License for the usage and distribution of the FhSim API

This license agreement covers the usage and distribution of the FhSim API.

What is the FhSim API?

The FhSim API is a collection of libraries and tools used for simulating and visualizing dynamic systems. Its core functionality is the integration engine. This is augmented by the Ogre3D API for visualizing the systems simulated.

Who is it for?

This version is for educational use only. It may not be used for commercial purposes or outside the educational institution which aquired it without full written permission from SINTEF Fisheries and Aquaculture.

Can i distribute it?

No. If others would like to try this out, please refer them to SINTEF Fisheries and Aquaculture for a trial license. If you use this software in a product, written permission and an acknowledgment in the product documentation is required.

Will it work?

Hopefully, but this software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

Conditions for use

By using this software you agree to the following conditions: - You will acknowledge the usage of this software in relevant presentations - You will include a copy of this license in the documentation - You will document the source code based on this API and make it available to others, including SINTEF Fisheries and Aquaculture.

Will it work?

Ogre

This is the license for the OGRE3D rendering engine, which the visualization in FhSim is based upon.

Download and Install

Requirements and recommendations

Here follows a list of the software which is required or recommended for FhSim development on Windows and Linux.

Function	Recommendation for Windows	Recommend
C++ compiler	**Microsoft Visual Studio 14 2015**	**gcc**
Git	**MsysGit (http://gitforwindows.org/)**	**apt-get ins**
Git GUI	**TortoiseGit (http://code.google.com/p/tortoisegit/wiki/Download)**	
Buildsystem	**CMake (http://www.cmake.org)**	**apt-get ins**

Getting the dependencies

The dependencies can be obtained as either source or binaries, according to wishes and needs. For internal use the sources are distributed through git and Stash (https://stash.code.sintef.no), and binaries are distributed through Artifactory (http://artifactory.ext.sintef.no:8081/artifactory). The correct privileges must first be obtained from an administrator.

Buildscripts

The buildscripts contains common resources for CMake and Doxygen, such as find scripts for often used packages and common settings. Be sure to set the environment variables according to the instructions in the Buildscripts user manual (see http://build.ext.sintef.no). In particular, the environment variables SFH_BUILDSCRIPTS_DIR and SFH_DEV_DIR should be set.

FhLib

The FhLb project contains libraries with functionality shared between projects.

Getting FhSim

Also FhSim can be obtained as either source or binary distribution from Gitorious or Artifactory.

Building and installing FhSim

Building the install target creates two folders in the directory `<SFH_DEV_DIR>/` with subdirectories, some of them as given in the table below.

Directory	Purpose
/FhSimPlayPen__	Run-time dependencies for running FhSim
/SDK__/FhSim/FhSimLightSDK	The minimum FhSim API
/SDK__/FhSim/FhSimSDK	The full FhSim API
/SDK__/FhSim/FhSimSimObjects	Libraries and binaries for SimObjects

Typical commands for building FhSim are as follows:

```
md build
cd build
cmake -D SFH_INSTALL_PREFIX=install ..
cmake --build . --target install --config Release
cmake --build . --target install --config Debug
cmake --build . --target install --config relwithdebinfo
```

Creating documentation

```
cd ../_doc
cmake -P installdoc.cmake
```

Creating archive

```
cd ..
cmake -D SFH_INSTALL_PREFIX=install -P package.cmake
```

Manually testing FhSim

General

To get an overview of the options for running FhSim, go to the directory /FhSimPlayPen_/bin and run

```
FhSim.exe
```

Run FhSim without visualization

In a command window from the directory /FhSimPlayPen_/bin, run

```
FhSim --input-file <input_file>
```

This runs the standard FhSim executable without visualization. Some sample input files are kept in XXX.

Run FhSim with visualization

In a command window from the directory /FhSimPlayPen_/bin, run

```
FhVis --input-file <input_file>
```

This runs the standard FhSim executable with visualization. (If this is the first time the visualization is run, you will get a dialog asking for visualization options. Choose OpenGl and Full screen = No .) A window should pop up, containing the visualization. How to control the camera is described in the Section **??**ec_camera_control. Press to exit.

Linux specific setup

Additions to .bashrc

The following may be added to the file .bashrc in your home directory (remember to restart the shell):

```
export LD_LIBRARY_PATH=$PLAYPEN_DIR/bin:$LD_LIBRARY_PATH
```

For OpenCL add something similar to

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib:$LD_LIBRARY_PATH
export PATH=/usr/local/cuda/bin:$PATH
```

To simplify Valgrind options, the standard options may be set, like

```
export VALGRIND_OPTS='--memcheck:xml=yes --memcheck:xml-file=log.xml'
```

Creating aliases for often used commands may speed up development:

```
alias sbd='cmake --build . --config Debug'
alias sbr='cmake --build . --config Release'
alias sbind='cmake --build . --target install --config Debug'
alias sbinr='cmake --build . --target install --config Release'
```

Various tools

Tool	Installation
Nautilus open terminal	apt-get install nautilus-open-terminal
gcc	apt-get install g++
magick++	apt-get install libmagick++-dev
Eclipse	Download from their webpage a version with the CDT.

Using Eclipse

Open a project in Eclipse as follows:

- Run the CMake configure step to create the makefiles.
- Start Eclipse
- Choose File -> New -> Project -> C/C++ -> C++ Project -> Next
- Specify Project name
- Uncheck Use default location
- Click Browse and find the location of the existing makefile (e.g. /__Fh-Sim/fhsim/build/Release/Debug).
- Choose Makefile Project and Empty Project.
- Choose the Linux GCC toolchain.
- Click Finish

Important Linux run-time specifics:

- Linux does not find shared libraries in “working directory”. Libraries must be found in standard directories, or settings must be set to make it search e.g. FhSimPlayPen_/bin.
- The dependencies of OGRE are placed under /usr/lib/OGRE. Use this as PluginFolder in FhSimPlayPen_/resources/Plugins.cfg.
- In resources.cfg: [Bootstrap] Zip=../../SDK/Ogre/media/packs/OgreCore.zip

Manually testing the build

```
LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH ./FhVis --input-file <input_file>
```

Developer tutorial

Getting started

FhSim is installed and you are ready to start on your first project. Let’s start out by looking at a tutorial project and how it works. The tutorial project models a spring, a cable and a point mass and creates a model based on an input file. This guide focuses on getting your source files up and running, not on how to write them.

The tutorial project can be found in XXX. It consists of two folders and some test files.

Item | Contents ————|——————— build | Contains all files for configuring and installing. The compiler’s project files will also end up in this folder. src | Contains all source files. This is also where you should add new ones. *.xml | Input files that demonstrate how to set up a simulation with the simobjects in the project.

Configure

From the root directory, run the following commands: md build cd build cmake ../src -A x64

This should configure the project, creating the make files or the Visual Studio solution files, according to your platform and settings.

If you want to edit and compile the project from within your IDE, see Section **??**ec_using_an_IDE. If you want to compile the project from the command line, see Section **??**ec_compiling_and_installing.

Using an IDE

Inside the build directory you will find the solution file (on Windows) or the make file (on Linux). For Windows, open the solution file in Visual Studio. For Linux, see Section ??sing_eclipse for how to use the Eclipse IDE.

Compiling and installing from the command line

To compile the project from the command line, run the following command (after the configure step):

```
cmake --build . --target install --config Debug
cmake --build . --target install --config Release
```

This will build and install both debug and release version of the library. The installation adds the library both to the playpen directory and the different SDK's.

Then go to the `FhSimPlayPen_<COMPILER_SPEC>/bin` directory and run FhSim with the input file test3 with visualization on. Look in Section ??ec_fhsim_run_windows and ??ec_fhsim_run_linux for how to do this on Windows and Linux, respectively. The example input file is found in ??ec_new_simobject_examples.

Troubleshooting

- Read through the error logs.
- Were all the files found? `../project/src/CMakeList.txt` holds the files that the script is looking for.
- Try running the script in debug mode. Write SET cmake_loglevel=4 (with no whitespace before or after the equals sign) in the command window and run the script again.
- Try delete the build directory or just delete the file `CMakeCache.txt` before running the configuration again.

The IDE

Only the configure part is needed to get the project set up for editing and debugging in your IDE of choice. Open the solution file in your IDE, i.e. `../TutorialProject/build/TutorialProject.sln` for Visual Studio, and browse the project. You can now compile the project in your IDE. To debug we need to edit a few options and unfortunately there is no way to automate this with a CMake script. See debugging for more information.

The source files

The tutorial project initially consists of several source files, defining classes which are all simobjects. Amongst these classes are CMass and CSpring. The main difference between these are that the CMass contains states and has a very simple output function, while CSpring has no states but a more complex output function. In short, the following are the most important functions in a SimObject:

Constructor

The constructor is responsible for reading the parameters belonging to the SimObject from the input file and defining the interface of the SimObject, in terms of:

- Input ports
- Output ports
- States

OdeFcn

This function is responsible for calculating the time derivative of the states of the SimObject. For the CMass object, this is done by setting the derivatives of the position equal to the velocity, and by setting the derivative of the velocity equal to the acceleration. The acceleration is found using Newton's laws.

Output functions

Each output port is implemented as a method responsible for returning a pointer to the data structure containing the value of the output port.

Adding a new SimObject

One of the strengths of FhSim is the ability to combine different models to create new ones or big systems of different models. Models you have created for one type of problem might be reused to solve another. Lets try to modify the tutorial project by adding another SimObject. We could write new source code or we could link to existing libraries. Lets look at both of these options.

Linking to existing SimObject

The FhSim shared library of projects consists of a number of different simobjects. Consult the documentation of FhSim libraries for a list of implemented simobjects. Sometimes there is no need to reinvent the wheel, sometimes you can just link to the libraries containing the source for the SimObject you need. All we have to do is add a few lines in `CMakeLists.txt` file to tell the compiler that we want to link to that specific library. We add the following at appropriate places in

CMakeLists.txt (in the tutorial project this has already been done). See the CMake part of the FhSim guide for more about these files.

Adding a new source file

For simplicity we copy the CMass SimObject and rename it to create a new class called CMassNew. Place this in the ../TutorialProject/src folder among the other source files. Copying and modifying an existing source file is usually a good way to start as the structure of simobjects are often similar. Add the new file to the project by editing CMakeLists.txt found in ../TutorialProject/src (should also show up in your IDE). Add CMassNew.h to the set(HEADER ...) line and CMassNew.cpp to the set(SRC ...) line. Configure the project again and you are set to write your new source files. Let's keep it simple for now and just rename any instance of CMass to CMassNew, including the #ifndef statement in the header file. When compiling the Tutorial Project, a library of the source files is made which is used when simulating. We need to add our new SimObject to the TutorialLib. Include the header file in SimObjectsLib.h and a reference to the SimObject in SimObjectsLib.cpp. Now we have a new SimObject, though the usefulness of many differently named, but otherwise equivalent objects can be debated. Install the project, and let's try testing it.

The input files

Each SimObject constructor takes as argument an instance of the class ISimObjectCreator. This class handles the input files and the connection between different models as specified in the given input file. Load up the input file named test. For more information see the FhSim user's guide. If you remember from the first simulation of TutorialProject we had a spring with a big mass attached to it. Extending from the big mass was a cable with a smaller mass at the end. We now want to add a second cable from the smaller mass and attach a CMassNew SimObject in the end of it.

-# In <OBJECTS>, copy the input for the MassPointCable named cable1 to create a new one called cable2 with the parameters of your choosing. -# In <OBJECTS>, copy the input for one of the Masses and create a new one called mass2 with parameters of your choosing. Also change the SimObject parameter so that this input will be to an instance of your newly created MassNew object. -# In <OBJECTS>, change the numInput parameter of sum from 2 to 3 -# In the <Connection> tag under <INTERCONNECTIONS>, add the following attributes to make the new connections:

```
mass2.Force = "cable2.ForceA"
cable2.PositionA="mass2.Position"
cable2.PositionB="mass1.Position"
cable2.VelocityA="mass2.Velocity"
cable2.VelocityB="mass1.Velocity"
```

```
    sum.component2_input="cable2.ForceB"
```

-# In `<INITIALIZATION>`, set some initial conditions for the positions and velocity of `mass2`.

Instead of our new Mass object we could add the mass object with 6 degrees of freedom. This mass object has a few more ports so you have to change the input file slightly in order to simulate with visualization. Try doing that to see if you've understood how simobjects are declared and how they are connected. Hint: Modify the existing file, exchange one mass object with 6D mass. One possible solution:

```
LibName="FhSimBaseLib" SimObject="Body/6DBody"
```

The names and types of the parameters for each SimObject instance are not arbitrary, but specified in the implementation of the SimObject. They can be declared with a default value which will be used if no value is given in the input file. If no default value was declared, this would raise an error.

Save this input file as e.g. `MyTest.xml` and try running the simulation with visualization. If you were successful, feel free to congratulate yourself. You have just created and visualized your first simobject!

FhSim development

FhSim related repositories and directories

Here follows an overview of repositories related to FhSim and their directory structure.

Repository	Availability	Directory	Contents
fhsim	Internal		The FhSim core project
fhsimbase	Open		Shared SimObjects and FhSim resources
sfhbase	Internal		Internal SimObjects and FhSim resources

Debugging a project

General IDE settings

-# Open the solution file in Visual Studio. This is usually found under `<project>/build`. -# Open the Solution Explorer by selecting View->Solution Explorer. -# In the Solution Explorer, right-click on the `INSTALL` project and choose Set as start-up project.

Debugging

-# The input file must be created according to standards. -# In the top of the IDE, choose Release or Debug in the drop down box. -# In the Solution Explorer, right-click on the project building the executable and choose Properties. -# Choose Configuration properties -> Debugging. The following values should be present: - Command (Use full path to executable, for example `C:/_work/FhSimPlayPen_<COMPILER_SPEC>/exe/FhVis_d` for visualization) - Command arguments (`<Absolute or relative path to input file>`) - Working directory (`FhSimPlayPen_<COMPILER_SPEC>/exe`) -# Run the program by pressing `<F5>` or by selecting Debug -> Start debugging. -# If you get a "No Debugging Information" warning here, don't worry, just press Yes to continue.

Miscellaneous

Setting the background colour of FhVis animation

The default background colour appears to be black. In order to change the background colour of an animation, the following lines can be added to the RenderInit method of an object:

```
Ogre::ColourValue backgroundColour(0.9, 0.9, 0.9);
m_pCamera = (CFhCamera*) m_pVisMgr->GetPtr("CFhCamera");
Ogre::RenderWindow * pRenderWindow =
m_pCamera->GetRenderWindow();
pRenderWindow->getViewport(0)->setBackgroundColour(backgroundColour);
```

Creating Mesh models from Solidworks

Ogre uses .mesh and .material files for 3D objects. This is a guideline for creating such models using a tool such as Solidworks.

-# Get someone to make a model for you in Solidworks -# Export this as a VRML file -# Download and install Blender, an open source 3D program. -# Download and install the Ogre export script as outlined on the Ogre forum -# You can now export the model to a `.mesh.xml` file -# Use the `OgreXmlConverter` tool to convert to `.mesh` and `.material` files. -# Place these files somewhere on the Ogre resources path and set the mesh name of the object to that of the .mesh file

Cable and rope properties

Table 2 (Synthetic Fiber Rope Properties) from http://www.tensiontech.com/papers/papers/deep_mor/synthetic
See also http://www.tensiontech.com/papers/papers/deep_mor/deep_mor.html.

Material	Rope construction	Strength kN/cm2	Stiffness kN/cm2	Remarks
Nylon	braided	25	82	Dry strengths only, wet strength 10-2

Material	Rope construction	Strength kN/cm2	Stiffness kN/cm2	Remarks
Nylon	plaited	20	87	
Polyester	braided	25	210	Wet and dry strengths, including jac
	plaited	20	160	
	7-strand	45	400	
	parallel strand	50	550	
	parallel fiber	35	1000	
Aramid	36-strand	70	3300	Including jacket
	parallel strand	65	2100	
	parallel fiber	95	4300	K29, including jacket
	parallel fiber	90	8000	K49, including jacket
HMPE	braided	55	1500	Unjacketed
	7-strand	55	2000	Including jacket
	parallel strand	65	3000	
Steel	7-strand	85	6800	6x36 IWRC
	bridge strand	110	14000	Not including jacket
	Solid bar	140	21000	4340 steel

NB! Strength and stiffness are given as approximate values. Actual properties may vary widely. Cross sections are based on cylindrical enclosed area including jacket where applicable. Multiply by 1430 to convert to lb/in2.

Troubleshooting

General troubleshooting

If you get compilation errors

- You may have to install the DirectX SDK (if compilation errors suggest this)

If you get run-time errors without or before

visualization - Check the logfile `LogOutput.txt` (name and directory specified in the FhSim executable. - Check that the input file does not contain `Vis.dll`, `.dll` or other extensions.

If you get run-time errors when visualization starts

- Check the logfile Resources/Ogre.log
- Delete the file `Resources/Ogre.cfg`, to be able to specify - the screen resolution - use of OpenGl or Direct3D - full screen or windowed mode
- Verify correct OpenGL drivers
- Verify correct DirectX drivers

Other

- Ensure that you have followed the installation and setup description closely.
- Ensure that your computer is up to date:
- Windows update
- Microsoft update (especially that Visual Studio 2005 with service pack 1 is installed)

Installation

- Be sure to have followed the steps in fhsim_prerequisites_installation and fhsim_installation

Compilation

Running FhSim

I receive the error: "The method RegRenderRes could not be found!"

- Be sure that all dll library loaded by the input file are of the same linking as the executable (debug/release, static/dynamic, with/without visualization).

I receive the error: "The libraryr Ois*.dll could not be found!" - Be sure that the working directory is correctly set, if starting via Visual Studio. - Try running the executable from the command line. - Check that the missing dll is present in the working directory. - Obsolete: Run CreateTests from a command window and check that there are no errors.

I receive the error: "Unhandled exception at 0x7c422e38 in sim.exe: 0xC0000005: Access violation reading location 0xcdcdcde5".

- Assure that the executable and the libraries mentioned in the input file are linked similarly (debug/release, static/dynamic, with/without visualization).

I receive the error: "ERROR! The license for the dll simulation object of type"

- You need to regenerate the license. To do this, run the `BuildAll.cmd` command in the `$WORK/fhSim/Build directory`.

I receive the error: "ERROR! Could not load library ....."

When a simulation is started, the following process happens: -# FhSim is told which input file to use, either in code or through the command line. -# The input file is parsed. -# From the parsed input file, it is found which libraries (dlls) to be used for each SimObject. -# If the library names do not end with `.dll`, the names are appended with: `- Vis` if visualization is enabled `- _d` if it is a debug build `- .dll` -# The simObjects are instantiated from the specified simObject libraries (dlls).

Any errors must be interpreted with this process in mind.

Hints and tricks

SimObject libraries

It is suggested to use CMake to create install targets or post-build events to copy the necessary binaries to the Playpen directory.

CMake and Visual Studio

- CMake creates the Visual Studio solutions and projects.
- Afterwards, Cmake is automatically rerun when compiling in Visual Studio.
- New source files must be added to `CMakeLists.txt` in the corresponding directory.
- Changes in project settings that should last must be provided in `CMakeLists.txt`.
- There is a shortcut to the corresponding `CMakeLists.txt` in Visual Studio for each project, found in the Solution Explorer.

Inserting special characters in Windows

If you know the keystroke value of the special character you want to insert, you can insert the special character directly into your document by using your keyboard. To do so, open the document and position the cursor where you want the special character to appear. Then, with `<NUM LOCK>` on, press and hold the `<ALT>` key, and then press the keys on the numeric keypad that represent the keystroke value of the character you want to input. After you finish typing, release the `<ALT>` key, and Windows generates the character you specified. NB! In Windows you need to include prepending zeros.

Example: With `<NUM LOCK>` on and `<ALT>` pressed, type at the numpad: "0169". This creates the copyright symbol Â©.

ASCII codes can be found at for example: - http://www.asciitable.com/. - http://www.petefreitag.com/cheatsheets/ascii-codes/ - http://www.ascii.cl/htmlcodes.htm - http://www.jimprice.com/jim-asc.shtml

Visual Studio macros

This section contains some macros which can be used in Visual Studio to make the workflow more efficient. The macros include mostly functionality to:

- Format documents
- Swap between header and source file
- Write Doxygen comment sections

Installation

To install these macros: -# Copy all of the section Source code. -# Open Visual Studio. -# Open the Macros IDE by pressing `<alt> + F11`. -# Create a new module by opening the Project Explorer, right-click on an item and select new -> module. Name this FormatDocs. You may use another name, but the name must be the same as that referenced as "Public Module". Open the module by double clicking on it. -# Select all text in the module and delete it. -# Paste the contents of the clipboard into the module. -# Replace names and initials in the macros with your own. -# Save the module by pressing `<ctrl> + s`. -# Set up the keyboard shortcuts as desired. The ones I use are mentioned in the leading comment in the source code of each macro. To do this: -# In Visual Studio: Tools -> Options -> Environment -> Keyboard. -# In Show commands containing, type in FormatDocs. -# Go down the list until you find the macro you are looking for. -# While this is selected: Press the shortcut keys in the field Press shortcut keys. -# Press Assign.

Important tools and software used within FhSim

Recommended tools

- doxygen (Automated documentation)
- stexbar (Windows explorer add-on)
- Notepad++(A text editor)
- Sysinternals (Lots of tools for monitoring file access, processes, TCP etc.)

Nice to have

- cygwin (Linux shell)
- automated_gui_testin
- beyondcompare3 (Comparing files and folders)
- which.bat (Which executable is run)

- 7-zip (A file compression tool)
- InfraRecorder CD/DVD burning software
- Simian (Finding duplicate code)
- Visual Assist X

Remote support

- RealVNC

CMake

Download CMake from the CMake homepage. Remember to let CMake update the path info when you install it.

Installing Win32::GuiTest

-# Install ActiveState Perl -# Start the Visual Studio command shell: Start -> All Programs -> Microsoft Visual Studio 2005 -> Visual Studio Tools -> Visual Studio 2005 Command Prompt -# Install GuiTest

```
‘perl -MCPAN -e shell‘   <br/>
‘cpan>force install Win32::GuiTest‘
```

Links

- http://www.perl.com/pub/a/2005/08/11/win32guitest.html?page=1
- http://sourceforge.net/projects/winguitest
- http://sourceforge.net/project/showfiles.php?group_id=104592&package_id=114918