it, at the height of offsetZ. So to use it in a proper way, you will have to build a ground model with holes were the user is not allowed to move to and set this model as the collision geometry (using the surface="X" attribute in the xmlIni file). Alternatively, you can use collisionSurface as collision for simple fly models, (mode=COLLSION_FLY), in this case any positve altitude-offsetZ over the ground plane will be allowed.

Author:
gf
Revision
2.1

Definition at line 154 of file veCollision.h.

12.10.2 Constructor & Destructor Documentation

12.10.2.1 ve::collisionSurface::collisionSurface ()

default constructor

12.10.2.2 ve::collisionSurface::collisionSurface (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor reading initialization values from an xml inifile, please see veLibXml.pdf for details.

12.10.2.3 virtual ve::collisionSurface::~collisionSurface () [inline, virtual]

destructor

Definition at line 161 of file veCollision.h.

References clear().

12.10.3 Member Function Documentation

12.10.3.1 int ve::collisionSurface::load (const std::string & *filename*)

loads collision geometry from file, filetype is chosen by suffix.

Recognized filteypes are the same as for ve::geoGroup.

12.10.3.2 void ve::collisionSurface::clear ()

clears current geometry definition.

Referenced by ~collisionSurface().

12.10.3.3 virtual int ve::collisionSurface::addObject (unsigned int *objectId*, const ve::vec6f & *pos*, float *offsetZ* = `0.0f`, float *deltaZMax* = `1.0f`, unsigned int *mode* = `ve::COLLISION_GROUND`) `[virtual]`

registers an object and sets its characteristic values

a _collisionSurfaceObj is allocated.

Parameters:
- ***objectId*** the object id to be registered
- ***pos*** current (allowed) position
- ***offsetZ*** (optional) the fixed mininal altitude over ground
- ***deltaZMax*** (optional) the maximum climb height when in ground mode
- ***mode*** (optional) one of the collisionMode ids, defining whether collision model works in ground or fly mode

Returns:
0 in case of success.

Reimplemented from ve::collision.

Referenced by addObject().

12.10.3.4 virtual int ve::collisionSurface::addObject (const ve::dataContainer & *object*, float *offsetZ* = `0.0f`, float *deltaZMax* = `1.0f`, unsigned int *mode* = `ve::COLLISION_GROUND`) `[inline, virtual]`

registers an object and sets its characteristic values

a _collisionSurfaceObj is allocated.

Parameters:
- ***object*** the dataContainer to be registered
- ***offsetZ*** (optional) the fixed minimal altitude over ground
- ***deltaZMax*** (optional) the maximum climb height when in ground mode
- ***mode*** (optional) one of the collisionMode ids, defining whether collision model works in ground or fly mode

Returns:
0 in case of success.

Definition at line 186 of file veCollision.h.

References addObject(), ve::dataChar::objectId(), and ve::dataContainer::position().

12.10.3.5 virtual float ve::collisionSurface::updateObject (unsigned int *objectId*, ve::vec6f & *pos*) `[virtual]`

applies the collision detection method.

Parameters:
- ***objectId*** the object's id that is subjected to collision detection
- ***pos*** the position to be tested, it might be changed by this routine.

Returns:
altitude over ground or NaN if no ground plane has been found

Reimplemented from ve::collision.

Referenced by updateObject().

12.10.3.6 virtual float ve::collisionSurface::updateObject (ve::dataContainer & *object*) [inline, virtual]

applies the collision detection method to a dataContainer.

Parameters:
object the dataContainer that is subjected to collision detection

Returns:
altitude over ground or NaN if no ground plane has been found

Definition at line 199 of file veCollision.h.

References ve::dataChar::objectId(), ve::dataContainer::position(), and updateObject().

12.10.3.7 virtual int ve::collisionSurface::dropObject (unsigned int *objectId*) [virtual]

unregisters an object

a _collisionSurfaceObj is deallocated.

Parameters:
objectId the object id to be unregistered

Returns:
0 in case of success.

Reimplemented from ve::collision.

Referenced by dropObject().

12.10.3.8 virtual int ve::collisionSurface::dropObject (const ve::dataContainer & *object*) [inline, virtual]

unregisters a dataContainer

a _collisionSurfaceObj is deallocated.

Parameters:
object the dataContainer to be unregistered

Returns:
0 in case of success.

Definition at line 210 of file veCollision.h.

References dropObject(), and ve::dataChar::objectId().

12.10.3.9 bool ve::collisionSurface::verbose () const [inline]

returns verbose state

Definition at line 214 of file veCollision.h.

References verb_.

12.10.3.10 void ve::collisionSurface::verbose (bool *newVerbose*) [inline]

sets verbose state

Definition at line 216 of file veCollision.h.

References verb_.

12.10.3.11 std::vector<ve::triangle>& ve::collisionSurface::triangles () [inline]

allows access to triangle vector

Definition at line 219 of file veCollision.h.

References vTriangle.

12.10.3.12 const ve::vec3f& ve::collisionSurface::minCoord () const [inline]

returns minimum coordinate

Definition at line 221 of file veCollision.h.

References minVtx.

12.10.3.13 const ve::vec3f& ve::collisionSurface::maxCoord () const [inline]

returns maximum coordinate

Definition at line 223 of file veCollision.h.

References maxVtx.

12.10.3.14 int ve::collisionSurface::loadModel (const std::string & *filename*) [protected]

loads collision geometry from a model file.

12.10.3.15 int ve::collisionSurface::loadAscii (const std::string & *filename*) [protected]

loads collision geometry from an ascii file.

12.10.3.16 float ve::collisionSurface::keepOnSurface (ve::vec6f & *pos*, ve::_collisionSurfaceObj & *obj*) const [protected]

test whether movement is allowed to new position pos

This method tests whether a new position is above the motion surface geometry.

Parameters:

pos the position to be tested

obj an internal structure holding the data of a collision object

Returns:

altitude over surface when pos is allowed, otherwise NaN

12.10.3.17 float ve::collisionSurface::keepOverSurface (ve::vec6f & *pos*, ve::_collisionSurfaceObj & *obj*) const [protected]

returns the z distance to a surface or NAN if pos is not over surface at all

This method is used for a flying motion model.

Parameters:

pos the position to be tested

obj an internal structure holding the data of a collision object

Returns:

altitude over surface when pos is allowed, otherwise NaN

12.10.4 Member Data Documentation

12.10.4.1 std::vector<ve::triangle> ve::collisionSurface::vTriangle [protected]

stores movement surfaces

Definition at line 246 of file veCollision.h.

Referenced by triangles().

12.10.4.2 ve::vec3f ve::collisionSurface::minVtx [protected]

stores minimum coordinate

Definition at line 248 of file veCollision.h.

Referenced by minCoord().

12.10.4.3 ve::vec3f ve::collisionSurface::maxVtx [protected]

stores maximum coordinate

Definition at line 250 of file veCollision.h.

Referenced by maxCoord().

12.10.4.4 std::map<unsigned int, ve::_collisionSurfaceObj> ve::collisionSurface::m_mObj [protected]

stores collision objects

Definition at line 252 of file veCollision.h.

12.10.4.5 bool ve::collisionSurface::verb_ [protected]

stores verbose state, allows more debug output

Definition at line 254 of file veCollision.h.

Referenced by verbose().

The documentation for this class was generated from the following file:

- veCollision.h

12.11 ve::connectionInfo Struct Reference

a struct with information on one network connection

`#include <veDeviceNetwork.h>`

Public Attributes

- bool inUse
- bool connected
- double deltaLastPing
- unsigned int ack
- double timeout
- sockaddr_in address

12.11.1 Detailed Description

a struct with information on one network connection

Definition at line 107 of file veDeviceNetwork.h.

12.11.2 Member Data Documentation

12.11.2.1 bool ve::connectionInfo::inUse

is the connection in use (will be true when connected or trying to connect, otherwise false)?

Definition at line 110 of file veDeviceNetwork.h.

12.11.2.2 bool ve::connectionInfo::connected

has connected been established (will be true when connected, otherwise false)?

Definition at line 112 of file veDeviceNetwork.h.

12.11.2.3 double ve::connectionInfo::deltaLastPing

how much ms passed since last ping was received (high value indicates connection loss)

Definition at line 114 of file veDeviceNetwork.h.

12.11.2.4 unsigned int ve::connectionInfo::ack

how many times has connection request been send without having received an ack message

Definition at line 116 of file veDeviceNetwork.h.

12.11.2.5 double ve::connectionInfo::timeout

time waited for ack after connection request has been send

Definition at line 118 of file veDeviceNetwork.h.

12.11.2.6 struct sockaddr_in ve::connectionInfo::address

the address of the remote machiene

Definition at line 120 of file veDeviceNetwork.h.

The documentation for this struct was generated from the following file:

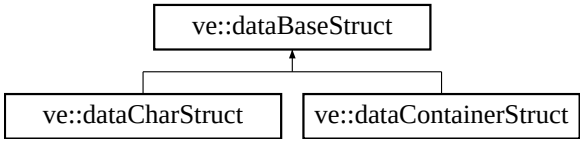
- veDeviceNetwork.h

12.12 ve::dataBaseStruct Class Reference

internal struct used as coomon data container header.

`#include <veDataContainer.h>`

Inheritance diagram for ve::dataBaseStruct::



Public Attributes

- unsigned int m_containerId
- unsigned int m_typeId
- unsigned int m_senderId
- unsigned int m_receiverId
- unsigned int m_deviceId
- unsigned int m_resourceId
- unsigned int m_objId
- unsigned int m_command
- unsigned int m_transfer
- unsigned int m_parent
- double m_timeStamp
- unsigned int m_uints [2]
- float m_floats [2]

12.12.1 Detailed Description

internal struct used as coomon data container header.

All variables that are shared by all container types are defined here.

Definition at line 40 of file veDataContainer.h.

12.12.2 Member Data Documentation

12.12.2.1 unsigned int ve::dataBaseStruct::m_containerId

the internal container Id will be generated automatically

Definition at line 43 of file veDataContainer.h.

12.12.2.2 unsigned int ve::dataBaseStruct::m_typeId

type of the data container, one of the DC_XYZ constants, e.g., DC_OBJECT for dataContainer

Definition at line 45 of file veDataContainer.h.

Referenced by ve::dataContainer::clear(), ve::dataChar::clear(), and ve::dataChar::type().

12.12.2.3 unsigned int ve::dataBaseStruct::m_senderId

the sender ID (combining device and instance tags). See ve::deviceId for a list of possible senders.

Definition at line 47 of file veDataContainer.h.

Referenced by ve::dataChar::sender().

12.12.2.4 unsigned int ve::dataBaseStruct::m_receiverId

the personal receiver ID (defined in the xml initialization id="XYZ")

Definition at line 49 of file veDataContainer.h.

Referenced by ve::dataChar::receiverId().

12.12.2.5 unsigned int ve::dataBaseStruct::m_deviceId

the device ID (message is supposed to be read by veDeviceXXXX). See ve::deviceId for a list of possible devices.

Definition at line 51 of file veDataContainer.h.

Referenced by ve::dataChar::receiverDevice().

12.12.2.6 unsigned int ve::dataBaseStruct::m_resourceId

assigns this data container to represent a specific object class

Definition at line 53 of file veDataContainer.h.

Referenced by ve::dataChar::resourceId().

12.12.2.7 unsigned int ve::dataBaseStruct::m_objId

assigns this data container to represent a specific object

Definition at line 55 of file veDataContainer.h.

Referenced by ve::dataChar::objectId().

12.12.2.8 unsigned int ve::dataBaseStruct::m_command

brief stores/transmits a specific command / message. See ve::dcCommandType for a list of possible commands.

Definition at line 57 of file veDataContainer.h.

Referenced by ve::dataContainer::clear(), and ve::dataChar::command().

12.12.2.9 unsigned int ve::dataBaseStruct::m_transfer

tranfer flag, see ve::dcTransfer for a list of possible flags.

Definition at line 59 of file veDataContainer.h.

Referenced by ve::dataChar::transferSetting().

12.12.2.10 unsigned int ve::dataBaseStruct::m_parent

an id to create a hierarchical object structure

Definition at line 61 of file veDataContainer.h.

Referenced by ve::dataChar::parent().

12.12.2.11 double ve::dataBaseStruct::m_timeStamp

timestamp

Definition at line 63 of file veDataContainer.h.

Referenced by ve::dataChar::timeStamp().

12.12.2.12 unsigned int ve::dataBaseStruct::m_uints[2]

stores command/message specific additional data

Definition at line 65 of file veDataContainer.h.

Referenced by ve::dataChar::dataUI().

12.12.2.13 float ve::dataBaseStruct::m_floats[2]

stores command/message specific additional data

Definition at line 67 of file veDataContainer.h.

Referenced by ve::dataChar::dataF().

The documentation for this class was generated from the following file:

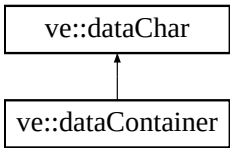
- veDataContainer.h

12.13 ve::dataChar Class Reference

veLib basic data class.

`#include <veDataContainer.h>`

Inheritance diagram for ve::dataChar::



Public Member Functions

- dataChar ()
- dataChar (unsigned int resourceId, const char *pData, unsigned int length)
- dataChar (unsigned int resourceId, const std::string &s)
- dataChar (const ve::xml &xs)
- dataChar (const std::string &s)
- const char * data () const
- void data (const char *pData, unsigned int length)
- void data (const std::string &s)
- int writeToBuffer (char *buffer) const
- int readFromBuffer (const char *buffer)
- double timeStamp () const
- void timeStamp (double timeStamp)
- unsigned int receiverId () const
- void receiverId (unsigned int id)
- unsigned int receiverDevice () const
- void receiverDevice (unsigned int id)
- unsigned int sender () const
- void sender (unsigned int id)
- unsigned int & type ()
- unsigned int type () const
- unsigned int & resourceId ()
- unsigned int resourceId () const
- unsigned int & objectId ()
- unsigned int objectId () const
- unsigned int & command ()
- unsigned int command () const
- unsigned int & parent ()
- unsigned int parent () const
- unsigned int & transferSetting ()
- unsigned int transferSetting () const
- unsigned int & id (unsigned int n=DC_ID_COMMAND)
- unsigned int id (unsigned int n=DC_ID_COMMAND) const
- ve::dataContainer & asContainer ()
- const ve::dataContainer & asContainer () const
- unsigned int & dataUI (unsigned int n=0)
- unsigned int dataUI (unsigned int n=0) const
- float & dataF (unsigned int n=0)
- float dataF (unsigned int n=0) const
- void userData (unsigned int uint0, unsigned int uint1=0, float float0=0.0f, float float1=0.0f)
- int interpret (const ve::xml &xs)
- int interpret (std::string s)
- ve::xml xml () const
- std::string str () const
- void clear ()

Static Public Member Functions

- static unsigned int size ()
- static unsigned int nData ()
- static std::string command2string (unsigned int cmd)
- static unsigned int string2command (const std::string &cmd)

Protected Member Functions

- int init ()

Protected Attributes

- dataUnion m_data

Static Protected Attributes

- static unsigned int currContainerId

12.13.1 Detailed Description

veLib basic data class.

This is a container class that contains and transfers raw character data. It is currently used to transfer text messages, or to dynamically define reources.

Definition at line 106 of file veDataContainer.h.

12.13.2 Constructor & Destructor Documentation

12.13.2.1 ve::dataChar::dataChar ()

default constructor

12.13.2.2 ve::dataChar::dataChar (unsigned int *resourceId*, const char * *pData*, unsigned int *length*)

constructor reading memory data into the data object

12.13.2.3 ve::dataChar::dataChar (unsigned int *resourceId*, const std::string & *s*)

constructor initializing data from a string

12.13.2.4 ve::dataChar::dataChar (const ve::xml & *xs*) [inline]

constructor from an xml statement

Definition at line 116 of file veDataContainer.h.

References interpret().

12.13.2.5 ve::dataChar::dataChar (const std::string & *s*) [inline]

constructor from a string

This constructor allows a scripting-like interactive syntax. It expects a string similar to a valid xml statement, except of following differences:

- no enclosing tags are allowed,
- the first word must be the mere command (a ve::dcCommandType without leading 'CMD_' prefix),
- the content data has to be specificied as attribute data="xyz".

  The container type is detected depending on the command, unspecified parameters are filled by default values.

Example

```
RESOURCE_ADD resource="7" mime="model/vrml" data="model.wrl"
OBJECT_ADD resource="7" object="5"
OBJECT_SET object="5" position="2.5 0 1.6 45 0 0"
```

Definition at line 134 of file veDataContainer.h.

References interpret().

12.13.3 Member Function Documentation

12.13.3.1 static unsigned int ve::dataChar::size () [inline, static]

returns the total number of bytes of a complete data container

Definition at line 137 of file veDataContainer.h.

References ve::totalSize.

12.13.3.2 static unsigned int ve::dataChar::nData () [inline, static]

returns the number of unspecified bytes available for user content

Reimplemented in ve::dataContainer.

Definition at line 139 of file veDataContainer.h.

References ve::headerSize, and ve::totalSize.

12.13.3.3 const char* ve::dataChar::data () const [inline]

returns a pointer to the data container's content

Reimplemented in ve::dataContainer.

Definition at line 142 of file veDataContainer.h.

References ve::dataUnion::chars, ve::dataCharStruct::m_chars, and m_data.

Referenced by data().

12.13.3.4 void ve::dataChar::data (const char * *pData*, unsigned int *length*)

sets the data container's content

Reimplemented in ve::dataContainer.

12.13.3.5 void ve::dataChar::data (const std::string & *s*) [inline]

sets the data container's content to a string

Reimplemented in ve::dataContainer.

Definition at line 146 of file veDataContainer.h.

References data().

12.13.3.6 int ve::dataChar::writeToBuffer (char * *buffer*) const

writes all data into this buffer.

Parameters:

buffer must be large enough (use size() method for memory allocation.) The byte order of the buffer is changed for TCP/IP transfer.

12.13.3.7 int ve::dataChar::readFromBuffer (const char * *buffer*)

reads all data from the buffer.

Parameters:

buffer must be large enough (use size() method for memory allocation.) The byte order of the buffer is changed for TCP/IP transfer.

12.13.3.8 double ve::dataChar::timeStamp () const [inline]

returns time stamp of creation / transfer

Definition at line 158 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_timeStamp.

12.13.3.9 void ve::dataChar::timeStamp (double *timeStamp*) [inline]

sets time stamp (done automatically whenever a network transfer is performed)

Definition at line 160 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_timeStamp.

12.13.3.10 unsigned int ve::dataChar::receiverId () const [inline]

returns unique receiver id

Definition at line 162 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_receiverId.

12.13.3.11 void ve::dataChar::receiverId (unsigned int *id*) [inline]

sets unique receiver id

Definition at line 164 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_receiverId.

12.13.3.12 unsigned int ve::dataChar::receiverDevice () const [inline]

returns receiver device id, see DEV_XYZ device identifiers

Definition at line 166 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_deviceId.

12.13.3.13 void ve::dataChar::receiverDevice (unsigned int *id*) [inline]

sets receiver device id, see DEV_XYZ device identifiers

Definition at line 168 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_deviceId.

12.13.3.14 unsigned int ve::dataChar::sender () const [inline]

returns sender id, see DEV_XYZ device identifiers

Definition at line 170 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_senderId.

12.13.3.15 void ve::dataChar::sender (unsigned int *id*) [inline]

sets sender id, see DEV_XYZ device identifiers

Definition at line 172 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_senderId.

12.13.3.16 unsigned int& ve::dataChar::type () [inline]

allows access to type (container class) id

Definition at line 174 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_typeId.

12.13.3.17 unsigned int ve::dataChar::type () const [inline]

returns type (container class) id

Definition at line 176 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_typeId.

12.13.3.18 unsigned int& ve::dataChar::resourceId () [inline]

allows access to the data container's resource id

Definition at line 179 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_resourceId.

12.13.3.19 unsigned int ve::dataChar::resourceId () const [inline]

allows read access to the data container's resource id

Definition at line 181 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_resourceId.

12.13.3.20 unsigned int& ve::dataChar::objectId () [inline]

allows access to the dataContainer's object id

Definition at line 183 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_objId.

Referenced by ve::collisionSurface::addObject(), ve::collision::addObject(), ve::collisionSurface::dropObject(), and ve::collisionSurface::updateObject().

12.13.3.21 unsigned int ve::dataChar::objectId () const [inline]

allows read access to the dataContainer's object id

Definition at line 185 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_objId.

12.13.3.22 unsigned int& ve::dataChar::command () [inline]

allows access to the data container's command id

Definition at line 187 of file veDataContainer.h.

References ve::dataUnion::chars, ve::dataBaseStruct::m_command, and m_data.

Referenced by ve::deviceContainer::setOutput().

12.13.3.23 unsigned int ve::dataChar::command () const [inline]

allows read access to the data container's command id

Definition at line 189 of file veDataContainer.h.

References ve::dataUnion::chars, ve::dataBaseStruct::m_command, and m_data.

12.13.3.24 unsigned int& ve::dataChar::parent () [inline]

allows access to the data container's parent

This feature may be used to create hierarchical scene graphs.

Definition at line 192 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_parent.

12.13.3.25 unsigned int ve::dataChar::parent () const [inline]

allows read access to the data container's parent

This feature may be used to create hierarchical scene graphs.

Definition at line 195 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_parent.

12.13.3.26 unsigned int& ve::dataChar::transferSetting () [inline]

allows access to the data container's network priority

Definition at line 198 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_transfer.

12.13.3.27 unsigned int ve::dataChar::transferSetting () const [inline]

allows read access to the data container's network priority

Definition at line 200 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_transfer.

12.13.3.28 unsigned int& ve::dataChar::id (unsigned int *n* = DC_ID_COMMAND)

allows access to the data container's various id fields

12.13.3.29 unsigned int ve::dataChar::id (unsigned int *n* = DC_ID_COMMAND) const

allows read access to the data container's various id fields

12.13.3.30 ve::dataContainer& ve::dataChar::asContainer () [inline]

casts the dataChar object into a dataContainer

this method is mainly for internal use in ve::devices. If you use it for your own purposes make sure to check the correct type() first.

Definition at line 211 of file veDataContainer.h.

12.13.3.31 const ve::dataContainer& ve::dataChar::asContainer () const [inline]

casts the dataChar object into a dataContainer, const

this method is mainly for internal use in ve::devices. If you use it for your own purposes make sure to check the correct type() first.

Definition at line 216 of file veDataContainer.h.

12.13.3.32 unsigned int& ve::dataChar::dataUI (unsigned int *n* = 0) [inline]

allows access to the data container's unsigned ints

Definition at line 219 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_uints.

12.13.3.33 unsigned int ve::dataChar::dataUI (unsigned int *n* = 0) const [inline]

allows read access to the data container's unsigned ints

Definition at line 221 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_uints.

12.13.3.34 float& ve::dataChar::dataF (unsigned int *n* = 0) [inline]

allows access to the data container's floats

Definition at line 223 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_floats.

12.13.3.35 float ve::dataChar::dataF (unsigned int *n* = 0) const [inline]

allows read access to the data container's floats

Definition at line 225 of file veDataContainer.h.

References ve::dataUnion::chars, m_data, and ve::dataBaseStruct::m_floats.

12.13.3.36 void ve::dataChar::userData (unsigned int *uint0*, unsigned int *uint1* = 0, float *float0* = 0.0f, float *float1* = 0.0f)

allows to set all special user data (dataUI, dataF) at once

12.13.3.37 int ve::dataChar::interpret (const ve::xml & *xs*)

interprets an xml statement and sets internal data fields accordingly

Compared to the corresponding constructor, this method has the advantage of returning an error code.

Parameters:
xs The xml statement to be interpreted

Returns:
0 in case of success, otherwise the number of occurred errors.

Referenced by dataChar().

12.13.3.38 int ve::dataChar::interpret (std::string *s*)

interprets a string and sets internal data fields accordingly

Compared to the corresponding constructor, this method has the advantage of returning an error code.

Parameters:
s The string to be interpreted

Returns:
0 in case of success, otherwise the number of occurred errors.

12.13.3.39 ve::xml ve::dataChar::xml () const

converts data container into an xml statement

Referenced by str().

12.13.3.40 std::string ve::dataChar::str () const [inline]

output of data container as human readable string

Definition at line 243 of file veDataContainer.h.

References ve::xml::str(), and xml().

12.13.3.41 static std::string ve::dataChar::command2string (unsigned int *cmd*) [static]

converts a command id int a command string

The strings and ids are the same as the enums defined in veTypes.h .

12.13.3.42 static unsigned int ve::dataChar::string2command (const std::string & *cmd*) [static]

converts a string into a command id

The strings and ids are the same as the enums defined in veTypes.h .

12.13.3.43 void ve::dataChar::clear () [inline]

resets the data container to its default state

Reimplemented in ve::dataContainer.

Definition at line 251 of file veDataContainer.h.

References ve::dataUnion::chars, init(), m_data, and ve::dataBaseStruct::m_typeId.

12.13.3.44 int ve::dataChar::init () [protected]

initializes all the data in the container

Referenced by ve::dataContainer::clear(), and clear().

12.13.4 Member Data Documentation

12.13.4.1 dataUnion ve::dataChar::m_data [protected]

this union stores all data in a flexible and expandable way

Definition at line 257 of file veDataContainer.h.

Referenced by ve::dataContainer::clear(), clear(), command(), ve::dataContainer::data(), data(), dataF(), dataUI(), objectId(), parent(), receiverDevice(), receiverId(), resourceId(), sender(), timeStamp(), transferSetting(), and type().

12.13.4.2 unsigned int ve::dataChar::currContainerId [static, protected]

works as counter for containers

Definition at line 260 of file veDataContainer.h.

The documentation for this class was generated from the following file:

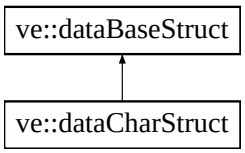
- veDataContainer.h

12.14 ve::dataCharStruct Class Reference

internal struct underlying the dataChar class

`#include <veDataContainer.h>`

Inheritance diagram for ve::dataCharStruct::



Public Attributes

- char m_chars [totalSize-headerSize]

12.14.1 Detailed Description

internal struct underlying the dataChar class

Definition at line 71 of file veDataContainer.h.

The documentation for this class was generated from the following file:

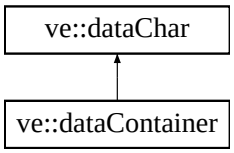
- veDataContainer.h

12.15 ve::dataContainer Class Reference

veLib data container representing a simulation object

`#include <veDataContainer.h>`

Inheritance diagram for ve::dataContainer::



Public Member Functions

- dataContainer ()
- dataContainer (unsigned int resourceId, unsigned int objectId)
- dataContainer (const ve::xml &xs)
- const char * data () const
- void data (const char *pData, unsigned int length)
- void data (const std::string &s)
- void clear ()

Generic access methods

- ve::vec6f & axes (unsigned int n=DC_AXES_INPUT)
- const ve::vec6f & axes (unsigned int n=DC_AXES_INPUT) const
- ve::flag128 & flags (unsigned int n=DC_FLAGS_INPUT)
- const ve::flag128 & flags (unsigned int n=DC_FLAGS_INPUT) const

Additional access methods for the most important data fields

- ve::vec6f & inputAxes ()
- const ve::vec6f & inputAxes () const
- ve::vec6f & position ()
- const ve::vec6f & position () const
- ve::vec6f & velocity ()
- const ve::vec6f & velocity () const
- ve::vec6f & acceleration ()
- const ve::vec6f & acceleration () const
- ve::vec6f & color ()
- const ve::vec6f & color () const
- ve::flag128 & buttons ()
- const ve::flag128 & buttons () const
- ve::flag128 & state ()
- const ve::flag128 & state () const

Static Public Member Functions

- static unsigned int nData ()

12.15.1 Detailed Description

veLib data container representing a simulation object

This class is designed to represent one entity of a simulation and to store all normally necessary data. For example it might represent the state of an input/output device, an observer, or a dynamic object. It is particularly designed for communication between the various ve::device descendants, including networks. Internally one of the dataContainerBaseType descendants is used to store the data.

Definition at line 272 of file veDataContainer.h.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 ve::dataContainer::dataContainer ()

default constructor

12.15.2.2 ve::dataContainer::dataContainer (unsigned int *resourceId*, unsigned int *objectId*)

constructor defining resource id and object id

12.15.2.3 ve::dataContainer::dataContainer (const ve::xml & *xs*) [inline]

constructor from an xml statement

Definition at line 279 of file veDataContainer.h.

12.15.3 Member Function Documentation

12.15.3.1 ve::vec6f& ve::dataContainer::axes (unsigned int *n* = DC_AXES_INPUT)

allows access to all the dataContainer's sixdofs

Referenced by acceleration(), color(), inputAxes(), position(), and velocity().

12.15.3.2 const ve::vec6f& ve::dataContainer::axes (unsigned int *n* = DC_AXES_INPUT) const

allows read access to all the dataContainer's sixdofs

12.15.3.3 ve::flag128& ve::dataContainer::flags (unsigned int *n* = DC_FLAGS_INPUT)

allows access to all the data container's flag containers

Referenced by buttons(), and state().

12.15.3.4 const ve::flag128& ve::dataContainer::flags (unsigned int *n* = DC_FLAGS_INPUT) const

allows read access to all the data container's flag containers

12.15.3.5 ve::vec6f& ve::dataContainer::inputAxes () [inline]

allows access to the dataContainer's input axes

Definition at line 296 of file veDataContainer.h.

References axes().

12.15.3.6 const ve::vec6f& ve::dataContainer::inputAxes () const [inline]

allows read access to the dataContainer's input axes

Definition at line 298 of file veDataContainer.h.

References axes().

12.15.3.7 ve::vec6f& ve::dataContainer::position () [inline]

allows access to the dataContainer's position axes

Definition at line 300 of file veDataContainer.h.

References axes().

Referenced by ve::collisionSurface::addObject(), ve::collision::addObject(), and ve::collisionSurface::updateObject().

12.15.3.8 const ve::vec6f& ve::dataContainer::position () const [inline]

allows read access to the dataContainer's position axes

Definition at line 302 of file veDataContainer.h.

References axes().

12.15.3.9 ve::vec6f& ve::dataContainer::velocity () [inline]

allows access to the dataContainer's velocity axes

Definition at line 304 of file veDataContainer.h.

References axes().

12.15.3.10 const ve::vec6f& ve::dataContainer::velocity () const [inline]

allows read access to the dataContainer's velocity axes

Definition at line 306 of file veDataContainer.h.

References axes().

12.15.3.11 ve::vec6f& ve::dataContainer::acceleration () [inline]

allows access to the dataContainer's acceleration axes

Definition at line 308 of file veDataContainer.h.

References axes().

12.15.3.12 const ve::vec6f& ve::dataContainer::acceleration () const [inline]

allows read access to the dataContainer's acceleration axes

Definition at line 310 of file veDataContainer.h.

References axes().

12.15.3.13 ve::vec6f& ve::dataContainer::color () [inline]

use this to set color for OVERLAY OBJECTS ONLY!! For all other objects, this sets the velocity!

Definition at line 312 of file veDataContainer.h.

References axes().

12.15.3.14 const ve::vec6f& ve::dataContainer::color () const [inline]

use this to read color for OVERLAY OBJECTS ONLY!! For all other objects, this reads the velocity!

Definition at line 314 of file veDataContainer.h.

References axes().

12.15.3.15 ve::flag128& ve::dataContainer::buttons () [inline]

allows access to the data container's input flags

Definition at line 316 of file veDataContainer.h.

References flags().

12.15.3.16 const ve::flag128& ve::dataContainer::buttons () const [inline]

allows read access to the data container's input flags

Definition at line 318 of file veDataContainer.h.

References flags().

12.15.3.17 ve::flag128& ve::dataContainer::state () [inline]

allows access to the data container's state flags

Definition at line 320 of file veDataContainer.h.

References flags().

12.15.3.18 const ve::flag128& ve::dataContainer::state () const [inline]

allows read access to the data container's state flags

Definition at line 322 of file veDataContainer.h.

References flags().

12.15.3.19 const char* ve::dataContainer::data () const [inline]

returns a pointer to the data unspecified bytes

Reimplemented from ve::dataChar.

Definition at line 326 of file veDataContainer.h.

References ve::dataContainerStruct::m_chars, ve::dataChar::m_data, and ve::dataUnion::mStd.

Referenced by data().

12.15.3.20 void ve::dataContainer::data (const char * *pData*, unsigned int *length*)

sets the data container's unspecified bytes

Reimplemented from ve::dataChar.

12.15.3.21 void ve::dataContainer::data (const std::string & *s*) [inline]

sets the data container's unspecified bytes to a string

Reimplemented from ve::dataChar.

Definition at line 330 of file veDataContainer.h.

References data().

12.15.3.22 static unsigned int ve::dataContainer::nData () [inline, static]

returns the number of unspecified bytes

Reimplemented from ve::dataChar.

Definition at line 332 of file veDataContainer.h.

References ve::DC_NUM_SIXDOFS, ve::headerSize, and ve::totalSize.

12.15.3.23 void ve::dataContainer::clear () [inline]

resets the data container to its initial state

Reimplemented from ve::dataChar.

Definition at line 336 of file veDataContainer.h.

References ve::CMD_OBJECT_SET, ve::dataChar::init(), ve::dataBaseStruct::m_command, ve::dataChar::m_data, ve::dataBaseStruct::m_typeId, and ve::dataUnion::mStd.

The documentation for this class was generated from the following file:

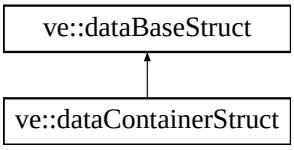
- veDataContainer.h

12.16 ve::dataContainerStruct Class Reference

internal struct underlying the dataContainer class

`#include <veDataContainer.h>`

Inheritance diagram for ve::dataContainerStruct::



Public Attributes

- float m_axes [ve::DC_NUM_SIXDOFS][6]
- unsigned int m_inputFlags [4]
- unsigned int m_stateFlags [4]
- char m_chars [totalSize-headerSize-ve::DC_NUM_SIXDOFS ∗6 ∗sizeof(float)-2 ∗4 ∗sizeof(unsigned int)]

12.16.1 Detailed Description

internal struct underlying the dataContainer class

Definition at line 79 of file veDataContainer.h.

12.16.2 Member Data Documentation

12.16.2.1 float ve::dataContainerStruct::m_axes[ve::DC_NUM_SIXDOFS][6]

stores 4 sixdof's This array provides storage for position, speed, acceleration, and input axes of one object.

Definition at line 84 of file veDataContainer.h.

12.16.2.2 unsigned int ve::dataContainerStruct::m_inputFlags[4]

This array provides storage for a ve::flag128 container storing input states.

Definition at line 87 of file veDataContainer.h.

12.16.2.3 unsigned int ve::dataContainerStruct::m_stateFlags[4]

This array provides storage for a ve::flag128 container storing state flags.

Definition at line 90 of file veDataContainer.h.

The documentation for this class was generated from the following file:

- veDataContainer.h

12.17 ve::dataUnion Union Reference

internal union comprising all low level data structures of various data containers

`#include <veDataContainer.h>`

Public Attributes

- dataCharStruct chars
- dataContainerStruct mStd

12.17.1 Detailed Description

internal union comprising all low level data structures of various data containers

Definition at line 96 of file veDataContainer.h.

The documentation for this union was generated from the following file:

- veDataContainer.h

12.18 ve::delayedData Struct Reference

Public Attributes

- ve::dataChar data
- double time

12.18.1 Detailed Description

Definition at line 85 of file veDeviceNetwork.h.

The documentation for this struct was generated from the following file:

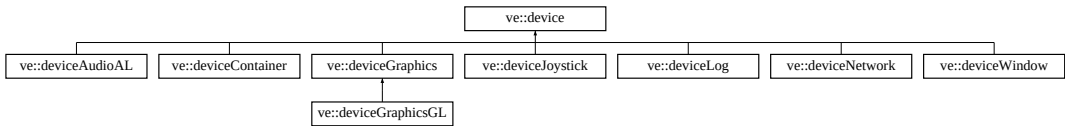
- veDeviceNetwork.h

12.19 ve::device Class Reference

The general device class.

`#include <veDevice.h>`

Inheritance diagram for ve::device::



Public Member Functions

- device ()
- virtual ~device ()
- virtual int update (double)
- unsigned int typeId () const
- unsigned int id () const
- virtual int reset ()
- int setPlugin (std::string filename)
- virtual int getInput (ve::dataChar &data, unsigned int mask=DC_MASK_ALL)
- virtual int setOutput (const ve::dataChar &data)
- virtual int setOutput (ve::dataChar &data, unsigned int command)
- virtual int getInput (ve::vec6f &axes, ve::flag128 &flags)
- virtual int setOutput (const ve::vec6f &pos, const ve::flag128 &flags)
- int setRemapping (ve::xmlIni &ini)
- virtual int observer (unsigned int)
- unsigned int resourceIdMax () const
- unsigned int objectIdMax () const

Protected Member Functions

- virtual int init ()
- virtual int shutdown ()
- virtual int queryDevice (ve::vec6f &, ve::flag128 &)
- virtual int updateDevice (const ve::vec6f &, const ve::flag128 &)
- virtual int updateObject (const ve::dataContainer &)
- virtual int updateResource (const ve::dataChar &)
- virtual int getVar (ve::dataChar &data)
- virtual int setVar (const ve::dataChar &data)
- void expose (int &i, const char *name, bool readOnly=false)
- void expose (unsigned int &i, const char *name, bool readOnly=false)
- void expose (float &f, const char *name, bool readOnly=false)
- void expose (std::string &s, const char *name, bool readOnly=false)
- void expose (bool &b, const char *name, bool readOnly=false)

Protected Attributes

- std::map< std::string, ve::_exposedVar > m_varTable
- unsigned int m_inputMapping [6]
- ve::vec6f m_inputScale
- ve::vec6f m_inputShift
- ve::vec6f m_inputDeadzone
- unsigned int m_typeId
- unsigned int m_id
- unsigned int m_resIdMax
- unsigned int m_objIdMax
- ve::plugin * m_pPlugin

12.19.1 Detailed Description

The general device class.

This class is used as a base class for all devices (joystick, keyboard, mouse, network, ...) in the veLib. For detailed information on the particular devices refer to their implementation. This class defines only the interface. So one can use any device later without knowing the exact one connected to the computer as a general device (this is called abstraction). The particular implementation handles then the actual hardware communication necessary. The getInput() and setOuput() calls the member function getInput. This is the place to put the actual hardware communication in the derived class.

Author:
MvdH / weyel / gf

Revision
2.9

Definition at line 61 of file veDevice.h.

12.19.2 Constructor & Destructor Documentation

12.19.2.1 ve::device::device ()

default constructor

12.19.2.2 virtual ve::device::~device () [virtual]

destructor

12.19.3 Member Function Documentation

12.19.3.1 virtual int ve::device::update (double) [inline, virtual]

updates device, interface definition

Reimplemented in ve::deviceAudioAL, ve::deviceContainer, ve::deviceGraphicsGL, and ve::deviceWindow.

Definition at line 69 of file veDevice.h.

12.19.3.2 unsigned int ve::device::typeId () const [inline]

returns device type id

Definition at line 72 of file veDevice.h.

References m_typeId.

12.19.3.3 unsigned int ve::device::id () const [inline]

returns individual id (often corresponding to iniSectionId)

Definition at line 74 of file veDevice.h.

References m_id.

12.19.3.4 virtual int ve::device::reset () [virtual]

reset the device by shutdown and new init

12.19.3.5 int ve::device::setPlugin (std::string *filename*)

sets a plugin to take over the device's main functionality.

Currently works only for graphics devices (devices that implement a draw() function). If the plugin file exports an init() function, it is called.

Parameters:

filename .so or .dll file to provide the plugin functions

Returns:

ERR_OK in case of success, ERR_ERROR in case plugin does not provide the right functions(see plugin class for details).

12.19.3.6 virtual int ve::device::getInput (ve::dataChar & *data*, unsigned int *mask* = DC_MASK_ALL) [virtual]

reads input from the device into a data container.

If the current parameters of an input device are requested, a dataContainer has to be passed, for reading variables (using CMD_DEVICE_VAR_GET/CMD_OBJECT_VAR_GET) a dataChar is required which has to carry the variable identifier in its data() part.

Parameters:

data the dataContainer that is overwritten

mask (optional) defines the data fields that are overwritten, a bitwise combination of DC_MASK_XYZ constants, see veTypes.h for further info.

Returns:
0 in case of success.

Reimplemented in ve::deviceLog, ve::deviceContainer, and ve::deviceNetwork.

12.19.3.7 virtual int ve::device::setOutput (const ve::dataChar & *data*) [virtual]

transmits a new output to the device.

Parameters:
data the data container that has to contain all necessary information.

Returns:
0 in case of success.

Reimplemented in ve::deviceLog, ve::deviceContainer, and ve::deviceNetwork.

12.19.3.8 virtual int ve::device::setOutput (ve::dataChar & *data*, unsigned int *command*) [virtual]

transmits a new output to the device and defines a command.

Parameters:
data the data container that contains all necessary information
command defines the command that shall be executed. Use one of the ve::CMD_XYZ constants (veTypes.h). It is stored in the dataContainer!

Returns:
0 in case of success.

Reimplemented in ve::deviceLog, ve::deviceContainer, and ve::deviceNetwork.

12.19.3.9 virtual int ve::device::getInput (ve::vec6f & *axes*, ve::flag128 & *flags*) [inline, virtual]

gets current input state of device.

Definition at line 107 of file veDevice.h.

References queryDevice().

12.19.3.10 virtual int ve::device::setOutput (const ve::vec6f & *pos*, const ve::flag128 & *flags*) [inline, virtual]

sets output of device.

Definition at line 110 of file veDevice.h.

References updateDevice().

12.19.3.11 int ve::device::setRemapping (ve::xmlIni & *ini*)

sets the remapping of input axes The method looks in ini for the first occurences of the tags "axesInputScale" and "axesInputMapping". axesInputScale must provide six float values in its content. axesInputMapping must provide the values 0..5 in the desired order

Parameters:

ini provides the remapping and scaling values.

Returns:

0 in case of success, 1 in case of an error, and 2 if no suitable initialization tags could be found.

12.19.3.12 virtual int ve::device::observer (unsigned *int*) [inline, virtual]

sets object id that determines the observer position, 0 means none. In case of success 0 is returned.

Definition at line 122 of file veDevice.h.

12.19.3.13 unsigned int ve::device::resourceIdMax () const [inline]

returns maximum resource id ever defined

useful for defining new unique resource ids on the fly. Note that currently this method may not work correctly for virtual and network devices.

Definition at line 126 of file veDevice.h.

References m_resIdMax.

12.19.3.14 unsigned int ve::device::objectIdMax () const [inline]

returns maximum object id ever defined

useful for defining new unique object ids on the fly. Note that currently this method may not work correctly for virtual and network devices.

Definition at line 130 of file veDevice.h.

References m_objIdMax.

12.19.3.15 virtual int ve::device::init () [protected, virtual]

Initialize the device.

12.19.3.16 virtual int ve::device::shutdown () [protected, virtual]

Shut down the device.

12.19.3.17 virtual int ve::device::queryDevice (ve::vec6f &, ve::flag128 &) [inline, protected, virtual]

handles get input requests, placeholder. This method should be overwritten by derived input devices. It is called by the public getInput methods of ve::device.

Reimplemented in ve::deviceWindow, and ve::deviceJoystick.

Definition at line 139 of file veDevice.h.

Referenced by getInput().

12.19.3.18 virtual int ve::device::updateDevice (const ve::vec6f &, const ve::flag128 &) [inline, protected, virtual]

handles set output requests, placeholder. This method should be overwritten by derived output devices. It is called by the public setOutput methods of ve::device.

Reimplemented in ve::deviceGraphics, ve::deviceAudioAL, and ve::deviceWindow.

Definition at line 143 of file veDevice.h.

Referenced by setOutput().

12.19.3.19 virtual int ve::device::updateObject (const ve::dataContainer &) [inline, protected, virtual]

defines an object instance's properties from a standard ve::dataContainer.

This is only an interface definition and should be overwritten by devices that use scene graphs! The method cannot be called directly but via the generic public setOutput() method. The command (add, drop, set, etc.) and all parameters are defined in the corresponding data fields of the container.

Parameters:

data passes the ve::dataContainer.

Returns:

0 in case of success.

Reimplemented in ve::deviceAudioAL, ve::deviceGraphicsGL, and ve::deviceWindow.

Definition at line 154 of file veDevice.h.

12.19.3.20 virtual int ve::device::updateResource (const ve::dataChar &) [inline, protected, virtual]

updates a resource at runtime

The method cannot be called directly but via the generic public setOutput() method.

Parameters:

data contains all data describing the resource.

Returns:

0 in case of success.

Reimplemented in ve::deviceAudioAL, ve::deviceGraphicsGL, and ve::deviceWindow.

Definition at line 160 of file veDevice.h.

12.19.3.21 virtual int ve::device::getVar (ve::dataChar & *data*) [protected, virtual]

reads an exposed variable of the device via a dataChar container

This feature is publicly accessible using getInput(CMD_DEVICE_VAR_GET). It has been added recently (veLib v1.1.1), therefore only a few internal variables are exposed up to now. If you need access to a particular feature at runtime, feel free to contact the developers, exposure is normally easily implemented, provided that a runtime access is sensible.

12.19.3.22 virtual int ve::device::setVar (const ve::dataChar & *data*) [protected, virtual]

sets an exposed variable of the device via a dataChar container

This feature is publicly accessible using setOutput(CMD_DEVICE_VAR_SET). It has been added recently (veLib v1.1.1), therefore only a few internal variables are exposed up to now. If you need access to a particular feature at runtime, feel free to contact the developers, exposure is normally easily implemented, provided that a runtime access is sensible.

Reimplemented in ve::deviceWindow.

12.19.3.23 void ve::device::expose (int & *i*, const char * *name*, bool *readOnly* = false) [inline, protected]

exposes an int variable

Definition at line 175 of file veDevice.h.

References m_varTable, and ve::TYPE_INT32.

12.19.3.24 void ve::device::expose (unsigned int & *i*, const char * *name*, bool *readOnly* = false) [inline, protected]

exposes an unsigned int variable

Definition at line 178 of file veDevice.h.

References m_varTable, and ve::TYPE_UINT32.

12.19.3.25 void ve::device::expose (float & *f*, const char * *name*, bool *readOnly* = false) [inline, protected]

exposes a float variable

Definition at line 181 of file veDevice.h.

References m_varTable, and ve::TYPE_FLOAT32.

12.19.3.26 void ve::device::expose (std::string & *s*, const char * *name*, bool *readOnly* = false) [inline, protected]

exposes a string variable

Definition at line 184 of file veDevice.h.

References m_varTable, and ve::TYPE_STRING.

12.19.3.27 void ve::device::expose (bool & *b*, const char * *name*, bool *readOnly* = false) [inline, protected]

exposes a boolean variable

Definition at line 187 of file veDevice.h.

12.19.4 Member Data Documentation

12.19.4.1 std::map<std::string,ve::_exposedVar> ve::device::m_varTable [protected]

stores exposed variables

Definition at line 188 of file veDevice.h.

Referenced by expose().

12.19.4.2 unsigned int ve::device::m_inputMapping[6] [protected]

stores the input axes (re)mapping

Definition at line 193 of file veDevice.h.

12.19.4.3 ve::vec6f ve::device::m_inputScale [protected]

stores the input axes scaling (multiplication) values

Definition at line 195 of file veDevice.h.

12.19.4.4 ve::vec6f ve::device::m_inputShift [protected]

stores the input axes shifting (addition) values

Definition at line 197 of file veDevice.h.

12.19.4.5 ve::vec6f ve::device::m_inputDeadzone [protected]

stores the input axes deadzone range values

Definition at line 199 of file veDevice.h.

12.19.4.6 unsigned int ve::device::m_typeId [protected]

stores the type of the device

Definition at line 201 of file veDevice.h.

Referenced by typeId().

12.19.4.7 unsigned int ve::device::m_id [protected]

stores individual id

Definition at line 203 of file veDevice.h.

Referenced by id().

12.19.4.8 unsigned int ve::device::m_resIdMax [protected]

stores maximum resource id ever defined

Definition at line 205 of file veDevice.h.

Referenced by resourceIdMax().

12.19.4.9 unsigned int ve::device::m_objIdMax [protected]

stores maximum object id ever defined

Definition at line 207 of file veDevice.h.

Referenced by objectIdMax().

12.19.4.10 ve::plugin* ve::device::m_pPlugin [protected]

pointer to an eventual plugin

Definition at line 210 of file veDevice.h.

The documentation for this class was generated from the following file:

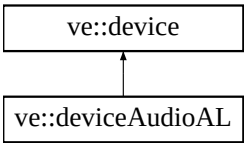
- veDevice.h

12.20 ve::deviceAudioAL Class Reference

a class for 3D audio simulation based on OpenAL

`#include <veDeviceAudioAL.h>`

Inheritance diagram for ve::deviceAudioAL::



Public Member Functions

- deviceAudioAL (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~deviceAudioAL ()
- virtual int update (double deltaT=0.0)
- virtual int observer (unsigned int objectId)

Protected Member Functions

- virtual int updateDevice (const ve::vec6f &pos, const ve::flag128 &flags)
- virtual int updateObject (const ve::dataContainer &data)
- virtual int updateResource (const ve::dataChar &data)

Protected Attributes

- std::vector< unsigned int > vSample
- std::vector< unsigned int > vResId
- std::vector< bool > vIsReference
- std::vector< float > vSampleGain
- std::vector< float > vSamplePitch
- std::vector< bool > vSampleLoop
- std::vector< float > vSampleAttDist
- std::vector< ve::dataContainer > m_vObj
- ve::vec6f observerPos
- ve::vec6f observerVel
- unsigned int observerId

12.20.1 Detailed Description

a class for 3D audio simulation based on OpenAL

Definition at line 24 of file veDeviceAudioAL.h.

12.20.2 Constructor & Destructor Documentation

12.20.2.1 ve::deviceAudioAL::deviceAudioAL (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor.

All initialization values are taken from the deviceAudio and scene definition sections of the xml statement.

Parameters:

ini a previously loaded xml ini file

iniSectionId (optional) specifies which sections of the xml statement are to be interpreted.

12.20.2.2 virtual ve::deviceAudioAL::~deviceAudioAL () [virtual]

destructor

12.20.3 Member Function Documentation

12.20.3.1 virtual int ve::deviceAudioAL::update (double *deltaT* = 0.0) [virtual]

updates device

Reimplemented from ve::device.

12.20.3.2 virtual int ve::deviceAudioAL::observer (unsigned int *objectId*) [virtual]

sets object id that determines the observer position, 0 means none. In case of success 0 is returned.

12.20.3.3 virtual int ve::deviceAudioAL::updateDevice (const ve::vec6f & *pos*, const ve::flag128 & *flags*) [protected, virtual]

sets output of device

In this case the listener position can be set.

Reimplemented from ve::device.

12.20.3.4 virtual int ve::deviceAudioAL::updateObject (const ve::dataContainer & *data*) [protected, virtual]

updates scene graph according to the information provided in the data container

Parameters:

data contains all data describing an object.

Returns:

0 in case of success.

Reimplemented from ve::device.

12.20.3.5 virtual int ve::deviceAudioAL::updateResource (const ve::dataChar & *data*) [protected, virtual]

updates a resource at runtime

Parameters:
data contains all data describing the resource.

Returns:
0 in case of success.

Reimplemented from ve::device.

12.20.4 Member Data Documentation

12.20.4.1 std::vector<unsigned int> ve::deviceAudioAL::vSample [protected]

stores internal ids of the sound samples

Definition at line 53 of file veDeviceAudioAL.h.

12.20.4.2 std::vector<unsigned int> ve::deviceAudioAL::vResId [protected]

stores external resource ids of the sound samples

Definition at line 55 of file veDeviceAudioAL.h.

12.20.4.3 std::vector<bool> ve::deviceAudioAL::vIsReference [protected]

stores whether the resource needs to be cleaned up or is only a reference

Definition at line 57 of file veDeviceAudioAL.h.

12.20.4.4 std::vector<float> ve::deviceAudioAL::vSampleGain [protected]

stores gain factor of samples

Definition at line 59 of file veDeviceAudioAL.h.

12.20.4.5 std::vector<float> ve::deviceAudioAL::vSamplePitch [protected]

stores pitch factor of samples

Definition at line 61 of file veDeviceAudioAL.h.

12.20.4.6 std::vector<bool> ve::deviceAudioAL::vSampleLoop [protected]

stores whether sample is played in an infinite loop
Definition at line 63 of file veDeviceAudioAL.h.

12.20.4.7 std::vector<float> ve::deviceAudioAL::vSampleAttDist [protected]

stores distances beyond which sources playing that sample get attenuated
Definition at line 65 of file veDeviceAudioAL.h.

12.20.4.8 std::vector<ve::dataContainer> ve::deviceAudioAL::m_vObj [protected]

vector for storing scene object data
Definition at line 68 of file veDeviceAudioAL.h.

12.20.4.9 ve::vec6f ve::deviceAudioAL::observerPos [protected]

stores listener (observer) position
Definition at line 71 of file veDeviceAudioAL.h.

12.20.4.10 ve::vec6f ve::deviceAudioAL::observerVel [protected]

stores listener (observer) velocity
Definition at line 73 of file veDeviceAudioAL.h.

12.20.4.11 unsigned int ve::deviceAudioAL::observerId [protected]

stores listener (observer) object id
Definition at line 75 of file veDeviceAudioAL.h.
The documentation for this class was generated from the following file:

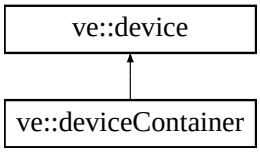
- veDeviceAudioAL.h

12.21 ve::deviceContainer Class Reference

The flexible device container.

`#include <veDeviceContainer.h>`

Inheritance diagram for ve::deviceContainer::



Public Member Functions

- deviceContainer ()
- deviceContainer (const ve::xmlIni &ini)
- deviceContainer (int argc, char **argv)
- virtual ~deviceContainer ()
- virtual int update (double deltaT=0.0)
- ve::xmlIni & ini ()
- virtual int getInput (ve::dataChar &data, unsigned int mask=DC_MASK_ALL)
- virtual int setOutput (const ve::dataChar &data)
- virtual int setOutput (ve::dataChar &data, unsigned int command)
- virtual int setOutput (ve::dataChar &data, unsigned int command, unsigned int receivers)
- unsigned int size () const
- ve::device * device (unsigned int n)
- int addDevice (ve::device *pDevice)
- int dropDevice (unsigned int n)
- int dropDevice (const ve::device *pDevice)
- int inputDevice (unsigned int n)

Protected Member Functions

- void initDevices ()

Protected Attributes

- ve::xmlIni m_ini
- std::vector< ve::device * > m_vDevice
- ve::device * m_pInput

12.21.1 Detailed Description

The flexible device container.

This class can be used as an abstract placeholder class for all sorts of devices (joystick, keyboard, mouse, network, ...) in the veLib. It just serves as a flexible intermediate interface, allowing to hide the hardware implementation details from simulation applications. The complete actual instanciation is defined in the xml-based ini file. So, ideally, switching the setup just means using a different ini file.

The ini-constructor parses the ini file for <deviceXYZ/> direct children tags. If it recognizes the device type, a corresponding device is instanciated. If the device has the attribute input="true", it is used as the input device.

Author:
gf

Revision
2.3

Definition at line 83 of file veDeviceContainer.h.

12.21.2 Constructor & Destructor Documentation

12.21.2.1 ve::deviceContainer::deviceContainer ()

default constructor creating an empty device container

12.21.2.2 ve::deviceContainer::deviceContainer (const ve::xmlIni & *ini*)

constructor from an xml ini file

Parameters:
ini the xml statement to be parsed

12.21.2.3 ve::deviceContainer::deviceContainer (int *argc*, char ** *argv*)

constructor interpreting the main(argc, argv) command line arguments

This is a convenience version of the standard constructor. The command line arguments are parsed for a -i(iniXYZ.xml) switch, if none is found, the default ini file name "ini.xml" in the applications' path is assumed and loaded.

Parameters:
argc the number of command line arguments
argv the pointer to the command line arguments

12.21.2.4 virtual ve::deviceContainer::~deviceContainer () [virtual]

destructor

12.21.3 Member Function Documentation

12.21.3.1 virtual int ve::deviceContainer::update (double *deltaT* = `0.0`) `[virtual]`

updates devices

Parameters:

deltaT (optional) the time elapsed since last call, in seconds.

Reimplemented from ve::device.

12.21.3.2 ve::xmlIni& ve::deviceContainer::ini () `[inline]`

allows access to the ini file

This method allows applications to parse the ini file themselves. It is especially useful in combination with the command line interpreting constructor, making additional ini file loading unnecessary.

Definition at line 109 of file veDeviceContainer.h.

References m_ini.

12.21.3.3 virtual int ve::deviceContainer::getInput (ve::dataChar & *data*, unsigned int *mask* = `DC_MASK_ALL`) `[virtual]`

reads input from the input device into a ve::dataContainer.

Parameters:

data the dataContainer that is overwritten

mask (optional) defines the data fields that are overwritten, a bitwise combination of DC_-MASK_XYZ constants, see veTypes.h for further info.

Returns:

the number of errors occured.

Reimplemented from ve::device.

12.21.3.4 virtual int ve::deviceContainer::setOutput (const ve::dataChar & *data*) `[virtual]`

transmits a new output to the devices.

Parameters:

data the data container that has to contain all necessary information.

Returns:

the number of errors occured.

Reimplemented from ve::device.

Referenced by setOutput().

12.21.3.5 virtual int ve::deviceContainer::setOutput (ve::dataChar & *data*, unsigned int *command*) [inline, virtual]

transmits a new output to the devices and defines a command.

Parameters:

data the data container that contains all necessary information

command defines the command that shall be executed. Use one of the ve::CMD_XYZ constants (veTypes.h). It is stored in the dataContainer!

Returns:

the number of errors occured.

Reimplemented from ve::device.

Definition at line 124 of file veDeviceContainer.h.

References ve::dataChar::command(), and setOutput().

12.21.3.6 virtual int ve::deviceContainer::setOutput (ve::dataChar & *data*, unsigned int *command*, unsigned int *receivers*) [virtual]

transmits a new output to selected devices, defines a command.

Parameters:

data the data container that contains all necessary information

command defines the command that shall be executed. Use one of the ve::CMD_XYZ constants (veTypes.h). It is stored in the dataContainer!

receivers defines a pattern of receiver devices. Use a combinatione of ve::DEV_XYZ constants (veTypes.h). It is stored in the dataContainer!

Returns:

the number of errors occured.

12.21.3.7 unsigned int ve::deviceContainer::size () const [inline]

returns number of devices

Definition at line 134 of file veDeviceContainer.h.

References m_vDevice.

12.21.3.8 ve::device* ve::deviceContainer::device (unsigned int *n*) [inline]

allows access to device number n

Definition at line 136 of file veDeviceContainer.h.

References m_vDevice.

12.21.3.9 int ve::deviceContainer::addDevice (ve::device * *pDevice*) [inline]

adds a device to the container
be aware that the device will be deleted when the deviceContainer constructor is called.
Definition at line 140 of file veDeviceContainer.h.
References m_vDevice.

12.21.3.10 int ve::deviceContainer::dropDevice (unsigned int *n*)

removes device n from the container
in this case the device's memory has to be deallocated by the application.

12.21.3.11 int ve::deviceContainer::dropDevice (const ve::device * *pDevice*)

removes device from the container
in this case the device's memory has to be deallocated by the application.

12.21.3.12 int ve::deviceContainer::inputDevice (unsigned int *n*) [inline]

sets input device
Definition at line 149 of file veDeviceContainer.h.

12.21.3.13 void ve::deviceContainer::initDevices () [protected]

initializes devices

12.21.4 Member Data Documentation

12.21.4.1 ve::xmlIni ve::deviceContainer::m_ini [protected]

stores the ini file
Definition at line 150 of file veDeviceContainer.h.
Referenced by ini().

12.21.4.2 std::vector<ve::device *> ve::deviceContainer::m_vDevice [protected]

stores pointers to the actually instanciated devices
Definition at line 157 of file veDeviceContainer.h.
Referenced by addDevice(), device(), and size().

12.21.4.3 ve::device* ve::deviceContainer::m_pInput [protected]

stores pointer to the (main) input device

Definition at line 159 of file veDeviceContainer.h.

The documentation for this class was generated from the following file:

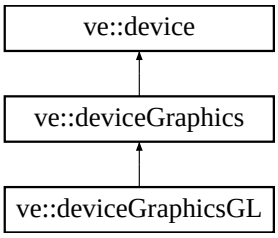
- veDeviceContainer.h

12.22 ve::deviceGraphics Class Reference

base class for 3D visualization classes.

`#include <veDevice.h>`

Inheritance diagram for ve::deviceGraphics::



Public Member Functions

- deviceGraphics (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~deviceGraphics ()
- ve::vec6f & camera ()
- ve::frustum & frustum ()
- float frustumLeft () const
- float frustumRight () const
- float frustumBottom () const
- float frustumTop () const
- float nearClipping () const
- float farClipping () const

Protected Member Functions

- virtual int draw ()=0
- virtual int updateDevice (const ve::vec6f &pos, const ve::flag128 &flags)

Protected Attributes

- ve::frustum m_frustum
- float m_frustumLeft
- float m_frustumRight
- float m_frustumBottom
- float m_frustumTop
- float m_nearClipping
- float m_farClipping
- ve::vec6f observerPos
- ve::vec6f observerVel
- unsigned int m_observerId

12.22.1 Detailed Description

base class for 3D visualization classes.

This is an abstract interface for different visualizations, e.g., scene graph toolkits, roundshot viewers. They allow to separate drawing functions from window handling.

Definition at line 219 of file veDevice.h.

12.22.2 Constructor & Destructor Documentation

12.22.2.1 ve::deviceGraphics::deviceGraphics (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor

12.22.2.2 virtual ve::deviceGraphics::~deviceGraphics () [virtual]

destructor

12.22.3 Member Function Documentation

12.22.3.1 ve::vec6f& ve::deviceGraphics::camera () [inline]

allows direct access to camera

Definition at line 227 of file veDevice.h.

References observerPos.

12.22.3.2 ve::frustum& ve::deviceGraphics::frustum () [inline]

allows direct access to frustum

Definition at line 229 of file veDevice.h.

References m_frustum.

12.22.3.3 float ve::deviceGraphics::frustumLeft () const [inline]

returns left frustum clipping distance

Definition at line 231 of file veDevice.h.

References m_frustumLeft.

12.22.3.4 float ve::deviceGraphics::frustumRight () const [inline]

returns right frustum clipping distance.

Definition at line 233 of file veDevice.h.

References m_frustumRight.

12.22.3.5 float ve::deviceGraphics::frustumBottom () const [inline]

returns bottom frustum clipping distance.

Definition at line 235 of file veDevice.h.

References m_frustumBottom.

12.22.3.6 float ve::deviceGraphics::frustumTop () const [inline]

returns top frustum clipping distance.

Definition at line 237 of file veDevice.h.

References m_frustumTop.

12.22.3.7 float ve::deviceGraphics::nearClipping () const [inline]

returns distance to near clipping plane.

Definition at line 239 of file veDevice.h.

References m_nearClipping.

12.22.3.8 float ve::deviceGraphics::farClipping () const [inline]

returns distance to far clipping plane.

Definition at line 241 of file veDevice.h.

References m_farClipping.

12.22.3.9 virtual int ve::deviceGraphics::draw () [protected, pure virtual]

draws scene, interface definition

Implemented in ve::deviceGraphicsGL.

12.22.3.10 virtual int ve::deviceGraphics::updateDevice (const ve::vec6f & *pos*, const ve::flag128 & *flags*) [protected, virtual]

sets output

Reimplemented from ve::device.

12.22.4 Member Data Documentation

12.22.4.1 ve::frustum ve::deviceGraphics::m_frustum [protected]

stores frustum

Definition at line 250 of file veDevice.h.

Referenced by frustum().

12.22.4.2 float ve::deviceGraphics::m_frustumLeft [protected]

left frustum clipping distance.

Definition at line 252 of file veDevice.h.

Referenced by frustumLeft().

12.22.4.3 float ve::deviceGraphics::m_frustumRight [protected]

right frustum clipping distance.

Definition at line 254 of file veDevice.h.

Referenced by frustumRight().

12.22.4.4 float ve::deviceGraphics::m_frustumBottom [protected]

bottom frustum clipping distance.

Definition at line 256 of file veDevice.h.

Referenced by frustumBottom().

12.22.4.5 float ve::deviceGraphics::m_frustumTop [protected]

top frustum clipping distance.

Definition at line 258 of file veDevice.h.

Referenced by frustumTop().

12.22.4.6 float ve::deviceGraphics::m_nearClipping [protected]

distance to near clipping plane.

Definition at line 260 of file veDevice.h.

Referenced by nearClipping().

12.22.4.7 float ve::deviceGraphics::m_farClipping [protected]

distance to far clipping plane.

Definition at line 262 of file veDevice.h.

Referenced by farClipping().

12.22.4.8 ve::vec6f ve::deviceGraphics::observerPos [protected]

stores camera (observer) position

Definition at line 265 of file veDevice.h.

Referenced by camera().

12.22.4.9 ve::vec6f ve::deviceGraphics::observerVel [protected]

stores camera (observer) velocity

Definition at line 267 of file veDevice.h.

12.22.4.10 unsigned int ve::deviceGraphics::m_observerId [protected]

stores camera (observer) object id

Definition at line 269 of file veDevice.h.

The documentation for this class was generated from the following file:

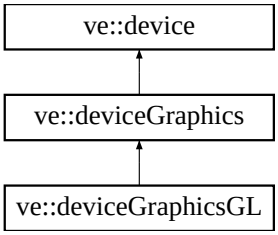
- veDevice.h

12.23 ve::deviceGraphicsGL Class Reference

OpenGL based graphics device.

`#include <veDeviceGraphicsGL.h>`

Inheritance diagram for ve::deviceGraphicsGL::



Public Member Functions

- deviceGraphicsGL (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~deviceGraphicsGL ()
- virtual int update (double deltaT=0.0)
- unsigned int addModel (ve::geoObj *pModel, unsigned int id=0)
- ve::geoObj * model (unsigned int n)
- unsigned int nModels () const
- unsigned int addObject (ve::geoObj *glObject)
- virtual void clear ()
- unsigned int nObjects () const
- ve::geoObj * object (unsigned int n)
- unsigned int nObjToDraw () const
- ve::xml xml () const
- int background (unsigned int classId, const ve::vec6f &pos=ve::vec6f(0, 0, 0))
- const ve::vec4f & lightPos (unsigned int n)

Protected Member Functions

- virtual int updateObject (const ve::dataContainer &data)
- virtual int updateResource (const ve::dataChar &data)
- virtual int draw ()
- void drawOpaqueObjects (ve::geoObj *pObj, const ve::vec6f &pos, std::vector< ve::_model-Ref > &vTransp)

Protected Attributes

- std::vector< ve::geoObj * > vModel
- std::vector< unsigned int > vResId
- std::vector< bool > vIsReference
- std::vector< ve::geoObj * > m_vScene
- std::vector< ve::dataContainer > m_vObj
- double tNow

- unsigned int objToDraw
- bool m_isFog
- unsigned int lightActive
- ve::vec4f m_lightPos [nLightMax]
- ve::geoObj * pBackgr
- ve::vec6f backgrPos
- ve::vec6f camOffset

Static Protected Attributes

- static const unsigned int nLightMax = 8

12.23.1 Detailed Description

OpenGL based graphics device.

This class implements a simple OpenGL based visualization device that does not depend on any other libraries outside of veLib. The loading and processing of predefined models (e.g., VRML, X3D, 3ds) is done via the ve::geoObj classes. Additional loaders can be added via an elegant plugin mechanism (see ve::geoFileHandler). The resource and camera concept is designed to be suitable as a remote network visualization server controlled by few standardized messages.

Definition at line 121 of file veDeviceGraphicsGL.h.

12.23.2 Constructor & Destructor Documentation

12.23.2.1 ve::deviceGraphicsGL::deviceGraphicsGL (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor

All initialization values are taken from the camera and scene definition sections of an xml statement.

Parameters:

ini a previously loaded xml ini file

iniSectionId (optional) specifies which scene sections are to be interpreted.

12.23.2.2 virtual ve::deviceGraphicsGL::~deviceGraphicsGL () [virtual]

destructor

12.23.3 Member Function Documentation

12.23.3.1 virtual int ve::deviceGraphicsGL::update (double *deltaT* = 0.0) [virtual]

updates scene and performs a redraw

Reimplemented from ve::device.

12.23.3.2 unsigned int ve::deviceGraphicsGL::addModel (ve::geoObj * *pModel*, unsigned int *id* = 0)

adds a model definition to the model vector.

Returns:
model's classId. Memory deallocation is left to the user.

12.23.3.3 ve::geoObj* ve::deviceGraphicsGL::model (unsigned int *n*) [inline]

allows access to member n of the model resource vector

Definition at line 140 of file veDeviceGraphicsGL.h.

References vModel.

12.23.3.4 unsigned int ve::deviceGraphicsGL::nModels () const [inline]

returns number of model resources

Definition at line 143 of file veDeviceGraphicsGL.h.

References vModel.

12.23.3.5 unsigned int ve::deviceGraphicsGL::addObject (ve::geoObj * *glObject*)

adds a model definition and object instance in one step.

This is mainly a convenience method for very simple scenes. It should not be used for dynamic scenes and scenes having multiple instances of the same model.

Returns:
the instance's object id.

12.23.3.6 virtual void ve::deviceGraphicsGL::clear () [virtual]

removes all object instances.

12.23.3.7 unsigned int ve::deviceGraphicsGL::nObjects () const [inline]

returns number of object instances

Definition at line 154 of file veDeviceGraphicsGL.h.

References m_vScene.

12.23.3.8 ve::geoObj* ve::deviceGraphicsGL::object (unsigned int *n*) [inline]

allows access to member n of the scene object instance vector

Definition at line 156 of file veDeviceGraphicsGL.h.

References m_vScene.

12.23.3.9 unsigned int ve::deviceGraphicsGL::nObjToDraw () const [inline]

returns number of objects that currently have to be drawn

Definition at line 159 of file veDeviceGraphicsGL.h.

References objToDraw.

12.23.3.10 ve::xml ve::deviceGraphicsGL::xml () const

returns the current scene as xml statement

12.23.3.11 int ve::deviceGraphicsGL::background (unsigned int *classId*, const ve::vec6f & *pos* = ve::vec6f(0, 0, 0))

adds a model instance as background to the scene.

It is identified by its classId. Background objects are not affected by lighting and translations of the camera. Currently only 1 backround is allowed at one time. Use classId 0 to unregister the current background.

Returns:
0 in case of success.

12.23.3.12 const ve::vec4f& ve::deviceGraphicsGL::lightPos (unsigned int *n*) [inline]

returns position of light source number n

Definition at line 171 of file veDeviceGraphicsGL.h.

References m_lightPos.

12.23.3.13 virtual int ve::deviceGraphicsGL::updateObject (const ve::dataContainer & *data*) [protected, virtual]

updates scene graph according to the information provided in the data container

Parameters:
data contains all data describing an object.

Returns:
0 in case of success.

Reimplemented from ve::device.

12.23.3.14 virtual int ve::deviceGraphicsGL::updateResource (const ve::dataChar & *data*) [protected, virtual]

updates a resource at runtime

Parameters:
data contains all data describing the resource.

Returns:
0 in case of success.

Reimplemented from ve::device.

12.23.3.15 virtual int ve::deviceGraphicsGL::draw () [protected, virtual]

draws scene

Implements ve::deviceGraphics.

12.23.3.16 void ve::deviceGraphicsGL::drawOpaqueObjects (ve::geoObj * *pObj*, const ve::vec6f & *pos*, std::vector< ve::_modelRef > & *vTransp*) [protected]

a helper method for recursively traversing scene trees

12.23.4 Member Data Documentation

12.23.4.1 std::vector<ve::geoObj*> ve::deviceGraphicsGL::vModel [protected]

vector for pointers to glObj model classes

Definition at line 189 of file veDeviceGraphicsGL.h.

Referenced by model(), and nModels().

12.23.4.2 std::vector<unsigned int> ve::deviceGraphicsGL::vResId [protected]

vector for storing the model ids

Definition at line 191 of file veDeviceGraphicsGL.h.

12.23.4.3 std::vector<bool> ve::deviceGraphicsGL::vIsReference [protected]

stores whether the resource needs to be cleaned up or is only a reference

Definition at line 193 of file veDeviceGraphicsGL.h.

12.23.4.4 std::vector<ve::geoObj*> ve::deviceGraphicsGL::m_vScene [protected]

vector for pointers to the current scene object instances

Definition at line 196 of file veDeviceGraphicsGL.h.

Referenced by nObjects(), and object().

12.23.4.5 std::vector<ve::dataContainer> ve::deviceGraphicsGL::m_vObj [protected]

vector for storing scene object data

Definition at line 198 of file veDeviceGraphicsGL.h.

12.23.4.6 double ve::deviceGraphicsGL::tNow [protected]

stores current time stamp

Definition at line 201 of file veDeviceGraphicsGL.h.

12.23.4.7 unsigned int ve::deviceGraphicsGL::objToDraw [protected]

stores number of objects that currently have to be drawn

Definition at line 203 of file veDeviceGraphicsGL.h.

Referenced by nObjToDraw().

12.23.4.8 bool ve::deviceGraphicsGL::m_isFog [protected]

stores fog state

Definition at line 205 of file veDeviceGraphicsGL.h.

12.23.4.9 const unsigned int ve::deviceGraphicsGL::nLightMax = 8 [static, protected]

defines maximum number of lights in the scene

Definition at line 208 of file veDeviceGraphicsGL.h.

12.23.4.10 unsigned int ve::deviceGraphicsGL::lightActive [protected]

stores bit flags of active lights in the scene

Definition at line 210 of file veDeviceGraphicsGL.h.

12.23.4.11 ve::vec4f ve::deviceGraphicsGL::m_lightPos[nLightMax] [protected]

stores position of lights

Definition at line 212 of file veDeviceGraphicsGL.h.

Referenced by lightPos().

12.23.4.12 ve::geoObj* ve::deviceGraphicsGL::pBackgr [protected]

stores pointer to background model

Definition at line 215 of file veDeviceGraphicsGL.h.

12.23.4.13 ve::vec6f ve::deviceGraphicsGL::backgrPos [protected]

stores relative position of background model

Definition at line 217 of file veDeviceGraphicsGL.h.

12.23.4.14 ve::vec6f ve::deviceGraphicsGL::camOffset [protected]

stores camera offset

Definition at line 220 of file veDeviceGraphicsGL.h.

The documentation for this class was generated from the following file:

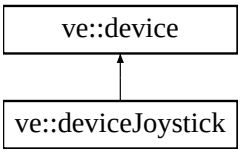
- veDeviceGraphicsGL.h

12.24 ve::deviceJoystick Class Reference

class for joystick input.

`#include <veDeviceSDL.h>`

Inheritance diagram for ve::deviceJoystick::



Public Member Functions

- deviceJoystick (unsigned int joystickNumber=0)
- deviceJoystick (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~deviceJoystick ()

Protected Member Functions

- virtual int queryDevice (ve::vec6f &axes, ve::flag128 &flags)
- int init (unsigned int joystickNumber)

Protected Attributes

- SDL_Joystick * pJoy
- unsigned int nButtons
- unsigned int nAxes

12.24.1 Detailed Description

class for joystick input.

This class handles standard joystick input in a system-independent way. The current implementation is based on libSDL. For advanced features as force feedback, veDeviceJoystickDirectX is recommended.

Definition at line 200 of file veDeviceSDL.h.

12.24.2 Constructor & Destructor Documentation

12.24.2.1 ve::deviceJoystick::deviceJoystick (unsigned int *joystickNumber* = `0`)

default constructor

12.24.2.2 ve::deviceJoystick::deviceJoystick (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor reading initialization values from an xml statement

12.24.2.3 virtual ve::deviceJoystick::~deviceJoystick () [virtual]

destructor

12.24.3 Member Function Documentation

12.24.3.1 virtual int ve::deviceJoystick::queryDevice (ve::vec6f & *axes*, ve::flag128 & *flags*) [protected, virtual]

gets current input state of device

Reimplemented from ve::device.

12.24.3.2 int ve::deviceJoystick::init (unsigned int *joystickNumber*) [protected]

initializes SDL joystick subsystem, and, if necessary, SDL itself.

12.24.4 Member Data Documentation

12.24.4.1 SDL_Joystick* ve::deviceJoystick::pJoy [protected]

pointer to SDL joystick

Definition at line 215 of file veDeviceSDL.h.

12.24.4.2 unsigned int ve::deviceJoystick::nButtons [protected]

stores number of buttons

Definition at line 217 of file veDeviceSDL.h.

12.24.4.3 unsigned int ve::deviceJoystick::nAxes [protected]

stores number of axes

Definition at line 219 of file veDeviceSDL.h.

The documentation for this class was generated from the following file:

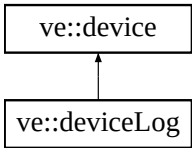
- veDeviceSDL.h

12.25 ve::deviceLog Class Reference

A (pseudo-)device that writes its traffic into log files or to stdout.

`#include <veDeviceContainer.h>`

Inheritance diagram for ve::deviceLog::



Public Member Functions

- deviceLog (const std::string &logFileName="")
- virtual ~deviceLog ()
- virtual int getInput (ve::dataChar &data, unsigned int mask=DC_MASK_ALL)
- virtual int setOutput (const ve::dataChar &data)
- virtual int setOutput (ve::dataChar &data, unsigned int command)

Protected Attributes

- bool m_writeToFile
- std::ofstream m_file

12.25.1 Detailed Description

A (pseudo-)device that writes its traffic into log files or to stdout.

This class can be used for debug and documentation purposes.

Author:
gf

Revision
2.3

Definition at line 32 of file veDeviceContainer.h.

12.25.2 Constructor & Destructor Documentation

12.25.2.1 ve::deviceLog::deviceLog (const std::string & *logFileName* = "")

constructor writing to logfile or stdout if no filename is provided

12.25.2.2 virtual ve::deviceLog::~deviceLog () [virtual]

destructor

12.25.3 Member Function Documentation

12.25.3.1 virtual int ve::deviceLog::getInput (ve::dataChar & *data*, unsigned int *mask* = DC_MASK_ALL) [virtual]

reads input from the input device into a ve::dataContainer.

Parameters:

data the dataContainer that is overwritten

mask (optional) defines the data fields that are overwritten, a bitwise combination of DC_-MASK_XYZ constants, see veTypes.h for further info.

Returns:

the number of errors occured.

Reimplemented from ve::device.

12.25.3.2 virtual int ve::deviceLog::setOutput (const ve::dataChar & *data*) [virtual]

transmits a new output to the devices.

Parameters:

data the data container that has to contain all necessary information.

Returns:

the number of errors occured.

Reimplemented from ve::device.

12.25.3.3 virtual int ve::deviceLog::setOutput (ve::dataChar & *data*, unsigned int *command*) [inline, virtual]

transmits a new output to the devices and defines a command.

Parameters:

data the data container that contains all necessary information

command defines the command that shall be executed. Use one of the ve::CMD_XYZ constants (veTypes.h). It is stored in the dataContainer!

Returns:

the number of errors occured.

Reimplemented from ve::device.

Definition at line 53 of file veDeviceContainer.h.

12.25.4 Member Data Documentation

12.25.4.1 bool ve::deviceLog::m_writeToFile [protected]

stores stream target, false=stdout, true=logfile

Definition at line 54 of file veDeviceContainer.h.

12.25.4.2 std::ofstream ve::deviceLog::m_file [protected]

log file stream

Definition at line 59 of file veDeviceContainer.h.

The documentation for this class was generated from the following file:

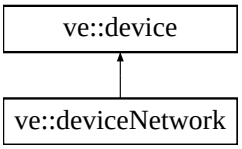
- veDeviceContainer.h

12.26 ve::deviceNetwork Class Reference

Network device class using pure UDP.

`#include <veDeviceNetwork.h>`

Inheritance diagram for ve::deviceNetwork::



Public Member Functions

- deviceNetwork (unsigned int mode=CLIENT)
- deviceNetwork (ve::xmlIni &ini, unsigned int iniSectionId=0, unsigned int mode=CLIENT)
- virtual ~deviceNetwork ()
- void startServer (int port)
- void connectToServer (int port, std::string host, bool autoReconnect=false)
- virtual int observer (unsigned int objectId)
- virtual int setOutput (const ve::dataChar &data)
- virtual int setOutput (ve::dataChar &data, unsigned int command)
- virtual void setOutputDelay (double delay)
- virtual int getInput (std::vector< ve::dataChar > &dataContainers, unsigned int receiverDevice=DEV_ALL, unsigned int receiverId=DEV_ALL)
- virtual int getInput (ve::dataChar &data, unsigned int mask=DC_MASK_ALL)
- virtual int getInput (ve::dataChar &data, unsigned int mask, unsigned int receiverDevice=DEV_ALL, unsigned int receiverId=DEV_ALL)
- void debugOutput (bool b)
- void useMulticast (bool b)
- int getConnectionStatus (unsigned int port, std::string host)

Protected Member Functions

- int handleData ()
- void updateConnections (ve::dataChar dataRecv, struct sockaddr_in sender, double deltaT)
- void checkForTimeouts (double deltaT)
- void updateData (ve::dataChar dataRecv, struct sockaddr_in sender)
- void predictMotion (ve::dataChar &data, double clientTime)
- double updateClientTime ()
- int sendData (const ve::dataChar &data)

Static Protected Member Functions

- static int runNetworkLoop (void *param)

Protected Attributes

- networkMode m_mode
- unsigned int m_numConnected
- connectionInfo m_connections [ve::NETWORK_MAX_CONNECTIONS]
- std::vector< ve::dataChar > m_redundantDataRecv
- std::vector< ve::dataChar > m_redundantDataSend
- std::vector< mandatoryData > m_mandatoryDataRecv
- std::vector< mandatoryData > m_mandatoryDataSend
- ve::dataChar m_observer
- unsigned int m_observerId
- bool m_output
- SDL_Thread * m_nwThread
- SDL_mutex * m_mutRecv
- SDL_mutex * m_mutSend
- int m_socket
- char m_inBuf [ve::NETWORK_BUFSIZE]
- char m_outBuf [ve::NETWORK_BUFSIZE]
- bool m_bPredictMotion
- in_addr m_mcGroupAddress
- bool m_bMulticast
- unsigned int m_serverPort
- std::vector< std::string > m_acceptClients
- double m_pingSendTime
- double m_pingRecvTime
- double m_clientTime
- std::list< double > m_roundTripTimes
- std::vector< int > m_connectPorts
- std::vector< std::string > m_connectHosts
- bool m_bAutoReconnect
- double m_outputDelay
- bool m_bDebugOutput
- std::stack< delayedData * > m_delayedDataPool
- std::queue< delayedData * > m_delayedDataQueue
- ve::chrono m_timer

Friends

- class ve::dataChar

12.26.1 Detailed Description

Network device class using pure UDP.

This class handles UDP network connections and data transfer, based on BSD sockets.

Author:
weyel

Definition at line 129 of file veDeviceNetwork.h.

12.26.2 Constructor & Destructor Documentation

12.26.2.1 ve::deviceNetwork::deviceNetwork (unsigned int *mode* = CLIENT)

constructor Makes all initialisations, but does not connect automatically. You have to call startServer() or connectToServer() resp. yourself.

12.26.2.2 ve::deviceNetwork::deviceNetwork (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0, unsigned int *mode* = CLIENT)

constructor Makes all initialisations. Based on mode (CLIENT or SERVER), it automatically opens a server port or tries to connect a client to a server if the necessary values are provided in the ini file.

12.26.2.3 virtual ve::deviceNetwork::~deviceNetwork () [virtual]

destructor

12.26.3 Member Function Documentation

12.26.3.1 void ve::deviceNetwork::startServer (int *port*)

start a server that listens on port for incoming data packets

12.26.3.2 void ve::deviceNetwork::connectToServer (int *port*, std::string *host*, bool *autoReconnect* = false)

client tries to connect on port to host

12.26.3.3 virtual int ve::deviceNetwork::observer (unsigned int *objectId*) [inline, virtual]

sets object id that determines the observer position, 0 means none. In case of success 0 is returned.

Definition at line 155 of file veDeviceNetwork.h.

References m_observerId.

12.26.3.4 virtual int ve::deviceNetwork::setOutput (const ve::dataChar & *data*) [virtual]

send a data container over the network

Reimplemented from ve::device.

12.26.3.5 virtual int ve::deviceNetwork::setOutput (ve::dataChar & *data*, unsigned int *command*) [virtual]

transfers a data container over the network and defines a command.

Parameters:

data the data container that contains all necessary information

command defines the command that shall be executed. Use one of the ve::CMD_XYZ constants (veTypes.h). It is stored in the dataContainer!

Returns:

0 in case of success.

Reimplemented from ve::device.

12.26.3.6 virtual void ve::deviceNetwork::setOutputDelay (double *delay*) [virtual]

sets a delay for all data sent using the connection

Parameters:

delay the delay in second.

12.26.3.7 virtual int ve::deviceNetwork::getInput (std::vector< ve::dataChar > & *dataContainers*, unsigned int *receiverDevice* = DEV_ALL, unsigned int *receiverId* = DEV_ALL) [virtual]

gives back all data containers that have been received and that have a matching receiverId and receiverDevice

Parameters:

dataContainers an empty vector, which will contain the received containers after the function returns. If motionPrediction is disabled (see veXml.pdf), this function will only return those containers that have been received from the network (may be none at all). If motionPrediction is enabled, it will always return at least the "observer"-data container, that is the last received container with objectId==m_observerId (0 by default, see observer()- function). Based on the synchronized network time and the position and velocity values of that container, a motion prediction for a new position is performed before the function returns, meaning you will always get an updated container, even if no new one has been received from the network. This can be used to overcome network lags and to produce smooth outputs for display servers. Of course, with or without prediction enabled, only data containers whose receiverId and receiverDevice match with the params are returned.

receiverId only containers with matching receiverId are returned

receiverDevice only containers with matching receiverDevice are returned

Returns:

the number of containers in dataContainers

Referenced by getInput().

12.26.3.8 virtual int ve::deviceNetwork::getInput (ve::dataChar & *data*, unsigned int *mask* = DC_MASK_ALL) [inline, virtual]

gives back only one data container at a time

Parameters:

data will contain a received container after the function returns. If motionPrediction is disabled (see veXml.pdf), this function will only return a container if at least one has been received from the network, otherwise data will be given back unchanged. If motionPrediction is enabled, it will always return at least the "observer"-data container, that is the last received container with objectId==m_observerId (0 by default, see observer()-function). Based on the synchronized network time and the position and velocity values of that container, a motion prediction for a new position is performed before the function returns, meaning you will always get an updated container, even if no new one has been received from the network. This can be used to overcome network lags and to produce smooth outputs for display servers.

mask defines the data fields that are overwritten, a bitwise combination of DC_MASK_XYZ constants, see veTypes.h for further info.

Returns:

the number of remaining data containers

Reimplemented from ve::device.

Definition at line 198 of file veDeviceNetwork.h.

References ve::DEV_ALL, and getInput().

12.26.3.9 virtual int ve::deviceNetwork::getInput (ve::dataChar & *data*, unsigned int *mask*, unsigned int *receiverDevice* = DEV_ALL, unsigned int *receiverId* = DEV_ALL) [virtual]

gives back only one data container at a time

Parameters:

data will contain a received container after the function returns. If motionPrediction is disabled (see veXml.pdf), this function will only return a container if at least one has been received from the network, otherwise data will be given back unchanged. If motionPrediction is enabled, it will always return at least the "observer"-data container, that is the last received container with objectId==m_observerId (0 by default, see observer()-function). Based on the synchronized network time and the position and velocity values of that container, a motion prediction for a new position is performed before the function returns, meaning you will always get an updated container, even if no new one has been received from the network. This can be used to overcome network lags and to produce smooth outputs for display servers. Of course, with or without prediction enabled, only data containers whose receiverId and receiverDevice match with the params are returned.

mask defines the data fields that are overwritten, a bitwise combination of DC_MASK_XYZ constants, see veTypes.h for further info.

receiverId only containers with matching receiverId are returned

receiverDevice only containers with matching receiverDevice are returned

Returns:

the number of remaining data containers

12.26.3.10 void ve::deviceNetwork::debugOutput (bool *b*) [inline]

Sets debug output flag.

When set to true, the veDeviceNetwork class will output basic information during connections and disconnections, and only error messages when set to false. Default is false.

Definition at line 226 of file veDeviceNetwork.h.

References m_bDebugOutput.

12.26.3.11 void ve::deviceNetwork::useMulticast (bool *b*) [inline]

Sets multicast flag.

When set to true, a veDeviceNetwork client will try to send data to servers on a multicast address. This can reduce network traffic, because each packet has to be send only once instead of to each server explicitly. However, network switches may be configured not to allow multicasting. Set this to false then. Currectly only works for clients.

Definition at line 232 of file veDeviceNetwork.h.

References m_bMulticast.

12.26.3.12 int ve::deviceNetwork::getConnectionStatus (unsigned int *port*, std::string *host*)

return the status of a connection

Parameters:

port the port that is used by this connection

host the hostname of the remote machiene to which the connection status should be returned

Returns:

ERR_OK, if the connection to host on port has been established and is working.
ERR_DEVICE_BUSY, if connection is not yet working, but trying to connect
ERR_DEVICE_NOT_WORKING, if host has been found but no connection is running on port
ERR_ERROR, if no matching connection has been found.

12.26.3.13 static int ve::deviceNetwork::runNetworkLoop (void * *param*) [static, protected]

runs the network loop after the network thread has been started

12.26.3.14 int ve::deviceNetwork::handleData () [protected]

checks for incoming data and calls functions accordingly, sends outgoing data

12.26.3.15 void ve::deviceNetwork::updateConnections (ve::dataChar *dataRecv*, struct sockaddr_in *sender*, double *deltaT*) [protected]

handles connection requests and ping messages, called by handle data

12.26.3.16 void ve::deviceNetwork::checkForTimeouts (double *deltaT*) [protected]

checks for connection losses and lost data, called by handle data

12.26.3.17 void ve::deviceNetwork::updateData (ve::dataChar *dataRecv*, struct sockaddr_in *sender*) [protected]

handles "ordinary" data containers, called by handleData

12.26.3.18 void ve::deviceNetwork::predictMotion (ve::dataChar & *data*, double *clientTime*) [protected]

when the device is in server mode, this function does a prediction on incoming data and elapsed time, using a synchronized time base from the client

12.26.3.19 double ve::deviceNetwork::updateClientTime () [protected]

when in server mode, this returns the current remote client time estimate

12.26.3.20 int ve::deviceNetwork::sendData (const ve::dataChar & *data*) [protected]

actually sends the data

12.26.4 Member Data Documentation

12.26.4.1 networkMode ve::deviceNetwork::m_mode [protected]

device runs in client or server mode

Definition at line 248 of file veDeviceNetwork.h.

12.26.4.2 unsigned int ve::deviceNetwork::m_numConnected [protected]

the number of connections

Definition at line 250 of file veDeviceNetwork.h.

12.26.4.3 connectionInfo ve::deviceNetwork::m_connections[ve::NETWORK_MAX_CONNECTIONS] [protected]

info structs on all possible connections

Definition at line 252 of file veDeviceNetwork.h.

12.26.4.4 std::vector<ve::dataChar> ve::deviceNetwork::m_redundantDataRecv [protected]

a buffer which holds redundant data that was last received

Definition at line 254 of file veDeviceNetwork.h.

12.26.4.5 std::vector<ve::dataChar> ve::deviceNetwork::m_redundantDataSend [protected]

a buffer which holds redundant data that is to be send

Definition at line 256 of file veDeviceNetwork.h.

12.26.4.6 std::vector<mandatoryData> ve::deviceNetwork::m_mandatoryDataRecv [protected]

a buffer that holds mandatory data that was received;

Definition at line 258 of file veDeviceNetwork.h.

12.26.4.7 std::vector<mandatoryData> ve::deviceNetwork::m_mandatoryDataSend [protected]

a buffer that holds mandatory data that is to be send;

Definition at line 260 of file veDeviceNetwork.h.

12.26.4.8 ve::dataChar ve::deviceNetwork::m_observer [protected]

the data container that represents the observer

Definition at line 262 of file veDeviceNetwork.h.

12.26.4.9 unsigned int ve::deviceNetwork::m_observerId [protected]

the id of the observer container

Definition at line 264 of file veDeviceNetwork.h.

Referenced by observer().

12.26.4.10 bool ve::deviceNetwork::m_output [protected]

is there new data to be send?

Definition at line 266 of file veDeviceNetwork.h.

12.26.4.11 SDL_Thread* ve::deviceNetwork::m_nwThread [protected]

thread for doing network IO

Definition at line 268 of file veDeviceNetwork.h.

12.26.4.12 SDL_mutex* ve::deviceNetwork::m_mutRecv [protected]

mutex for thread sync on received data

Definition at line 270 of file veDeviceNetwork.h.

12.26.4.13 SDL_mutex* ve::deviceNetwork::m_mutSend [protected]

mutex for thread sync on data that is to be send

Definition at line 272 of file veDeviceNetwork.h.

12.26.4.14 int ve::deviceNetwork::m_socket [protected]

the socket for sending/receiving data

Definition at line 277 of file veDeviceNetwork.h.

12.26.4.15 char ve::deviceNetwork::m_inBuf[ve::NETWORK_BUFSIZE] [protected]

incoming data buffer

Definition at line 280 of file veDeviceNetwork.h.

12.26.4.16 char ve::deviceNetwork::m_outBuf[ve::NETWORK_BUFSIZE] [protected]

outgoing data buffer

Definition at line 282 of file veDeviceNetwork.h.

12.26.4.17 bool ve::deviceNetwork::m_bPredictMotion [protected]

enable/disable motion prediction

Definition at line 284 of file veDeviceNetwork.h.

12.26.4.18 struct in_addr ve::deviceNetwork::m_mcGroupAddress [protected]

the address of the multicast group

Definition at line 286 of file veDeviceNetwork.h.

12.26.4.19 bool ve::deviceNetwork::m_bMulticast [protected]

use multicasts?

Definition at line 288 of file veDeviceNetwork.h.

Referenced by useMulticast().

12.26.4.20 unsigned int ve::deviceNetwork::m_serverPort [protected]

the port on which the server listens for incoming packets (SERVER mode only)

Definition at line 292 of file veDeviceNetwork.h.

12.26.4.21 std::vector< std::string > ve::deviceNetwork::m_acceptClients [protected]

which clients are allowed to connect (all if acceptClients.size() == 0), NOT YET IMPLEMENTED

Definition at line 294 of file veDeviceNetwork.h.

12.26.4.22 double ve::deviceNetwork::m_pingSendTime [protected]

variables for determining network lag and doing motion prediction

Definition at line 296 of file veDeviceNetwork.h.

12.26.4.23 double ve::deviceNetwork::m_pingRecvTime [protected]

variables for determining network lag and doing motion prediction

Definition at line 298 of file veDeviceNetwork.h.

12.26.4.24 double ve::deviceNetwork::m_clientTime [protected]

the servers estimate of the client time

Definition at line 300 of file veDeviceNetwork.h.

12.26.4.25 std::list<double> ve::deviceNetwork::m_roundTripTimes [protected]

the roundtrip times of the last 32 pings

Definition at line 302 of file veDeviceNetwork.h.

12.26.4.26 std::vector<int> ve::deviceNetwork::m_connectPorts [protected]

vector of ports on which the client tries connects to servers

Definition at line 307 of file veDeviceNetwork.h.

12.26.4.27 std::vector<std::string> ve::deviceNetwork::m_connectHosts [protected]

vector of hostnames to which the client tries to connect

Definition at line 309 of file veDeviceNetwork.h.

12.26.4.28 bool ve::deviceNetwork::m_bAutoReconnect [protected]

client will automatically try to reconnect to a server if set to true

Definition at line 311 of file veDeviceNetwork.h.

12.26.4.29 double ve::deviceNetwork::m_outputDelay [protected]

delay applied to data sent from the connection

Definition at line 313 of file veDeviceNetwork.h.

12.26.4.30 bool ve::deviceNetwork::m_bDebugOutput [protected]

Stores debug output.

Definition at line 315 of file veDeviceNetwork.h.

Referenced by debugOutput().

12.26.4.31 std::stack<delayedData*> ve::deviceNetwork::m_delayedDataPool [protected]

Stack of delayedData structures, used for delayed data (avoids runtime memory allocations).

Definition at line 317 of file veDeviceNetwork.h.

12.26.4.32 std::queue<delayedData*> ve::deviceNetwork::m_delayedDataQueue [protected]

Queue of delayedData structures, waiting for the right time to be sent.

Definition at line 319 of file veDeviceNetwork.h.

12.26.4.33 ve::chrono ve::deviceNetwork::m_timer [protected]

Timer needed for delayed data.

Definition at line 321 of file veDeviceNetwork.h.

The documentation for this class was generated from the following file:

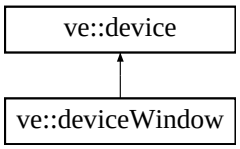
- veDeviceNetwork.h

12.27 ve::deviceWindow Class Reference

class for window handling, keyboard and mouse input.

`#include <veDeviceSDL.h>`

Inheritance diagram for ve::deviceWindow::



Public Member Functions

- deviceWindow (ve::xmlIni &ini, unsigned int iniSectionId=0)
- virtual ~deviceWindow ()
- virtual int update (double deltaT=0.0)
- int w () const
- int h () const
- Window windowHandle () const
- GLXContext glContext () const
- virtual int setVar (const ve::dataChar &data)
- void clearOverlay ()
- const ve::vec4f & fgNormalColor () const
- const ve::vec4f & bgNormalColor () const
- const ve::vec4f & fgSelectColor () const
- const ve::vec4f & bgSelectColor () const
- ve::glText * textRenderer (unsigned int id)
- const ve::flag128 & inputState () const
- float screenX (float x) const
- float screenY (float y) const
- float ovlX (float x) const
- float ovlY (float y) const
- float minX () const
- float minY () const
- float maxX () const
- float maxY () const

Static Public Attributes

- static ve::deviceWindow * window

Protected Member Functions

- virtual int queryDevice (ve::vec6f &axes, ve::flag128 &flags)
- virtual int updateDevice (const ve::vec6f &pos, const ve::flag128 &flags)
- virtual int updateObject (const ve::dataContainer &data)
- virtual int updateResource (const ve::dataChar &data)

Protected Attributes

- Window m_hWindow
- GLXContext m_hGlContext
- int m_winSizeX
- int m_winSizeY
- std::string m_winTitle
- int m_glClearBits
- float m_mouseNeutral
- float m_mouseRelative
- bool m_mouseVisible
- ve::flag128 m_inputState
- ve::vec6f m_inputAxes
- unsigned int m_wheel
- ve::vec4f m_fgNormalCol
- ve::vec4f m_fgSelectCol
- ve::vec4f m_bgNormalCol
- ve::vec4f m_bgSelectCol
- std::vector< ve::glText * > m_vTxtRenderer
- std::vector< unsigned int > m_vTxtRendererId
- std::vector< ve::dataChar > m_vRes
- std::vector< ve::ovlObj * > m_vOvlObj
- std::vector< ve::dataContainer > m_vObj
- float m_ovlX0
- float m_ovlY0
- float m_ovlX1
- float m_ovlY1

12.27.1 Detailed Description

class for window handling, keyboard and mouse input.

This class provides an OpenGL window and handles basic user input in a system-independent way. The current implementation is based on libSDL.

Definition at line 41 of file veDeviceSDL.h.

12.27.2 Constructor & Destructor Documentation

12.27.2.1 ve::deviceWindow::deviceWindow (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0)

constructor

All initialization values are taken from the deviceWindow section of an xml statement.

Parameters:

ini a previously loaded xml ini file

iniSectionId (optional) specifies which deviceWindow section is to be interpreted.

12.27.2.2 virtual ve::deviceWindow::~deviceWindow () [virtual]

destructor

12.27.3 Member Function Documentation

12.27.3.1 virtual int ve::deviceWindow::update (double *deltaT* = 0.0) [virtual]

performs drawing, adds overlay plane, swaps buffers, and polls events

Reimplemented from ve::device.

12.27.3.2 int ve::deviceWindow::w () const [inline]

returns window width

Definition at line 54 of file veDeviceSDL.h.

References m_winSizeX.

12.27.3.3 int ve::deviceWindow::h () const [inline]

returns window height

Definition at line 56 of file veDeviceSDL.h.

References m_winSizeY.

12.27.3.4 Window ve::deviceWindow::windowHandle () const [inline]

returns (platform specific) window handle

Definition at line 66 of file veDeviceSDL.h.

References m_hWindow.

12.27.3.5 GLXContext ve::deviceWindow::glContext () const [inline]

returns (platform specific) OpenGL context

Definition at line 68 of file veDeviceSDL.h.

References m_hGlContext.

12.27.3.6 virtual int ve::deviceWindow::setVar (const ve::dataChar & *data*) [virtual]

sets an exposed variable of the device via a dataChar container

This feature is publicly accessible using setOutput(CMD_DEVICE_VAR_SET). It has been added recently (veLib v1.1.1), therefore only a few internal variables are exposed up to now. If you need access to a particular feature at runtime, feel free to contact the developers, exposure is normally easily implemented, provided that a runtime access is sensible.

Reimplemented from ve::device.

12.27.3.7 void ve::deviceWindow::clearOverlay ()

drops all overlay objects

12.27.3.8 const ve::vec4f& ve::deviceWindow::fgNormalColor () const [inline]

returns standard foreground color
Definition at line 82 of file veDeviceSDL.h.
References m_fgNormalCol.

12.27.3.9 const ve::vec4f& ve::deviceWindow::bgNormalColor () const [inline]

returns standard background color
Definition at line 84 of file veDeviceSDL.h.
References m_bgNormalCol.

12.27.3.10 const ve::vec4f& ve::deviceWindow::fgSelectColor () const [inline]

returns selected foreground color
Definition at line 86 of file veDeviceSDL.h.
References m_fgSelectCol.

12.27.3.11 const ve::vec4f& ve::deviceWindow::bgSelectColor () const [inline]

returns selected background color
Definition at line 88 of file veDeviceSDL.h.
References m_bgSelectCol.

12.27.3.12 ve::glText* ve::deviceWindow::textRenderer (unsigned int *id*)

returns pointer to text renderer with suitable id, or a default text renderer if available, otherwise 0

12.27.3.13 const ve::flag128& ve::deviceWindow::inputState () const [inline]

makes input device state available for interactive overlay objects
Definition at line 92 of file veDeviceSDL.h.
References m_inputState.

12.27.3.14 float ve::deviceWindow::screenX (float *x*) const [inline]

transforms x overlay ordinate in x screen ordinate
Definition at line 94 of file veDeviceSDL.h.
References m_ovlX0, m_ovlX1, and m_winSizeX.

12.27.3.15 float ve::deviceWindow::screenY (float *y*) const [inline]

transforms y overlay ordinate in y screen ordinate
Definition at line 96 of file veDeviceSDL.h.
References m_ovlY0, m_ovlY1, and m_winSizeY.

12.27.3.16 float ve::deviceWindow::ovlX (float *x*) const [inline]

transforms x screen ordinate in x ovl ordinate
Definition at line 98 of file veDeviceSDL.h.
References m_ovlX0, m_ovlX1, and m_winSizeX.

12.27.3.17 float ve::deviceWindow::ovlY (float *y*) const [inline]

transforms y overlay ordinate in y ovl ordinate
Definition at line 100 of file veDeviceSDL.h.
References m_ovlY0, m_ovlY1, and m_winSizeY.

12.27.3.18 float ve::deviceWindow::minX () const [inline]

returns minimum x coordinate
Definition at line 102 of file veDeviceSDL.h.
References m_ovlX0.

12.27.3.19 float ve::deviceWindow::minY () const [inline]

returns minimum x coordinate
Definition at line 104 of file veDeviceSDL.h.
References m_ovlY0.

12.27.3.20 float ve::deviceWindow::maxX () const [inline]

returns maximum x coordinate
Definition at line 106 of file veDeviceSDL.h.
References m_ovlX1.

12.27.3.21 float ve::deviceWindow::maxY () const [inline]

returns maximum x coordinate

Definition at line 108 of file veDeviceSDL.h.

12.27.3.22 virtual int ve::deviceWindow::queryDevice (ve::vec6f & *axes*, ve::flag128 & *flags*) [protected, virtual]

gets current input state of device

Reimplemented from ve::device.

12.27.3.23 virtual int ve::deviceWindow::updateDevice (const ve::vec6f & *pos*, const ve::flag128 & *flags*) [protected, virtual]

sets output of device

In this case only the mouse pointer can be set (pos[H] and pos[P]).

Reimplemented from ve::device.

12.27.3.24 virtual int ve::deviceWindow::updateObject (const ve::dataContainer & *data*) [protected, virtual]

updates an overlay object by using the standard interface

Parameters:
data contains all data describing an object.

Returns:
0 in case of success.

Reimplemented from ve::device.

12.27.3.25 virtual int ve::deviceWindow::updateResource (const ve::dataChar & *data*) [protected, virtual]

updates a resource at runtime

Parameters:
data contains all data describing the resource.

Returns:
0 in case of success.

Reimplemented from ve::device.

12.27.4 Member Data Documentation

12.27.4.1 ve::deviceWindow* ve::deviceWindow::window [static]

stores a pointer to the current window.

Only 1 window is allowed at a time.

Definition at line 108 of file veDeviceSDL.h.

12.27.4.2 Window ve::deviceWindow::m_hWindow [protected]

stores (platform specific) window handle

Definition at line 136 of file veDeviceSDL.h.

Referenced by windowHandle().

12.27.4.3 GLXContext ve::deviceWindow::m_hGlContext [protected]

stores (platform specific) window handle

Definition at line 138 of file veDeviceSDL.h.

Referenced by glContext().

12.27.4.4 int ve::deviceWindow::m_winSizeX [protected]

window width

Definition at line 141 of file veDeviceSDL.h.

Referenced by ovlX(), screenX(), and w().

12.27.4.5 int ve::deviceWindow::m_winSizeY [protected]

window height

Definition at line 143 of file veDeviceSDL.h.

Referenced by h(), ovlY(), and screenY().

12.27.4.6 std::string ve::deviceWindow::m_winTitle [protected]

window title

Definition at line 145 of file veDeviceSDL.h.

12.27.4.7 int ve::deviceWindow::m_glClearBits [protected]

stores bit mask for clearing screen

Definition at line 147 of file veDeviceSDL.h.

12.27.4.8 float ve::deviceWindow::m_mouseNeutral [protected]

stores relative neutral range for the mouse interpretation
Definition at line 149 of file veDeviceSDL.h.

12.27.4.9 float ve::deviceWindow::m_mouseRelative [protected]

stores whether the mouse returns relative or absolute data
The variable additionally is a scale factor for relative mouse events.
Definition at line 152 of file veDeviceSDL.h.

12.27.4.10 bool ve::deviceWindow::m_mouseVisible [protected]

stores whether the mouse pointer is visible
Definition at line 154 of file veDeviceSDL.h.

12.27.4.11 ve::flag128 ve::deviceWindow::m_inputState [protected]

stores recent input events of mouse and keyboard
Definition at line 157 of file veDeviceSDL.h.
Referenced by inputState().

12.27.4.12 ve::vec6f ve::deviceWindow::m_inputAxes [protected]

stores recent position of keyboard and mouse axes
Definition at line 159 of file veDeviceSDL.h.

12.27.4.13 unsigned int ve::deviceWindow::m_wheel [protected]

stores mouse wheel events to make them last at least one frame
Definition at line 161 of file veDeviceSDL.h.

12.27.4.14 ve::vec4f ve::deviceWindow::m_fgNormalCol [protected]

stores standard foreground (font) color
Definition at line 164 of file veDeviceSDL.h.
Referenced by fgNormalColor().

12.27.4.15 ve::vec4f ve::deviceWindow::m_fgSelectCol [protected]

stores selected foreground (font) color

Definition at line 166 of file veDeviceSDL.h.

Referenced by fgSelectColor().

12.27.4.16 ve::vec4f ve::deviceWindow::m_bgNormalCol [protected]

stores standard background color

Definition at line 168 of file veDeviceSDL.h.

Referenced by bgNormalColor().

12.27.4.17 ve::vec4f ve::deviceWindow::m_bgSelectCol [protected]

stores selected background color

Definition at line 170 of file veDeviceSDL.h.

Referenced by bgSelectColor().

12.27.4.18 std::vector<ve::glText *> ve::deviceWindow::m_vTxtRenderer [protected]

stores pointers to text renderers which are used for overlay labels

Definition at line 172 of file veDeviceSDL.h.

12.27.4.19 std::vector<unsigned int> ve::deviceWindow::m_vTxtRendererId [protected]

stores ids of text renderers

Definition at line 174 of file veDeviceSDL.h.

12.27.4.20 std::vector<ve::dataChar> ve::deviceWindow::m_vRes [protected]

stores data about overlay object resources

Definition at line 177 of file veDeviceSDL.h.

12.27.4.21 std::vector<ve::ovlObj *> ve::deviceWindow::m_vOvlObj [protected]

stores pointers to overlay objects that are drawn

Definition at line 179 of file veDeviceSDL.h.

12.27.4.22 std::vector<ve::dataContainer> ve::deviceWindow::m_vObj [protected]

vector for storing the object data of the current scene object instances

Definition at line 181 of file veDeviceSDL.h.

12.27.4.23 float ve::deviceWindow::m_ovlX0 [protected]

stores overlay viewport minimum x ordinate

Definition at line 184 of file veDeviceSDL.h.

Referenced by minX(), ovlX(), and screenX().

12.27.4.24 float ve::deviceWindow::m_ovlY0 [protected]

stores overlay viewport minimum y ordinate

Definition at line 186 of file veDeviceSDL.h.

Referenced by minY(), ovlY(), and screenY().

12.27.4.25 float ve::deviceWindow::m_ovlX1 [protected]

stores overlay viewport maximum x ordinate

Definition at line 188 of file veDeviceSDL.h.

Referenced by maxX(), ovlX(), and screenX().

12.27.4.26 float ve::deviceWindow::m_ovlY1 [protected]

stores overlay viewport maximum y ordinate

Definition at line 190 of file veDeviceSDL.h.

Referenced by ovlY(), and screenY().

The documentation for this class was generated from the following file:

- veDeviceSDL.h

12.28 ve::fileInfo Struct Reference

Public Attributes

- std::string name
- unsigned int offset
- unsigned int size
- bool isCompressed

12.28.1 Detailed Description

Definition at line 120 of file veUtils.h.

The documentation for this struct was generated from the following file:

- veUtils.h

12.29 ve::fileIo Class Reference

class facilitating file input/output operations.

`#include <veUtils.h>`

Static Public Member Functions

- static std::string exec (const std::string &cmdName, const std::string &cmdArgs="", const std::string &cmdPath="./")
- static int dir (std::vector< std::string > &target, const std::string &path=".", const std::string &filter="*")
- static bool isDir (const std::string &path)
- static int openZip (const std::string &path)
- static int closeZip ()
- static int readZipDir (FILE *fp, std::vector< fileInfo > &vZip)
- static FILE * open (const std::string &path, const char *mode)
- static int close (FILE *stream)
- static int eof (FILE *stream)
- static unsigned int getline (FILE *fp, std::string &s)
- static bool fileExist (const std::string &filename)
- static unsigned int fileSize (FILE *fp)
- static bool wildcardMatch (const std::string &fname, const std::string &filter)
- static std::string unifyPath (const std::string &source)
- static unsigned int mime (const std::string &filename)
- static std::string mime2string (unsigned int mime)
- static unsigned int string2mime (const std::string &s)
- static std::string cwd ()
- static int chdir (const std::string &path)

Static Protected Attributes

- static FILE * zipFile
- static std::vector< fileInfo > zipDir
- static unsigned int zipFileId

12.29.1 Detailed Description

class facilitating file input/output operations.

Definition at line 136 of file veUtils.h.

12.29.2 Member Function Documentation

12.29.2.1 static std::string ve::fileIo::exec (const std::string & *cmdName*, const std::string & *cmdArgs* = "", const std::string & *cmdPath* = "./") [static]

opens a pipe from an executed external command and returns its standard output.

12.29.2.2 static int ve::fileIo::dir (std::vector< std::string > & *target*, const std::string & *path* = ".", const std::string & *filter* = "*") [static]

puts all directory entries of path in target string vector.

this method is planned to become transparent for zip archives. Not yet implemented.

Parameters:

target is filled with directory entries

path (optional) defines directory path

filter (optional) sets filter with * wildcards

Returns:

0 if no errors occured, otherwise 1.

12.29.2.3 static bool ve::fileIo::isDir (const std::string & *path*) [static]

determines whether path points to a directory

12.29.2.4 static int ve::fileIo::openZip (const std::string & *path*) [static]

opens a zip archive for further reading.

Currently only uncompressed archives are supported. Only one archive can be open at the same time. This is NOT thread safe!

12.29.2.5 static int ve::fileIo::closeZip () [static]

closes currently opened zip archive, if any.

12.29.2.6 static int ve::fileIo::readZipDir (FILE * *fp*, std::vector< fileInfo > & *vZip*) [static]

reads information about files in a zip archive

12.29.2.7 static FILE* ve::fileIo::open (const std::string & *path*, const char * *mode*) [static]

opens a FILE stream

This static method makes reading of zip archive member files "transparent" (i.e. hides them). It tests whether the requested file is member of a currently opened ZIP archive. If so, the read position is moved at the correct position. If it is no member of the current archive, or no archive is open at all, a normal fopen is performed. Only reading from zip is supported (i.e. "r" or "rb").

12.29.2.8 static int ve::fileIo::close (FILE * *stream*) [static]

closes a FILE stream

This static method makes reading of zip archive member files "transparent" (i.e. hides them). It tests whether the requested file is member of a currently open ZIP archive. If so, the read position is resetted. If it is no member of the current archive, or no archive is open at all, a normal fclose is performed.

12.29.2.9 static int ve::fileIo::eof (FILE * *stream*) [static]

tests a FILE stream for end of file

This static method makes reading of zip archive member files "transparent" (i.e. hides them). It tests whether the requested file stream is readable. If it is no member of the current archive, or no archive is open at all, a normal feof is performed.

12.29.2.10 static unsigned int ve::fileIo::getline (FILE * *fp*, std::string & *s*) [static]

reads a line from a FILE* into a c++ string

Returns:
the number of read chars.

12.29.2.11 static bool ve::fileIo::fileExist (const std::string & *filename*) [static]

tests whether file filename exists and returns TRUE in case of existence.

12.29.2.12 static unsigned int ve::fileIo::fileSize (FILE * *fp*) [static]

returns the file size in bytes.

This function is transparent for files in zip archives.

12.29.2.13 static bool ve::fileIo::wildcardMatch (const std::string & *fname*, const std::string & *filter*) [static]

tests whether a filename matches to a filename filter that may contain stars (*)

12.29.2.14 static std::string ve::fileIo::unifyPath (const std::string & *source*) [static]

removes OS dependencies from a filepath string

12.29.2.15 static unsigned int ve::fileIo::mime (const std::string & *filename*) [static]

returns the mime type of a file name if recognized

12.29.2.16 static std::string ve::fileIo::mime2string (unsigned int *mime*) [static]

this function converts mime type constants to literal strings

12.29.2.17 static unsigned int ve::fileIo::string2mime (const std::string & s) [static]

this function converts literal strings to mime type constants

12.29.2.18 static std::string ve::fileIo::cwd () [static]

returns current working directory

12.29.2.19 static int ve::fileIo::chdir (const std::string & *path*) [static]

changes the current working directory

12.29.3 Member Data Documentation

12.29.3.1 FILE* ve::fileIo::zipFile [static, protected]

stores file handle to currently opened zip archive
Definition at line 204 of file veUtils.h.

12.29.3.2 std::vector<fileInfo> ve::fileIo::zipDir [static, protected]

stores information about members of currently opened zip archive
Definition at line 206 of file veUtils.h.

12.29.3.3 unsigned int ve::fileIo::zipFileId [static, protected]

stores id of current opened zipfile member
Definition at line 208 of file veUtils.h.
The documentation for this class was generated from the following file:

- veUtils.h

12.30 ve::flag128 Class Reference

a class for storing a large number of flags.

`#include <veTypes.h>`

Public Member Functions

- flag128 ()
- flag128 (const std::string &s)
- bool operator[ ] (unsigned int n) const
- bool operator== (const ve::flag128 &f) const
- bool operator!= (const ve::flag128 &f) const
- void on (unsigned int n)
- void off (unsigned int n)
- void clear ()
- void set (const std::string &s)
- unsigned int size () const
- unsigned int & word (unsigned int n)
- unsigned int word (unsigned int n) const
- unsigned int nWords () const
- std::string str () const

Protected Attributes

- unsigned int m_word [4]

12.30.1 Detailed Description

a class for storing a large number of flags.

This small inline class is mainly designed to store a complete state of any input device, e.g., all keys of a keyboard. To allow effective memory packing and copying, no virtual methods or additional variables should be added.

Definition at line 215 of file veTypes.h.

12.30.2 Constructor & Destructor Documentation

12.30.2.1 ve::flag128::flag128 () [inline]

default constructor

Definition at line 218 of file veTypes.h.

References clear().

12.30.2.2 ve::flag128::flag128 (const std::string & *s*) [inline]

constructor from a string containing 4 unsigned ints separated by whitespace

Definition at line 220 of file veTypes.h.

References set().

12.30.3 Member Function Documentation

12.30.3.1]

bool ve::flag128::operator[] (unsigned int *n*) const [inline]

returns value n.

Returns:
bool value of flag n or false if n is out of range.

Definition at line 223 of file veTypes.h.

References ve::BIT, m_word, and size().

Referenced by off().

12.30.3.2 bool ve::flag128::operator== (const ve::flag128 & *f*) const [inline]

comparison operator equality

Definition at line 226 of file veTypes.h.

References m_word.

Referenced by operator!=().

12.30.3.3 bool ve::flag128::operator!= (const ve::flag128 & *f*) const [inline]

comparison operator inequality

Definition at line 230 of file veTypes.h.

References operator==().

12.30.3.4 void ve::flag128::on (unsigned int *n*) [inline]

turns flag n on

Definition at line 232 of file veTypes.h.

References ve::BIT, m_word, and size().

12.30.3.5 void ve::flag128::off (unsigned int *n*) [inline]

turns flag n off

Definition at line 235 of file veTypes.h.
References ve::BIT, m_word, operator[](), and size().

12.30.3.6 void ve::flag128::clear () [inline]

sets all flags to false
Definition at line 238 of file veTypes.h.
References m_word.
Referenced by flag128(), and set().

12.30.3.7 void ve::flag128::set (const std::string & *s*) [inline]

sets flags from a string containing 4 unsigned ints separated by whitespace
Definition at line 240 of file veTypes.h.
References clear(), m_word, and ve::s2ui().
Referenced by flag128().

12.30.3.8 unsigned int ve::flag128::size () const [inline]

returns number of flags in one object
Definition at line 243 of file veTypes.h.
Referenced by off(), on(), and operator[]().

12.30.3.9 unsigned int& ve::flag128::word (unsigned int *n*) [inline]

allows direct access to the unsigned ints, no range check!
Definition at line 245 of file veTypes.h.
References m_word.

12.30.3.10 unsigned int ve::flag128::word (unsigned int *n*) const [inline]

allows direct reading of the unsigned ints, no range check!
Definition at line 247 of file veTypes.h.
References m_word.

12.30.3.11 unsigned int ve::flag128::nWords () const [inline]

returns the number of words needed to store a flag128 object
Definition at line 249 of file veTypes.h.

12.30.3.12 std::string ve::flag128::str () const [inline]

returns a string containing all data

Definition at line 251 of file veTypes.h.

References ve::i2s(), and m_word.

12.30.4 Member Data Documentation

12.30.4.1 unsigned int ve::flag128::m_word[4] [protected]

stores the flags

Definition at line 251 of file veTypes.h.

Referenced by clear(), off(), on(), operator==(), operator[](), set(), str(), and word().

The documentation for this class was generated from the following file:

- veTypes.h

12.31 ve::frustum Class Reference

class for frustum (clipping) operations.

`#include <veMath.h>`

Public Member Functions

- frustum ()
- frustum (float clipLeft, float clipRight, float clipBottom, float clipTop, float clipNear, float clipFar)
- frustum (const frustum &source)
- const frustum & operator= (const frustum &source)
- void set (float clipLeft, float clipRight, float clipBottom, float clipTop, float clipNear, float clipFar)
- bool intersects (const sphere &sph) const
- bool intersects (const ve::vec3f &vecMin, const ve::vec3f &vecMax) const
- void translate (float x, float y, float z=0.0f)
- void translate (const vec3f &v)
- void rotate (float angle, const vec3f &p)
- void rotate (float h, float p, float r)
- void transform (const ve::vec6f &sdof)
- ve::plane & plane (unsigned int i)
- const ve::plane & plane (unsigned int i) const

Protected Attributes

- ve::plane pl [6]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::frustum &fr)

12.31.1 Detailed Description

class for frustum (clipping) operations.

Definition at line 878 of file veMath.h.

12.31.2 Constructor & Destructor Documentation

12.31.2.1 ve::frustum::frustum ()

default constructor

12.31.2.2 ve::frustum::frustum (float *clipLeft*, float *clipRight*, float *clipBottom*, float *clipTop*, float *clipNear*, float *clipFar*)

constructor

12.31.2.3 ve::frustum::frustum (const frustum & *source*)

copy constructor

12.31.3 Member Function Documentation

12.31.3.1 const frustum& ve::frustum::operator= (const frustum & *source*)

copy operator

12.31.3.2 void ve::frustum::set (float *clipLeft*, float *clipRight*, float *clipBottom*, float *clipTop*, float *clipNear*, float *clipFar*)

sets frustum geometry

12.31.3.3 bool ve::frustum::intersects (const sphere & *sph*) const

tests whether sphere sph at least partially intersects this frustum.

12.31.3.4 bool ve::frustum::intersects (const ve::vec3f & *vecMin*, const ve::vec3f & *vecMax*) const

tests whether an axis-aligned box, defined by its min and max coordinates, at least partially intersects this frustum.

12.31.3.5 void ve::frustum::translate (float *x*, float *y*, float *z* = `0.0f`) `[inline]`

translates object by x|y|z.

Definition at line 897 of file veMath.h.

12.31.3.6 void ve::frustum::translate (const vec3f & *v*)

translates object by vector v.

12.31.3.7 void ve::frustum::rotate (float *angle*, const vec3f & *p*)

rotates object around arbitrary axis from origin to p by angle.

12.31.3.8 void ve::frustum::rotate (float *h*, float *p*, float *r*)

rotates object according to heading, pitch, and roll.

12.31.3.9 void ve::frustum::transform (const ve::vec6f & *sdof*)

transforms this frustum by applying the provided sixdof transformation.

The vector is rotated first according to r,p,h, afterwards translated by x,y,z.

12.31.3.10 ve::plane& ve::frustum::plane (unsigned int *i*) [inline]

allows access to plane i

Definition at line 909 of file veMath.h.

References pl.

12.31.3.11 const ve::plane& ve::frustum::plane (unsigned int *i*) const [inline]

allows reading of plane i

Definition at line 911 of file veMath.h.

References pl.

12.31.4 Friends And Related Function Documentation

12.31.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::frustum & *fr*) [friend]

operator for output in streams

12.31.5 Member Data Documentation

12.31.5.1 ve::plane ve::frustum::pl[6] [protected]

stores clipping planes

Definition at line 917 of file veMath.h.

Referenced by plane().

The documentation for this class was generated from the following file:

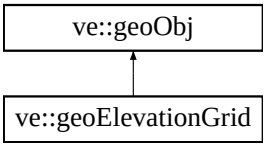
- veMath.h

12.32 ve::geoElevationGrid Class Reference

a class for interpreting and displaying elevation grids / terrain models

`#include <veGeoObj.h>`

Inheritance diagram for ve::geoElevationGrid::



Public Member Functions

- geoElevationGrid (const float *pElev, unsigned int dimX, unsigned int dimY, float spacingX, float spacingY, float scaleZ=1.0f)
- geoElevationGrid (const ve::xml &xs)
- virtual void initGraphics ()
- virtual void draw ()
- void triangles (std::vector< ve::triangle > &tr) const
- virtual void transform (const ve::mat4f &m)
- virtual void calcBounding ()
- unsigned int dimX () const
- unsigned int dimY () const
- float zValue (unsigned int x, unsigned int y) const
- float & zValue (unsigned int x, unsigned int y)
- float elevation (float x, float y) const

Protected Attributes

- unsigned int m_dimX
- unsigned int m_dimY
- float m_spaceX
- float m_spaceY
- std::vector< ve::vec3f > m_vCoord
- std::vector< ve::vec3f > m_vNormal
- std::vector< ve::vec2f > m_vTexCoord
- std::string m_texName
- unsigned int m_texId
- ve::vec2f m_texCoordScale

12.32.1 Detailed Description

a class for interpreting and displaying elevation grids / terrain models

Definition at line 428 of file veGeoObj.h.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 ve::geoElevationGrid::geoElevationGrid (const float * *pElev*, unsigned int *dimX*, unsigned int *dimY*, float *spacingX*, float *spacingY*, float *scaleZ* = `1.0f`)

constructor from memory data

12.32.2.2 ve::geoElevationGrid::geoElevationGrid (const ve::xml & *xs*)

constructor interpreting an X3D defined ElevationGrid node.

12.32.3 Member Function Documentation

12.32.3.1 virtual void ve::geoElevationGrid::initGraphics () `[virtual]`

performs OpenGL initializations.

Reimplemented from ve::geoObj.

12.32.3.2 virtual void ve::geoElevationGrid::draw () `[virtual]`

draws the elevation grid

Reimplemented from ve::geoObj.

12.32.3.3 void ve::geoElevationGrid::triangles (std::vector< ve::triangle > & *tr*) const `[virtual]`

fills tr with corresponding triangles

Reimplemented from ve::geoObj.

12.32.3.4 virtual void ve::geoElevationGrid::transform (const ve::mat4f & *m*) `[inline, virtual]`

transforms this object by multiplying it with matrix m, not for realtime!.

Reimplemented from ve::geoObj.

Definition at line 442 of file veGeoObj.h.

References m_vCoord, and ve::mat4f::transform().

12.32.3.5 virtual void ve::geoElevationGrid::calcBounding () `[virtual]`

computes the bounding geometry, not for realtime!

Reimplemented from ve::geoObj.

12.32.3.6 unsigned int ve::geoElevationGrid::dimX () const [inline]

returns x dimension

Definition at line 448 of file veGeoObj.h.

References m_dimX.

12.32.3.7 unsigned int ve::geoElevationGrid::dimY () const [inline]

returns y dimension

Definition at line 450 of file veGeoObj.h.

References m_dimY.

12.32.3.8 float ve::geoElevationGrid::zValue (unsigned int *x*, unsigned int *y*) const [inline]

returns z value of grid cell x|y

Definition at line 452 of file veGeoObj.h.

References m_dimX, m_dimY, and m_vCoord.

12.32.3.9 float& ve::geoElevationGrid::zValue (unsigned int *x*, unsigned int *y*) [inline]

allows access to z value of grid cell x|y

Definition at line 454 of file veGeoObj.h.

References m_dimX, m_dimY, and m_vCoord.

12.32.3.10 float ve::geoElevationGrid::elevation (float *x*, float *y*) const

returns elevation at spatial coordinate x|y

12.32.4 Member Data Documentation

12.32.4.1 unsigned int ve::geoElevationGrid::m_dimX [protected]

stores x dimension

Definition at line 459 of file veGeoObj.h.

Referenced by dimX(), and zValue().

12.32.4.2 unsigned int ve::geoElevationGrid::m_dimY [protected]

stores y dimension

Definition at line 461 of file veGeoObj.h.

Referenced by dimY(), and zValue().

12.32.4.3 float ve::geoElevationGrid::m_spaceX [protected]

stores x spacing

Definition at line 463 of file veGeoObj.h.

12.32.4.4 float ve::geoElevationGrid::m_spaceY [protected]

stores y spacing

Definition at line 465 of file veGeoObj.h.

12.32.4.5 std::vector<ve::vec3f> ve::geoElevationGrid::m_vCoord [protected]

stores vertices

Definition at line 467 of file veGeoObj.h.

Referenced by transform(), and zValue().

12.32.4.6 std::vector<ve::vec3f> ve::geoElevationGrid::m_vNormal [protected]

stores the object's normals

Definition at line 469 of file veGeoObj.h.

12.32.4.7 std::vector<ve::vec2f> ve::geoElevationGrid::m_vTexCoord [protected]

stores the elevation grid's texture coordinates

Definition at line 472 of file veGeoObj.h.

12.32.4.8 std::string ve::geoElevationGrid::m_texName [protected]

stores texture filename

Definition at line 474 of file veGeoObj.h.

12.32.4.9 unsigned int ve::geoElevationGrid::m_texId [protected]

stores texture id

Definition at line 476 of file veGeoObj.h.

12.32.4.10 ve::vec2f ve::geoElevationGrid::m_texCoordScale [protected]

stores texture coordinate scaling

Definition at line 478 of file veGeoObj.h.

The documentation for this class was generated from the following file:

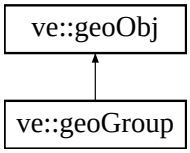
- veGeoObj.h

12.33 ve::geoGroup Class Reference

base class for organizing ve::geoObjects in a tree-like structure.

`#include <veGeoObj.h>`

Inheritance diagram for ve::geoGroup::



Public Member Functions

- geoGroup (unsigned int newId=0)
- geoGroup (const std::string &filename, unsigned int newId=0)
- geoGroup (const ve::geoGroup &source)
- virtual ~geoGroup ()
- virtual void draw ()
- virtual void initGraphics ()
- void regionalize (unsigned int currRecursion=0)
- const ve::mat4f & transform () const
- virtual void transform (const ve::mat4f &m)
- virtual void setTransform (const ve::mat4f &m)
- virtual void setTransform (const ve::vec6f &sixdof)
- virtual void resetTransform ()
- virtual void addChild (ve::geoObj *geoObject)
- virtual void dropChild (ve::geoObj *geoObject)
- virtual unsigned int nChildren () const
- const std::vector< ve::geoObj * > & children () const
- virtual void calcBounding ()
- const ve::vec3f minCoord () const
- const ve::vec3f maxCoord () const
- virtual void triangles (std::vector< ve::triangle > &tr) const
- virtual void vertices (std::vector< ve::vec3f > &vVertices) const
- virtual std::string vrml (unsigned int nTabs=0) const

Protected Attributes

- std::vector< ve::geoObj * > vChildren
- bool m_cleanupChildren
- ve::mat4f m_mat
- bool m_isIdentity

12.33.1 Detailed Description

base class for organizing ve::geoObjects in a tree-like structure.

This class makes use of the ve::io classes to load and interpret X3D, VRML and 3ds files.

Definition at line 227 of file veGeoObj.h.

12.33.2 Constructor & Destructor Documentation

12.33.2.1 ve::geoGroup::geoGroup (unsigned int *newId* = 0) [inline]

default constructor, optional argument is user definable class id

Definition at line 230 of file veGeoObj.h.

References m_cleanupChildren, and m_isIdentity.

12.33.2.2 ve::geoGroup::geoGroup (const std::string & *filename*, unsigned int *newId* = 0)

constructor loading a model from a file.

Parameters:

filename must contain the filename plus path,

newId (optional) is a user definable class id.

12.33.2.3 ve::geoGroup::geoGroup (const ve::geoGroup & *source*)

copy constructor

12.33.2.4 virtual ve::geoGroup::~geoGroup () [virtual]

destructor

12.33.3 Member Function Documentation

12.33.3.1 virtual void ve::geoGroup::draw () [virtual]

draws object.

calls draw methods of children.

Reimplemented from ve::geoObj.

12.33.3.2 virtual void ve::geoGroup::initGraphics () [virtual]

performs OpenGL initializations.

calls initGraphics() of all children.

Reimplemented from ve::geoObj.

12.33.3.3 void ve::geoGroup::regionalize (unsigned int *currRecursion* = 0)

regionalizes this group in a boxtree structure.

The boxtree structure is similar to a quadtree or octree, but assigns objects intersected by the clipping plane to the best sector if possible. That means, it may be suboptimal for carefully modeled scenes, but tolerant for others.

12.33.3.4 const ve::mat4f& ve::geoGroup::transform () const [inline]

returns transformation matrix m

Definition at line 254 of file veGeoObj.h.

References m_mat.

12.33.3.5 virtual void ve::geoGroup::transform (const ve::mat4f & *m*) [inline, virtual]

transforms this object by multiplying it with matrix m.

beware of scalings or shears!

Reimplemented from ve::geoObj.

Definition at line 257 of file veGeoObj.h.

References m_isIdentity, and m_mat.

12.33.3.6 virtual void ve::geoGroup::setTransform (const ve::mat4f & *m*) [inline, virtual]

sets transformation to matrix m.

beware of scalings or shears!

Definition at line 261 of file veGeoObj.h.

References m_isIdentity, and m_mat.

12.33.3.7 virtual void ve::geoGroup::setTransform (const ve::vec6f & *sixdof*) [inline, virtual]

sets transformation to vec6f sixdof.

Definition at line 264 of file veGeoObj.h.

References m_isIdentity, and m_mat.

12.33.3.8 virtual void ve::geoGroup::addChild (ve::geoObj * *geoObject*) [virtual]

adds a child glObj

12.33.3.9 virtual void ve::geoGroup::dropChild (ve::geoObj * *geoObject*) [virtual]

removes a child glObj

12.33.3.10 virtual unsigned int ve::geoGroup::nChildren () const [inline, virtual]

returns number of direct children

Reimplemented from ve::geoObj.

Definition at line 274 of file veGeoObj.h.

References vChildren.

12.33.3.11 const std::vector<ve::geoObj*>& ve::geoGroup::children () const [inline]

allows read access to children vector, always empty

Definition at line 276 of file veGeoObj.h.

References vChildren.

12.33.3.12 virtual void ve::geoGroup::calcBounding () [virtual]

computes the bounding sphere, not for realtime!

Reimplemented from ve::geoObj.

12.33.3.13 const ve::vec3f ve::geoGroup::minCoord () const

computes and returns the minimum coordinates

12.33.3.14 const ve::vec3f ve::geoGroup::maxCoord () const

computes and returns the maximum coordinates

12.33.3.15 virtual void ve::geoGroup::triangles (std::vector< ve::triangle > & *tr*) const [virtual]

fills tr with triangles of all children

Reimplemented from ve::geoObj.

12.33.3.16 virtual void ve::geoGroup::vertices (std::vector< ve::vec3f > & *vVertices*) const [virtual]

puts all vertices in vVertices.

Reimplemented from ve::geoObj.

12.33.3.17 virtual std::string ve::geoGroup::vrml (unsigned int *nTabs* = 0) const [virtual]

returns geometry as VRML

Reimplemented from ve::geoObj.

12.33.4 Member Data Documentation

12.33.4.1 std::vector<ve::geoObj*> ve::geoGroup::vChildren [protected]

vector for pointers to geoObj children

Definition at line 293 of file veGeoObj.h.

Referenced by children(), and nChildren().

12.33.4.2 bool ve::geoGroup::m_cleanupChildren [protected]

stores whether this group has been loaded from a file.

Definition at line 295 of file veGeoObj.h.

Referenced by geoGroup().

12.33.4.3 ve::mat4f ve::geoGroup::m_mat [protected]

stores current transformation

Definition at line 297 of file veGeoObj.h.

Referenced by resetTransform(), setTransform(), and transform().

12.33.4.4 bool ve::geoGroup::m_isIdentity [protected]

stores whether current transformation is for sure an identity matrix

Definition at line 299 of file veGeoObj.h.

Referenced by geoGroup(), resetTransform(), setTransform(), and transform().

The documentation for this class was generated from the following file:

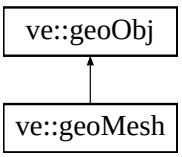
- veGeoObj.h

12.34 ve::geoMesh Class Reference

a class for static indexedFaceSet mesh objects.

`#include <veGeoObj.h>`

Inheritance diagram for ve::geoMesh::



Public Member Functions

- geoMesh ()
- geoMesh (const ve::geoMesh &source)
- geoMesh (const ve::xml &xs)
- virtual void draw ()
- virtual void initGraphics ()
- virtual void transform (const ve::mat4f &m)
- virtual void calcBounding ()
- virtual void vertices (std::vector< ve::vec3f > &vVertices) const
- virtual void triangles (std::vector< ve::triangle > &tr) const
- void triangulate ()
- std::vector< ve::vec3f > & coords ()
- const std::vector< ve::vec3f > & coords () const
- std::vector< ve::vec3f > & normals ()
- const std::vector< ve::vec3f > & normals () const
- std::vector< ve::vec2f > & texCoords ()
- const std::vector< ve::vec2f > & texCoords () const
- const std::vector< ve::vec3f > & vertexColors () const
- std::vector< ve::vec3f > & vertexColors ()
- std::vector< unsigned int > & indices ()
- const std::vector< unsigned int > & indices () const
- std::vector< unsigned int > & faceEnds ()
- const std::vector< unsigned int > & faceEnds () const
- std::vector< unsigned int > & texIndices ()
- const std::vector< unsigned int > & texIndices () const
- const std::vector< unsigned int > & colorIndices () const
- std::vector< unsigned int > & colorIndices ()
- const std::vector< unsigned int > & normalIndices () const
- std::vector< unsigned int > & normalIndices ()
- int addTexture (const std::string &filename, const std::vector< float > &texCoords, bool repeatTexture=true)
- bool isTextured () const
- std::string & textureFileName ()
- const std::string & textureFileName () const
- bool textureRepeat () const
- bool & textureRepeat ()
- virtual std::string vrml (unsigned int nTabs=0) const

Protected Member Functions

- virtual void trianglesRaw (std::vector< ve::triangle > &tr) const

Protected Attributes

- std::vector< ve::vec3f > m_vCoord
- std::vector< ve::vec2f > m_vTexCoords
- std::vector< ve::vec3f > m_vColor
- std::vector< ve::vec3f > m_vNormal
- std::vector< unsigned int > m_vIndices
- std::vector< unsigned int > m_vTexIndices
- std::vector< unsigned int > m_vColorIndices
- std::vector< unsigned int > m_vNormalIndex
- std::vector< unsigned int > m_vFaceEnds
- std::string m_texName
- bool m_texRepeat
- unsigned int m_texId
- unsigned int m_listId

12.34.1 Detailed Description

a class for static indexedFaceSet mesh objects.

Definition at line 305 of file veGeoObj.h.

12.34.2 Constructor & Destructor Documentation

12.34.2.1 ve::geoMesh::geoMesh ()

default constructor, empty mesh.

12.34.2.2 ve::geoMesh::geoMesh (const ve::geoMesh & *source*)

copy constructor

12.34.2.3 ve::geoMesh::geoMesh (const ve::xml & *xs*)

constructor interpreting an X3D defined IndexedFaceSet node.

12.34.3 Member Function Documentation

12.34.3.1 virtual void ve::geoMesh::draw () [virtual]

draws object

Reimplemented from ve::geoObj.

12.34.3.2 virtual void ve::geoMesh::initGraphics () [virtual]

initializes GL, uploads textures to OpenGL and creates display list.

Reimplemented from ve::geoObj.

12.34.3.3 virtual void ve::geoMesh::transform (const ve::mat4f & *m*) [inline, virtual]

transforms this object by multiplying it with matrix m, not for realtime!.

Reimplemented from ve::geoObj.

Definition at line 320 of file veGeoObj.h.

References m_vCoord, and ve::mat4f::transform().

12.34.3.4 virtual void ve::geoMesh::calcBounding () [virtual]

computes the bounding geometry, not for realtime!

Reimplemented from ve::geoObj.

12.34.3.5 virtual void ve::geoMesh::vertices (std::vector< ve::vec3f > & *vVertices*) const [virtual]

puts all vertices in vVertices.

Reimplemented from ve::geoObj.

12.34.3.6 virtual void ve::geoMesh::triangles (std::vector< ve::triangle > & *tr*) const [virtual]

fills tr with corresponding triangles, triangle coordinates are transformed

Reimplemented from ve::geoObj.

12.34.3.7 void ve::geoMesh::triangulate ()

triangulates mesh polygons.

12.34.3.8 std::vector<ve::vec3f>& ve::geoMesh::coords () [inline]

allows direct access to coordinate data.

Definition at line 333 of file veGeoObj.h.

References m_vCoord.

12.34.3.9 const std::vector<ve::vec3f>& ve::geoMesh::coords () const [inline]

allows direct reading of coordinate data.

Definition at line 335 of file veGeoObj.h.

References m_vCoord.

12.34.3.10 std::vector<ve::vec3f>& ve::geoMesh::normals () [inline]

allows direct access to normals.

Definition at line 337 of file veGeoObj.h.

References m_vNormal.

12.34.3.11 const std::vector<ve::vec3f>& ve::geoMesh::normals () const [inline]

allows direct reading of normals.

Definition at line 339 of file veGeoObj.h.

References m_vNormal.

12.34.3.12 std::vector<ve::vec2f>& ve::geoMesh::texCoords () [inline]

allows direct access to texture coordinate data.

Definition at line 341 of file veGeoObj.h.

References m_vTexCoords.

12.34.3.13 const std::vector<ve::vec2f>& ve::geoMesh::texCoords () const [inline]

allows direct reading of texture coordinate data.

Definition at line 343 of file veGeoObj.h.

References m_vTexCoords.

12.34.3.14 const std::vector<ve::vec3f>& ve::geoMesh::vertexColors () const [inline]

allows direct reading of vertex colors.

Definition at line 345 of file veGeoObj.h.

References m_vColor.

12.34.3.15 std::vector<ve::vec3f>& ve::geoMesh::vertexColors () [inline]

allows direct access to vertex colors.

Definition at line 347 of file veGeoObj.h.

References m_vColor.

12.34.3.16 std::vector<unsigned int>& ve::geoMesh::indices () [inline]

allows direct access to indices.

Definition at line 350 of file veGeoObj.h.

References m_vIndices.

12.34.3.17 const std::vector<unsigned int>& ve::geoMesh::indices () const [inline]

allows direct reading of indices.

Definition at line 352 of file veGeoObj.h.

References m_vIndices.

12.34.3.18 std::vector<unsigned int>& ve::geoMesh::faceEnds () [inline]

allows direct access to face ends.

Definition at line 354 of file veGeoObj.h.

References m_vFaceEnds.

12.34.3.19 const std::vector<unsigned int>& ve::geoMesh::faceEnds () const [inline]

allows direct reading of face ends.

Definition at line 356 of file veGeoObj.h.

References m_vFaceEnds.

12.34.3.20 std::vector<unsigned int>& ve::geoMesh::texIndices () [inline]

allows direct access to texture indices.

Definition at line 358 of file veGeoObj.h.

References m_vTexIndices.

12.34.3.21 const std::vector<unsigned int>& ve::geoMesh::texIndices () const [inline]

allows direct reading of texture indices.

Definition at line 360 of file veGeoObj.h.

References m_vTexIndices.

12.34.3.22 const std::vector<unsigned int>& ve::geoMesh::colorIndices () const [inline]

allows direct reading of vertex color indices.

Definition at line 362 of file veGeoObj.h.

References m_vColorIndices.

12.34.3.23 std::vector<unsigned int>& ve::geoMesh::colorIndices () [inline]

allows direct access to vertex color indices.

Definition at line 364 of file veGeoObj.h.

References m_vColorIndices.

12.34.3.24 const std::vector<unsigned int>& ve::geoMesh::normalIndices () const [inline]

allows direct reading of normal indices.

Definition at line 366 of file veGeoObj.h.

References m_vNormalIndex.

12.34.3.25 std::vector<unsigned int>& ve::geoMesh::normalIndices () [inline]

allows direct access to normal indices.

Definition at line 368 of file veGeoObj.h.

References m_vNormalIndex.

12.34.3.26 int ve::geoMesh::addTexture (const std::string & *filename*, const std::vector< float > & *texCoords*, bool *repeatTexture* = true)

adds a complete texture definition.

12.34.3.27 bool ve::geoMesh::isTextured () const [inline]

returns true if mesh is textured.

Definition at line 373 of file veGeoObj.h.

References m_texName.

12.34.3.28 std::string& ve::geoMesh::textureFileName () [inline]

allows access to texture filename

Definition at line 375 of file veGeoObj.h.

References m_texName.

12.34.3.29 const std::string& ve::geoMesh::textureFileName () const [inline]

returns texture filename

Definition at line 377 of file veGeoObj.h.

References m_texName.

12.34.3.30 bool ve::geoMesh::textureRepeat () const [inline]

returns whether texture is repeated

Definition at line 379 of file veGeoObj.h.

References m_texRepeat.

12.34.3.31 bool& ve::geoMesh::textureRepeat () [inline]

allows to change whether texture is repeated

Definition at line 381 of file veGeoObj.h.

References m_texRepeat.

12.34.3.32 virtual std::string ve::geoMesh::vrml (unsigned int *nTabs* = 0) const [virtual]

returns a string containing all data as VRML.

Reimplemented from ve::geoObj.

12.34.3.33 virtual void ve::geoMesh::trianglesRaw (std::vector< ve::triangle > & *tr*) const [protected, virtual]

fills tr with corresponding triangles, triangle coordinates are untransformed

12.34.4 Member Data Documentation

12.34.4.1 std::vector<ve::vec3f> ve::geoMesh::m_vCoord [protected]

stores coordinates

Definition at line 390 of file veGeoObj.h.

Referenced by coords(), and transform().

12.34.4.2 std::vector<ve::vec2f> ve::geoMesh::m_vTexCoords [protected]

stores texture coords

Definition at line 392 of file veGeoObj.h.

Referenced by texCoords().

12.34.4.3 std::vector<ve::vec3f> ve::geoMesh::m_vColor [protected]

stores color values, if color per vertex

Definition at line 394 of file veGeoObj.h.

Referenced by vertexColors().

12.34.4.4 std::vector<ve::vec3f> ve::geoMesh::m_vNormal [protected]

stores the object's normals

Definition at line 396 of file veGeoObj.h.

Referenced by normals().

12.34.4.5 std::vector<unsigned int> ve::geoMesh::m_vIndices [protected]

stores coordinate indices

Definition at line 399 of file veGeoObj.h.

Referenced by indices().

12.34.4.6 std::vector<unsigned int> ve::geoMesh::m_vTexIndices [protected]

stores texture indices

Definition at line 401 of file veGeoObj.h.

Referenced by texIndices().

12.34.4.7 std::vector<unsigned int> ve::geoMesh::m_vColorIndices [protected]

stores color indices, if color per vertex

Definition at line 403 of file veGeoObj.h.

Referenced by colorIndices().

12.34.4.8 std::vector<unsigned int> ve::geoMesh::m_vNormalIndex [protected]

stores the object's normal indices

Definition at line 405 of file veGeoObj.h.

Referenced by normalIndices().

12.34.4.9 std::vector<unsigned int> ve::geoMesh::m_vFaceEnds [protected]

stores indices of face ends

Definition at line 408 of file veGeoObj.h.

Referenced by faceEnds().

12.34.4.10 std::string ve::geoMesh::m_texName [protected]

stores texture filename

Definition at line 410 of file veGeoObj.h.

Referenced by isTextured(), and textureFileName().

12.34.4.11 bool ve::geoMesh::m_texRepeat [protected]

stores wrap mode for texture

Definition at line 412 of file veGeoObj.h.

Referenced by textureRepeat().

12.34.4.12 unsigned int ve::geoMesh::m_texId [protected]

stores texture id

Definition at line 414 of file veGeoObj.h.

12.34.4.13 unsigned int ve::geoMesh::m_listId [protected]

stores OpenGL display list id.

Definition at line 416 of file veGeoObj.h.

The documentation for this class was generated from the following file:

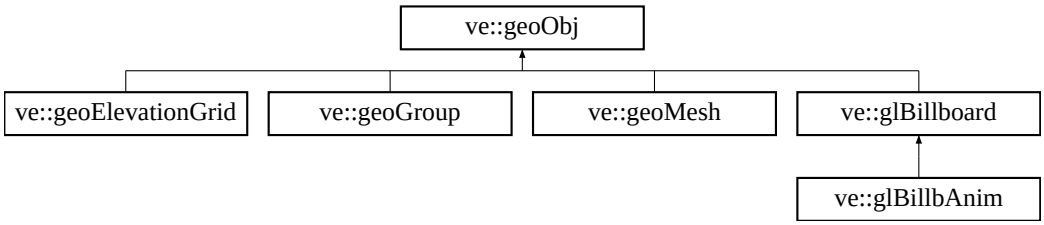
- veGeoObj.h

12.35 ve::geoObj Class Reference

base class for all derived geometry objects.

`#include <veGeoObj.h>`

Inheritance diagram for ve::geoObj::



Public Member Functions

- geoObj (unsigned int newId=0)
- geoObj (const ve::geoObj &source)
- virtual ~geoObj ()
- virtual void parent (ve::geoGroup *glO)
- virtual ve::geoGroup * parent ()
- virtual unsigned int nChildren () const
- unsigned int state () const
- const ve::vec4f & color () const
- ve::vec4f & color ()
- bool isTransparent () const
- int id () const
- void classId (unsigned int newId)
- unsigned int classId () const
- const std::string & name () const
- void name (const std::string &s)
- const void * userData () const
- void * userData ()
- virtual void transform (const ve::mat4f &m)
- virtual void draw ()
- virtual void initGeometry ()
- virtual void initGraphics ()
- virtual void closeGraphics ()
- virtual void calcBounding ()
- virtual bool testBounding (const ve::frustum &fr, const ve::vec6f &pos=ve::vec6f(0, 0, 0)) const
- virtual const ve::sphere & boundingSphere () const
- virtual void drawBounding ()
- virtual void triangles (std::vector< ve::triangle > &) const
- virtual void vertices (std::vector< ve::vec3f > &) const
- virtual std::string vrml (unsigned int nTabs=0) const
- virtual ve::xml xml () const

Static Public Member Functions

- static void setCurrTime (double tNow)
- static void setCamera (ve::vec6f *pCamera)

Public Attributes

- ve::plugin * m_plugin

Protected Attributes

- ve::sphere m_bndSphere
- ve::geoGroup * m_pParent
- void * m_pUserData
- ve::vec4f m_color
- bool m_isTransp
- unsigned int m_state
- int m_id
- unsigned int m_classId
- std::string m_name

Static Protected Attributes

- static double s_tNow
- static ve::vec6f * s_pCamera

12.35.1 Detailed Description

base class for all derived geometry objects.

This class defines the shared methods of all geometry objects. It also defines the interface for drawing and graphics initialization.

Definition at line 98 of file veGeoObj.h.

12.35.2 Constructor & Destructor Documentation

12.35.2.1 ve::geoObj::geoObj (unsigned int *newId* = 0)

default constructor, optional argument is user definable class id.

12.35.2.2 ve::geoObj::geoObj (const ve::geoObj & *source*)

copy constructor

12.35.2.3 virtual ve::geoObj::~geoObj () [virtual]

destructor

12.35.3 Member Function Documentation

12.35.3.1 virtual void ve::geoObj::parent (ve::geoGroup * *glO*) [inline, virtual]

sets the parent object

Definition at line 108 of file veGeoObj.h.

References m_pParent.

12.35.3.2 virtual ve::geoGroup* ve::geoObj::parent () [inline, virtual]

returns a pointer to the parent object

Definition at line 110 of file veGeoObj.h.

References m_pParent.

12.35.3.3 virtual unsigned int ve::geoObj::nChildren () const [inline, virtual]

returns number of direct children

Reimplemented in ve::geoGroup.

Definition at line 112 of file veGeoObj.h.

12.35.3.4 unsigned int ve::geoObj::state () const [inline]

returns state

The codes used are the generic veLib errorCodes .

Definition at line 115 of file veGeoObj.h.

References m_state.

12.35.3.5 const ve::vec4f& ve::geoObj::color () const [inline]

returns color

Definition at line 118 of file veGeoObj.h.

References m_color.

12.35.3.6 ve::vec4f& ve::geoObj::color () [inline]

allows access to color

Definition at line 120 of file veGeoObj.h.

References m_color.

12.35.3.7 bool ve::geoObj::isTransparent () const [inline]

returns transparency state
Definition at line 122 of file veGeoObj.h.
References m_isTransp.

12.35.3.8 int ve::geoObj::id () const [inline]

returns automatically generated unique id
Definition at line 125 of file veGeoObj.h.
References m_id.

12.35.3.9 void ve::geoObj::classId (unsigned int *newId*) [inline]

sets user definable id
Definition at line 127 of file veGeoObj.h.
References m_classId.

12.35.3.10 unsigned int ve::geoObj::classId () const [inline]

returns user definable id
Definition at line 129 of file veGeoObj.h.
References m_classId.

12.35.3.11 const std::string& ve::geoObj::name () const [inline]

returns name
Definition at line 131 of file veGeoObj.h.
References m_name.

12.35.3.12 void ve::geoObj::name (const std::string & *s*) [inline]

sets name
Definition at line 133 of file veGeoObj.h.
References m_name.

12.35.3.13 const void* ve::geoObj::userData () const [inline]

returns user data

Definition at line 135 of file veGeoObj.h.

References m_pUserData.

12.35.3.14 void* ve::geoObj::userData () [inline]

allows access to user data

Definition at line 137 of file veGeoObj.h.

References m_pUserData.

12.35.3.15 virtual void ve::geoObj::transform (const ve::mat4f & *m*) [inline, virtual]

transforms this object by multiplying it with matrix m, interface definition.

Reimplemented in ve::geoGroup, ve::geoMesh, and ve::geoElevationGrid.

Definition at line 140 of file veGeoObj.h.

12.35.3.16 virtual void ve::geoObj::draw () [inline, virtual]

draws object, interface definition.

Reimplemented in ve::glBillboard, ve::glBillbAnim, ve::geoGroup, ve::geoMesh, and ve::geoElevationGrid.

Definition at line 143 of file veGeoObj.h.

12.35.3.17 virtual void ve::geoObj::initGeometry () [inline, virtual]

performs geometry calculations, interface definition.

Definition at line 145 of file veGeoObj.h.

12.35.3.18 virtual void ve::geoObj::initGraphics () [inline, virtual]

performs graphics initializations, interface definition.

Reimplemented in ve::glBillboard, ve::geoGroup, ve::geoMesh, and ve::geoElevationGrid.

Definition at line 147 of file veGeoObj.h.

12.35.3.19 virtual void ve::geoObj::closeGraphics () [inline, virtual]

performs graphics cleanups, interface definition.

Definition at line 149 of file veGeoObj.h.

12.35.3.20 virtual void ve::geoObj::calcBounding () [inline, virtual]

computes the bounding geometry.

interface definition, not for realtime!

Reimplemented in ve::geoGroup, ve::geoMesh, and ve::geoElevationGrid.

Definition at line 153 of file veGeoObj.h.

12.35.3.21 virtual bool ve::geoObj::testBounding (const ve::frustum & *fr*, const ve::vec6f & *pos* = `ve::vec6f(0, 0, 0)`) const [virtual]

tests for intersection with the bounding geometry

Parameters:

fr the frustum to be tested

pos (optional) additional translation of the object

Returns:

true if the bounding geometry is intersected, otherwise false.

12.35.3.22 virtual const ve::sphere& ve::geoObj::boundingSphere () const [inline, virtual]

returns the bounding sphere

Definition at line 161 of file veGeoObj.h.

References m_bndSphere.

12.35.3.23 virtual void ve::geoObj::drawBounding () [virtual]

draws bounding geometry, mainly for debug purposes

12.35.3.24 virtual void ve::geoObj::triangles (std::vector< ve::triangle > &) const [inline, virtual]

fills tr with triangles, interface definition

Reimplemented in ve::geoGroup, ve::geoMesh, and ve::geoElevationGrid.

Definition at line 166 of file veGeoObj.h.

12.35.3.25 virtual void ve::geoObj::vertices (std::vector< ve::vec3f > &) const [inline, virtual]

puts all vertices in vVertices, interface definition.

Reimplemented in ve::geoGroup, and ve::geoMesh.

Definition at line 168 of file veGeoObj.h.

12.35.3.26 virtual std::string ve::geoObj::vrml (unsigned int *nTabs* = 0) const [inline, virtual]

returns geometry as VRML, interface definition.

Reimplemented in ve::glBillboard, ve::geoGroup, and ve::geoMesh.

Definition at line 170 of file veGeoObj.h.

12.35.3.27 virtual ve::xml ve::geoObj::xml () const [virtual]

returns an xml statement, containing all data, interface definition.

12.35.3.28 static void ve::geoObj::setCurrTime (double *tNow*) [inline, static]

sets the current time

this static method allows to set a global time for the rendering of time-dependent objects such as animated billboards or characters. The idea is to call the method once per frame, as done by deviceGraphicsX3D.

Parameters:

tNow the current time stamp in seconds

Definition at line 179 of file veGeoObj.h.

References s_tNow.

12.35.3.29 static void ve::geoObj::setCamera (ve::vec6f * *pCamera*) [inline, static]

sets the current camera position

this static method allows to set a global camera for the rendering of view-dependent objects such as billboards. Since the method takes a pointer, it has to be called normally only once at the beginning of the program execution, as done by deviceGraphicsX3D.

Parameters:

pCamera pointer to the camera sixdof. Use 0 to disable view-dependent rendering.

Definition at line 186 of file veGeoObj.h.

References s_pCamera.

12.35.4 Member Data Documentation

12.35.4.1 ve::plugin* ve::geoObj::m_plugin

pointer to an eventual plugin

Definition at line 186 of file veGeoObj.h.

12.35.4.2 ve::sphere ve::geoObj::m_bndSphere [protected]

stores bounding sphere

Definition at line 194 of file veGeoObj.h.

Referenced by boundingSphere().

12.35.4.3 ve::geoGroup* ve::geoObj::m_pParent [protected]

pointer to parent glObj

Definition at line 196 of file veGeoObj.h.

Referenced by parent().

12.35.4.4 void* ve::geoObj::m_pUserData [protected]

pointer to user data

Definition at line 198 of file veGeoObj.h.

Referenced by userData().

12.35.4.5 ve::vec4f ve::geoObj::m_color [protected]

stores color

Definition at line 201 of file veGeoObj.h.

Referenced by color().

12.35.4.6 bool ve::geoObj::m_isTransp [protected]

stores whether object is transparent

Definition at line 203 of file veGeoObj.h.

Referenced by isTransparent().

12.35.4.7 unsigned int ve::geoObj::m_state [protected]

stores state

Definition at line 205 of file veGeoObj.h.

Referenced by state().

12.35.4.8 int ve::geoObj::m_id [protected]

stores automatically generated unique number

Definition at line 208 of file veGeoObj.h.

Referenced by id().

12.35.4.9 unsigned int ve::geoObj::m_classId [protected]

stores user definable id

Definition at line 210 of file veGeoObj.h.

Referenced by classId().

12.35.4.10 std::string ve::geoObj::m_name [protected]

stores name

Definition at line 212 of file veGeoObj.h.

Referenced by name().

12.35.4.11 double ve::geoObj::s_tNow [static, protected]

stores current time

Definition at line 215 of file veGeoObj.h.

Referenced by setCurrTime().

12.35.4.12 ve::vec6f* ve::geoObj::s_pCamera [static, protected]

stores pointer to current scene camera

Definition at line 217 of file veGeoObj.h.

Referenced by setCamera().

The documentation for this class was generated from the following file:

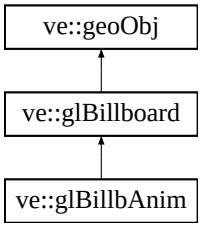
- veGeoObj.h

12.36 ve::glBillbAnim Class Reference

a class for rendering animated billboards

`#include <veDeviceGraphicsGL.h>`

Inheritance diagram for ve::glBillbAnim::



Public Member Functions

- glBillbAnim (const std::string &texName, const ve::vec3f &position, float sizeX=1.0f, float sizeY=1.0f, bool usePitch=false)
- glBillbAnim (const ve::xml &xs)
- void animate (unsigned int nX, unsigned int nY, double sampleLen=1.0, bool loop=true)
- virtual void draw ()

Protected Attributes

- unsigned int tileX
- unsigned int tileY
- double sampleLength
- double tStart
- unsigned int nLoops
- unsigned int nFrames
- double tFrame

12.36.1 Detailed Description

a class for rendering animated billboards

Definition at line 60 of file veDeviceGraphicsGL.h.

12.36.2 Constructor & Destructor Documentation

12.36.2.1 ve::glBillbAnim::glBillbAnim (const std::string & *texName*, const ve::vec3f & *position*, float *sizeX* = `1.0f`, float *sizeY* = `1.0f`, bool *usePitch* = `false`)

constructor

12.36.2.2 ve::glBillbAnim::glBillbAnim (const ve::xml & *xs*)

constructor from an xml statement

12.36.3 Member Function Documentation

12.36.3.1 void ve::glBillbAnim::animate (unsigned int *nX*, unsigned int *nY*, double *sampleLen* = `1.0`, bool *loop* = `true`)

sets animation parameters.

12.36.3.2 virtual void ve::glBillbAnim::draw () `[virtual]`

draws object

Reimplemented from ve::glBillboard.

12.36.4 Member Data Documentation

12.36.4.1 unsigned int ve::glBillbAnim::tileX `[protected]`

stores number of X tiles

Definition at line 73 of file veDeviceGraphicsGL.h.

12.36.4.2 unsigned int ve::glBillbAnim::tileY `[protected]`

stores number of Y tiles

Definition at line 75 of file veDeviceGraphicsGL.h.

12.36.4.3 double ve::glBillbAnim::sampleLength `[protected]`

stores length of sample in secs

Definition at line 77 of file veDeviceGraphicsGL.h.

12.36.4.4 double ve::glBillbAnim::tStart `[protected]`

stores start time

Definition at line 79 of file veDeviceGraphicsGL.h.

12.36.4.5 unsigned int ve::glBillbAnim::nLoops `[protected]`

stores number of loops

Definition at line 81 of file veDeviceGraphicsGL.h.

12.36.4.6 unsigned int ve::glBillbAnim::nFrames [protected]

stores total number of frames

Definition at line 83 of file veDeviceGraphicsGL.h.

12.36.4.7 double ve::glBillbAnim::tFrame [protected]

length of one frame

Definition at line 85 of file veDeviceGraphicsGL.h.

The documentation for this class was generated from the following file:

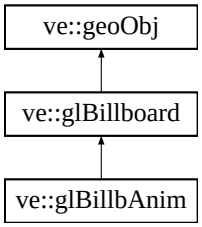
- veDeviceGraphicsGL.h

12.37 ve::glBillboard Class Reference

a class for rendering billboards

`#include <veDeviceGraphicsGL.h>`

Inheritance diagram for ve::glBillboard::



Public Member Functions

- glBillboard (const std::string &texName, const ve::vec3f &position, float sizeX=1.0f, float sizeY=1.0f, bool usePitch=false)
- glBillboard (const ve::xml &xs)
- glBillboard (const ve::glBillboard &source)
- virtual void draw ()
- virtual void initGraphics ()
- virtual std::string vrml (unsigned int nTabs=0) const

Protected Attributes

- ve::vec6f m_pos
- unsigned int m_texId
- float szX
- float szY
- bool isPitch

12.37.1 Detailed Description

a class for rendering billboards

Definition at line 30 of file veDeviceGraphicsGL.h.

12.37.2 Constructor & Destructor Documentation

12.37.2.1 ve::glBillboard::glBillboard (const std::string & *texName*, const ve::vec3f & *position*, float *sizeX* = `1.0f`, float *sizeY* = `1.0f`, bool *usePitch* = `false`)

constructor

12.37.2.2 ve::glBillboard::glBillboard (const ve::xml & *xs*)

constructor from an xml statement

12.37.2.3 ve::glBillboard::glBillboard (const ve::glBillboard & *source*)

copy constructor

12.37.3 Member Function Documentation

12.37.3.1 virtual void ve::glBillboard::draw () [virtual]

draws object

Reimplemented from ve::geoObj.

Reimplemented in ve::glBillbAnim.

12.37.3.2 virtual void ve::glBillboard::initGraphics () [virtual]

performs OpenGL initializations.

Reimplemented from ve::geoObj.

12.37.3.3 virtual std::string ve::glBillboard::vrml (unsigned int *nTabs* = 0) const [virtual]

returns geometry as VRML

Reimplemented from ve::geoObj.

12.37.4 Member Data Documentation

12.37.4.1 ve::vec6f ve::glBillboard::m_pos [protected]

stores position

Definition at line 47 of file veDeviceGraphicsGL.h.

12.37.4.2 unsigned int ve::glBillboard::m_texId [protected]

stores texture id

Definition at line 49 of file veDeviceGraphicsGL.h.

12.37.4.3 float ve::glBillboard::szX [protected]

stores X size

Definition at line 51 of file veDeviceGraphicsGL.h.

12.37.4.4 float ve::glBillboard::szY [protected]

stores Y size

Definition at line 53 of file veDeviceGraphicsGL.h.

12.37.4.5 bool ve::glBillboard::isPitch [protected]

stores whether billboard shall be pitched according to camera

Definition at line 55 of file veDeviceGraphicsGL.h.

The documentation for this class was generated from the following file:

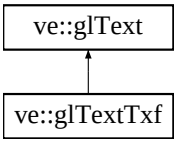
- veDeviceGraphicsGL.h

12.38 ve::glText Class Reference

an abstract base class for OpenGL font renderers.

`#include <veGlUtils.h>`

Inheritance diagram for ve::glText::



Public Member Functions

- glText (const std::string &fontName, float fontSize)
- virtual void draw (const std::string &s, float x, float y, float z=0)=0
- virtual void multiLineDraw (const std::string &s, float x, float y, float z=0, ve::align_t align=ve::ALIGN_LEFT)=0
- virtual float w (const std::string &str) const =0
- float ascent () const
- float descent () const
- float fontSize () const

Protected Attributes

- float fntSize
- float maxAscent
- float maxDescent

12.38.1 Detailed Description

an abstract base class for OpenGL font renderers.

Definition at line 58 of file veGlUtils.h.

12.38.2 Constructor & Destructor Documentation

12.38.2.1 ve::glText::glText (const std::string & *fontName*, float *fontSize*)

constructor.

Parameters:

fontName defines font file name and path,

fontSize (optional, default=24) defines font size.

12.38.3 Member Function Documentation

12.38.3.1 virtual void ve::glText::draw (const std::string & *s*, float *x*, float *y*, float *z* = 0) [pure virtual]

draws string s at position x|y|z.

Implemented in ve::glTextTxf.

12.38.3.2 virtual void ve::glText::multiLineDraw (const std::string & *s*, float *x*, float *y*, float *z* = 0, ve::align_t *align* = ve::ALIGN_LEFT) [pure virtual]

draws a string containing new lines at position x|y|z.

Implemented in ve::glTextTxf.

12.38.3.3 virtual float ve::glText::w (const std::string & *str*) const [pure virtual]

returns width of a string

Implemented in ve::glTextTxf.

12.38.3.4 float ve::glText::ascent () const [inline]

returns maximum ascent of the current font

Definition at line 73 of file veGlUtils.h.

References maxAscent.

12.38.3.5 float ve::glText::descent () const [inline]

returns maximum descent of the current font

Definition at line 75 of file veGlUtils.h.

References maxDescent.

12.38.3.6 float ve::glText::fontSize () const [inline]

returns font size in point

Definition at line 77 of file veGlUtils.h.

References fntSize.

12.38.4 Member Data Documentation

12.38.4.1 float ve::glText::fntSize [protected]

stores font size

Definition at line 77 of file veGlUtils.h.

Referenced by fontSize().

12.38.4.2 float ve::glText::maxAscent [protected]

stores maxAscent of the font.

Definition at line 82 of file veGlUtils.h.

Referenced by ascent().

12.38.4.3 float ve::glText::maxDescent [protected]

stores maxDescent of the font.

Definition at line 84 of file veGlUtils.h.

Referenced by descent().

The documentation for this class was generated from the following file:

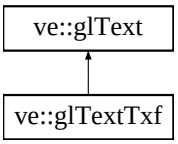
- veGlUtils.h

12.39 ve::glTextTxf Class Reference

a class for rendering txf texture fonts in OpenGL.

`#include <veGlUtils.h>`

Inheritance diagram for ve::glTextTxf::



Public Member Functions

- glTextTxf (const std::string &fontName, float fontSize=24)
- virtual ~glTextTxf ()
- virtual void draw (const std::string &s, float x, float y, float z=0)
- virtual void multiLineDraw (const std::string &s, float x, float y, float z=0, ve::align_t align=ve::ALIGN_LEFT)
- virtual float w (const std::string &str) const

Protected Attributes

- TexFont * txf

12.39.1 Detailed Description

a class for rendering txf texture fonts in OpenGL.

This class is mainly an improved wrapper for MJKs standard C txf functions. For more information about the txf format in general, also about the creation of font sets, see `http://www.sgi.com/software/opengl/examples/glut/texfont/`

The original license statement of MJK:

This program part is freely distributable without licensing fees and is provided without guarantee or warrantee expressed or implied. This program part is -not- in the public domain.

Copyright (c) partially by Mark J. Kilgard, 1997

Copyright (c) partially by Gerald Franz, 2002

Author:
gf, based on the work of Mark J. Kilgard
Revision
2.5

Definition at line 108 of file veGlUtils.h.

12.39.2 Constructor & Destructor Documentation

12.39.2.1 ve::glTextTxf::glTextTxf (const std::string & *fontName*, float *fontSize* = 24)

constructor.

Parameters:

fontName defines font file name and path,

fontSize (optional, default=24) defines font size.

12.39.2.2 virtual ve::glTextTxf::~glTextTxf () [virtual]

destructor

12.39.3 Member Function Documentation

12.39.3.1 virtual void ve::glTextTxf::draw (const std::string & *s*, float *x*, float *y*, float *z* = 0) [virtual]

draws the string str at position x|y|z.

Implements ve::glText.

12.39.3.2 virtual void ve::glTextTxf::multiLineDraw (const std::string & *s*, float *x*, float *y*, float *z* = 0, ve::align_t *align* = ve::ALIGN_LEFT) [virtual]

draws a string containing new lines at position x|y|z.

Implements ve::glText.

12.39.3.3 virtual float ve::glTextTxf::w (const std::string & *str*) const [virtual]

returns width of a string

Implements ve::glText.

12.39.4 Member Data Documentation

12.39.4.1 TexFont* ve::glTextTxf::txf [protected]

pointer to the internal TexFont class

Definition at line 126 of file veGlUtils.h.

The documentation for this class was generated from the following file:

- veGlUtils.h

12.40 ve::glTexture Class Reference

a class for OpenGL image and texture operations.

`#include <veImage.h>`

Public Member Functions

- int byteDepth () const

Static Public Member Functions

- static int load (const std::string &name, unsigned int &id, int &bytesPerPixel, int &width, int &height, bool isRepeated=true)
- static int load (const std::string &name)
- static int load (const std::string &name, unsigned int &id, const ve::image &img, bool isRepeated=true)
- static void del (unsigned int id)
- static void clear ()
- static int grabScreen (const std::string &fileName, unsigned int screenW, unsigned int screenH)
- static void addSearchPath (const std::string &path)
- static ve::glTexture * get (const std::string &name)
- static void enableTextureCompression (bool enable)
- static void enableAnisotropicFilter (float quality)

Protected Attributes

- std::string url
- unsigned int id
- int nRefs
- int bytesPerPixel
- int width
- int height
- bool repeat

Static Protected Attributes

- static std::vector< ve::glTexture > table
- static std::vector< std::string > texPath
- static bool m_bCompressTextures
- static float m_fAFQuality

12.40.1 Detailed Description

a class for OpenGL image and texture operations.

This class simplifies the loading of OpenGL textures and internally manages multiple instances of the same texture. For file input/output ve::image is used.

Author:
gf

Version:
2.3

Definition at line 121 of file veImage.h.

12.40.2 Member Function Documentation

12.40.2.1 static int ve::glTexture::load (const std::string & *name*, unsigned int & *id*, int & *bytesPerPixel*, int & *width*, int & *height*, bool *isRepeated* = `true`) `[static]`

loads up an image file into texture memory.

Supported are all ve::image types that can be loaded by ve::image.

Parameters:
- ***name*** is the filename, it is also used as identifier for checking whether the texture is already loaded.
- ***id*** is the OpenGL texture id, necessary for binding.
- ***bytesPerPixel*** as defined by OpenGL
- ***width*** of the image
- ***height*** of the image
- ***isRepeated*** (optional) defines whether the texture is repeated or clamped.

Returns:
0, if all is ok, otherwise an error code.

12.40.2.2 static int ve::glTexture::load (const std::string & *name*) `[static]`

loads up an image file into texture memory, simplified form.

Supported are all ve::image types that can be loaded by ve::image.

Returns:
the texture id if all is ok, otherwise -1

12.40.2.3 static int ve::glTexture::load (const std::string & *name*, unsigned int & *id*, const **ve::image** & *img*, bool *isRepeated* = `true`) `[static]`

loads up a image object into texture memory.

Parameters:

name is used as identifier for checking whether the texture is already loaded.

id is the OpenGL texture id, necessary for binding.

img the actual image

isRepeated (optional) defines whether the texture is repeated or clamped.

Returns:

0, if all is ok, otherwise an error code.

12.40.2.4 static void ve::glTexture::del (unsigned int *id*) [static]

deletes a previously defined texture.

The texture is deleted in the veTexture table and glDeleteTextures is called for freeing the OpenGL texture memory.

Parameters:

id is the OpenGL texture id for identifying the texture.

12.40.2.5 static void ve::glTexture::clear () [static]

deletes all previously defined textures

calls del() for all textures in the texture table.

12.40.2.6 static int ve::glTexture::grabScreen (const std::string & *fileName*, unsigned int *screenW*, unsigned int *screenH*) [static]

grabs screen content and saves it as a TGA file named fileName

12.40.2.7 static void ve::glTexture::addSearchPath (const std::string & *path*) [static]

adds a texture search path

12.40.2.8 static ve::glTexture* ve::glTexture::get (const std::string & *name*) [static]

returns a pointer to texture defined by name or NULL

12.40.2.9 int ve::glTexture::byteDepth () const [inline]

returns the number of bytes per pixel

Definition at line 165 of file veImage.h.

12.40.2.10 static void ve::glTexture::enableTextureCompression (bool *enable*) [static]

enables/disables load time texture compression

12.40.2.11 static void ve::glTexture::enableAnisotropicFilter (float *quality*) [static]

enables/disables anisotropic texture filtering

Parameters:

quality The desired quality level of the filtering. Common values are 2.0, 4.0 and 8.0. 1.0 means no AF. Higher values will produce nicer images but may slow down rendering significantly. If you set this to a higher value than your graphics hardware supports, this function will automatically reduce AF to the highest possible level.

12.40.3 Member Data Documentation

12.40.3.1 std::string ve::glTexture::url [protected]

stores filename

Definition at line 176 of file veImage.h.

12.40.3.2 unsigned int ve::glTexture::id [protected]

stores OpenGL texture id

Definition at line 178 of file veImage.h.

12.40.3.3 int ve::glTexture::nRefs [protected]

stores number of references

Definition at line 180 of file veImage.h.

12.40.3.4 int ve::glTexture::bytesPerPixel [protected]

stores number of bytes of texture

Definition at line 182 of file veImage.h.

12.40.3.5 int ve::glTexture::width [protected]

stores texture's width in pixels

Definition at line 184 of file veImage.h.

12.40.3.6 int ve::glTexture::height [protected]

stores texture's height in pixels

Definition at line 186 of file veImage.h.

12.40.3.7 bool ve::glTexture::repeat [protected]

stores texture's OpenGL wrap mode.
true means GL_REPEAT, false GL_CLAMP.
Definition at line 189 of file veImage.h.

12.40.3.8 std::vector<ve::glTexture> ve::glTexture::table [static, protected]

stores references for checking which texture is already in memory
Definition at line 191 of file veImage.h.

12.40.3.9 std::vector<std::string> ve::glTexture::texPath [static, protected]

stores texture search paths
Definition at line 193 of file veImage.h.

12.40.3.10 bool ve::glTexture::m_bCompressTextures [static, protected]

compress textures at load time?
Definition at line 195 of file veImage.h.

12.40.3.11 float ve::glTexture::m_fAFQuality [static, protected]

anisotropic texture filter quality setting
Definition at line 197 of file veImage.h.
The documentation for this class was generated from the following file:

- veImage.h

12.41 ve::image Class Reference

ve::image is a class for basic image file input/output.

`#include <veImage.h>`

Public Member Functions

- image (const std::string &filename="")
- image (const unsigned char *pData, unsigned int w, unsigned int h, unsigned int byteDepth)
- image (const char **ch)
- image (const ve::image &source)
- ~image ()
- const ve::image & operator= (const ve::image &source)
- const unsigned char *const data () const
- unsigned char *& data ()
- unsigned int w () const
- unsigned int h () const
- unsigned int bytesPerPixel () const
- unsigned int pixel (unsigned int x, unsigned int y) const
- void pixel (unsigned int x, unsigned int y, unsigned int value)
- unsigned int valueAt (float x, float y) const
- void rgb2gray ()
- void resize (unsigned int sizeX, unsigned int sizeY)
- int save (const std::string &filename) const
- int read (const std::string &filename)
- int readPng (FILE *fp)
- int readJpg (FILE *fp)
- int readBmp (const std::string &fileName)

Protected Attributes

- unsigned char * Data
- unsigned int Width
- unsigned int Height
- unsigned int BytesPerPixel

Friends

- std::ostream & operator<< (std::ostream &os, const ve::image &img)

12.41.1 Detailed Description

ve::image is a class for basic image file input/output.

ve::image provides a unified interface for low level graphics loader libraries and a few image manipulating methods. JPEG and PNG images can be loaded via libjpeg or libpng and zlib. For this file format bindings, corresponding settings have to be defined in veConfig.h, the library headers have to be provided for compiling the veLib, and finally, the executables have to be linked with the low level loader libs.

Author:
gf
Revision
2.3

Definition at line 35 of file veImage.h.

12.41.2 Constructor & Destructor Documentation

12.41.2.1 ve::image::image (const std::string & *filename* = "")

default constructor or constructor loading from file.

12.41.2.2 ve::image::image (const unsigned char * *pData*, unsigned int *w*, unsigned int *h*, unsigned int *byteDepth*)

constructor from memory data.

12.41.2.3 ve::image::image (const char ** *ch*)

constructor from XPM memory data.

12.41.2.4 ve::image::image (const ve::image & *source*) [inline]

copy constructor.

Definition at line 44 of file veImage.h.

References Data, and operator=().

12.41.2.5 ve::image::~image ()

destructor.

12.41.3 Member Function Documentation

12.41.3.1 const ve::image& ve::image::operator= (const ve::image & *source*)

copy operator.

Referenced by image().

12.41.3.2 const unsigned char* const ve::image::data () const [inline]

allows read access to raw data, const.

Definition at line 51 of file veImage.h.

References Data.

12.41.3.3 unsigned char* & ve::image::data () [inline]

allows full access to raw data.

Definition at line 53 of file veImage.h.

References Data.

12.41.3.4 unsigned int ve::image::w () const [inline]

returns image width in pixels

Definition at line 55 of file veImage.h.

References Width.

12.41.3.5 unsigned int ve::image::h () const [inline]

returns image height in pixels

Definition at line 57 of file veImage.h.

References Height.

12.41.3.6 unsigned int ve::image::bytesPerPixel () const [inline]

returns bytes per pixel

Definition at line 59 of file veImage.h.

References BytesPerPixel.

12.41.3.7 unsigned int ve::image::pixel (unsigned int *x*, unsigned int *y*) const

returns pixel value at x|y.

If coordinate is out of range, 0 is returned. If it is a color image, return value encodes RGB values: byte0=R, byte1=G, byte2=B, byte3=A

12.41.3.8 void ve::image::pixel (unsigned int *x*, unsigned int *y*, unsigned int *value*)

sets pixel value at x|y.

If coordinate is out of range, nothing happens. If it is a color image, value has to encode the RGB values: byte0=R, byte1=G, byte2=B, byte3=A

12.41.3.9 unsigned int ve::image::valueAt (float *x*, float *y*) const

returns linearly interpolated value at x|y.

The image's limits are treated as 0|0 and 1|1. If coordinate is out of this range, the next border value is returned. If it is a color image, return value encodes RGB values: byte0=R, byte1=G, byte2=B, byte3=A

12.41.3.10 void ve::image::rgb2gray ()

converts color image int grayscale image

12.41.3.11 void ve::image::resize (unsigned int *sizeX*, unsigned int *sizeY*)

resizes image using bilinear filtering.

12.41.3.12 int ve::image::save (const std::string & *filename*) const

saves image data as tga file. under devolopment, experimental!

12.41.3.13 int ve::image::read (const std::string & *filename*)

tries to detect file type and reads image

12.41.3.14 int ve::image::readPng (FILE * *fp*)

reads a PNG image via libpng and zlib.
veImage must be compiled with _HAVE_LIBPNG anv _HAVE_LIBZ defined in veConfig.h.

12.41.3.15 int ve::image::readJpg (FILE * *fp*)

reads a JPEG image via libjpeg.
veImage must be compiled with _HAVE_LIBJPEG defined in veConfig.h.

12.41.3.16 int ve::image::readBmp (const std::string & *fileName*)

reads a BMP image via libSDL.
veImage must be compiled with _HAVE_SDL defined in veConfig.h.

12.41.4 Friends And Related Function Documentation

12.41.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::image & *img*) [friend]

operator for output in streams

12.41.5 Member Data Documentation

12.41.5.1 unsigned char* ve::image::Data [protected]

internal pointer to texture memory

Definition at line 96 of file veImage.h.

Referenced by data(), and image().

12.41.5.2 unsigned int ve::image::Width [protected]

stores image width and height

Definition at line 98 of file veImage.h.

Referenced by w().

12.41.5.3 unsigned int ve::image::Height [protected]

stores image width and height

Definition at line 98 of file veImage.h.

Referenced by h().

12.41.5.4 unsigned int ve::image::BytesPerPixel [protected]

stores bytes per pixel

Definition at line 100 of file veImage.h.

Referenced by bytesPerPixel().

The documentation for this class was generated from the following file:

- veImage.h

12.42 ve::io3ds Class Reference

this class handles the loading of 3ds files

`#include <veIo3ds.h>`

Public Member Functions

- io3ds ()
- int load (const std::string &strFileName, std::vector< ve::geoObj * > &vMesh)

Protected Member Functions

- int getString (char *)
- void readChunk (_chunk *)
- void processNextChunk (_chunk *)
- void processNextObjectChunk (_3dsObject *pObject, _chunk *)
- void processNextMaterialChunk (_chunk *)
- void readColorChunk (_materialInfo *pMaterial, _chunk *pChunk)
- void readTranspChunk (_materialInfo *pMaterial, _chunk *pChunk)
- void readVertices (_3dsObject *pObject, _chunk *)
- void readVertexIndices (_3dsObject *pObject, _chunk *)
- void readUVCoordinates (_3dsObject *pObject, _chunk *)
- void readObjectMaterial (_3dsObject *pObject, _chunk *pPreviousChunk)
- void cleanUp ()

Protected Attributes

- std::vector< _3dsObject > vChildren
- std::vector< _materialInfo > vMaterial
- FILE * m_FilePointer
- _chunk * m_CurrentChunk
- _chunk * m_TempChunk
- int buffer [50000]

12.42.1 Detailed Description

this class handles the loading of 3ds files

Definition at line 87 of file veIo3ds.h.

12.42.2 Constructor & Destructor Documentation

12.42.2.1 ve::io3ds::io3ds ()

constructor initializing the data members

12.42.3 Member Function Documentation

12.42.3.1 int ve::io3ds::load (const std::string & *strFileName*, std::vector< ve::geoObj * > & *vMesh*)

Loads the 3ds geometry into a vector of pointers to geoMesh objects.

12.42.3.2 int ve::io3ds::getString (char *) [protected]

This reads in a string and saves it in the char array passed in.

12.42.3.3 void ve::io3ds::readChunk (_chunk *) [protected]

This reads the next chunk.

12.42.3.4 void ve::io3ds::processNextChunk (_chunk *) [protected]

This reads the next large chunk.

12.42.3.5 void ve::io3ds::processNextObjectChunk (_3dsObject * *pObject*, _chunk *) [protected]

This reads the object chunks.

12.42.3.6 void ve::io3ds::processNextMaterialChunk (_chunk *) [protected]

This reads the material chunks.

12.42.3.7 void ve::io3ds::readColorChunk (_materialInfo * *pMaterial*, _chunk * *pChunk*) [protected]

This reads the RGB value for the object's color.

12.42.3.8 void ve::io3ds::readTranspChunk (_materialInfo * *pMaterial*, _chunk * *pChunk*) [protected]

this method reads transparency data

12.42.3.9 void ve::io3ds::readVertices (_3dsObject * *pObject*, _chunk *) [protected]

This reads the objects vertices.

12.42.3.10 void ve::io3ds::readVertexIndices (_3dsObject * *pObject*, _chunk *) [protected]

This reads the objects face information.

12.42.3.11 void ve::io3ds::readUVCoordinates (_3dsObject * *pObject*, _chunk *) [protected]

This reads the texture coodinates of the object.

12.42.3.12 void ve::io3ds::readObjectMaterial (_3dsObject * *pObject*, _chunk * *pPreviousChunk*) [protected]

This reads in the material name assigned to the object and sets the materialID.

12.42.3.13 void ve::io3ds::cleanUp () [protected]

This frees memory and closes the file.

12.42.4 Member Data Documentation

12.42.4.1 std::vector<_3dsObject> ve::io3ds::vChildren [protected]

stores children

Definition at line 95 of file veIo3ds.h.

12.42.4.2 std::vector<_materialInfo> ve::io3ds::vMaterial [protected]

stores materials

Definition at line 97 of file veIo3ds.h.

12.42.4.3 FILE* ve::io3ds::m_FilePointer [protected]

The file pointer.

Definition at line 125 of file veIo3ds.h.

12.42.4.4 _chunk* ve::io3ds::m_CurrentChunk [protected]

this pointer is used through the loading process to hold the chunk information

Definition at line 127 of file veIo3ds.h.

12.42.4.5 _chunk* ve::io3ds::m_TempChunk [protected]

this pointer is used through the loading process to hold the chunk information

Definition at line 129 of file velo3ds.h.

12.42.4.6 int ve::io3ds::buffer[50000] [protected]

this buffer is used to skip unwanted data when reading

Definition at line 131 of file velo3ds.h.

The documentation for this class was generated from the following file:

- velo3ds.h

12.43 ve::ioFileHandler Class Reference

a small auxilliary class that allows the writing of plugin file handlers

`#include <veGeoObj.h>`

Public Member Functions

- ioFileHandler (geoObj *(*loadFunc)(const std::string &), const std::string &suffix)

Static Public Member Functions

- static ve::geoObj * load (const std::string &filename)

Static Protected Attributes

- static std::vector< ve::geoObj *(*)(const std::string &)> vLoadFunc
- static std::vector< std::string > vSuffix

12.43.1 Detailed Description

a small auxilliary class that allows the writing of plugin file handlers

Definition at line 27 of file veGeoObj.h.

12.43.2 Constructor & Destructor Documentation

12.43.2.1 ve::ioFileHandler::ioFileHandler (geoObj *(*)(const std::string &) *loadFunc*, const std::string & *suffix*) [inline]

constructor that just registers a loader function

use a static object to register the file handler function at start up. The function has to be of type geoObj* loadXYZ(const std::string & filename).

Definition at line 32 of file veGeoObj.h.

References vLoadFunc, and vSuffix.

12.43.3 Member Function Documentation

12.43.3.1 static ve::geoObj* ve::ioFileHandler::load (const std::string & *filename*) [static]

tries to load a file using previously registered loader functions

Parameters:

filename path to the file to be loaded, type will be identified by suffix

Returns:
pointer to loaded model or 0.

12.43.4 Member Data Documentation

12.43.4.1 std::vector<ve::geoObj* (*)(const std::string &)> ve::ioFileHandler::vLoadFunc [static, protected]

vector to store registered loader functions

Definition at line 40 of file veGeoObj.h.

Referenced by ioFileHandler().

12.43.4.2 std::vector<std::string> ve::ioFileHandler::vSuffix [static, protected]

vector to store registered file suffixes

Definition at line 42 of file veGeoObj.h.

Referenced by ioFileHandler().

The documentation for this class was generated from the following file:

- veGeoObj.h

12.44 ve::ioVrml Class Reference

a class for VRML input/output

`#include <veGeoObj.h>`

Static Public Member Functions

- static int toX3d (const std::string &filename, ve::xml &xs)
- static int toGeo (const std::string &filename, std::vector< ve::geoObj * > &vObj)
- static void vrml2ve (std::vector< ve::vec3f > &vCoord)

12.44.1 Detailed Description

a class for VRML input/output

Definition at line 48 of file veGeoObj.h.

12.44.2 Member Function Documentation

12.44.2.1 static int ve::ioVrml::toX3d (const std::string & *filename*, ve::xml & *xs*) [static]

converts vrml geometry from a file to x3d.

Parameters:

filename the file to be converted,

xs reference of ve::xml object that is filled with the geometry information,

Returns:

0 in case of success.

12.44.2.2 static int ve::ioVrml::toGeo (const std::string & *filename*, std::vector< ve::geoObj * > & *vObj*) [static]

interprets VRML file as geoObjects.

Currently solely IndexedFaceSet, ElevationGrid, Material, and Transform nodes are interpreted. This static method first converts the VRML file into X3D, then the X3D geometry definition is interpreted.

Parameters:

filename the file to be loaded,

vObj is filled with interpreted geoObj.

Returns:

0 in case of success.

12.44.2.3 static void ve::ioVrml::vrml2ve (std::vector< ve::vec3f > & *vCoord*) [static]

converts a vector of vertices from vrml coordinate system to veLib coordinate system

The documentation for this class was generated from the following file:

- veGeoObj.h

12.45 ve::ioX3d Class Reference

a class for X3d input/output

`#include <veGeoObj.h>`

Static Public Member Functions

- static int toGeo (const std::string &filename, std::vector< ve::geoObj * > &vObj)
- static int toGeo (ve::xml &xs, std::vector< ve::geoObj * > &vObj)
- static int interpretNode (ve::xml &xs, std::vector< ve::geoObj * > &vObj, ve::matStack4f &m)
- static ve::mat4f interpretTransform (const ve::xml &xs, bool vrml2ve=true)

12.45.1 Detailed Description

a class for X3d input/output

Definition at line 69 of file veGeoObj.h.

12.45.2 Member Function Documentation

12.45.2.1 static int ve::ioX3d::toGeo (const std::string & *filename*, std::vector< ve::geoObj * > & *vObj*) [static]

interprets X3D file as geoObjects.

Currently solely IndexedFaceSet, ElevationGrid, Material, and Transform nodes are interpreted.

Parameters:

filename the file to be loaded.

vObj is filled with interpreted geoObj.

Returns:

0 in case of success.

12.45.2.2 static int ve::ioX3d::toGeo (ve::xml & *xs*, std::vector< ve::geoObj * > & *vObj*) [static]

interprets x3d defined geometry as geoObjects.

Currently solely IndexedFaceSet, ElevationGrid, Material, and Transform nodes are interpreted.

Parameters:

xs an xml statement that is interpreted and potentially changed, if DEFINEs have to be resolved,

vObj is filled with interpreted geoObj.

Returns:

0 in case of success.

12.45.2.3 static int ve::ioX3d::interpretNode (ve::xml & *xs*, std::vector< ve::geoObj * > & *vObj*, ve::matStack4f & *m*) [static]

tries to interprets xs as X3D node

12.45.2.4 static ve::mat4f ve::ioX3d::interpretTransform (const ve::xml & *xs*, bool *vrml2ve* = true) [static]

interprets a single X3D defined transform node

Parameters:

xs the xml statement to be parsed

vrml2ve defines whether the transformation shall be translated from VRML to the veLib coordinate system

Returns:

a matrix containing all transformations.

The documentation for this class was generated from the following file:

- veGeoObj.h

12.46 ve::line Class Reference

a class for line mathematics.

`#include <veMath.h>`

Public Member Functions

- line ()
- line (float x1_, float y1_, float z1_, float x2_, float y2_, float z2_)
- line (float x1_, float y1_, float x2_, float y2_)
- line (const vec3f &p1, const vec3f &p2)
- line (const line &source)
- ~line ()
- const line & operator= (const line &source)
- void set (float x1_, float y1_, float z1_, float x2_, float y2_, float z2_)
- void set (const vec3f &p1, const vec3f &p2)
- void get (vec3f &p1, vec3f &p2) const
- void translate (float dx, float dy, float dz=0)
- void translate (const vec3f &v, float n=1.0f)
- void rotate (float angle, float x, float y, float z)
- void scale (float sx, float sy, float sz=1)
- void transform (const ve::mat4f &m)
- void transform (const ve::vec6f &sdof)
- vec3f & operator[] (unsigned int n)
- const vec3f & operator[] (unsigned int n) const
- float distTo (const vec3f &p) const
- float length () const
- float sqrLength () const
- float intersect2d (line &l) const
- vec3f * intersection (const triangle &tr, bool isRay=true) const
- bool intersects (const triangle &tr, bool isRay=true) const
- vec3f * intersection (const ve::vec3f &tr0, const ve::vec3f &tr1, const ve::vec3f &tr2, bool isRay=true) const
- bool intersects (const ve::vec3f &tr0, const ve::vec3f &tr1, const ve::vec3f &tr2, bool isRay=true) const
- vec3f * intersection (const std::vector< ve::triangle > &vTr, bool isRay=true) const
- bool intersects (const std::vector< ve::triangle > &vTr, bool isRay=true) const
- vec3f * intersection (const sphere &sph, bool isRay=true) const
- vec3f * intersection (const ve::vec3f &center, float radius, bool isRay=true) const

Protected Attributes

- vec3f * pt

Friends

- std::ostream & operator<< (std::ostream &os, const ve::line &l)

12.46.1 Detailed Description

a class for line mathematics.

Depending on the method, the line is treated as a

1. line segment (limited by the 2 control vertices): distTo, length, squaredLength, intersect2d, intersects(isRay=false)
2. ray with pt[0] as origin and pt[0]pt[1] as direction: intersects(isRay=true)
3. infinite line.

Definition at line 628 of file veMath.h.

12.46.2 Constructor & Destructor Documentation

12.46.2.1 ve::line::line ()

default constructor

12.46.2.2 ve::line::line (float *x1_*, float *y1_*, float *z1_*, float *x2_*, float *y2_*, float *z2_*)

explicit 3d constructor

12.46.2.3 ve::line::line (float *x1_*, float *y1_*, float *x2_*, float *y2_*)

explicit 2d constructor

12.46.2.4 ve::line::line (const vec3f & *p1*, const vec3f & *p2*)

constructor taking vertices as argument

12.46.2.5 ve::line::line (const line & *source*)

copy constructor

12.46.2.6 ve::line::~line ()

destructor

12.46.3 Member Function Documentation

12.46.3.1 const line& ve::line::operator= (const line & *source*)

copy operator

12.46.3.2 void ve::line::set (float *x1_*, float *y1_*, float *z1_*, float *x2_*, float *y2_*, float *z2_*)

sets the coordinates of the two control vertices

12.46.3.3 void ve::line::set (const vec3f & *p1*, const vec3f & *p2*)

sets the two control vertices to p1 & p2

12.46.3.4 void ve::line::get (vec3f & *p1*, vec3f & *p2*) const

returns the current control vec3f values in p1 and p2

12.46.3.5 void ve::line::translate (float *dx*, float *dy*, float *dz* = `0`)

translates object by dx,dy,dz
Referenced by translate().

12.46.3.6 void ve::line::translate (const vec3f & *v*, float *n* = `1.0f`) `[inline]`

translates object by vector v, optionally scaled by factor n
Definition at line 654 of file veMath.h.
References translate().

12.46.3.7 void ve::line::rotate (float *angle*, float *x*, float *y*, float *z*) `[inline]`

rotates object around arbitrary axis from origin to p by angle.
Definition at line 656 of file veMath.h.
References pt, and ve::vec3f::rotate().

12.46.3.8 void ve::line::scale (float *sx*, float *sy*, float *sz* = `1`)

scales object by sx,sy and optionally sz

12.46.3.9 void ve::line::transform (const ve::mat4f & *m*)

transforms this line by multiplying it with matrix m

12.46.3.10 void ve::line::transform (const ve::vec6f & *sdof*)

transforms this line by applying the provided sixdof transformation.
The vector is rotated first according to r,p,h, afterwards translated by x,y,z.

12.46.3.11]

vec3f& ve::line::operator[] (unsigned int *n*) [inline]

returns control vec3f n

Definition at line 667 of file veMath.h.

References pt.

12.46.3.12]

const vec3f& ve::line::operator[] (unsigned int *n*) const [inline]

returns control vec3f n, const

Definition at line 669 of file veMath.h.

References pt.

12.46.3.13 float ve::line::distTo (const vec3f & *p*) const

returns distance to vertex p

12.46.3.14 float ve::line::length () const [inline]

returns length

Definition at line 673 of file veMath.h.

References pt.

12.46.3.15 float ve::line::sqrLength () const [inline]

returns the squared length.

This method is provided mainly for efficiency reasons for avoiding square root computations.

Definition at line 676 of file veMath.h.

References pt.

12.46.3.16 float ve::line::intersect2d (line & *l*) const

tests for intersection, projected in xy plane.

Parameters:

l the line segment to tested

Returns:

distance to intersection point or -1 if no intersection has been found.

12.46.3.17 vec3f* ve::line::intersection (const triangle & *tr*, bool *isRay* = `true`) const [inline]

calculates intersection with triangle tr.

Parameters:

tr is a reference to the triangle that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

NULL or intersection point. The memory of this vec has to be deallocated by the user!

Definition at line 689 of file veMath.h.

Referenced by intersection().

12.46.3.18 bool ve::line::intersects (const triangle & *tr*, bool *isRay* = `true`) const [inline]

tests for intersection between a line (segment) and a triangle.

This method is much faster than the corresponding intersection() method, because it does not compute the exact intersection.

Parameters:

tr is a reference to the triangle that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

false or true

Definition at line 698 of file veMath.h.

12.46.3.19 vec3f* ve::line::intersection (const ve::vec3f & *tr0*, const ve::vec3f & *tr1*, const ve::vec3f & *tr2*, bool *isRay* = `true`) const

calculates intersection with triangle (tr0|tr1|tr2).

Parameters:

tr0 is a reference to the first vertex of the triangle that is tested

tr1 is a reference to the second vertex of the triangle that is tested

tr2 is a reference to the third vertex of the triangle that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

NULL or intersection point. The memory of this vec has to be deallocated by the user!

12.46.3.20 bool ve::line::intersects (const ve::vec3f & *tr0*, const ve::vec3f & *tr1*, const ve::vec3f & *tr2*, bool *isRay* = `true`) const

tests for intersection between a line (segment) and triangle (tr0|tr1|tr2).

This method is much faster than the corresponding intersection() method, because it does not compute the exact intersection.

Parameters:

tr0 is a reference to the first vertex of the triangle that is tested

tr1 is a reference to the second vertex of the triangle that is tested

tr2 is a reference to the third vertex of the triangle that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

false or true

12.46.3.21 vec3f* ve::line::intersection (const std::vector< ve::triangle > & *vTr*, bool *isRay* = `true`) const

returns intersection point between a line (segment) and a triangle array.

Parameters:

vTr is a reference to the triangle array that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

NULL or intersection point. The memory of this vec has to be deallocated by the user!

12.46.3.22 bool ve::line::intersects (const std::vector< ve::triangle > & *vTr*, bool *isRay* = `true`) const

tests for intersection between a line (segment) and a triangle array.

This method is much faster than the corresponding intersection() method, because it does not compute the exact intersection.

Parameters:

vTr is a reference to the triangle array that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

false or true

12.46.3.23 vec3f* ve::line::intersection (const sphere & *sph*, bool *isRay* = `true`) const `[inline]`

tests for intersection with sphere sph.

Parameters:

sph is a reference to the sphere that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

NULL or intersection point. The memory of this vec has to be deallocated by the user!

Definition at line 745 of file veMath.h.

References intersection(), and ve::sphere::radius().

12.46.3.24 vec3f* ve::line::intersection (const ve::vec3f & *center*, float *radius*, bool *isRay* = `true`) const

tests for intersection with sphere (center|radius).

Parameters:

center the center of the sphere that is tested

radius the radius of the sphere that is tested

isRay (optional) defines whether the line is treated as as infinite ray starting from pt[0].

Returns:

NULL or intersection point. The memory of this vec has to be deallocated by the user!

12.46.4 Friends And Related Function Documentation

12.46.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::line & *l*) `[friend]`

operator for output in streams

12.46.5 Member Data Documentation

12.46.5.1 vec3f* ve::line::pt `[protected]`

pointer to vertices

Definition at line 761 of file veMath.h.

Referenced by length(), operator[](), rotate(), and sqrLength().

The documentation for this class was generated from the following file:

- veMath.h

12.47 ve::mandatoryData Struct Reference

Public Attributes

- ve::dataChar dataCont
- bool send
- unsigned int ack
- double timeout

12.47.1 Detailed Description

Definition at line 93 of file veDeviceNetwork.h.

12.47.2 Member Data Documentation

12.47.2.1 ve::dataChar ve::mandatoryData::dataCont

the data

Definition at line 96 of file veDeviceNetwork.h.

12.47.2.2 bool ve::mandatoryData::send

has it been send?

Definition at line 98 of file veDeviceNetwork.h.

12.47.2.3 unsigned int ve::mandatoryData::ack

how many times has data been send without having received an ack message

Definition at line 100 of file veDeviceNetwork.h.

12.47.2.4 double ve::mandatoryData::timeout

time waited for ack after data has been send

Definition at line 102 of file veDeviceNetwork.h.

The documentation for this struct was generated from the following file:

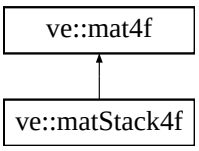
- veDeviceNetwork.h

12.48 ve::mat4f Class Reference

a class for typical 3D geometry 4x4 matrix operations.

`#include <veMath.h>`

Inheritance diagram for ve::mat4f::



Public Member Functions

- mat4f ()
- mat4f (const ve::vec6f &sdof)
- mat4f (const ve::mat4f &source)
- const ve::mat4f & operator= (const ve::mat4f &source)
- float & operator[] (unsigned int i)
- float operator[] (unsigned int i) const
- const ve::vec4f col (unsigned int i) const
- const ve::vec4f row (unsigned int i) const
- void identity ()
- bool isIdentity () const
- void nan ()
- bool isNan () const
- void transpose ()
- void inverse ()
- void set (float m0, float m1, float m2, float m3, float m4, float m5, float m6, float m7, float m8, float m9, float m10, float m11, float m12, float m13, float m14, float m15)
- void set (float *f)
- void set (const ve::vec6f &sdof)
- void vrml2ve ()
- void ve2vrml ()
- const mat4f operator * (const mat4f &m) const
- void operator *= (const ve::mat4f &m2)
- void translate (float x, float y, float z=0)
- void translate (const ve::vec3f &v)
- void rotate (float angle, vec3f p)
- void rotate (float angle, float ax, float ay, float az)
- void scale (float sx, float sy, float sz)
- void transform (std::vector< ve::vec3f > &vV) const
- void transform (std::vector< float > &vF) const
- void transform (float *pF, unsigned int n) const
- std::string str () const

Protected Attributes

- float m [16]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::mat4f &m)

12.48.1 Detailed Description

a class for typical 3D geometry 4x4 matrix operations.

The matrix class is intentionally free of virtual methods and takes exactly 64 bytes memory. This allows for using a pointer to a matrix identically to using float pointers, e.g., for OpenGL. Therefore also the order of tranformations (e.g., rotations, scale) correspond exactly to OpenGL's factor order.

Also the order of its members is equivalent to OpenGL:

```
mat4f M = [ m0  m4 m8  m12
            m1  m5 m9  m13
            m2  m6 m10 m14
            m3  m7 m11 m15 ]
```

Note that this order has changed between veLib v1.0 and veLib 1.0.1! The normal user should not be affected, as long as matrices have not been used directly. However, the switch of the order could also be the origin of subtle bugs. In order to make the user explicitly aware of this change, the class has been renamed from matrix4f to mat4f (which also is equivalent to GLSL).

Definition at line 995 of file veMath.h.

12.48.2 Constructor & Destructor Documentation

12.48.2.1 ve::mat4f::mat4f () [inline]

default constructor, creating identity matrix

Definition at line 998 of file veMath.h.

References identity().

12.48.2.2 ve::mat4f::mat4f (const ve::vec6f & *sdof*) [inline]

constructor from sixdof

Definition at line 1000 of file veMath.h.

References set().

12.48.2.3 ve::mat4f::mat4f (const ve::mat4f & *source*)

copy constructor, bitwise copy.

12.48.3 Member Function Documentation

12.48.3.1 const ve::mat4f& ve::mat4f::operator= (const ve::mat4f & *source*)

copy operator, bitwise copy.

12.48.3.2]

float& ve::mat4f::operator[] (unsigned int *i*) `[inline]`

returns reference of single ordinate

Definition at line 1006 of file veMath.h.

References m.

12.48.3.3]

float ve::mat4f::operator[] (unsigned int *i*) const `[inline]`

returns single ordinate value

Definition at line 1008 of file veMath.h.

References m.

12.48.3.4 const ve::vec4f ve::mat4f::col (unsigned int *i*) const `[inline]`

returns column vector i

Definition at line 1010 of file veMath.h.

References m.

12.48.3.5 const ve::vec4f ve::mat4f::row (unsigned int *i*) const `[inline]`

returns row vector i

Definition at line 1013 of file veMath.h.

References m.

12.48.3.6 void ve::mat4f::identity ()

transforms this matrix to an identity matrix

Referenced by mat4f(), and ve::geoGroup::resetTransform().

12.48.3.7 bool ve::mat4f::isIdentity () const

returns true if this matrix is an identity matrix

12.48.3.8 void ve::mat4f::nan ()

makes this matrix to a NaN matrix

The NaN matrix is used to indicate that no mathematical solution could be found in an operation. To test for NaN, use the mat4f::isNan() method.

12.48.3.9 bool ve::mat4f::isNan () const [inline]

returns true if this matrix is a NaN matrix

Definition at line 1025 of file veMath.h.

References m.

12.48.3.10 void ve::mat4f::transpose ()

transposes this matrix

12.48.3.11 void ve::mat4f::inverse ()

transforms this matrix into its inverse if possible, otherwise into a NaN matrix

12.48.3.12 void ve::mat4f::set (float *m0*, float *m1*, float *m2*, float *m3*, float *m4*, float *m5*, float *m6*, float *m7*, float *m8*, float *m9*, float *m10*, float *m11*, float *m12*, float *m13*, float *m14*, float *m15*)

sets this matrix to the 16 provided values

the values are assumed to be in the OpenGL order.

Referenced by mat4f().

12.48.3.13 void ve::mat4f::set (float * *f*)

sets this matrix to the values provided in a float array

the values are assumed to be in the OpenGL order.

12.48.3.14 void ve::mat4f::set (const ve::vec6f & *sdof*)

sets this matrix to transformation stored in a sixdof

12.48.3.15 void ve::mat4f::vrml2ve ()

flips this matrix from the vrml/x3d coordinate system to the veLib coordinate system

12.48.3.16 void ve::mat4f::ve2vrml ()

flips this matrix from the veLib coordinate system to the vrml/x3d coordinate system

12.48.3.17 const mat4f ve::mat4f::operator ∗ (const mat4f & *m*) const

operator multiplying two matrixes in the order (∗this) ∗ m

12.48.3.18 void ve::mat4f::operator ∗= (const ve::mat4f & *m2*) [inline]

multiplies this matrix by m2, (∗this) = (∗this) ∗ m2

Definition at line 1050 of file veMath.h.

12.48.3.19 void ve::mat4f::translate (float *x*, float *y*, float *z* = 0) [inline]

adds the translation x|y|z to the transformation

Definition at line 1052 of file veMath.h.

12.48.3.20 void ve::mat4f::translate (const ve::vec3f & *v*)

adds the translation v to the transformation

12.48.3.21 void ve::mat4f::rotate (float *angle*, vec3f *p*)

multiplies matrix with a rotation matrix around arbitrary axis from origin to p by angle, (∗this) = (∗this) ∗ mRot

Referenced by rotate().

12.48.3.22 void ve::mat4f::rotate (float *angle*, float *ax*, float *ay*, float *az*) [inline]

multiplies matrix with a rotation matrix around arbitrary axis from origin to (x|y|z) by angle, (∗this) = (∗this) ∗ mRot

Definition at line 1058 of file veMath.h.

References rotate().

12.48.3.23 void ve::mat4f::scale (float *sx*, float *sy*, float *sz*)

multiplies matrix with a scale matrix, (∗this) = (∗this) ∗ mScale

12.48.3.24 void ve::mat4f::transform (std::vector< ve::vec3f > & *vV*) const

transforms a vector of ve::vec3f by applying this matrix

Referenced by ve::geoElevationGrid::transform(), and ve::geoMesh::transform().

12.48.3.25 void ve::mat4f::transform (std::vector< float > & *vF*) const

transforms a vector of float coordinates n*(x|y|z) by applying this matrix

12.48.3.26 void ve::mat4f::transform (float * *pF*, unsigned int *n*) const

transforms an array of coherent float coordinates n*(x|y|z) starting at pF by applying this matrix

12.48.3.27 std::string ve::mat4f::str () const

returns a string containing all data, optional arguments set separator between ordinates and number of digits per ordinate.

Referenced by ve::operator<<().

12.48.4 Friends And Related Function Documentation

12.48.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::mat4f & *m*) [friend]

operator for output in streams

Definition at line 1080 of file veMath.h.

12.48.5 Member Data Documentation

12.48.5.1 float ve::mat4f::m[16] [protected]

stores values.

Definition at line 1075 of file veMath.h.

Referenced by col(), isNan(), operator[](), and row().

The documentation for this class was generated from the following file:

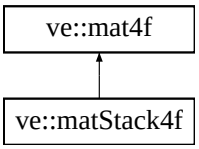
- veMath.h

12.49 ve::matStack4f Class Reference

a class for typical 3D geometry 4x4 matrix stack operations such as in OpenGL.

`#include <veMath.h>`

Inheritance diagram for ve::matStack4f::



Public Member Functions

- matStack4f ()
- matStack4f (const ve::vec6f &sdof)
- matStack4f (const ve::mat4f &source)
- matStack4f (const ve::matStack4f &source)
- void push ()
- void pop ()

Protected Attributes

- std::vector< ve::mat4f > m_stack

12.49.1 Detailed Description

a class for typical 3D geometry 4x4 matrix stack operations such as in OpenGL.

Definition at line 1088 of file veMath.h.

12.49.2 Constructor & Destructor Documentation

12.49.2.1 ve::matStack4f::matStack4f () `[inline]`

default constructor, creating empty stack and identity matrix

Definition at line 1091 of file veMath.h.

12.49.2.2 ve::matStack4f::matStack4f (const ve::vec6f & *sdof*) `[inline]`

constructor from sixdof

Definition at line 1093 of file veMath.h.

12.49.2.3 ve::matStack4f::matStack4f (const ve::mat4f & *source*) [inline]

constructor from normal matrix

Definition at line 1095 of file veMath.h.

12.49.2.4 ve::matStack4f::matStack4f (const ve::matStack4f & *source*) [inline]

copy constructor from matrix stack

Definition at line 1097 of file veMath.h.

References m_stack.

12.49.3 Member Function Documentation

12.49.3.1 void ve::matStack4f::push () [inline]

pushes current matrix onto the stack

Definition at line 1100 of file veMath.h.

References m_stack.

12.49.3.2 void ve::matStack4f::pop () [inline]

pops current matrix and replaces it by last stored matrix

Definition at line 1102 of file veMath.h.

12.49.4 Member Data Documentation

12.49.4.1 std::vector<ve::mat4f> ve::matStack4f::m_stack [protected]

stores pushed matrices

Definition at line 1103 of file veMath.h.

Referenced by matStack4f(), and push().

The documentation for this class was generated from the following file:

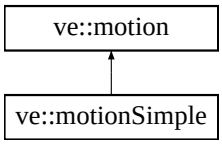
- veMath.h

12.50 ve::motion Class Reference

Base class for all motion model classes.

`#include <veMotion.h>`

Inheritance diagram for ve::motion::



Public Member Functions

- motion (ve::collision *pCollision=NULL)
- virtual ~motion ()
- virtual int updateObject (ve::dataContainer &data, double deltaT) const
- virtual int updateObject (unsigned int objectId, const ve::vec6f &input, ve::vec6f &pos, ve::vec6f &speed, ve::vec6f &accel, double deltaT) const
- virtual int setCollisionClass (ve::collision *pCollision)

Protected Member Functions

- virtual int calculateStatus (unsigned int objectId, const ve::vec6f &input, ve::vec6f &pos, ve::vec6f &vel, ve::vec6f &acc, double deltaT) const

Protected Attributes

- ve::collision * collision_p

12.50.1 Detailed Description

Base class for all motion model classes.

The public user interface of this class is mainly its update() methods that change the values of a dataContainer according to the input and the motion model.

Both update() methods calculate the actual observer status by the same (protected) method calculateStatus(). This method can also (if necessary) apply a collision detection which can be configured by providing a veCollision class. Derived classes (i.e. new motion models) only have to overwrite this method.

Author:
MvdH & gf

Definition at line 43 of file veMotion.h.

12.50.2 Constructor & Destructor Documentation

12.50.2.1 ve::motion::motion (ve::collision * *pCollision* = NULL)

The constructor can be initialized with or without collision detection.

Parameters:

pCollision if set to something != NULL the collision class is configured.

12.50.2.2 virtual ve::motion::~motion () [inline, virtual]

destructor dummy

Definition at line 51 of file veMotion.h.

12.50.3 Member Function Documentation

12.50.3.1 virtual int ve::motion::updateObject (ve::dataContainer & *data*, double *deltaT*) const [virtual]

updates an object. pos, speed and acceleration of the dataContainer are changed according to its input values. This method does not rely on an internal representation and thus is capable of handling multiple objects.

Returns:

number of errors which occured.

12.50.3.2 virtual int ve::motion::updateObject (unsigned int *objectId*, const ve::vec6f & *input*, ve::vec6f & *pos*, ve::vec6f & *speed*, ve::vec6f & *accel*, double *deltaT*) const [virtual]

updates an object. pos, speed and acceleration of an object are changed according to its input values. This method does not rely on an internal representation and thus is capable of handling multiple objects.

Returns:

number of errors which occured.

12.50.3.3 virtual int ve::motion::setCollisionClass (ve::collision * *pCollision*) [virtual]

Set the class which is used for the collision detection in the calculateStatus() function. To switch off collision use value NULL.

Parameters:

pCollision is a pointer to new collision class

Returns:

number of errors which occured

12.50.3.4 virtual int ve::motion::calculateStatus (unsigned int *objectId*, const ve::vec6f & *input*, ve::vec6f & *pos*, ve::vec6f & *vel*, ve::vec6f & *acc*, double *deltaT*) const [protected, virtual]

main motion computation method.

This function represents the dynamic implemented in the motion model. It calcutates the new position, velocity, acceleration and applies the collision detection.

Reimplemented in ve::motionSimple.

12.50.4 Member Data Documentation

12.50.4.1 ve::collision* ve::motion::collision_p [protected]

this pointer keeps the collision detection class used by calculateStatus(). It is set by setCollisionClass().

Definition at line 83 of file veMotion.h.

The documentation for this class was generated from the following file:

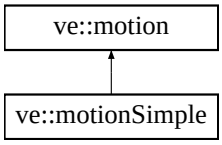
- veMotion.h

12.51 ve::motionSimple Class Reference

A very simple motion model. This class implements a basic generic motion model, all speed and acceleration factors are widely adjustable by an xml initialization file.

`#include <veMotion.h>`

Inheritance diagram for ve::motionSimple::



Public Member Functions

- motionSimple (ve::collision *pCollision=NULL)
- motionSimple (ve::xmlIni &ini, unsigned int iniSectionId=0, ve::collision *pCollision=NULL)

Protected Member Functions

- virtual int calculateStatus (unsigned int objectId, const ve::vec6f &input, ve::vec6f &pos, ve::vec6f &vel, ve::vec6f &acc, double deltaT) const

Protected Attributes

- float translSpeedMax
- float translAccFactor
- float translDecFactor
- float rotSpeedMax
- float rotAccFactor

12.51.1 Detailed Description

A very simple motion model. This class implements a basic generic motion model, all speed and acceleration factors are widely adjustable by an xml initialization file.

Author:
gf

Revision
2.0

Definition at line 96 of file veMotion.h.

12.51.2 Constructor & Destructor Documentation

12.51.2.1 ve::motionSimple::motionSimple (ve::collision * *pCollision* = NULL)

default constructor, optionally taking a pointer to a collision model

12.51.2.2 ve::motionSimple::motionSimple (ve::xmlIni & *ini*, unsigned int *iniSectionId* = 0, ve::collision * *pCollision* = NULL)

constructor interpreting an xml ini statement

12.51.3 Member Function Documentation

12.51.3.1 virtual int ve::motionSimple::calculateStatus (unsigned int *objectId*, const ve::vec6f & *input*, ve::vec6f & *pos*, ve::vec6f & *vel*, ve::vec6f & *acc*, double *deltaT*) const [protected, virtual]

main motion computation method.

This function represents the dynamics implemented in the motion model. It calcutates the new position, velocity, acceleration based on input and applies the collision detection.

Reimplemented from ve::motion.

12.51.4 Member Data Documentation

12.51.4.1 float ve::motionSimple::translSpeedMax [protected]

maximum translation speed

Definition at line 104 of file veMotion.h.

12.51.4.2 float ve::motionSimple::translAccFactor [protected]

acceleration factor

Definition at line 106 of file veMotion.h.

12.51.4.3 float ve::motionSimple::translDecFactor [protected]

deceleration factor

Definition at line 108 of file veMotion.h.

12.51.4.4 float ve::motionSimple::rotSpeedMax [protected]

maximum rotation speed

Definition at line 110 of file veMotion.h.

12.51.4.5 float ve::motionSimple::rotAccFactor [protected]

rotation acceleration

Definition at line 112 of file veMotion.h.

The documentation for this class was generated from the following file:

- veMotion.h

12.52 ve::networkTime Struct Reference

Public Attributes

- double nwTime
- double lastLocalUpdate

12.52.1 Detailed Description

Definition at line 79 of file veDeviceNetwork.h.

The documentation for this struct was generated from the following file:

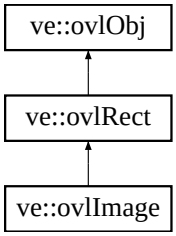
- veDeviceNetwork.h

12.53 ve::ovlImage Class Reference

a class for displaying images in the overlay plane.

`#include <veGlUtils.h>`

Inheritance diagram for ve::ovlImage::



Public Member Functions

- ovlImage (ve::deviceWindow &window, const std::string &filename, float xPos=0, float yPos=0, float width=1.0, float height=1.0, ve::align_t alH=ve::ALIGN_LEFT, ve::align_t alV=ve::ALIGN_BOTTOM, bool repeat=true)
- ovlImage (ve::deviceWindow &window, const std::string &filename, float xPos, float yPos, ve::align_t alH=ve::ALIGN_LEFT, ve::align_t alV=ve::ALIGN_BOTTOM, bool repeat=true)
- virtual void draw ()
- virtual const std::string & filename () const

Protected Attributes

- int byteDepth
- unsigned int m_texId
- std::string strFile

12.53.1 Detailed Description

a class for displaying images in the overlay plane.

Definition at line 268 of file veGlUtils.h.

12.53.2 Constructor & Destructor Documentation

12.53.2.1 ve::ovlImage::ovlImage (ve::deviceWindow & *window*, const std::string & *filename*, float *xPos* = `0`, float *yPos* = `0`, float *width* = `1.0`, float *height* = `1.0`, ve::align_t *alH* = `ve::ALIGN_LEFT`, ve::align_t *alV* = `ve::ALIGN_BOTTOM`, bool *repeat* = `true`)

constructor with texture file, image will be displayed scaled to fit the defined width and height.

12.53.2.2 ve::ovlImage::ovlImage (ve::deviceWindow & *window*, const std::string & *filename*, float *xPos*, float *yPos*, ve::align_t *alH* = `ve::ALIGN_LEFT`, ve::align_t *alV* = `ve::ALIGN_BOTTOM`, bool *repeat* = `true`)

constructor with texture file, image will be displayed unscaled.

12.53.3 Member Function Documentation

12.53.3.1 virtual void ve::ovlImage::draw () `[virtual]`

draws widget

Reimplemented from ve::ovlRect.

12.53.3.2 virtual const std::string& ve::ovlImage::filename () const `[inline, virtual]`

returns the texture filename

Definition at line 279 of file veGlUtils.h.

References strFile.

12.53.4 Member Data Documentation

12.53.4.1 int ve::ovlImage::byteDepth `[protected]`

stores byte depth of texture

Definition at line 279 of file veGlUtils.h.

12.53.4.2 unsigned int ve::ovlImage::m_texId `[protected]`

stores texture id

Definition at line 285 of file veGlUtils.h.

12.53.4.3 std::string ve::ovlImage::strFile `[protected]`

stores the texture filename

Definition at line 287 of file veGlUtils.h.

Referenced by filename().

The documentation for this class was generated from the following file:

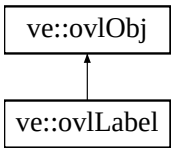
- veGlUtils.h

12.54 ve::ovlLabel Class Reference

a single line text overlay widget.

`#include <veGlUtils.h>`

Inheritance diagram for ve::ovlLabel::



Public Member Functions

- ovlLabel (ve::deviceWindow &window, const std::string &str="", float xPos=0, float yPos=0, float width=0.0f, float height=0.0f, ve::align_t alH=ve::ALIGN_LEFT, ve::align_t alV=ve::ALIGN_BOTTOM, unsigned int textRendererId=0)
- virtual void draw ()
- virtual const std::string & str () const
- virtual void str (const std::string &txt)
- void fgNormalColor (float r, float g, float b, float a=1.0f)
- void bgNormalColor (float r, float g, float b, float a=1.0f)
- void fgSelectColor (float r, float g, float b, float a=1.0f)
- void bgSelectColor (float r, float g, float b, float a=1.0f)
- bool isSelected () const

Protected Attributes

- std::string text
- ve::vec4f m_fgNormalCol
- ve::vec4f m_fgSelectCol
- ve::vec4f m_bgNormalCol
- ve::vec4f m_bgSelectCol
- ve::glText * m_pTextRenderer
- bool m_selected

12.54.1 Detailed Description

a single line text overlay widget.

Definition at line 194 of file veGlUtils.h.

12.54.2 Constructor & Destructor Documentation

12.54.2.1 ve::ovlLabel::ovlLabel (ve::deviceWindow & *window*, const std::string & *str* = "", float *xPos* = 0, float *yPos* = 0, float *width* = 0.0f, float *height* = 0.0f, ve::align_t *alH* = ve::ALIGN_LEFT, ve::align_t *alV* = ve::ALIGN_BOTTOM, unsigned int *textRendererId* = 0)

constructor

Parameters:

window the window device the label will be attached to

str (optional) initial string

xPos (optional) x position in overlay coordinates

yPos (optional) y position in overlay coordinates

width (optional) width in overlay coordinates, use 0.0 for automatic width according to initial string

height (optional) height in overlay coordinates, use 0.0 for automatic height

alH (optional) horizontal align

alV (optional) vertical align

textRendererId (optional) resource id of font

12.54.3 Member Function Documentation

12.54.3.1 virtual void ve::ovlLabel::draw () [virtual]

draws widget

Reimplemented from ve::ovlObj.

12.54.3.2 virtual const std::string& ve::ovlLabel::str () const [inline, virtual]

returns current label text

Definition at line 212 of file veGlUtils.h.

References text.

12.54.3.3 virtual void ve::ovlLabel::str (const std::string & *txt*) [virtual]

sets current label text

12.54.3.4 void ve::ovlLabel::fgNormalColor (float *r*, float *g*, float *b*, float *a* = 1.0f) [inline]

sets text color

Definition at line 216 of file veGlUtils.h.

References m_fgNormalCol, and ve::vec4f::set().

12.54.3.5 void ve::ovlLabel::bgNormalColor (float *r*, float *g*, float *b*, float *a* = 1.0f) [inline]

sets background color

Definition at line 219 of file veGlUtils.h.

References m_bgNormalCol, and ve::vec4f::set().

12.54.3.6 void ve::ovlLabel::fgSelectColor (float *r*, float *g*, float *b*, float *a* = 1.0f) [inline]

sets selected foreground color

Definition at line 222 of file veGlUtils.h.

References m_fgSelectCol, and ve::vec4f::set().

12.54.3.7 void ve::ovlLabel::bgSelectColor (float *r*, float *g*, float *b*, float *a* = 1.0f) [inline]

sets selected background color

Definition at line 225 of file veGlUtils.h.

References m_bgSelectCol, and ve::vec4f::set().

12.54.3.8 bool ve::ovlLabel::isSelected () const [inline]

returns true if widget is currently selected, otherwise false

Definition at line 228 of file veGlUtils.h.

12.54.4 Member Data Documentation

12.54.4.1 std::string ve::ovlLabel::text [protected]

stores currently displayed text

Definition at line 228 of file veGlUtils.h.

Referenced by str().

12.54.4.2 ve::vec4f ve::ovlLabel::m_fgNormalCol [protected]

stores text color

Definition at line 233 of file veGlUtils.h.

Referenced by fgNormalColor().

12.54.4.3 ve::vec4f ve::ovlLabel::m_fgSelectCol [protected]

foreground color when selected

Definition at line 235 of file veGlUtils.h.

Referenced by fgSelectColor().

12.54.4.4 ve::vec4f ve::ovlLabel::m_bgNormalCol [protected]

stores text background color, only used in derived classes

Definition at line 237 of file veGlUtils.h.

Referenced by bgNormalColor().

12.54.4.5 ve::vec4f ve::ovlLabel::m_bgSelectCol [protected]

selected background color, only used in derived classes

Definition at line 239 of file veGlUtils.h.

Referenced by bgSelectColor().

12.54.4.6 ve::glText* ve::ovlLabel::m_pTextRenderer [protected]

stores pointer to suitable text renderer

Definition at line 241 of file veGlUtils.h.

12.54.4.7 bool ve::ovlLabel::m_selected [protected]

stores state of selection

Definition at line 243 of file veGlUtils.h.

The documentation for this class was generated from the following file:

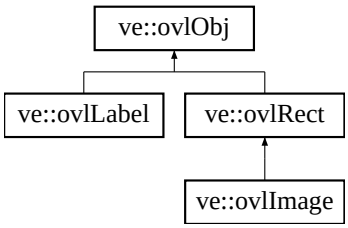
- veGlUtils.h

12.55 ve::ovlObj Class Reference

parent class for 2D overlay objects.

`#include <veGlUtils.h>`

Inheritance diagram for ve::ovlObj::



Public Member Functions

- ovlObj (ve::deviceWindow &window, float xPos=0, float yPos=0, ve::align_t alH=ve::ALIGN_-LEFT, ve::align_t alV=ve::ALIGN_BOTTOM)
- virtual ~ovlObj ()
- virtual void draw ()
- virtual float x () const
- virtual float y () const
- virtual void x (float xPos)
- virtual void y (float yPos)
- virtual float w () const
- virtual float h () const
- virtual void w (float width)
- virtual void h (float height)
- virtual void alH (ve::align_t hAl)
- virtual void alV (ve::align_t vAl)
- virtual void update ()

Protected Member Functions

- virtual void init ()

Protected Attributes

- float m_x
- float m_y
- float m_w
- float m_h
- ve::align_t alignH
- ve::align_t alignV
- float dAlignH
- float dAlignV
- ve::deviceWindow & wnd

12.55.1 Detailed Description

parent class for 2D overlay objects.

Its subclasses allow to build a simple but very portable GUI entirely based on OpenGL. ovlObj are always managed by an ovlPlane object.

Definition at line 136 of file veGlUtils.h.

12.55.2 Constructor & Destructor Documentation

12.55.2.1 ve::ovlObj::ovlObj (ve::deviceWindow & *window*, float *xPos* = 0, float *yPos* = 0, ve::align_t *alH* = ve::ALIGN_LEFT, ve::align_t *alV* = ve::ALIGN_BOTTOM)

constructor

12.55.2.2 virtual ve::ovlObj::~ovlObj () [inline, virtual]

destructor

Definition at line 141 of file veGlUtils.h.

12.55.3 Member Function Documentation

12.55.3.1 virtual void ve::ovlObj::draw () [inline, virtual]

draws widget dummy

Reimplemented in ve::ovlLabel, ve::ovlRect, and ve::ovlImage.

Definition at line 143 of file veGlUtils.h.

12.55.3.2 virtual float ve::ovlObj::x () const [inline, virtual]

returns x position

Definition at line 145 of file veGlUtils.h.

References m_x.

12.55.3.3 virtual float ve::ovlObj::y () const [inline, virtual]

returns y position

Definition at line 147 of file veGlUtils.h.

References m_y.

12.55.3.4 virtual void ve::ovlObj::x (float *xPos*) [inline, virtual]

sets x position

Definition at line 149 of file veGlUtils.h.

References m_x.

12.55.3.5 virtual void ve::ovlObj::y (float *yPos*) [inline, virtual]

sets y position

Definition at line 151 of file veGlUtils.h.

References m_y.

12.55.3.6 virtual float ve::ovlObj::w () const [inline, virtual]

returns width

Definition at line 153 of file veGlUtils.h.

References m_w.

12.55.3.7 virtual float ve::ovlObj::h () const [inline, virtual]

returns height

Definition at line 155 of file veGlUtils.h.

References m_h.

12.55.3.8 virtual void ve::ovlObj::w (float *width*) [inline, virtual]

sets width

Definition at line 157 of file veGlUtils.h.

References init(), and m_w.

12.55.3.9 virtual void ve::ovlObj::h (float *height*) [inline, virtual]

sets height

Definition at line 159 of file veGlUtils.h.

References init(), and m_h.

12.55.3.10 virtual void ve::ovlObj::alH (ve::align_t *hAl*) [inline, virtual]

sets horizontal alignment

Definition at line 161 of file veGlUtils.h.

References alignH, and init().

12.55.3.11 virtual void ve::ovlObj::alV (ve::align_t *vAl*) [inline, virtual]

sets vertical alignment

Definition at line 163 of file veGlUtils.h.

References alignV, and init().

12.55.3.12 virtual void ve::ovlObj::update () [inline, virtual]

handles events, interface definition

automatically called by overlay plane

Definition at line 167 of file veGlUtils.h.

12.55.3.13 virtual void ve::ovlObj::init () [protected, virtual]

performs internal initializations, e. g. computes alignment values

Referenced by alH(), alV(), h(), and w().

12.55.4 Member Data Documentation

12.55.4.1 float ve::ovlObj::m_x [protected]

stores x position

Definition at line 172 of file veGlUtils.h.

Referenced by x().

12.55.4.2 float ve::ovlObj::m_y [protected]

stores y position

Definition at line 174 of file veGlUtils.h.

Referenced by y().

12.55.4.3 float ve::ovlObj::m_w [protected]

stores width

Definition at line 176 of file veGlUtils.h.

Referenced by w().

12.55.4.4 float ve::ovlObj::m_h [protected]

stores height

Definition at line 178 of file veGlUtils.h.

Referenced by h().

12.55.4.5 ve::align_t ve::ovlObj::alignH [protected]

stores horizontal alignment state

Definition at line 180 of file veGlUtils.h.

Referenced by alH().

12.55.4.6 ve::align_t ve::ovlObj::alignV [protected]

stores vertical alignment state

Definition at line 182 of file veGlUtils.h.

Referenced by alV().

12.55.4.7 float ve::ovlObj::dAlignH [protected]

stores actual horizontal translation caused by alignment

Definition at line 184 of file veGlUtils.h.

12.55.4.8 float ve::ovlObj::dAlignV [protected]

stores actual vertical translation caused by alignment

Definition at line 186 of file veGlUtils.h.

12.55.4.9 ve::deviceWindow& ve::ovlObj::wnd [protected]

stores reference to current window

Definition at line 188 of file veGlUtils.h.

The documentation for this class was generated from the following file:

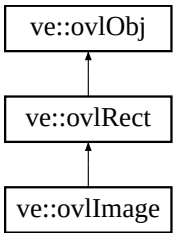
- veGlUtils.h

12.56 ve::ovlRect Class Reference

a class for displaying untextextured rectangles in the overlay plane.

`#include <veGlUtils.h>`

Inheritance diagram for ve::ovlRect::



Public Member Functions

- ovlRect (ve::deviceWindow &window, float xPos=0, float yPos=0, float width=1.0, float height=1.0, ve::align_t alH=ve::ALIGN_LEFT, ve::align_t alV=ve::ALIGN_BOTTOM, float r=1.0, float g=0.0, float b=1.0, float a=1.0)
- virtual void draw ()
- void color (float r, float g, float b, float a=1.0f)

Protected Attributes

- bool sizeAbs
- ve::vec4f m_bgNormalCol

12.56.1 Detailed Description

a class for displaying untextextured rectangles in the overlay plane.

Definition at line 249 of file veGlUtils.h.

12.56.2 Constructor & Destructor Documentation

12.56.2.1 ve::ovlRect::ovlRect (ve::deviceWindow & *window*, float *xPos* = `0`, float *yPos* = `0`, float *width* = `1.0`, float *height* = `1.0`, ve::align_t *alH* = `ve::ALIGN_LEFT`, ve::align_t *alV* = `ve::ALIGN_BOTTOM`, float *r* = `1.0`, float *g* = `0.0`, float *b* = `1.0`, float *a* = `1.0`)

standard constructor

12.56.3 Member Function Documentation

12.56.3.1 virtual void ve::ovlRect::draw () `[virtual]`

draws widget

Reimplemented from ve::ovlObj.

Reimplemented in ve::ovlImage.

12.56.3.2 void ve::ovlRect::color (float *r*, float *g*, float *b*, float *a* = `1.0f`) `[inline]`

sets color

Definition at line 257 of file veGlUtils.h.

12.56.4 Member Data Documentation

12.56.4.1 bool ve::ovlRect::sizeAbs `[protected]`

is true if size of object is absolute, otherwise false

Definition at line 258 of file veGlUtils.h.

12.56.4.2 ve::vec4f ve::ovlRect::m_bgNormalCol `[protected]`

current color of widget

Definition at line 263 of file veGlUtils.h.

The documentation for this class was generated from the following file:

- veGlUtils.h

12.57 ve::plane Class Reference

a class representing a plane.

`#include <veMath.h>`

Public Member Functions

- plane (const vec3f &v=vec3f(0, 0, 1), float d_=0)
- plane (const vec3f &v, const vec3f &p)
- plane (const vec3f &p0, const vec3f &p1, const vec3f &p2)
- plane (const ve::triangle &tr)
- plane (const plane &source)
- const plane & operator= (const plane &source)
- float signedDistTo (const vec3f &p) const
- float distTo (const vec3f &p) const
- vec3f & normalVector ()
- const vec3f & normalVector () const
- float D () const
- float & D ()
- void translate (float x, float y, float z=0.0f)
- void translate (const vec3f &v)
- void rotate (float angle, vec3f p)
- void rotate (float h, float p, float r)
- void transform (const ve::vec6f &sdof)

Protected Attributes

- vec3f n
- float d

Friends

- std::ostream & operator<< (std::ostream &os, const ve::plane &pl)

12.57.1 Detailed Description

a class representing a plane.

Definition at line 492 of file veMath.h.

12.57.2 Constructor & Destructor Documentation

12.57.2.1 ve::plane::plane (const vec3f & *v* = `vec3f(0, 0, 1)`, float *d_* = `0`)

default constructor from a normal vector v and optional d component.

12.57.2.2 ve::plane::plane (const vec3f & *v*, const vec3f & *p*)

constructor from a normal vector v and a point p.

12.57.2.3 ve::plane::plane (const vec3f & *p0*, const vec3f & *p1*, const vec3f & *p2*)

constructor from three points

12.57.2.4 ve::plane::plane (const ve::triangle & *tr*)

constructor from triangle

12.57.2.5 ve::plane::plane (const plane & *source*)

copy constructor

12.57.3 Member Function Documentation

12.57.3.1 const plane& ve::plane::operator= (const plane & *source*)

copy operator

12.57.3.2 float ve::plane::signedDistTo (const vec3f & *p*) const [inline]

returns signed distance to point p
Definition at line 507 of file veMath.h.
References d, and n.
Referenced by distTo().

12.57.3.3 float ve::plane::distTo (const vec3f & *p*) const [inline]

returns absolute distance to point p
Definition at line 509 of file veMath.h.
References signedDistTo().

12.57.3.4 vec3f& ve::plane::normalVector () [inline]

returns normal vector
Definition at line 511 of file veMath.h.
References n.

12.57.3.5 const vec3f& ve::plane::normalVector () const [inline]

returns normal vector, const

Definition at line 513 of file veMath.h.

References n.

12.57.3.6 float ve::plane::D () const [inline]

returns offset component d, const

Definition at line 515 of file veMath.h.

References d.

12.57.3.7 float& ve::plane::D () [inline]

returns offset component d

Definition at line 517 of file veMath.h.

References d.

12.57.3.8 void ve::plane::translate (float *x*, float *y*, float *z* = 0.0f) [inline]

translates object by x|y|z.

Definition at line 519 of file veMath.h.

12.57.3.9 void ve::plane::translate (const vec3f & *v*)

translates object by vector v.

12.57.3.10 void ve::plane::rotate (float *angle*, vec3f *p*) [inline]

rotates object around arbitrary axis from origin to p by angle.

Definition at line 523 of file veMath.h.

References n, and ve::vec3f::rotate().

12.57.3.11 void ve::plane::rotate (float *h*, float *p*, float *r*) [inline]

rotates object according to heading, pitch, and roll.

Definition at line 525 of file veMath.h.

References n, and ve::vec3f::rotate().

12.57.3.12 void ve::plane::transform (const ve::vec6f & *sdof*)

transforms this plane by applying the provided sixdof transformation.

The plane is rotated first according to r,p,h, afterwards translated by x,y,z.

12.57.4 Friends And Related Function Documentation

12.57.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::plane & *pl*) [friend]

operator for output in streams

12.57.5 Member Data Documentation

12.57.5.1 vec3f ve::plane::n [protected]

stores normal vector

Definition at line 535 of file veMath.h.

Referenced by normalVector(), rotate(), and signedDistTo().

12.57.5.2 float ve::plane::d [protected]

stores offset component d of plane equation.

Definition at line 537 of file veMath.h.

Referenced by D(), and signedDistTo().

The documentation for this class was generated from the following file:

- veMath.h

12.58 ve::plugin Class Reference

Public Member Functions

- plugin ()
- plugin (const char *pluginName)
- virtual ~plugin ()
- bool isOpen ()
- const string & name ()
- int init ()
- void update (void *obj, double deltaT)
- int close ()
- bool drawFunctionLinked ()
- void draw (void *obj)

Protected Member Functions

- virtual void classInit ()

Protected Attributes

- void * m_libHandle
- string m_name
- pluginInitFunc libInit
- bool m_libInitPlugged
- pluginUpdateFunc libUpdate
- bool m_libUpdatePlugged
- pluginCloseFunc libClose
- bool m_libClosePlugged
- pluginDrawFunc libDraw
- bool m_libDrawPlugged

12.58.1 Detailed Description

Definition at line 23 of file vePlugins.h.

12.58.2 Constructor & Destructor Documentation

12.58.2.1 ve::plugin::plugin ()

Default constructor.

12.58.2.2 ve::plugin::plugin (const char * *pluginName*)

Constructor that takes the name of the plugin file.

12.58.2.3 virtual ve::plugin::~plugin () [virtual]

Destructor.

12.58.3 Member Function Documentation

12.58.3.1 bool ve::plugin::isOpen () [inline]

Returns true if a handle to a plugin exists.

Definition at line 33 of file vePlugins.h.

References m_libHandle.

12.58.3.2 const string& ve::plugin::name () [inline]

Returns the name of the plugin as found in the library, or NULL if char *vePluginVarName not found.

Definition at line 35 of file vePlugins.h.

References m_name.

12.58.3.3 int ve::plugin::init ()

Calls vePluginInit.

12.58.3.4 void ve::plugin::update (void * *obj*, double *deltaT*)

Calls vePluginUpdate.

12.58.3.5 int ve::plugin::close ()

Calls vePluginClose.

12.58.3.6 bool ve::plugin::drawFunctionLinked () [inline]

Returns true if vePluginDraw was found and linked properly.

Definition at line 43 of file vePlugins.h.

References m_libDrawPlugged.

12.58.3.7 void ve::plugin::draw (void * *obj*)

Calls vePluginGeoDraw.

12.58.3.8 virtual void ve::plugin::classInit () [inline, protected, virtual]

Class initialisation function.

Definition at line 70 of file vePlugins.h.

12.58.4 Member Data Documentation

12.58.4.1 void* ve::plugin::m_libHandle [protected]

Handle to the dynamic library.

Definition at line 49 of file vePlugins.h.

Referenced by isOpen().

12.58.4.2 string ve::plugin::m_name [protected]

Plugin name, as found in library.

Definition at line 51 of file vePlugins.h.

Referenced by name().

12.58.4.3 pluginInitFunc ve::plugin::libInit [protected]

Pointer to plugin initialisation function.

Definition at line 53 of file vePlugins.h.

12.58.4.4 bool ve::plugin::m_libInitPlugged [protected]

true if vePluginGeoInit has been linked

Definition at line 55 of file vePlugins.h.

12.58.4.5 pluginUpdateFunc ve::plugin::libUpdate [protected]

Pointer to plugin update function.

Definition at line 57 of file vePlugins.h.

12.58.4.6 bool ve::plugin::m_libUpdatePlugged [protected]

true if vePluginGeoUpdate has been linked

Definition at line 59 of file vePlugins.h.

12.58.4.7 pluginCloseFunc ve::plugin::libClose [protected]

Pointer to plugin closing function.

Definition at line 61 of file vePlugins.h.

12.58.4.8 bool ve::plugin::m_libClosePlugged [protected]

true if vePluginGeoClose has been linked

Definition at line 63 of file vePlugins.h.

12.58.4.9 pluginDrawFunc ve::plugin::libDraw [protected]

Drawing function (to render a geoObj for instance).

Definition at line 65 of file vePlugins.h.

12.58.4.10 bool ve::plugin::m_libDrawPlugged [protected]

true if vePluginDraw has been linked

Definition at line 67 of file vePlugins.h.

Referenced by drawFunctionLinked().

The documentation for this class was generated from the following file:

- vePlugins.h

12.59 ve::pluginHandler Class Reference

Public Member Functions

- plugin * getPlugin (const string &name)
- void update (double deltaT)
- bool registerPlugin (const char *path)
- void closePlugin (unsigned int handle)

Protected Member Functions

- pluginHandler ()
- ~pluginHandler ()

Protected Attributes

- map< string, plugin * > m_pluginMap

Static Protected Attributes

- static pluginHandler * s_instance

12.59.1 Detailed Description

Definition at line 75 of file vePlugins.h.

12.59.2 Constructor & Destructor Documentation

12.59.2.1 ve::pluginHandler::pluginHandler () [protected]

Default constructor.

12.59.2.2 ve::pluginHandler::~pluginHandler () [protected]

Destructor.

12.59.3 Member Function Documentation

12.59.3.1 plugin* ve::pluginHandler::getPlugin (const string & *name*)

Returns a pointer to the instance of a plugin referenced by handle. Plugin functions should only be accessed using this function.

12.59.3.2 void ve::pluginHandler::update (double *deltaT*)

Updates all plugins.

12.59.3.3 bool ve::pluginHandler::registerPlugin (const char * *path*)

Loads a plugin from the given file and returns a handle to it. Returns true when successful. Only one instance of a given plugin can be loaded in a given application.

12.59.3.4 void ve::pluginHandler::closePlugin (unsigned int *handle*)

Closes an instance of a plugin.

12.59.4 Member Data Documentation

12.59.4.1 map<string, plugin*> ve::pluginHandler::m_pluginMap [protected]

linked list of plugins

Definition at line 99 of file vePlugins.h.

The documentation for this class was generated from the following file:

- vePlugins.h

12.60 ve::rnd Class Reference

an extension of c random number functions.

`#include <veMath.h>`

Public Member Functions

- rnd (unsigned int seed=0)
- int get ()
- int get (int max)
- float getf ()
- float getf (float f)

Static Public Member Functions

- static void permutate (std::vector< unsigned int > &vec, unsigned int n=0)
- static void permutate (std::vector< std::string > &vec)
- static void permutate (std::vector< float > &vec)
- static void rndEqual (std::vector< float > &vec, unsigned int n, float loBound, float hiBound, unsigned int intervals)

Static Protected Attributes

- static unsigned int defaultSeed

12.60.1 Detailed Description

an extension of c random number functions.

Definition at line 929 of file veMath.h.

12.60.2 Constructor & Destructor Documentation

12.60.2.1 ve::rnd::rnd (unsigned int *seed* = `0`)

default constructor.

12.60.3 Member Function Documentation

12.60.3.1 int ve::rnd::get ()

returns a random integer value.

12.60.3.2 int ve::rnd::get (int *max*)

returns an int number between 0 < max.

12.60.3.3 float ve::rnd::getf ()

returns a float number between 0.0 and 1.0.

Referenced by getf().

12.60.3.4 float ve::rnd::getf (float *f*) [inline]

returns a float number between 0.0 and f.

Definition at line 940 of file veMath.h.

References getf().

12.60.3.5 static void ve::rnd::permutate (std::vector< unsigned int > & *vec*, unsigned int *n* = 0) [static]

randomizes an unsigned int vector, optionally fills in n Elements 0..n-1.

12.60.3.6 static void ve::rnd::permutate (std::vector< std::string > & *vec*) [static]

randomizes a string vector.

12.60.3.7 static void ve::rnd::permutate (std::vector< float > & *vec*) [static]

randomizes a float vector.

12.60.3.8 static void ve::rnd::rndEqual (std::vector< float > & *vec*, unsigned int *n*, float *loBound*, float *hiBound*, unsigned int *intervals*) [static]

fills vec3f with n equal-distributed pseudo random numbers.

This random algorithm provides equally-distributed random numbers for small quantitities by dividing the range between loBound and hiBound in even intervals and using each interval equally often for generating random numbers.

12.60.4 Member Data Documentation

12.60.4.1 unsigned int ve::rnd::defaultSeed [static, protected]

stores a seed for the case no seed is provided by the constructor call.

The default seed will be generated from system time or from previously instanciated ve::rnd object constructors.

Definition at line 957 of file veMath.h.

The documentation for this class was generated from the following file:

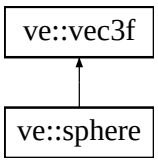
- veMath.h

12.61 ve::sphere Class Reference

a class representing a sphere.

`#include <veMath.h>`

Inheritance diagram for ve::sphere::



Public Member Functions

- sphere (float x=0, float y=0, float z=0, float rd=0)
- sphere (const vec3f &center, float rd=0)
- sphere (const sphere &source)
- float radius () const
- void radius (float newRadius)

Protected Attributes

- float r

Friends

- std::ostream & operator<< (std::ostream &os, const ve::sphere &sph)

12.61.1 Detailed Description

a class representing a sphere.

Definition at line 462 of file veMath.h.

12.61.2 Constructor & Destructor Documentation

12.61.2.1 ve::sphere::sphere (float *x* = 0, float *y* = 0, float *z* = 0, float *rd* = 0) [inline]

default constructor

Definition at line 465 of file veMath.h.

References r.

12.61.2.2 ve::sphere::sphere (const vec3f & *center*, float *rd* = 0) [inline]

constructor from a vec

Definition at line 467 of file veMath.h.

References r.

12.61.2.3 ve::sphere::sphere (const sphere & *source*) [inline]

copy constructor

Definition at line 469 of file veMath.h.

References r.

12.61.3 Member Function Documentation

12.61.3.1 float ve::sphere::radius () const [inline]

returns radius

Definition at line 471 of file veMath.h.

References r.

Referenced by ve::line::intersection().

12.61.3.2 void ve::sphere::radius (float *newRadius*) [inline]

sets radius

Definition at line 473 of file veMath.h.

References r.

12.61.4 Friends And Related Function Documentation

12.61.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::sphere & *sph*) [friend]

operator for output in streams

12.61.5 Member Data Documentation

12.61.5.1 float ve::sphere::r [protected]

stores radius

Definition at line 479 of file veMath.h.

Referenced by radius(), and sphere().

The documentation for this class was generated from the following file:

- veMath.h

12.62 ve::triangle Class Reference

a class for triangle geometry.

`#include <veMath.h>`

Public Member Functions

- triangle (float x1, float y1, float x2, float y2, float x3, float y3)
- triangle (float x1, float y1, float z1, float x2, float y2, float z2, float x3, float y3, float z3)
- triangle (const vec3f &p0, const vec3f &p1, const vec3f &p2)
- triangle (const float *pCoords)
- triangle (const triangle &source)
- void set (float x1, float y1, float x2, float y2, float x3, float y3)
- void set (float x1, float y1, float z1, float x2, float y2, float z2, float x3, float y3, float z3)
- void set (const ve::vec3f &v0, const ve::vec3f &v1, const ve::vec3f &v2)
- void translate (float dx, float dy, float dz=0)
- void translate (const vec3f &v, float n=1.0f)
- void rotate (float angle, float x, float y, float z)
- void scale (float sx, float sy, float sz=1)
- void transform (const ve::mat4f &m)
- void transform (const ve::vec6f &sdof)
- vec3f & operator[] (unsigned int n)
- const vec3f & operator[] (unsigned int n) const
- void getCoords (double *coords) const
- bool isElemXY (const vec3f &p) const
- float distZ (const vec3f &p) const
- vec3f normalVector () const
- double area () const
- bool inRange (int axis, double minValue, double maxValue) const
- void getABCD (float &a, float &b, float &c, float &d) const

Protected Attributes

- vec3f pt [3]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::triangle &t)

12.62.1 Detailed Description

a class for triangle geometry.

Definition at line 550 of file veMath.h.

12.62.2 Constructor & Destructor Documentation

12.62.2.1 ve::triangle::triangle (float *x1*, float *y1*, float *x2*, float *y2*, float *x3*, float *y3*) [inline]

2d constructor

Definition at line 553 of file veMath.h.

References set().

12.62.2.2 ve::triangle::triangle (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*, float *x3*, float *y3*, float *z3*) [inline]

3d constructor

Definition at line 555 of file veMath.h.

References set().

12.62.2.3 ve::triangle::triangle (const vec3f & *p0*, const vec3f & *p1*, const vec3f & *p2*) [inline]

constructor taking vec3f references

Definition at line 558 of file veMath.h.

References set().

12.62.2.4 ve::triangle::triangle (const float * *pCoords*)

low level constructor taking binary data of 9 float values, no range checks, beware of segfaults!

12.62.2.5 ve::triangle::triangle (const triangle & *source*)

copy constructor

12.62.3 Member Function Documentation

12.62.3.1 void ve::triangle::set (float *x1*, float *y1*, float *x2*, float *y2*, float *x3*, float *y3*) [inline]

sets triangle to new 2D vec3f values

Definition at line 564 of file veMath.h.

Referenced by triangle().

12.62.3.2 void ve::triangle::set (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*, float *x3*, float *y3*, float *z3*)

sets triangle to new 3D vec3f values

12.62.3.3 void ve::triangle::set (const ve::vec3f & *v0*, const ve::vec3f & *v1*, const ve::vec3f & *v2*) [inline]

sets triangle to new vec3f values

Definition at line 569 of file veMath.h.

References pt.

12.62.3.4 void ve::triangle::translate (float *dx*, float *dy*, float *dz* = 0) [inline]

translates object by dx,dy,dz

Definition at line 572 of file veMath.h.

References pt, and ve::vec3f::translate().

Referenced by translate().

12.62.3.5 void ve::triangle::translate (const vec3f & *v*, float *n* = 1.0f) [inline]

translates object by vector v, optionally scaled by factor n

Definition at line 575 of file veMath.h.

References translate().

12.62.3.6 void ve::triangle::rotate (float *angle*, float *x*, float *y*, float *z*) [inline]

rotates object around arbitrary axis from origin to p by angle.

Definition at line 577 of file veMath.h.

References pt, and ve::vec3f::rotate().

12.62.3.7 void ve::triangle::scale (float *sx*, float *sy*, float *sz* = 1)

scales object by sx,sy and optionally sz

12.62.3.8 void ve::triangle::transform (const ve::mat4f & *m*)

transforms this triangle by multiplying it with matrix m

12.62.3.9 void ve::triangle::transform (const ve::vec6f & *sdof*)

transforms this triangle by applying the provided sixdof transformation.

The vector is rotated first according to r,p,h, afterwards translated by x,y,z.

12.62.3.10]

vec3f& ve::triangle::operator[] (unsigned int *n*) [inline]

returns control point n

Definition at line 588 of file veMath.h.

References pt.

12.62.3.11]

const vec3f& ve::triangle::operator[] (unsigned int *n*) const [inline]

returns control point n, const

Definition at line 590 of file veMath.h.

References pt.

12.62.3.12 void ve::triangle::getCoords (double * *coords*) const

fills coords with coordinate values of all 3 control points, no memory allocation.

12.62.3.13 bool ve::triangle::isElemXY (const vec3f & *p*) const

tests v for being in the same xy area than the triangle

12.62.3.14 float ve::triangle::distZ (const vec3f & *p*) const

returns the z distance from v to the triangle plane, positive if v is above, otherwise negative

12.62.3.15 vec3f ve::triangle::normalVector () const

returns normal vector of the triangle plane

12.62.3.16 double ve::triangle::area () const

returns area of triangle.

12.62.3.17 bool ve::triangle::inRange (int *axis*, double *minValue*, double *maxValue*) const

tests whether triangle intersects defined range between minValue and maxValue with respect to axis.

12.62.3.18 void ve::triangle::getABCD (float & *a*, float & *b*, float & *c*, float & *d*) const

returns plane equation factors A,B,C,D.

12.62.4 Friends And Related Function Documentation

12.62.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::triangle & *t*) [friend]

operator for output in streams

12.62.5 Member Data Documentation

12.62.5.1 vec3f ve::triangle::pt[3] [protected]

vertices

Definition at line 610 of file veMath.h.

Referenced by operator[](), rotate(), set(), and translate().

The documentation for this class was generated from the following file:

- veMath.h

12.63 ve::vec2f Class Reference

a class for 2d vector and vertex geometry operations.

`#include <veMath.h>`

Public Member Functions

- vec2f (float x_=0, float y_=0)
- vec2f (const vec2f &v)
- vec2f (const vec2f &p1, const vec2f &p2)
- vec2f (const std::string &s, const std::string &separator=", \t\n\015")
- const vec2f & operator= (const vec2f &source)
- void set (float x_, float y_)
- void set (float *x_)
- void set (const std::string &s, const std::string &separator=", \t\n\015")
- std::string str (const std::string &separator=" ", unsigned char nDigits=8) const
- bool operator== (const vec2f &vt) const
- bool operator!= (const vec2f &vt) const
- float & operator[ ] (unsigned int i)
- float operator[ ] (unsigned int i) const
- const float * coords () const
- void translate (float dx, float dy)
- void translate (const vec2f &v, float n=1.0f)
- void operator+= (const vec2f &v)
- const vec2f & operator *= (float f)
- const vec2f & operator/= (float f)
- float operator * (const ve::vec2f &v) const
- const vec2f & normalize ()
- float sqrLength () const
- float length () const
- const vec2f operator+ (const vec2f &v) const
- const vec2f operator- (const vec2f &v) const
- const vec2f operator * (float f) const

Protected Attributes

- float coord [2]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::vec2f &v)

12.63.1 Detailed Description

a class for 2d vector and vertex geometry operations.

The vec2f class is intentionally free of virtual methods and takes exactly 8 bytes of memory. It is not meant as generic vector class (use vec3f instead), but only for efficient storing of 2D mass coordinate information.

Definition at line 152 of file veMath.h.

12.63.2 Constructor & Destructor Documentation

12.63.2.1 ve::vec2f::vec2f (float *x_* = 0, float *y_* = 0) [inline]

default constructor

Definition at line 155 of file veMath.h.

References coord.

Referenced by operator ∗(), operator+(), and operator-().

12.63.2.2 ve::vec2f::vec2f (const vec2f & *v*) [inline]

copy constructor

Definition at line 157 of file veMath.h.

References coord.

12.63.2.3 ve::vec2f::vec2f (const vec2f & *p1*, const vec2f & *p2*) [inline]

constructor for vector between two given vertices

Definition at line 159 of file veMath.h.

References coord.

12.63.2.4 ve::vec2f::vec2f (const std::string & *s*, const std::string & *separator* = ", \t\n\015") [inline]

constructor interpreting a string, values are separated by optionally definable separators

Definition at line 162 of file veMath.h.

References set().

12.63.3 Member Function Documentation

12.63.3.1 const vec2f& ve::vec2f::operator= (const vec2f & *source*)

copy operator

12.63.3.2 void ve::vec2f::set (float *x_*, float *y_*) [inline]

sets vector ordinates to new values (x_|y_).

Definition at line 167 of file veMath.h.

References coord.

Referenced by vec2f().

12.63.3.3 void ve::vec2f::set (float * *x_*) [inline]

sets vector ordinates to new values (x_[0]|x_[1]).

Definition at line 169 of file veMath.h.

References coord.

12.63.3.4 void ve::vec2f::set (const std::string & *s*, const std::string & *separator* = ", \t\n\015")

sets all ordinates by a string, values are separated by optionally definable separators.

12.63.3.5 std::string ve::vec2f::str (const std::string & *separator* = " ", unsigned char *nDigits* = 8) const

returns a string containing all data, optional arguments set separator between ordinates and number of digits per ordinate.

12.63.3.6 bool ve::vec2f::operator== (const vec2f & *vt*) const

comparison operator equality

12.63.3.7 bool ve::vec2f::operator!= (const vec2f & *vt*) const [inline]

comparison operator inequality

Definition at line 178 of file veMath.h.

12.63.3.8]

float& ve::vec2f::operator[] (unsigned int *i*) [inline]

returns reference to single ordinate

Definition at line 180 of file veMath.h.

References coord.

12.63.3.9]

float ve::vec2f::operator[] (unsigned int *i*) const [inline]

returns single ordinate value

Definition at line 182 of file veMath.h.

References coord.

12.63.3.10 const float* ve::vec2f::coords () const [inline]

returns coordinate array

Definition at line 184 of file veMath.h.

References coord.

12.63.3.11 void ve::vec2f::translate (float *dx*, float *dy*) [inline]

translates object by dx,dy

Definition at line 187 of file veMath.h.

References coord.

Referenced by translate().

12.63.3.12 void ve::vec2f::translate (const vec2f & *v*, float *n* = 1.0f) [inline]

translates object by vector v, optionally scaled by factor n

Definition at line 189 of file veMath.h.

References translate().

12.63.3.13 void ve::vec2f::operator+= (const vec2f & *v*) [inline]

+= operator. Sums correspondend ordinates. Identical to translate(v).

Definition at line 191 of file veMath.h.

References coord.

12.63.3.14 const vec2f& ve::vec2f::operator *= (float *f*) [inline]

scales this vector by f.

Definition at line 193 of file veMath.h.

References coord.

12.63.3.15 const vec2f& ve::vec2f::operator/= (float *f*) [inline]

devides this vector by f.

Definition at line 195 of file veMath.h.

References coord.

12.63.3.16 float ve::vec2f::operator * (const ve::vec2f & *v*) const [inline]

vector scalar product operator

Definition at line 197 of file veMath.h.

References coord.

12.63.3.17 const vec2f& ve::vec2f::normalize () [inline]

scales this vector to unit vector of length 1.0.

Definition at line 200 of file veMath.h.

References length().

12.63.3.18 float ve::vec2f::sqrLength () const [inline]

returns the squared length.

Definition at line 203 of file veMath.h.

References coord.

Referenced by length().

12.63.3.19 float ve::vec2f::length () const [inline]

returns absolute length.

Definition at line 205 of file veMath.h.

References sqrLength().

Referenced by normalize().

12.63.3.20 const vec2f ve::vec2f::operator+ (const vec2f & *v*) const [inline]

vector addition operator

Definition at line 208 of file veMath.h.

References coord, and vec2f().

12.63.3.21 const vec2f ve::vec2f::operator- (const vec2f & *v*) const [inline]

vector subtraction operator

Definition at line 210 of file veMath.h.

References coord, and vec2f().

12.63.3.22 const vec2f ve::vec2f::operator ∗ (float *f*) const [inline]

operator multiplying a vector v with a scalar f

Definition at line 212 of file veMath.h.

References coord, and vec2f().

12.63.4 Friends And Related Function Documentation

12.63.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::vec2f & *v*) [friend]

operator for output of vec2f objects in streams

12.63.5 Member Data Documentation

12.63.5.1 float ve::vec2f::coord[2] [protected]

stores coordinate values.

Definition at line 218 of file veMath.h.

Referenced by coords(), operator ∗(), operator ∗=(), operator+(), operator+=(), operator-(), operator/=(), operator[](), set(), sqrLength(), translate(), and vec2f().

The documentation for this class was generated from the following file:

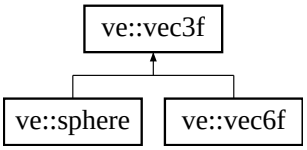
- veMath.h

12.64 ve::vec3f Class Reference

a class for vector and 3D vertex geometry operations.

`#include <veMath.h>`

Inheritance diagram for ve::vec3f::



Public Member Functions

- vec3f (float x_=0, float y_=0, float z_=0)
- vec3f (const vec3f &v)
- vec3f (const vec3f &p1, const vec3f &p2)
- vec3f (const std::string &s, const std::string &separator=", \t\n\015")
- void set (const std::string &s, const std::string &separator=", \t\n\015")
- std::string str (const std::string &separator=" ", unsigned char nDigits=8) const
- const vec3f & operator= (const vec3f &source)
- void set (float x_, float y_)
- void set (float x_, float y_, float z_)
- void set (const float *x_)
- void set (const vec3f &p1, const vec3f &p2)
- void setPolar (float angleXYpl, float angleZ=0, float r=1.0)
- bool operator== (const vec3f &vt) const
- bool operator!= (const vec3f &vt) const
- float & operator[ ] (unsigned int i)
- float operator[ ] (unsigned int i) const
- const float * coords () const
- void translate (float dx, float dy)
- void translate (float dx, float dy, float dz)
- void translate (const vec3f &v, float n=1.0f)
- void operator+= (const vec3f &v)
- void rotate (float angle, vec3f p)
- void rotate (float h, float p, float r)
- void scale (float sx, float sy, float sz=1)
- const vec3f & operator *= (float f)
- const vec3f & operator/= (float f)
- const vec3f operator+ (const vec3f &v) const
- const vec3f operator- (const vec3f &v) const
- const vec3f operator * (float f) const
- const vec3f operator/ (float f) const
- float operator * (const vec3f &v) const
- const vec3f & normalize ()
- void project (const vec3f &v2)
- void transform (const ve::mat4f &m)

- void transform (const ve::vec6f &sdof)
- float sqrLength () const
- float length () const
- vec3f crossProduct (const vec3f &v2) const
- float sqrDistTo (float x, float y, float z=0.0f) const
- float sqrDistTo (const vec3f &v) const
- float distTo (const vec3f &v) const
- float distTo (float x, float y, float z=0.0f) const
- float angleToXY (const vec3f &v) const
- float angleToXY (float x, float y) const

Static Public Member Functions

- static void translate (float *x, float dx, float dy, float dz, unsigned int n=1)
- static void rotate (float &x, float &y, float &z, float angle, float ax, float ay, float az)
- static void rotate (float *x, float angle, float ax, float ay, float az, unsigned int n=1)

Protected Attributes

- float coord [3]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::vec3f &v)

12.64.1 Detailed Description

a class for vector and 3D vertex geometry operations.

The vec3f class is intentionally free of virtual methods and takes exactly twelve bytes of memory. This allows for using a pointer to a vec3f array instead of using pointers to floats, e.g., for OpenGL vertex arrays.

Definition at line 236 of file veMath.h.

12.64.2 Constructor & Destructor Documentation

12.64.2.1 ve::vec3f::vec3f (float *x*_ = 0, float *y*_ = 0, float *z*_ = 0) [inline]

default constructor

Definition at line 239 of file veMath.h.

References coord.

Referenced by operator *(), operator+(), operator-(), and operator/().

12.64.2.2 ve::vec3f::vec3f (const vec3f & *v*) [inline]

copy constructor

Definition at line 241 of file veMath.h.

References coord.

12.64.2.3 ve::vec3f::vec3f (const vec3f & *p1*, const vec3f & *p2*) [inline]

constructor for vector between two given vertices

Definition at line 243 of file veMath.h.

References coord.

12.64.2.4 ve::vec3f::vec3f (const std::string & *s*, const std::string & *separator* = ", \t\n\015") [inline]

constructor interpreting a string, values are separated by optionally definable separators

Definition at line 246 of file veMath.h.

References set().

12.64.3 Member Function Documentation

12.64.3.1 void ve::vec3f::set (const std::string & *s*, const std::string & *separator* = ", \t\n\015")

sets all ordinates by a string, values are separated by optionally definable separators.

Reimplemented in ve::vec6f.

Referenced by setPolar(), and vec3f().

12.64.3.2 std::string ve::vec3f::str (const std::string & *separator* = " ", unsigned char *nDigits* = 8) const

returns a string containing all data, optional arguments set separator between ordinates and number of digits per ordinate.

Reimplemented in ve::vec6f.

12.64.3.3 const vec3f& ve::vec3f::operator= (const vec3f & *source*)

copy operator

12.64.3.4 void ve::vec3f::set (float *x_*, float *y_*) [inline]

sets vector ordinates to new values (x_|y_).

Definition at line 254 of file veMath.h.

References coord.

12.64.3.5 void ve::vec3f::set (float *x_*, float *y_*, float *z_*) [inline]

sets vector ordinates to new values (x_|y_|z_).

Definition at line 256 of file veMath.h.

References coord.

12.64.3.6 void ve::vec3f::set (const float * *x_*) [inline]

sets vector ordinates to new values (x_[0]|x_[1]|x_[2]).

Definition at line 259 of file veMath.h.

References coord.

12.64.3.7 void ve::vec3f::set (const vec3f & *p1*, const vec3f & *p2*) [inline]

sets vector to difference between two given vertices

Definition at line 262 of file veMath.h.

References coord.

12.64.3.8 void ve::vec3f::setPolar (float *angleXYpl*, float *angleZ* = 0, float *r* = 1.0) [inline]

sets vec3f to the specified polar position

Definition at line 265 of file veMath.h.

References ve::dcos(), ve::dsin(), and set().

12.64.3.9 bool ve::vec3f::operator== (const vec3f & *vt*) const

comparison operator equality

12.64.3.10 bool ve::vec3f::operator!= (const vec3f & *vt*) const [inline]

comparison operator inequality

Definition at line 273 of file veMath.h.

12.64.3.11]

float& ve::vec3f::operator[] (unsigned int *i*) [inline]

returns reference to single ordinate

Reimplemented in ve::vec6f.

Definition at line 275 of file veMath.h.

References coord.

12.64.3.12]

float ve::vec3f::operator[] (unsigned int *i*) const [inline]

returns single ordinate value

Reimplemented in ve::vec6f.

Definition at line 277 of file veMath.h.

References coord.

12.64.3.13 const float* ve::vec3f::coords () const [inline]

returns coordinate array

Definition at line 279 of file veMath.h.

References coord.

12.64.3.14 void ve::vec3f::translate (float *dx*, float *dy*) [inline]

translates object by dx,dy

Reimplemented in ve::vec6f.

Definition at line 282 of file veMath.h.

References coord.

Referenced by ve::triangle::translate(), and translate().

12.64.3.15 void ve::vec3f::translate (float *dx*, float *dy*, float *dz*) [inline]

translates object by dx,dy,dz

Reimplemented in ve::vec6f.

Definition at line 284 of file veMath.h.

References coord.

12.64.3.16 void ve::vec3f::translate (const vec3f & *v*, float *n* = 1.0f) [inline]

translates object by vector v, optionally scaled by factor n

Definition at line 287 of file veMath.h.

References translate().

12.64.3.17 static void ve::vec3f::translate (float * *x*, float *dx*, float *dy*, float *dz*, unsigned int *n* = 1) [static]

translates n points starting at ordinate *x by dx,dy,dz.

This is an optimized low level translation function for large numbers of coherent coordinate data.

12.64.3.18 void ve::vec3f::operator+= (const vec3f & *v*) [inline]

+= operator. Sums correspondent ordinates. Identical to translate(v).

Definition at line 293 of file veMath.h.

References coord.

12.64.3.19 void ve::vec3f::rotate (float *angle*, vec3f *p*)

rotates object around arbitrary axis from origin to p by angle.

Referenced by ve::line::rotate(), ve::triangle::rotate(), and ve::plane::rotate().

12.64.3.20 void ve::vec3f::rotate (float *h*, float *p*, float *r*)

rotates vector according to heading, pitch, and roll.

12.64.3.21 static void ve::vec3f::rotate (float & *x*, float & *y*, float & *z*, float *angle*, float *ax*, float *ay*, float *az*) [static]

rotates point (x|y|z) by angle around axis (origin|(ax|ay|az)).

Vertex (ax|ay|az) has to be normalized before applying this static method. Otherwise an uncontrolled scaling occurs.

12.64.3.22 static void ve::vec3f::rotate (float * *x*, float *angle*, float *ax*, float *ay*, float *az*, unsigned int *n* = 1) [static]

rotates n vertices starting at ordinate *x by angle around axis (origin|(ax|ay|az)).

This is an optimized low level rotation function for large numbers of coherent coordinate data. Vertex (ax|ay|az) has to be normalized before applying this static method. Otherwise an uncontrolled scaling occurs.

12.64.3.23 void ve::vec3f::scale (float *sx*, float *sy*, float *sz* = 1)

scales object by sx,sy and optionally sz

12.64.3.24 const vec3f& ve::vec3f::operator *= (float *f*) [inline]

scales this vector by f.

Definition at line 310 of file veMath.h.

References coord.

12.64.3.25 const vec3f& ve::vec3f::operator/= (float *f*) [inline]

divides this vector by f.

Definition at line 312 of file veMath.h.

References coord.

12.64.3.26 const vec3f ve::vec3f::operator+ (const vec3f & *v*) const [inline]

vector addition operator

Reimplemented in ve::vec6f.

Definition at line 314 of file veMath.h.

References coord, and vec3f().

12.64.3.27 const vec3f ve::vec3f::operator- (const vec3f & *v*) const [inline]

vector subtraction operator

Definition at line 316 of file veMath.h.

References coord, and vec3f().

12.64.3.28 const vec3f ve::vec3f::operator * (float *f*) const [inline]

operator multiplying a vector v with a scalar f

Reimplemented in ve::vec6f.

Definition at line 318 of file veMath.h.

References coord, and vec3f().

12.64.3.29 const vec3f ve::vec3f::operator/ (float *f*) const [inline]

operator dividing a vector v by a scalar f

Definition at line 320 of file veMath.h.

References coord, and vec3f().

12.64.3.30 float ve::vec3f::operator * (const vec3f & *v*) const [inline]

vector scalar product operator

Definition at line 322 of file veMath.h.

References coord.

12.64.3.31 const vec3f& ve::vec3f::normalize () [inline]

scales this vector to unit vector of length 1.0.
Definition at line 325 of file veMath.h.
References length().

12.64.3.32 void ve::vec3f::project (const vec3f & *v2*) [inline]

transforms this vector to an orthogonal projection of v2.
Definition at line 327 of file veMath.h.
References sqrLength().

12.64.3.33 void ve::vec3f::transform (const ve::mat4f & *m*)

transforms this vec3f by multiplying it with matrix m
Reimplemented in ve::vec6f.

12.64.3.34 void ve::vec3f::transform (const ve::vec6f & *sdof*)

transforms this vec3f by applying the provided sixdof transformation.
The vector is rotated first according to r,p,h, afterwards translated by x,y,z.

12.64.3.35 float ve::vec3f::sqrLength () const [inline]

returns the squared length.
Definition at line 336 of file veMath.h.
References coord.
Referenced by length(), and project().

12.64.3.36 float ve::vec3f::length () const [inline]

returns absolute length.
Definition at line 338 of file veMath.h.
References sqrLength().
Referenced by normalize().

12.64.3.37 vec3f ve::vec3f::crossProduct (const vec3f & *v2*) const

returns cross product vector with vec3f v2.

12.64.3.38 float ve::vec3f::sqrDistTo (float *x*, float *y*, float *z* = `0.0f`) const [inline]

returns squared distance to coordinate x|y|(z).

Definition at line 342 of file veMath.h.

References coord.

Referenced by distTo(), and sqrDistTo().

12.64.3.39 float ve::vec3f::sqrDistTo (const vec3f & *v*) const [inline]

returns squared distance to vertex v.

Definition at line 345 of file veMath.h.

References sqrDistTo().

12.64.3.40 float ve::vec3f::distTo (const vec3f & *v*) const [inline]

returns distance to vertex v

Definition at line 347 of file veMath.h.

References sqrDistTo().

12.64.3.41 float ve::vec3f::distTo (float *x*, float *y*, float *z* = `0.0f`) const [inline]

returns distance to vertex coordinate x|y|(z).

Definition at line 349 of file veMath.h.

References sqrDistTo().

12.64.3.42 float ve::vec3f::angleToXY (const vec3f & *v*) const [inline]

returns angle in xy plane to vertex v

Definition at line 351 of file veMath.h.

References ve::angleXY(), and coord.

12.64.3.43 float ve::vec3f::angleToXY (float *x*, float *y*) const [inline]

returns angle in xy plane to vertex x|y

Definition at line 353 of file veMath.h.

References ve::angleXY(), and coord.

12.64.4 Friends And Related Function Documentation

12.64.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::vec3f & *v*) [friend]

operator for output in streams

12.64.5 Member Data Documentation

12.64.5.1 float ve::vec3f::coord[3] [protected]

stores coordinate values.

Definition at line 359 of file veMath.h.

Referenced by angleToXY(), coords(), ve::vec6f::operator *(), operator *(), operator *=(), ve::vec6f::operator+(), operator+(), operator+=(), ve::vec6f::operator-(), operator-(), operator/(), operator/=(), operator[]() , ve::vec6f::reset(), set(), sqrDistTo(), sqrLength(), ve::vec6f::translate(), translate(), vec3f(), and ve::vec6f::vec6f().

The documentation for this class was generated from the following file:

- veMath.h

12.65 ve::vec4f Class Reference

a class for 4D vector geometry operations.

`#include <veMath.h>`

Public Member Functions

- vec4f (float x=0.0f, float y=0.0f, float z=0.0f, float w=1.0f)
- vec4f (const vec4f &v)
- vec4f (const float *f)
- vec4f (const std::string &s, const std::string &separator=", \t\n\015")
- const vec4f & operator= (const vec4f &source)
- void set (float x, float y, float z, float w)
- void set (const std::string &s, const std::string &separator=", \t\n\015")
- std::string str (const std::string &separator=" ", unsigned char nDigits=8) const
- bool operator== (const vec4f &vt) const
- bool operator!= (const vec4f &vt) const
- float & operator[] (unsigned int i)
- float operator[] (unsigned int i) const
- const float * coords () const
- void translate (float dx, float dy, float dz, float dw)
- void translate (const vec4f &v, float n=1.0f)
- void operator+= (const vec4f &v)
- void scale (float sx, float sy, float sz, float sw)
- const vec4f & operator *= (float f)
- const vec4f & operator/= (float f)
- const vec4f & normalize ()
- void transform (const ve::mat4f &m)
- float sqrLength () const
- float length () const
- float scalarProduct (const vec4f &v) const

Protected Attributes

- float coord [4]

Friends

- std::ostream & operator<< (std::ostream &os, const ve::vec4f &v)

12.65.1 Detailed Description

a class for 4D vector geometry operations.

The vec4f class is intentionally free of virtual methods and takes exactly twelve bytes of memory. This allows for using a pointer to a vec4f array instead of using pointers to floats, e.g., for OpenGL vertex arrays. NOTE: This class has been added very recently (Revision 2.1) and is far from complete and not yet thoroughly tested!

Definition at line 376 of file veMath.h.

12.65.2 Constructor & Destructor Documentation

12.65.2.1 ve::vec4f::vec4f (float *x* = 0.0f, float *y* = 0.0f, float *z* = 0.0f, float *w* = 1.0f) [inline]

default constructor

Definition at line 379 of file veMath.h.

References coord.

12.65.2.2 ve::vec4f::vec4f (const vec4f & *v*) [inline]

copy constructor

Definition at line 382 of file veMath.h.

References coord.

12.65.2.3 ve::vec4f::vec4f (const float * *f*) [inline]

constructor from a float pointer

Definition at line 385 of file veMath.h.

References coord.

12.65.2.4 ve::vec4f::vec4f (const std::string & *s*, const std::string & *separator* = ", \t\n\015") [inline]

constructor interpreting a string, values are separated by optionally definable separators

Definition at line 388 of file veMath.h.

References set().

12.65.3 Member Function Documentation

12.65.3.1 const vec4f& ve::vec4f::operator= (const vec4f & *source*)

copy operator

12.65.3.2 void ve::vec4f::set (float *x*, float *y*, float *z*, float *w*) [inline]

sets vector ordinates to new values (x|y|z|w).

Definition at line 393 of file veMath.h.

References coord.

Referenced by ve::ovlLabel::bgNormalColor(), ve::ovlLabel::bgSelectColor(), ve::ovlLabel::fgNormalColor(), ve::ovlLabel::fgSelectColor(), and vec4f().

12.65.3.3 void ve::vec4f::set (const std::string & *s*, const std::string & *separator* = ", \t\n\015")

sets all ordinates by a string, values are separated by optionally definable separators.

12.65.3.4 std::string ve::vec4f::str (const std::string & *separator* = " ", unsigned char *nDigits* = 8) const

returns a string containing all data, optional arguments set separator between ordinates and number of digits per ordinate.

12.65.3.5 bool ve::vec4f::operator== (const vec4f & *vt*) const

comparison operator equality

12.65.3.6 bool ve::vec4f::operator!= (const vec4f & *vt*) const [inline]

comparison operator inequality

Definition at line 403 of file veMath.h.

12.65.3.7]

float& ve::vec4f::operator[] (unsigned int *i*) [inline]

returns reference to single ordinate

Definition at line 405 of file veMath.h.

References coord.

12.65.3.8]

float ve::vec4f::operator[] (unsigned int *i*) const [inline]

returns single ordinate value

Definition at line 407 of file veMath.h.

References coord.

12.65.3.9 const float* ve::vec4f::coords () const [inline]

returns coordinate array

Definition at line 409 of file veMath.h.

References coord.

12.65.3.10 void ve::vec4f::translate (float *dx*, float *dy*, float *dz*, float *dw*) [inline]

translates object by dx,dy,dz,dw

Definition at line 412 of file veMath.h.

References coord.

Referenced by translate().

12.65.3.11 void ve::vec4f::translate (const vec4f & *v*, float *n* = 1.0f) [inline]

translates object by vector v, optionally scaled by factor n

Definition at line 415 of file veMath.h.

References translate().

12.65.3.12 void ve::vec4f::operator+= (const vec4f & *v*) [inline]

+= operator. Sums correspondent ordinates. Identical to translate(v).

Definition at line 418 of file veMath.h.

References coord.

12.65.3.13 void ve::vec4f::scale (float *sx*, float *sy*, float *sz*, float *sw*) [inline]

scales object by sx,sy,sz,sw

Definition at line 421 of file veMath.h.

References coord.

12.65.3.14 const vec4f& ve::vec4f::operator *= (float *f*) [inline]

scales this vector by f.

Definition at line 424 of file veMath.h.

References coord.

12.65.3.15 const vec4f& ve::vec4f::operator/= (float *f*) [inline]

divides this vector by f.

Definition at line 427 of file veMath.h.

References coord.

12.65.3.16 const vec4f& ve::vec4f::normalize () [inline]

scales this vector to unit vector of length 1.0.

Definition at line 430 of file veMath.h.

References length().

12.65.3.17 void ve::vec4f::transform (const ve::mat4f & *m*)

transforms this vec3f by multiplying it with matrix m

12.65.3.18 float ve::vec4f::sqrLength () const [inline]

returns the squared length.

Definition at line 434 of file veMath.h.

References coord.

Referenced by length().

12.65.3.19 float ve::vec4f::length () const [inline]

returns absolute length.

Definition at line 436 of file veMath.h.

References sqrLength().

Referenced by normalize().

12.65.3.20 float ve::vec4f::scalarProduct (const vec4f & *v*) const [inline]

computes scalar product between this vector and vec3f v.

Definition at line 438 of file veMath.h.

References coord.

12.65.4 Friends And Related Function Documentation

12.65.4.1 std::ostream& operator<< (std::ostream & *os*, const ve::vec4f & *v*) [friend]

operator for output in streams

12.65.5 Member Data Documentation

12.65.5.1 float ve::vec4f::coord[4] [protected]

stores coordinate values.

Definition at line 445 of file veMath.h.

Referenced by coords(), operator *=(), operator+=(), operator/=(), operator[](), scalarProduct(), scale(), set(), sqrLength(), translate(), and vec4f().

The documentation for this class was generated from the following file:

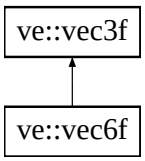
- veMath.h

12.66 ve::vec6f Class Reference

a class representing a six degree of freedom coordinate.

`#include <veMath.h>`

Inheritance diagram for ve::vec6f::



Public Member Functions

- vec6f ()
- vec6f (float x, float y, float z, float h=0, float p=0, float r=0)
- vec6f (const vec6f &source)
- vec6f (const vec3f &source)
- vec6f (const ve::mat4f &m)
- vec6f (const std::string &s, const std::string &separator=", \t\n\015")
- void set (const std::string &s, const std::string &separator=", \t\n\015")
- std::string str (const std::string &separator=" ", unsigned char nDigits=8) const
- vec6f & operator= (const vec6f &source)
- void operator+= (const vec6f &summand)
- bool operator== (const vec6f &sd) const
- bool operator!= (const vec6f &sd) const
- void transform (const ve::mat4f &m)
- void translate (float dx, float dy)
- void translate (float dx, float dy, float dz)
- void translate (const vec6f &v, float linearscale=1.0f, float angularscale=1.0f)
- void set (float x=0, float y=0, float z=0, float h=0, float p=0, float r=0)
- void set (const ve::mat4f &m)
- void get (float *f) const
- float & operator[] (unsigned int i)
- const float & operator[] (unsigned int i) const
- const vec6f operator * (float f) const
- const vec6f operator+ (const vec6f &v) const
- const vec6f operator+ (const vec3f &v) const
- const vec6f operator- (const vec6f &v) const
- void reset ()
- void remap (const unsigned int *mapping, const ve::vec6f &scale=ve::vec6f(1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f), const ve::vec6f &shift=ve::vec6f(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f), const ve::vec6f &deadzone=ve::vec6f(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f))

Static Public Member Functions

- static unsigned int size ()

Protected Attributes

- float ang [3]

Friends

- std::ostream & operator<< (std::ostream &os, const vec6f &sdof)

12.66.1 Detailed Description

a class representing a six degree of freedom coordinate.

Note that unfortunately there is no generally accepted standard for sixdofs rotations. Since the veLib normally defines forward along the +Y axis and upward along +Z axis, roll means a rotation around the Y axis, pitch around the X axis, and heading (i.e. yaw) around the Z axis. Euler rotations shall be preformed in exactly this order.

Definition at line 779 of file veMath.h.

12.66.2 Constructor & Destructor Documentation

12.66.2.1 ve::vec6f::vec6f () [inline]

default constructor, all ordinates are initialized as 0.

Definition at line 782 of file veMath.h.

References ang, and ve::vec3f::coord.

Referenced by operator ∗(), operator+(), and operator-().

12.66.2.2 ve::vec6f::vec6f (float *x*, float *y*, float *z*, float *h* = 0, float *p* = 0, float *r* = 0)

constructor from at least 3 float values

12.66.2.3 ve::vec6f::vec6f (const vec6f & *source*)

copy constructor

12.66.2.4 ve::vec6f::vec6f (const vec3f & *source*)

constructor from a vec3f

12.66.2.5 ve::vec6f::vec6f (const ve::mat4f & *m*) [inline]

constructor from a transformation matrix

Definition at line 790 of file veMath.h.

References set().

12.66.2.6 ve::vec6f::vec6f (const std::string & *s*, const std::string & *separator* = ", \t\n\015") [inline]

constructor interpreting a string, values are separated by optionally definable separators

Definition at line 792 of file veMath.h.

References set().

12.66.3 Member Function Documentation

12.66.3.1 void ve::vec6f::set (const std::string & *s*, const std::string & *separator* = ", \t\n\015")

sets all ordinates by a string, values are separated by optionally definable separators.

Reimplemented from ve::vec3f.

Referenced by vec6f().

12.66.3.2 std::string ve::vec6f::str (const std::string & *separator* = " ", unsigned char *nDigits* = 8) const

returns a string containing all data, optional arguments set separator between ordinates and number of digits per ordinate.

Reimplemented from ve::vec3f.

12.66.3.3 vec6f& ve::vec6f::operator= (const vec6f & *source*)

copy operator

12.66.3.4 void ve::vec6f::operator+= (const vec6f & *summand*)

+= operator. Sums correspondend ordinates.

12.66.3.5 bool ve::vec6f::operator== (const vec6f & *sd*) const

comparison operator equality

12.66.3.6 bool ve::vec6f::operator!= (const vec6f & *sd*) const [inline]

comparison operator inequality

Definition at line 804 of file veMath.h.

12.66.3.7 void ve::vec6f::transform (const ve::mat4f & *m*)

transforms this sixdof by multiplying it with matrix m

Note that this operation is computationally expensive, since this vec6f has to be transformed into a matrix first, then the matrices are multiplied, and then the result is retransformed.

Reimplemented from ve::vec3f.

12.66.3.8 void ve::vec6f::translate (float *dx*, float *dy*) [inline]

translates object by dx,dy

Reimplemented from ve::vec3f.

Definition at line 811 of file veMath.h.

References ve::vec3f::coord.

12.66.3.9 void ve::vec6f::translate (float *dx*, float *dy*, float *dz*) [inline]

translates object by dx,dy,dz

Reimplemented from ve::vec3f.

Definition at line 813 of file veMath.h.

References ve::vec3f::coord.

12.66.3.10 void ve::vec6f::translate (const vec6f & *v*, float *linearscale* = 1.0f, float *angularscale* = 1.0f)

translates the vec6f in the direction of v (basically the same as in class vec3f). It also rotates with respective scaling factor.

12.66.3.11 void ve::vec6f::set (float *x* = 0, float *y* = 0, float *z* = 0, float *h* = 0, float *p* = 0, float *r* = 0)

sets all ordinates

12.66.3.12 void ve::vec6f::set (const ve::mat4f & *m*)

sets sixdof as far as possible to an equivalent transformation as in matrix m

12.66.3.13 void ve::vec6f::get (float * *f*) const

copies values into a float pointer

12.66.3.14]

float& ve::vec6f::operator[] (unsigned int *i*)

returns single ordinate reference

Reimplemented from ve::vec3f.

12.66.3.15]

const float& ve::vec6f::operator[] (unsigned int *i*) const

returns single ordinate value, const

Reimplemented from ve::vec3f.

12.66.3.16 const vec6f ve::vec6f::operator * (float *f*) const [inline]

product operator. All ordinates of the return value are scaled by f.

Reimplemented from ve::vec3f.

Definition at line 828 of file veMath.h.

References ve::vec3f::coord, and vec6f().

12.66.3.17 const vec6f ve::vec6f::operator+ (const vec6f & *v*) const [inline]

addition operator vec6f+vec6f

Definition at line 830 of file veMath.h.

References ve::vec3f::coord, and vec6f().

12.66.3.18 const vec6f ve::vec6f::operator+ (const vec3f & *v*) const [inline]

addition operator vec6f+vec3f

Reimplemented from ve::vec3f.

Definition at line 834 of file veMath.h.

References ve::vec3f::coord, and vec6f().

12.66.3.19 const vec6f ve::vec6f::operator- (const vec6f & *v*) const [inline]

subtraction operator

Definition at line 838 of file veMath.h.

References ve::vec3f::coord, and vec6f().

12.66.3.20 void ve::vec6f::reset () [inline]

sets all ordinates to 0.0

Definition at line 843 of file veMath.h.

References ang, and ve::vec3f::coord.

12.66.3.21 static unsigned int ve::vec6f::size () [inline, static]

returns vec6f size (number of ordinates), mainly for "for" statements

Definition at line 845 of file veMath.h.

12.66.3.22 void ve::vec6f::remap (const unsigned int * *mapping*, const ve::vec6f & *scale* = `ve::vec6f(1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f)`, const ve::vec6f & *shift* = `ve::vec6f(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f)`, const ve::vec6f & *deadzone* = `ve::vec6f(0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f)`)

applies a remapping and an optional scaling The axes are first remapped, scaling, shifting and appliance of deadzone occures afterwards.

Parameters:

mapping must provide an array of 6 values from 0..5 in the desired order.

scale (optional) holds the linear scaling values.

shift (optional) holds values that are added to the final result.

deadzone (optional) holds range in which axes will be reported as zero.

12.66.4 Friends And Related Function Documentation

12.66.4.1 std::ostream& operator<< (std::ostream & *os*, const vec6f & *sdof*) `[friend]`

operator for output in streams

12.66.5 Member Data Documentation

12.66.5.1 float ve::vec6f::ang[3] `[protected]`

stores angle ordinates

Definition at line 861 of file veMath.h.

Referenced by reset(), and vec6f().

The documentation for this class was generated from the following file:

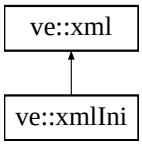
- veMath.h

12.67 ve::xml Class Reference

basic XML class.

`#include <veXml.h>`

Inheritance diagram for ve::xml::



Public Member Functions

- xml ()
- xml (const std::string &tag, const std::string &id="", const std::string &content="")
- xml (const ve::xml &source)
- virtual ~xml ()
- ve::xml & operator= (const ve::xml &source)
- virtual const std::string & getAttribute (const std::string &name, const std::string &defStr="") const
- virtual const std::string & getAttribute (const char *name, const std::string &defStr="") const
- virtual const std::string & getAttribute (unsigned int n, const std::string &defStr="") const
- virtual std::string attributeName (unsigned int n) const
- virtual void setAttribute (const std::string &name, const std::string &value)
- virtual void setAttribute (const std::string &name, const char *value)
- virtual void setAttribute (const std::string &name, int value)
- virtual void setAttribute (const std::string &name, unsigned int value)
- virtual void setAttribute (const std::string &name, float value)
- virtual void setAttribute (const std::string &name, bool value, bool asText=false)
- virtual int dropAttribute (const std::string &name)
- virtual void addChild (xml *xmlSt, int asCopy=0)
- virtual void addChild (const xml &xmlSt)
- virtual int dropChild (xml *xmlSt)
- virtual xml * parent () const
- virtual void tag (const std::string &name)
- virtual const std::string & tag () const
- virtual void content (const std::string &cont)
- virtual std::string & content ()
- virtual const std::string & content () const
- unsigned int nChildren () const
- unsigned int nAttributes () const
- virtual ve::xml * child (unsigned int number)
- virtual const ve::xml * child (unsigned int number) const
- virtual ve::xml * child (const std::string &tagName, const std::string &id="")
- virtual std::string str (unsigned int nTabs=0) const
- virtual ve::xml * subTag (const std::string &tagName, const std::string &id="")
- virtual const ve::xml * subTag (const std::string &tagName, const std::string &id="") const

- virtual ve::xml * subTag (const std::string &tagName, unsigned int id)
- virtual const ve::xml * subTag (const std::string &tagName, unsigned int id) const
- virtual ve::xml * subTagAttribute (const std::string &attribute, const std::string &value)
- virtual void clear ()
- virtual int load (const std::string &filename)
- void interpret (const std::string &xmlString)
- int save (const std::string &filename) const

Static Public Member Functions

- static std::string encode (std::string source)
- static std::string decode (std::string source)

Protected Member Functions

- void parse (const std::vector< std::string > &token)

Protected Attributes

- std::string tagStr
- std::vector< std::string > attr
- std::string contentStr
- std::vector< xml * > vChildren
- ve::xml * pParent
- int delByParent

12.67.1 Detailed Description

basic XML class.

This class implements the basic tree-like structure of XML and provides methods for accessing all different kind of data, file input-output and searching and parsing. A typical xml statement consists of a tag, some attributes, a content, and may contain substatements:

```
<tag attributeName1="attributeValue1" attributeName2="attributeValue2">
  content, may be arbitrary long
  <substatement1/>
  <substatement2/>
  ...
</tag>
```

Author:
gf
Revision
2.2

Definition at line 46 of file veXml.h.

12.67.2 Constructor & Destructor Documentation

12.67.2.1 ve::xml::xml ()

standard constructor

12.67.2.2 ve::xml::xml (const std::string & *tag*, const std::string & *id* = "", const std::string & *content* = "")

initializing constructor

12.67.2.3 ve::xml::xml (const ve::xml & *source*)

copy constructor

12.67.2.4 virtual ve::xml::~xml () [virtual]

destructor

12.67.3 Member Function Documentation

12.67.3.1 ve::xml& ve::xml::operator= (const ve::xml & *source*)

copy operator=

12.67.3.2 virtual const std::string& ve::xml::getAttribute (const std::string & *name*, const std::string & *defStr* = "") const [virtual]

returns attribute value as string. If it does not exist, defStr is returned.

Reimplemented in ve::xmlIni.

Referenced by getAttribute().

12.67.3.3 virtual const std::string& ve::xml::getAttribute (const char * *name*, const std::string & *defStr* = "") const [inline, virtual]

returns attribute value as string. If it does not exist, defStr is returned.

Reimplemented in ve::xmlIni.

Definition at line 62 of file veXml.h.

References getAttribute().

12.67.3.4 virtual const std::string& ve::xml::getAttribute (unsigned int *n*, const std::string & *defStr* = "") const [virtual]

returns value of attribute n as string. If it does not exist, defStr is returned.

Reimplemented in ve::xmlIni.

12.67.3.5 virtual std::string ve::xml::attributeName (unsigned int *n*) const [virtual]

returns name of attribute n as string. If attribute n does not exist, an empty string is returned.

12.67.3.6 virtual void ve::xml::setAttribute (const std::string & *name*, const std::string & *value*) [virtual]

sets a string attribute value. If it does not exist, a new attribute is added

Reimplemented in ve::xmlIni.

Referenced by setAttribute().

12.67.3.7 virtual void ve::xml::setAttribute (const std::string & *name*, const char * *value*) [inline, virtual]

sets a char * attribute value. If it does not exist, a new attribute is added

Reimplemented in ve::xmlIni.

Definition at line 70 of file veXml.h.

References setAttribute().

12.67.3.8 virtual void ve::xml::setAttribute (const std::string & *name*, int *value*) [inline, virtual]

sets an int attribute value. If it does not exist, a new attribute is added

Reimplemented in ve::xmlIni.

Definition at line 72 of file veXml.h.

References ve::i2s(), and setAttribute().

12.67.3.9 virtual void ve::xml::setAttribute (const std::string & *name*, unsigned int *value*) [inline, virtual]

sets an unsigned int attribute value. If it does not exist, a new attribute is added

Definition at line 74 of file veXml.h.

References ve::i2s(), and setAttribute().

12.67.3.10 virtual void ve::xml::setAttribute (const std::string & *name*, float *value*) [inline, virtual]

sets a float attribute value. If it does not exist, a new attribute is added

Reimplemented in ve::xmlIni.

Definition at line 76 of file veXml.h.

References ve::f2s(), and setAttribute().

12.67.3.11 virtual void ve::xml::setAttribute (const std::string & *name*, bool *value*, bool *asText* = false) [inline, virtual]

sets a bool attribute value. If it does not exist, a new attribute is added

Reimplemented in ve::xmlIni.

Definition at line 78 of file veXml.h.

References ve::b2s(), and setAttribute().

12.67.3.12 virtual int ve::xml::dropAttribute (const std::string & *name*) [virtual]

remove attribute

12.67.3.13 virtual void ve::xml::addChild (xml * *xmlSt*, int *asCopy* = 0) [virtual]

adds child statement either as pointer reference or as new copy, which will be only deleted via the parent

12.67.3.14 virtual void ve::xml::addChild (const xml & *xmlSt*) [virtual]

adds a copy of xmlSt as new child statement, its scope is identical to its parent's scope

12.67.3.15 virtual int ve::xml::dropChild (xml * *xmlSt*) [virtual]

drops child statement. If delByParent, the memory is actually freed.

12.67.3.16 virtual xml* ve::xml::parent () const [inline, virtual]

returns address of parent statement or NULL if toplevel

Definition at line 88 of file veXml.h.

References pParent.

12.67.3.17 virtual void ve::xml::tag (const std::string & *name*) [inline, virtual]

sets tag name

Definition at line 90 of file veXml.h.

References tagStr.

12.67.3.18 virtual const std::string& ve::xml::tag () const [inline, virtual]

returns tag name

Definition at line 92 of file veXml.h.

References tagStr.

12.67.3.19 virtual void ve::xml::content (const std::string & *cont*) [inline, virtual]

sets content

Definition at line 94 of file veXml.h.

References contentStr.

12.67.3.20 virtual std::string& ve::xml::content () [inline, virtual]

returns content, content is editable.

Definition at line 96 of file veXml.h.

References contentStr.

12.67.3.21 virtual const std::string& ve::xml::content () const [inline, virtual]

returns content, const.

Definition at line 98 of file veXml.h.

References contentStr.

12.67.3.22 unsigned int ve::xml::nChildren () const [inline]

returns number of children statements

Reimplemented in ve::xmlIni.

Definition at line 100 of file veXml.h.

References vChildren.

12.67.3.23 unsigned int ve::xml::nAttributes () const [inline]

returns number of attributes

Reimplemented in ve::xmlIni.

Definition at line 102 of file veXml.h.

References attr.

12.67.3.24 virtual ve::xml* ve::xml::child (unsigned int *number*) [virtual]

returns the nth child statement or NULL

Reimplemented in ve::xmlIni.

12.67.3.25 virtual const ve::xml* ve::xml::child (unsigned int *number*) const [virtual]

returns the nth child statement or NULL, const

Reimplemented in ve::xmlIni.

12.67.3.26 virtual ve::xml* ve::xml::child (const std::string & *tagName*, const std::string & *id* = "") [virtual]

returns a specified child statement or NULL

12.67.3.27 virtual std::string ve::xml::str (unsigned int *nTabs* = 0) const [virtual]

returns a preformatted xml string

Reimplemented in ve::xmlIni.

Referenced by ve::dataChar::str().

12.67.3.28 virtual ve::xml* ve::xml::subTag (const std::string & *tagName*, const std::string & *id* = "") [virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found

Reimplemented in ve::xmlIni.

12.67.3.29 virtual const ve::xml* ve::xml::subTag (const std::string & *tagName*, const std::string & *id* = "") const [virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found, const

Reimplemented in ve::xmlIni.

12.67.3.30 virtual ve::xml* ve::xml::subTag (const std::string & *tagName*, unsigned int *id*) [virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found

Reimplemented in ve::xmlIni.

12.67.3.31 virtual const ve::xml* ve::xml::subTag (const std::string & *tagName*, unsigned int *id*) const [virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found, const

Reimplemented in ve::xmlIni.

12.67.3.32 virtual ve::xml* ve::xml::subTagAttribute (const std::string & *attribute*, const std::string & *value*) [virtual]

returns a pointer to a subtag with a suitable attribute and value, or NULL if none is found

12.67.3.33 virtual void ve::xml::clear () [virtual]

clears all existing information.

12.67.3.34 virtual int ve::xml::load (const std::string & *filename*) [virtual]

loads xml from disk, previous information is cleared.

Parameters:
filename url of the file to be loaded

Returns:
number of detected errors.

Reimplemented in ve::xmlIni.

12.67.3.35 void ve::xml::interpret (const std::string & *xmlString*)

interprets xmlString as xml, previous information is cleared.

12.67.3.36 int ve::xml::save (const std::string & *filename*) const

writes xml to disk.

12.67.3.37 static std::string ve::xml::encode (std::string *source*) [static]

encodes special characters in xml

12.67.3.38 static std::string ve::xml::decode (std::string *source*) [static]

translates encoded special characters from XML to ascii

12.67.3.39 void ve::xml::parse (const std::vector< std::string > & *token*) [protected]

converts preparsed tokens to an xml

12.67.4 Member Data Documentation

12.67.4.1 std::string ve::xml::tagStr [protected]

stores xml tag

Definition at line 141 of file veXml.h.

Referenced by tag().

12.67.4.2 std::vector<std::string> ve::xml::attr [protected]

stores xml attributes, keys have even indizes, values odd indizes.

Definition at line 143 of file veXml.h.

Referenced by nAttributes().

12.67.4.3 std::string ve::xml::contentStr [protected]

content of the xml statement

Definition at line 145 of file veXml.h.

Referenced by content().

12.67.4.4 std::vector<xml *> ve::xml::vChildren [protected]

stores pointers to subordinate statements

Definition at line 147 of file veXml.h.

Referenced by nChildren().

12.67.4.5 ve::xml* ve::xml::pParent [protected]

stores pointer to superordinate statement

Definition at line 149 of file veXml.h.

Referenced by parent().

12.67.4.6 int ve::xml::delByParent [protected]

stores whether statement should be deleted when parent gets deleted

Definition at line 151 of file veXml.h.

The documentation for this class was generated from the following file:

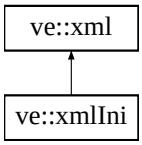
- veXml.h

12.68 ve::xmlIni Class Reference

A class for reading variable values from XML inifiles. This class complements the basic xml class with convenient parsing functions for basic data types. It is a good way for reading initialization information and for transfering information between programs in temporary files.

`#include <veXml.h>`

Inheritance diagram for ve::xmlIni::



Public Member Functions

- xmlIni ()
- xmlIni (const std::string &id, int content)
- xmlIni (const std::string &id, float content)
- xmlIni (const std::string &id, const std::string &content)
- xmlIni (const std::string &id, std::vector< float > vContent)
- xmlIni (const std::string &id, std::vector< int > vContent)
- xmlIni (const std::string &id, std::vector< std::vector< int > > vvContent)
- xmlIni (const std::string &id, std::vector< std::vector< float > > vvContent)
- xmlIni (const ve::xmlIni &source)
- xmlIni (const ve::xml &source)
- virtual int load (const std::string &filename)
- xmlIni & operator= (const xmlIni &source)
- int read (int &pInt, const std::string &id, int defValue=0) const
- int read (unsigned int &uInt, const std::string &id, unsigned int defValue=0) const
- int read (float &fl, const std::string &id, float defValue=0.0f) const
- int read (bool &yesno, const std::string &id, bool defValue=false) const
- int read (char *&pChar, const std::string &id, char *defValue="") const
- int read (std::string &str, const std::string &id, std::string defValue="") const
- int read (std::vector< std::vector< int > > &vvInt, const std::string &id) const
- int read (std::vector< int > &vInt, const std::string &id) const
- int read (int *pInt, unsigned int n, const std::string &id, int defValue=0) const
- int read (std::vector< std::vector< float > > &vvFloat, const std::string &id) const
- int read (std::vector< float > &vFloat, const std::string &id) const
- int read (float *pFloat, unsigned int n, const std::string &id, float defValue=0.0) const
- int read (std::vector< char * > &vStr, const std::string &id) const
- int read (std::vector< std::string > &vStr, const std::string &id) const
- int focusOn (const std::string &tagName, const std::string &id="")
- int focusOn (const std::string &tagName, unsigned int id)
- void focusOff ()
- unsigned int nChildren () const
- virtual xml * child (unsigned int number)
- virtual const xml * child (unsigned int number) const

- virtual xml * child (std::string tagName, std::string id="")
- unsigned int nAttributes () const
- virtual void setAttribute (const std::string &name, const std::string &value)
- virtual void setAttribute (const std::string &name, const char *value)
- virtual void setAttribute (const std::string &name, int value)
- virtual void setAttribute (const std::string &name, float value)
- virtual void setAttribute (const std::string &name, bool value, bool asText=false)
- virtual const std::string & getAttribute (const std::string &name, const std::string &defStr="") const
- virtual const std::string & getAttribute (const char *name, const std::string &defStr="") const
- virtual const std::string & getAttribute (unsigned int n, const std::string &defStr="") const
- virtual std::string str (unsigned int nTabs=0) const
- virtual ve::xml * subTag (const std::string &tagName, const std::string &id="")
- virtual const ve::xml * subTag (const std::string &tagName, const std::string &id="") const
- virtual ve::xml * subTag (const std::string &tagName, unsigned int id)
- virtual const ve::xml * subTag (const std::string &tagName, unsigned int id) const

Protected Attributes

- ve::xml * focus

12.68.1 Detailed Description

A class for reading variable values from XML inifiles. This class complements the basic xml class with convenient parsing functions for basic data types. It is a good way for reading initialization information and for transfering information between programs in temporary files.

brief usage: xml.load("filename.xml") loads the content of the xml file filename.xml into memory. After that the content of this file is parsable using the read methods.

All the read methods share a similar interface: The first argument is a reference of the variable where the data will be stored. The second argument is a char * or a c++ string which contains the id of the searched statement. An optional third argument contains a standard value, which is assigned if a suitable statement is not found.

Example

```
<xml_code_snippet>
<int id="trials"> 5 </int>
<string id="mail_address">agbu\@tuebingen.mpg.de</string>
</xml_code_snippet>

// c code snippet for reading that data:
int numTrials=0;   // these vars will give storage for our read data
string replyTo;

ve::xmlIni myIniFile;                       // create an xml file object
myIniFile.load("../info/test.xml");     // open a certain file for reading
myIniFile.read(numTrials,"trials",3);   // this method call searches for an
// xml statement of type int with the
// attribute id="trials", and stores
// the found data in numTrials. For
// stability an optional default
// initialization should be provided.
myIniFile.read(replyTo,"mail_address"); // dito, only with a string
```

Author:
gf & jmw
Revision
2.2

Definition at line 203 of file veXml.h.

12.68.2 Constructor & Destructor Documentation

12.68.2.1 ve::xmlIni::xmlIni () [inline]

standard constructor

Definition at line 206 of file veXml.h.

References focus.

12.68.2.2 ve::xmlIni::xmlIni (const std::string & *id*, int *content*)

constructor taking int argument

12.68.2.3 ve::xmlIni::xmlIni (const std::string & *id*, float *content*)

constructor taking float argument

12.68.2.4 ve::xmlIni::xmlIni (const std::string & *id*, const std::string & *content*)

constructor taking std::string argument

12.68.2.5 ve::xmlIni::xmlIni (const std::string & *id*, std::vector< float > *vContent*)

constructor taking float vector

12.68.2.6 ve::xmlIni::xmlIni (const std::string & *id*, std::vector< int > *vContent*)

constructor taking int vector

12.68.2.7 ve::xmlIni::xmlIni (const std::string & *id*, std::vector< std::vector< int > > *vvContent*)

constructor taking vector of vector of int

12.68.2.8 ve::xmlIni::xmlIni (const std::string & *id*, std::vector< std::vector< float > > *vvContent*)

constructor taking vector of vector of float

12.68.2.9 ve::xmlIni::xmlIni (const ve::xmlIni & *source*)

copy constructor

12.68.2.10 ve::xmlIni::xmlIni (const ve::xml & *source*)

copy constructor with an xml as source

12.68.3 Member Function Documentation

12.68.3.1 virtual int ve::xmlIni::load (const std::string & *filename*) [virtual]

loads xml inifile from disk

in addition to xml::load, xml <include url="xyz"/> statements are resolved.

Parameters:
filename url of the file to be loaded

Returns:
number of detected errors.

Reimplemented from ve::xml.

12.68.3.2 xmlIni& ve::xmlIni::operator= (const xmlIni & *source*)

copy operator=

12.68.3.3 int ve::xmlIni::read (int & *pInt*, const std::string & *id*, int *defValue* = 0) const

reads an int value, if not found in file assign default value

12.68.3.4 int ve::xmlIni::read (unsigned int & *uInt*, const std::string & *id*, unsigned int *defValue* = 0) const

reads an unsigned int value, if not found in file default value is assigned

12.68.3.5 int ve::xmlIni::read (float & *fl*, const std::string & *id*, float *defValue* = 0.0f) const

reads a float value, if not found in file assign default value

12.68.3.6 int ve::xmlIni::read (bool & *yesno*, const std::string & *id*, bool *defValue* = false) const

reads a bool value, if not found in file assign default value

12.68.3.7 int ve::xmlIni::read (char *& *pChar*, const std::string & *id*, char * *defValue* = "") const

reads a c string value, if not found in file assign default value

12.68.3.8 int ve::xmlIni::read (std::string & *str*, const std::string & *id*, std::string *defValue* = "") const

reads a c++ string value, if not found in file assign default value

12.68.3.9 int ve::xmlIni::read (std::vector< std::vector< int > > & *vvInt*, const std::string & *id*) const

reads a 2 dimensional int vector

12.68.3.10 int ve::xmlIni::read (std::vector< int > & *vInt*, const std::string & *id*) const

reads a 1 dimensional int vector

12.68.3.11 int ve::xmlIni::read (int * *pInt*, unsigned int *n*, const std::string & *id*, int *defValue* = 0) const

reads n values into a 1 dimensional int array. If not enough values can be extracted, defValue is assigned.

12.68.3.12 int ve::xmlIni::read (std::vector< std::vector< float > > & *vvFloat*, const std::string & *id*) const

reads a 2 dimensional float vector

12.68.3.13 int ve::xmlIni::read (std::vector< float > & *vFloat*, const std::string & *id*) const

reads a 1 dimensional float vector

12.68.3.14 int ve::xmlIni::read (float * *pFloat*, unsigned int *n*, const std::string & *id*, float *defValue* = 0.0) const

reads n values into a 1 dimensional float array. If not enough values can be extracted, defValue is assigned.

12.68.3.15 int ve::xmlIni::read (std::vector< char * > & *vStr*, const std::string & *id*) const

reads a char array vector

12.68.3.16 int ve::xmlIni::read (std::vector< std::string > & *vStr*, const std::string & *id*) const

reads a string vector

12.68.3.17 int ve::xmlIni::focusOn (const std::string & *tagName*, const std::string & *id* = "")

tries to restrict search to the given tag, if successful 0 is returned, otherwise ve::ERR_NOT_-FOUND

12.68.3.18 int ve::xmlIni::focusOn (const std::string & *tagName*, unsigned int *id*)

tries to restrict search to the given tag, if successful 0 is returned, otherwise ve::ERR_NOT_-FOUND

12.68.3.19 void ve::xmlIni::focusOff () [inline]

releases restriction of search

Definition at line 266 of file veXml.h.

References focus.

12.68.3.20 unsigned int ve::xmlIni::nChildren () const [inline]

returns number of children statements, using current focus

Reimplemented from ve::xml.

Definition at line 268 of file veXml.h.

References focus.

12.68.3.21 virtual xml* ve::xmlIni::child (unsigned int *number*) [inline, virtual]

returns a pointer to the nth child statement or NULL

Reimplemented from ve::xml.

Definition at line 270 of file veXml.h.

References focus.

12.68.3.22 virtual const xml* ve::xmlIni::child (unsigned int *number*) const [inline, virtual]

returns a pointer to the nth child statement or NULL, const

Reimplemented from ve::xml.

Definition at line 272 of file veXml.h.

References focus.

12.68.3.23 virtual xml* ve::xmlIni::child (std::string *tagName*, std::string *id* = "") [inline, virtual]

returns a pointer to a specified child statement or NULL

Definition at line 274 of file veXml.h.

References focus.

12.68.3.24 unsigned int ve::xmlIni::nAttributes () const [inline]

returns the number of attributes, using current focus

Reimplemented from ve::xml.

Definition at line 276 of file veXml.h.

References focus.

12.68.3.25 virtual void ve::xmlIni::setAttribute (const std::string & *name*, const std::string & *value*) [inline, virtual]

sets a string attribute value, using current focus. If it does not exist, a new attribute is added

Reimplemented from ve::xml.

Definition at line 279 of file veXml.h.

References focus.

Referenced by setAttribute().

12.68.3.26 virtual void ve::xmlIni::setAttribute (const std::string & *name*, const char * *value*) [inline, virtual]

sets a char * attribute value, using current focus. If it does not exist, a new attribute is added

Reimplemented from ve::xml.

Definition at line 282 of file veXml.h.

References setAttribute().

12.68.3.27 virtual void ve::xmlIni::setAttribute (const std::string & *name*, int *value*) [inline, virtual]

sets a int attribute value, using current focus. If it does not exist, a new attribute is added

Reimplemented from ve::xml.

Definition at line 284 of file veXml.h.

References ve::i2s(), and setAttribute().

12.68.3.28 virtual void ve::xmlIni::setAttribute (const std::string & *name*, float *value*) [inline, virtual]

sets a float attribute value, using current focus. If it does not exist, a new attribute is added

Reimplemented from ve::xml.

Definition at line 286 of file veXml.h.

References ve::f2s(), and setAttribute().

12.68.3.29 virtual void ve::xmlIni::setAttribute (const std::string & *name*, bool *value*, bool *asText* = false) [inline, virtual]

sets a bool attribute value, using current focus. If it does not exist, a new attribute is added

Reimplemented from ve::xml.

Definition at line 288 of file veXml.h.

References ve::b2s(), and setAttribute().

12.68.3.30 virtual const std::string& ve::xmlIni::getAttribute (const std::string & *name*, const std::string & *defStr* = "") const [inline, virtual]

returns an attribute value as string. If it does not exist, defStr is returned.

Reimplemented from ve::xml.

Definition at line 290 of file veXml.h.

References focus.

12.68.3.31 virtual const std::string& ve::xmlIni::getAttribute (const char * *name*, const std::string & *defStr* = "") const [inline, virtual]

returns an attribute value as string. If it does not exist, defStr is returned.

Reimplemented from ve::xml.

Definition at line 293 of file veXml.h.

References focus.

12.68.3.32 virtual const std::string& ve::xmlIni::getAttribute (unsigned int *n*, const std::string & *defStr* = "") const [inline, virtual]

returns an value of attribute n as string, using current focus. If it does not exist, defStr is returned.

Reimplemented from ve::xml.

Definition at line 296 of file veXml.h.

References focus.

12.68.3.33 virtual std::string ve::xmlIni::str (unsigned int *nTabs* = 0) const [inline, virtual]

returns a preformatted xml string

Reimplemented from ve::xml.

Definition at line 299 of file veXml.h.

References focus.

12.68.3.34 virtual ve::xml* ve::xmlIni::subTag (const std::string & *tagName*, const std::string & *id* = "") [inline, virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found

Reimplemented from ve::xml.

Definition at line 302 of file veXml.h.

References focus.

12.68.3.35 virtual const ve::xml* ve::xmlIni::subTag (const std::string & *tagName*, const std::string & *id* = "") const [inline, virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found, const

Reimplemented from ve::xml.

Definition at line 305 of file veXml.h.

References focus.

12.68.3.36 virtual ve::xml* ve::xmlIni::subTag (const std::string & *tagName*, unsigned int *id*) [inline, virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found

Reimplemented from ve::xml.

Definition at line 308 of file veXml.h.

References focus.

12.68.3.37 virtual const ve::xml* ve::xmlIni::subTag (const std::string & *tagName*, unsigned int *id*) const [inline, virtual]

returns a pointer to a subtag with suitable tag and id, or NULL if none is found, const

Reimplemented from ve::xml.

Definition at line 311 of file veXml.h.

12.68.4 Member Data Documentation

12.68.4.1 ve::xml* ve::xmlIni::focus [protected]

stores current search restriction or NULL if none

Definition at line 312 of file veXml.h.

Referenced by child(), focusOff(), getAttribute(), nAttributes(), nChildren(), setAttribute(), str(), subTag(), and xmlIni().

The documentation for this class was generated from the following file:

- veXml.h

Chapter 13

veLib File Documentation

13.1 veCollision.h File Reference

Contains the classes ve::collision and ve::collisionSurface.

```
#include "veStd.h"
#include "veMath.h"
#include "veDataContainer.h"
#include <map>
```

Namespaces

- namespace ve

Classes

- class ve::collision
  base class for collision models.
- class ve::_collisionSurfaceObj
- class ve::collisionSurface
  collision model based on motion surfaces.

Enumerations

- enum ve::collisionType { ve::COLLISION_GROUND = 0, ve::COLLISION_FLY }

13.1.1 Detailed Description

Contains the classes ve::collision and ve::collisionSurface.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
MvdH & gf

Revision
2.1

Definition in file veCollision.h.

13.2 veConfig.h File Reference

Central configuration file for the Virtual Environments Library (veLib).

Defines

- #define _HAVE_GL 1
- #define _HAVE_SDL 1
- #define _HAVE_LIBPNG 1
- #define _HAVE_LIBZ 1
- #define _HAVE_LIBJPEG 1
- #define _HAVE_X 1
- #define _HAVE_OPEN_AL 1

13.2.1 Detailed Description

Central configuration file for the Virtual Environments Library (veLib).

This file is the central configuration instance for all external library bindings of veLib. Here you can define which external libraries veLib should use and should provide interfaces. Its standard settings are fairly traditional and should work with all existing veLib programs.

Important note: If you manually edit this file, it might be necessary to adjust the library linking options in the file VELIB_TOP_DIR/src/Malerules. Further advice is given at the particular defines that follow.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
2.6

Definition in file veConfig.h.

13.2.2 Define Documentation

13.2.2.1 #define _HAVE_GL 1

Do we have OpenGL? Undef this if you don't have or plan to use OpenGL, and veLib will not work. ;-) Sorry, OpenGL is currently mandatory. More Information about OpenGL can be found at `http://www.opengl.org`

Definition at line 32 of file veConfig.h.

13.2.2.2 #define _HAVE_SDL 1

Do we have SDL? Undef this if you don't have or plan to use SDL, a cross-platform windowing and input handling toolkit, and veSDL will not be built. SDL is open source and free software. It can be obtained from http://www.libsdl.org. SDL support is currently experimental! If this option is used, the applications have also to be linked with -lSDL and correct library path settings.

Definition at line 44 of file veConfig.h.

13.2.2.3 #define _HAVE_LIBPNG 1

Do we have libPNG? This free image library allows veImage to load PNG files together with zlib (needs -lpng -lz linking options). The homepage of libPNG is http://www.libpng.org/pub/png/ .

Definition at line 53 of file veConfig.h.

13.2.2.4 #define _HAVE_LIBZ 1

Do we have zLib? zLib is a free compression library (.gz) that for instance is used by libPNG, or by veGeoObj to load compressed VRML files. zLib needs the -lz linking option and can be obtained from http://www.gzip.org/zlib/ .

Definition at line 62 of file veConfig.h.

13.2.2.5 #define _HAVE_LIBJPEG 1

Do we have libJPEG? This free image library allows veImage to load jpeg images (needs -ljpeg linking option). libJPEG can be found at http://www.ijg.org/ .

Definition at line 70 of file veConfig.h.

13.2.2.6 #define _HAVE_X 1

Do we have X-Window support ? This is set automatically if you are not using a Win32-Compiler. Under Win32, veDeviceKbdX and veDeviceMouseX will not be built.

Definition at line 92 of file veConfig.h.

13.2.2.7 #define _HAVE_OPEN_AL 1

Do we have OpenAL? Undef this if you don't have or plan to use OpenAL, a cross-platform 3D audio library, and ve::deviceAudioAL will not be built. OpenAL is open source and free software, it can be obtained from http://www.openal.org. If this option is used, the applications have also to be linked with -lopenal under UNIX and -lopenal32 under Windows, and correct library path settings are required.

Definition at line 131 of file veConfig.h.

13.3 veDataContainer.h File Reference

Definition of the standard veLib data container.

```
#include "veTypes.h"
#include "veMath.h"
#include "veXml.h"
```

Namespaces

- namespace ve

Classes

- class ve::dataBaseStruct
  internal struct used as coomon data container header.
- class ve::dataCharStruct
  internal struct underlying the dataChar class
- class ve::dataContainerStruct
  internal struct underlying the dataContainer class
- union ve::dataUnion
  internal union comprising all low level data structures of various data containers
- class ve::dataChar
  veLib basic data class.
- class ve::dataContainer
  veLib data container representing a simulation object

Variables

- const unsigned int ve::totalSize = 256
- const unsigned int ve::headerSize = 64

13.3.1 Detailed Description

Definition of the standard veLib data container.

The general data container for all data transfers between classes. The dataContainer class realizes the standadized data transfer and normation between all ve classes.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf & weyel & malte & mvdh

Revision
2.4

Definition in file veDataContainer.h.

13.4 veDevice.h File Reference

This file contains the basic general ve::device and ve::deviceGraphics class declarations.

```
#include "veTypes.h"
#include "veMath.h"
#include "vePlugins.h"
#include <map>
```

Namespaces

- namespace ve

Classes

- class ve::_exposedVar
- class ve::device
  The general device class.
- class ve::deviceGraphics
  base class for 3D visualization classes.

13.4.1 Detailed Description

This file contains the basic general ve::device and ve::deviceGraphics class declarations.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh / weyel / gf

Revision
2.9

Definition in file veDevice.h.

13.5 veDeviceAudioAL.h File Reference

contains the class ve::deviceAudioAL.

```
#include "veDevice.h"
#include "veXml.h"
#include "veMath.h"
```

Namespaces

- namespace ve

Classes

- class ve::deviceAudioAL
  a class for 3D audio simulation based on OpenAL

13.5.1 Detailed Description

contains the class ve::deviceAudioAL.

veLib Copyright 2003-2005 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.1

Definition in file veDeviceAudioAL.h.

13.6 veDeviceContainer.h File Reference

This file contains the classes ve::deviceLog ve::deviceContainer.

```
#include "veStd.h"
#include "veDevice.h"
#include "veDataContainer.h"
#include "veXml.h"
```

Namespaces

- namespace ve

Classes

- class ve::deviceLog
  A (pseudo-)device that writes its traffic into log files or to stdout.
- class ve::deviceContainer
  The flexible device container.

13.6.1 Detailed Description

This file contains the classes ve::deviceLog ve::deviceContainer.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
2.3

Definition in file veDeviceContainer.h.

13.7 veDeviceDirectX.h File Reference

This file contains the veDeviceDirectX class.

```
#include "veDevice.h"
```

13.7.1 Detailed Description

This file contains the veDeviceDirectX class.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
Michael Weyel

Revision
2.0

Definition in file veDeviceDirectX.h.

13.8 veDeviceGraphicsGL.h File Reference

Contains classes for graphics primitives and a the ve::deviceGraphicsGL visualization.

```
#include "veStd.h"
#include <veMath.h>
#include <veXml.h>
#include <GL/gl.h>
#include "veDevice.h"
#include "veGeoObj.h"
```

Namespaces

- namespace ve

Classes

- class ve::glBillboard
  a class for rendering billboards
- class ve::glBillbAnim
  a class for rendering animated billboards
- class ve::_modelRef
- class ve::deviceGraphicsGL
  OpenGL based graphics device.

13.8.1 Detailed Description

Contains classes for graphics primitives and a the ve::deviceGraphicsGL visualization.

veLib Copyright 2003-2005 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.8

Definition in file veDeviceGraphicsGL.h.

13.9 veDeviceNetwork.h File Reference

This file contains the veDeviceNetwork class interface.

```
#include "veConfig.h"
#include "veStd.h"
#include "veDevice.h"
#include "veTypes.h"
#include "veDataContainer.h"
#include "veUtils.h"
#include <SDL.h>
#include <SDL_thread.h>
#include <list>
#include <stack>
#include <queue>
```

Namespaces

- namespace ve

Classes

- struct ve::networkTime
- struct ve::delayedData
- struct ve::mandatoryData
- struct ve::connectionInfo

  a struct with information on one network connection

- class ve::deviceNetwork

  Network device class using pure UDP.

Defines

- #define DEFAULT_SERVER_PORT 5000
- #define DEFAULT_SDISPLAY_PORT 5009
- #define DEFAULT_SKARTSOUND_PORT 5010
- #define DEFAULT_SJOYSTICK_PORT 5011
- #define DEFAULT_SSOUND_PORT 5012
- #define DEFAULT_STRACK_PORT 5013
- #define DEFAULT_SMOUSE_PORT 5014
- #define DEFAULT_ARTUDP_PORT 5592
- #define NETWORK_MAGIC 0x47110815
- #define MAX_THREADS 128
- #define MSG_NOSIGNAL 0

Enumerations

- enum ve::networkMode { ve::CLIENT, ve::SERVER }

Variables

- const unsigned int ve::NETWORK_MAX_CONNECTIONS = 32
- const unsigned int ve::NETWORK_NUM_CONTAINERS = 5
- const unsigned int ve::NETWORK_BUFSIZE = NETWORK_NUM_CONTAINERS * ve::total-Size
- const unsigned int ve::NETWORK_BUF_CONTAINERS = 10

13.9.1 Detailed Description

This file contains the veDeviceNetwork class interface.

The network device allows a transparent transport of ve::data containers over the network from one/or multiple servers to one/or multiple clients. This file contains also the veNetworkInputTask and veNetworkOutputTask classes.

veLib Copyright 2003, 2004 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh / weyel

Revision
2.12

Definition in file veDeviceNetwork.h.

13.9.2 Define Documentation

13.9.2.1 #define DEFAULT_SERVER_PORT 5000

default port for any kind of server device

Definition at line 36 of file veDeviceNetwork.h.

13.9.2.2 #define DEFAULT_SDISPLAY_PORT 5009

default port for any kind of diplay server

Definition at line 38 of file veDeviceNetwork.h.

13.9.2.3 #define DEFAULT_SKARTSOUND_PORT 5010

default port for the kart sound server

Definition at line 40 of file veDeviceNetwork.h.

13.9.2.4 #define DEFAULT_SJOYSTICK_PORT 5011

default port for any kind of joystick server
Definition at line 42 of file veDeviceNetwork.h.

13.9.2.5 #define DEFAULT_SSOUND_PORT 5012

default port for any kind of sound server
Definition at line 44 of file veDeviceNetwork.h.

13.9.2.6 #define DEFAULT_STRACK_PORT 5013

default port for any kind of tracking server
Definition at line 46 of file veDeviceNetwork.h.

13.9.2.7 #define DEFAULT_SMOUSE_PORT 5014

default port for a mouse server
Definition at line 48 of file veDeviceNetwork.h.

13.9.2.8 #define DEFAULT_ARTUDP_PORT 5592

default port for the ART udp protocol
Definition at line 50 of file veDeviceNetwork.h.

13.9.2.9 #define NETWORK_MAGIC 0x47110815

magic number for connection test
Definition at line 55 of file veDeviceNetwork.h.

13.9.2.10 #define MAX_THREADS 128

howmany threads we ever run in parallel
Definition at line 72 of file veDeviceNetwork.h.

13.10 veDeviceSDL.h File Reference

Contains the libSDL based input system. Started 2003-12-13 by Gerald.Franz@tuebingen.mpg.de.

```
#include "veConfig.h"
#include "veStd.h"
#include "veMath.h"
#include "veXml.h"
#include "veTypes.h"
#include "veDevice.h"
#include "veGlUtils.h"
#include <GL/glx.h>
```

Namespaces

- namespace ve

Classes

- class ve::deviceWindow
  class for window handling, keyboard and mouse input.
- class ve::deviceJoystick
  class for joystick input.

Typedefs

- typedef _SDL_Joystick SDL_Joystick

13.10.1 Detailed Description

Contains the libSDL based input system. Started 2003-12-13 by Gerald.Franz@tuebingen.mpg.de.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.4

Definition in file veDeviceSDL.h.

13.11 veGeoObj.h File Reference

Contains geometry primitives classes.

```
#include "veStd.h"
#include "veXml.h"
#include "veMath.h"
```

Namespaces

- namespace ve

Classes

- class ve::ioFileHandler
  a small auxilliary class that allows the writing of plugin file handlers
- class ve::ioVrml
  a class for VRML input/output
- class ve::ioX3d
  a class for X3d input/output
- class ve::geoObj
  base class for all derived geometry objects.
- class ve::geoGroup
  base class for organizing ve::geoObjects in a tree-like structure.
- class ve::geoMesh
  a class for static indexedFaceSet mesh objects.
- class ve::geoElevationGrid
  a class for interpreting and displaying elevation grids / terrain models

Functions

- std::ostream & operator<< (std::ostream &os, const ve::geoMesh &mesh)

13.11.1 Detailed Description

Contains geometry primitives classes.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.12

Definition in file veGeoObj.h.

13.11.2 Function Documentation

13.11.2.1 std::ostream& operator<< (std::ostream & *os*, const ve::geoMesh & *mesh*)

operator for output of geoMeshes in ostreams.

13.12 veGlUtils.h File Reference

A collection of OpenGL auxilliary classes.

```
#include <GL/gl.h>
#include "veStd.h"
#include "veTypes.h"
#include "veMath.h"
#include "veDataContainer.h"
#include "veDeviceSDL.h"
```

Namespaces

- namespace ve

Classes

- class ve::glText
  an abstract base class for OpenGL font renderers.
- class ve::glTextTxf
  a class for rendering txf texture fonts in OpenGL.
- class ve::ovlObj
  parent class for 2D overlay objects.
- class ve::ovlLabel
  a single line text overlay widget.
- class ve::ovlRect
  a class for displaying untextextured rectangles in the overlay plane.
- class ve::ovlImage
  a class for displaying images in the overlay plane.

Enumerations

- enum ve::align_t

Functions

- bool ve::hasGlExtension (const std::string &which)

13.12.1 Detailed Description

A collection of OpenGL auxilliary classes.

glText an abstract base class for OpenGL font renderers

glTextTxf loading and low level rendering of txf texture fonts

ovlPlane framework for managing overlay objects

ovlObj parent class for overlay 2d objects

ovlLabel single line text rendering

ovlRect display of 2D rectangles

ovlImage display of images.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
2.5

Definition in file veGlUtils.h.

13.13 veImage.h File Reference

contains the classes veImage and veTexture.

```
#include "veStd.h"
```

Namespaces

- namespace ve
- namespace std

Classes

- class ve::image
  ve::image is a class for basic image file input/output.
- class ve::glTexture
  a class for OpenGL image and texture operations.

Functions

- std::ostream & std::operator<< (std::ostream &os, const ve::image &img)

13.13.1 Detailed Description

contains the classes veImage and veTexture.

veLib Copyright 2003, 2004 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.3

Definition in file veImage.h.

13.14 veIo3ds.h File Reference

A basic 3ds loader class.

`#include <veStd.h>`

`#include <veMath.h>`

Namespaces

- namespace ve

Classes

- class ve::_materialInfo
- class ve::_materialRef
- class ve::_3dsObject
- class ve::io3ds

  this class handles the loading of 3ds files

13.14.1 Detailed Description

A basic 3ds loader class.

This file is substancially based on a tutorial by

Ben Humphrey (DigiBen)

Game Programmer

DigiBen@GameTutorials.com

Co-Web Host of www.GameTutorials.com

veLib Copyright 2003-2005 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
1.2

Definition in file veIo3ds.h.

13.15 veLib.h File Reference

Main include file for the whole veLib.

```
#include "veConfig.h"
#include "veStd.h"
#include "veTypes.h"
#include "veGlUtils.h"
#include "veImage.h"
#include "veStrUtils.h"
#include "veXml.h"
#include "veMath.h"
#include "veUtils.h"
#include "veDataContainer.h"
#include "veCollision.h"
#include "vePlugins.h"
#include "veDevice.h"
#include "veDeviceContainer.h"
#include "veDeviceGraphicsGL.h"
#include "veDeviceNetwork.h"
#include "veMotion.h"
#include "veDeviceAudioAL.h"
#include "veDeviceSDL.h"
```

13.15.1 Detailed Description

Main include file for the whole veLib.

This file contains includes all core veLib headers.

veLib Copyright 2003, 2004 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh
Revision
2.4

Definition in file veLib.h.

13.16 veMath.h File Reference

mathematical classes and utility functions.

```
#include "veStd.h"
#include <cmath>
```

Namespaces

- namespace ve

Classes

- class ve::vec2f
  a class for 2d vector and vertex geometry operations.
- class ve::vec3f
  a class for vector and 3D vertex geometry operations.
- class ve::vec4f
  a class for 4D vector geometry operations.
- class ve::sphere
  a class representing a sphere.
- class ve::plane
  a class representing a plane.
- class ve::triangle
  a class for triangle geometry.
- class ve::line
  a class for line mathematics.
- class ve::vec6f
  a class representing a six degree of freedom coordinate.
- class ve::frustum
  class for frustum (clipping) operations.
- class ve::rnd
  an extension of c random number functions.
- class ve::mat4f
  a class for typical 3D geometry 4x4 matrix operations.
- class ve::matStack4f
  a class for typical 3D geometry 4x4 matrix stack operations such as in OpenGL.

Defines

- #define M_PI PI
- #define NAN 0.0f/0.0f
- #define HUGE_VALF 1.0f/0.0f
- #define fsign(fnum) ((fnum)<0.0?-1:1)
- #define max(a, b) (((a) > (b)) ? (a) : (b))
- #define min(a, b) (((a) < (b)) ? (a) : (b))

Enumerations

- enum ve::axis

Functions

- template<class T> T ve::sqr (T x)
- template<class T> int ve::sgn (T x)
- template<class T> double ve::dsin (T x)
- template<class T> double ve::dcos (T x)
- template<class T> double ve::dtan (T x)
- template<class T> double ve::datan (T x)
- template<class T> double ve::angle (T angle)
- template<class T> double ve::dAngle (T ang1, T ang2)
- template<class T> void ve::swap (T &t1, T &t2)
- template<class T> T ve::min3 (T value0, T value1, T value2)
- template<class T> T ve::max3 (T value0, T value1, T value2)
- float ve::dist (float x1, float y1, float z1, float x2, float y2, float z2)
- float ve::minAbs (float f1, float f2)
- float ve::distPointSeg (float x1, float y1, float x2, float y2, float x3, float y3)
- float ve::angleXY (const ve::vec3f &p0, const ve::vec3f &p1, const ve::vec3f &p2)
- float ve::scalarProduct (float x1, float y1, float x2, float y2)
- std::ostream & ve::operator<< (std::ostream &os, const ve::vec2f &v)
- const ve::vec4f operator * (const ve::vec4f &v, float f)
- const ve::vec6f operator+ (const ve::vec3f &v1, const ve::vec6f &v2)
- int ve::veRandi (int max)
- float ve::veRandf (double max)

Variables

- const float ve::PI = 3.14159265358979323846f
- const float ve::PI_180 = PI/180.0f
- const float ve::DEG2RAD = PI_180
- const float ve::RAD2DEG = 180.0f/PI
- const double ve::EPSILON = 0.00000001

13.16.1 Detailed Description

mathematical classes and utility functions.

This is the veMath auxilliary classes library. started 2000 as gf_math.h 2002-02-01 generously granted to the veLib by Gerald.Franz@tuebingen.mpg.de .

The templates defined here are small inline functions mainly for convenient handling of degree angle values.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf
Revision
2.19

Definition in file veMath.h.

13.16.2 Define Documentation

13.16.2.1 #define M_PI PI

share the normal math.h definition

Definition at line 43 of file veMath.h.

13.16.2.2 #define NAN 0.0f/0.0f

defines C99 NaN constant if necessary

Definition at line 50 of file veMath.h.

13.16.2.3 #define HUGE_VALF 1.0f/0.0f

defines C99 HUGE_VALF constant if necessary

Definition at line 60 of file veMath.h.

13.16.2.4 #define fsign(fnum) ((fnum)<0.0?-1:1)

simple signum function for floats

Macro for extracting the sign of float or double values. It is as simple as the normal sign function for integers.

Definition at line 96 of file veMath.h.

13.16.2.5 #define max(a, b) (((a) > (b)) ? (a) : (b))

returns maximum of 2 values

Definition at line 99 of file veMath.h.

Referenced by ve::max3().

13.16.2.6 #define min(a, b) (((a) < (b)) ? (a) : (b))

returns minimum of 2 values

Definition at line 103 of file veMath.h.

Referenced by ve::min3().

13.16.3 Function Documentation

13.16.3.1 const ve::vec4f operator * (const ve::vec4f & *v*, float *f*) [inline]

operator multiplying a vec4f v with a scalar f

Definition at line 453 of file veMath.h.

13.16.3.2 const ve::vec6f operator+ (const ve::vec3f & *v1*, const ve::vec6f & *v2*) [inline]

addition operator vec3f+vec6f

Definition at line 869 of file veMath.h.

13.17 veMotion.h File Reference

The general veMotion class declaration.

Namespaces

- namespace ve

Classes

- class ve::motion

  Base class for all motion model classes.

- class ve::motionSimple

  A very simple motion model. This class implements a basic generic motion model, all speed and acceleration factors are widely adjustable by an xml initialization file.

13.17.1 Detailed Description

The general veMotion class declaration.

This file contains the declaration of the veMotion class. This virtual class constitutes a base for all motion models used in the veLib. It is meant to offer a general interface for any kind of motion attributed to a virtual observer. Furthermore a most basic implementation (veMotionSimple) is provided.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh & gf

Revision
2.0

Definition in file veMotion.h.

13.18 veStd.h File Reference

Central definition of included standard headers.

```
#include "veConfig.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <functional>
#include <string>
#include <vector>
#include <cstdlib>
#include <cstdio>
#include <cassert>
```

Namespaces

- namespace std

13.18.1 Detailed Description

Central definition of included standard headers.

This file hides platform dependent naming differences of C++ standard headers from the user. It should be used instead of direct includes.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh

Revision
2.1

Definition in file veStd.h.

13.19 veStrUtils.h File Reference

string utility functions.

`#include "veStd.h"`

Namespaces

- namespace ve

Functions

- unsigned int ve::split (const std::string &input, std::vector< std::string > &output, const std::string &separators=" \t\n\015")
- std::string ve::trim (const std::string &s, const std::string &pattern=", \t\n\015")
- bool ve::isWhiteSpace (char ch)
- std::string ve::replaceAll (std::string s, const std::string &search, const std::string &repl)
- std::string ve::replaceChars (const std::string &s, const std::string &pattern=" \t\n\015", char ch=' ')
- void ve::operator-= (std::string &s, unsigned int n)
- std::string ve::i2s (long i)
- std::string ve::f2s (double f)
- std::string ve::b2s (bool b, bool asText=false)
- std::string ve::c2s (char ch)
- int ve::s2i (const std::string &s)
- unsigned int ve::s2ui (const std::string &s)
- float ve::s2f (const std::string &s)
- bool ve::s2b (const std::string &s)
- std::string ve::toUpper (const std::string &s)
- std::string ve::toLower (const std::string &s)
- unsigned int ve::hex2ui (const std::string &s)
- std::string ve::load (const std::string &filename)
- void ve::save (const std::string &s, const std::string &filename)
- template<class T> std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)

13.19.1 Detailed Description

string utility functions.

These functions allow a convenient string processing, mainly for internal use in the veXml and xmlStatement classes.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
2.2

Definition in file veStrUtils.h.

13.19.2 Function Documentation

13.19.2.1 template<class T> std::ostream& operator<< (std::ostream & *os*, const std::vector< T > & *v*)

simple standard output of vectors

Definition at line 86 of file veStrUtils.h.

13.20 veTypes.h File Reference

Central repository for defined simple types.

```
#include "veStd.h"
#include "veStrUtils.h"
```

Namespaces

- namespace ve

Classes

- class ve::flag128
  a class for storing a large number of flags.

definitions and constants

some handy definitions of bitmasks and bytemasks for packing multiple flags into variables.

- const unsigned int ve::BIT [ ]
- const unsigned int ve::BYTE [ ] = { 0xFF, 0xFF00, 0xFF0000, 0xFF000000 }

Enumerations

- enum ve::buttonId
- enum ve::dataContainerType
- enum ve::dcAxesType
- enum ve::dcFlagContainerType
- enum ve::dcFlagsType { ve::DC_FLAG_STATIC = 0, ve::DC_NUM_FLAGS }
- enum ve::dcIdType
- enum ve::dcCommandType {
  ve::CMD_OBJECT_SET = 0, ve::CMD_OBJECT_ADD, ve::CMD_OBJECT_DROP, ve::CMD_OBJECT_VAR_GET,
  ve::CMD_OBJECT_VAR_SET, ve::CMD_OBSERVERID_SET, ve::CMD_SCENE_CLEAR, ve::CMD_SCENE_LOAD,
  ve::CMD_SCENE_RESET, ve::CMD_RESOURCE_SET, ve::CMD_RESOURCE_ADD, ve::CMD_RESOURCE_DROP,
  ve::CMD_DEVICE_TERM, ve::CMD_DEVICE_STATUS, ve::CMD_DEVICE_VAR_GET, ve::CMD_DEVICE_VAR_SET ,
  ve::CMD_UNKNOWN }
- enum ve::dcMaskType { ve::DC_MASK_ALL = 0, ve::DC_MASK_INPUT = 1, ve::DC_-MASK_STATE = 2 }
- enum ve::dcTransfer

- enum ve::mimeType {

  ve::MIME_NONE = 0, ve::MIME_IMAGE_PNG, ve::MIME_IMAGE_JPEG, ve::MIME_-IMAGE_BMP,

  ve::MIME_IMAGE_SVG, ve::MIME_TEXT_PLAIN, ve::MIME_MODEL_VRML, ve::MIME_-MODEL_X3D,

  ve::MIME_APPLICATION_3DS, ve::MIME_AUDIO_WAV, ve::MIME_GEOM_RECT, ve::MIME_FONT_TXF }
- enum ve::deviceId {

  ve::DEV_NO_DEVICE = 0x1, ve::DEV_UNKNOWN = 0x2, ve::DEV_INPUT = 0x4, ve::DEV_OUTPUT = 0x8,

  ve::DEV_MOUSE = 0x10, ve::DEV_KEYBOARD = 0x20, ve::DEV_JOYSTICK = 0x40, ve::DEV_TRACKER = 0x80,

  ve::DEV_NETWORK = 0x100, ve::DEV_SIMULATION = 0x200, ve::DEV_SOUND = 0x400, ve::DEV_WINDOW = 0x800,

  ve::DEV_GRAPHICS = 0x1000, ve::DEV_CONTAINER = 0x2000 , ve::DEV_ALL = 0x-FFFFFFFF }
- enum ve::errorCodes {

  ve::ERR_OK = 0, ve::ERR_ERROR, ve::ERR_WARNING, ve::ERR_FATAL,

  ve::ERR_DEVICE_BUSY, ve::ERR_DEVICE_NOT_WORKING, ve::ERR_OUT_OF_-MEMORY, ve::ERR_IO }
- enum ve::varTypes {

  ve::TYPE_UNKNOWN = 0, ve::TYPE_INT32, ve::TYPE_UINT32, ve::TYPE_FLOAT32,

  ve::TYPE_STRING, ve::TYPE_BOOL }

Variables

- const unsigned int ve::DC_NUM_BUTTONS = ve::BUTTON_ID_MAX + 1
- const unsigned int ve::DC_NUM_SIXDOFS = 4
- const unsigned int ve::DC_NUM_AXES = 6 ∗ ve::DC_NUM_SIXDOFS

13.20.1 Detailed Description

Central repository for defined simple types.

veLib Copyright 2003-2005 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf

Revision
2.7

Definition in file veTypes.h.

13.21 veUtils.h File Reference

A collection of utility functions and classes.

```
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include "veStd.h"
```

Namespaces

- namespace ve

Classes

- class ve::chrono
  time / timer class
- struct ve::fileInfo
- class ve::fileIo
  class facilitating file input/output operations.
- class ve::cmdLine
  a simple static class for preparsing command line arguments and options

Defines

- #define veDelete(object) { if (NULL!= (object)) { talk1(5,"veDelete object %x",(object)); delete ((object)); object = NULL; } }

Functions

- short ve::net2hosts (const char *buffer)
- int ve::host2nets (char *buffer, short number)
- void ve::byteSwap (char *b, int n)

Variables

- const unsigned int ve::ZIP_HEADER_ID = 67324752
- const unsigned int ve::ZIP_DIR_ID = 33639248

13.21.1 Detailed Description

A collection of utility functions and classes.

It currently contains byte order related conversion functions, portable time function wrappers, file input/output functions, and a framework for command line argument parsing.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
mvdh / gf

Revision
2.3

Definition in file veUtils.h.

13.21.2 Define Documentation

13.21.2.1 #define veDelete(object) { if (NULL!= (object)) { talk1(5,"veDelete object %x",(object)); delete ((object)); object = NULL; } }

A simple but secure delete macro.

Definition at line 47 of file veUtils.h.

13.22 veXml.h File Reference

contains the classes ve::xml and ve::xmlIni.

```
#include "veStd.h"
#include "veStrUtils.h"
```

Namespaces

- namespace ve

Classes

- class ve::xml
  basic XML class.
- class ve::xmlIni
  A class for reading variable values from XML inifiles. This class complements the basic xml class with convenient parsing functions for basic data types. It is a good way for reading initialization information and for transfering information between programs in temporary files.

Functions

- std::ostream & operator<< (std::ostream &os, const ve::xml &xs)

13.22.1 Detailed Description

contains the classes ve::xml and ve::xmlIni.

veLib Copyright 2003 - 2005 by Reinhard Feiler for the Max Planck Institute of Biological Cybernetics, Tuebingen.

Please report all bugs and problems to "weyel\@tuebingen.mpg.de".

Author:
gf & jmw
Revision
2.2

Definition in file veXml.h.

13.22.2 Function Documentation

13.22.2.1 std::ostream& operator<< (std::ostream & *os*, const ve::xml & *xs*)

just a good friend of xml, performs output in a C++ standard stream.

Appendix A

Appendix

A.1 Contact

The veLib is currently mainly maintained by `Michael Weyel` (weyel) and `Gerald Franz` (gf). If you find bugs or have severe troubles, FIRST please carefully read the corresponding documentation and exclude other sources of errors, afterwards you are welcome to contact us or post to the veLib mailing list.

A.2 Acknowledgements

The veLib was initially designed and is still developed at the department of Professor Heinrich H. Bülthoff at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany. The contributors are (in rough temporal order of involvement):

- Markus von der Heyde
- Jan M. Wiener
- Gerald Franz
- Boris Searles
- Cornelius Raths
- Michael Weyel
- Franck Caniard

The veLib makes itself widely use of Open Source software and libraries. This is further acknowledged in the respective classes or file documentations. General packages are:

- gcc - the GNU compiler collection and build tools
- MinGW - a great port of the GNU compiler and tools to Windows
- libSDL - the Simple DirectMedia Layer
- OpenGL - platform-independent 3D graphics library
- OpenAL - platform-independent 3D audio library
- libJPEG - the reference library for loading JPEG images
- libpng - the standard library for loading PNG images
- zlib - input/output of compressed data (.zip/.gz)

This documentation was created using LaTeX and doxygen.

A.3 GNU Public License (GPL)

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of

having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

A.4 veLicense for Contributors

veLib Library License for Contributors May 9, 2003

The veLib and included programs are provided under the terms of the GNU Library General Public License (LGPL) with the following exceptions:

1. Modifications to the veLib configure script, config header file, and makefiles by themselves to support a specific platform do not constitute a modified or derivative work.

The authors do request that such modifications be contributed to the veLib project

2. Classes that are subclassed from veLib classes do not constitute a derivative work.

3. Static linking of applications and widgets to the veLib does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared veLib libraries, or link their applications or against a user-supplied version of veLib.

If you link the application to a modified version of veLib, then the changes to veLib must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the veLib license with programs that are linked to the veLib library, nor do you have to identify the veLib license in your program or documentation as required by section 6 of the LGPL.

However, programs must still identify their use of veLib. The following example statement can be included in user documentation to satisfy this requirement:

program is based in part on the work of the veLib project.

GNU LIBRARY GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that

work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so,

and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Library General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Library General Public License for more details.

You should have received a copy of the GNU Library General Public
License along with this library; if not, write to the Free
Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That’s all there is to it!

Index