ESyS-Particle Build and Installation Notes

Justin Rahardjo, Dr. Vince Boros

October 2014

Contents

Chapter 1

Building ESyS-Particle

1.1 Easy Install

An extract from `https://answers.launchpad.net/esys-particle/+faq/1792`

To install a recent released version of ESyS-Particle on Ubuntu from prebuilt packages, first remove any existing ESyS-Particle installation, and then type in a terminal:

```
$ sudo apt-get update
$ sudo apt-get install esys-particle
```

Or for the latest trunk revision, including the most recent bug fixes:

```
$ sudo add-apt-repository ppa:esys-p-dev/daily
$ sudo apt-get update
$ sudo apt-get install esys-particle-daily
```

You will get updates automatically during the normal system update procedure.

1.2 Guide to a source-build

ESyS-Particle needs to be built against:

- Python 2.6.x, 2.7.x & 3.x
- Boost
- Povray

- VTK (currently only supported on Python 2.x)
- epydoc (currently only supported on Python 2.x)

All these packages need to be installed under a directory that is separate from `/usr/` and `/usr/local/`. This will help in testing the build for each version of the software.

1.2.1 Building the dependencies

Firstly, create a new directory to contain the various folders. For the rest of this document, it is assumed that the `~/BUILD/` directory is the base directory for the installations.

```
$ cd ~
$ mkdir -p BUILD/sources
$ cd BUILD
$ mkdir bin lib share include
```

And don't forget to add these folders to the environment variables. For a more permanent install, add these to the end of the `~/.bashrc` file.

```
$ export PATH=$HOME/BUILD/bin:$PATH
$ export LD_LIBRARY_PATH=$HOME/BUILD/lib:$LD_LIBRARY_PATH
$ export LIBRARY_PATH=$HOME/BUILD/lib:$LIBRARY_PATH
```

Installing Python from Source

Get the Python sourcecode from `https://www.python.org/downloads/source/` and place into the base directory. These instructions will be using Python 2.6.9 release. Adjust the commands accordingly to the release required.

```
$ cd ~/BUILD/sources
$ tar xfz Python-2.6.9.tgz
$ rm Python-2.6.9.tgz
$ cd Python-2.6.9
$ ./configure --prefix=$HOME/BUILD --enable-shared
$ make
$ make altinstall
```

NOTE:

- `--prefix=x` installs all platform-independent files in `x/lib`

- `--enabled-shared` allows installation of the Python library as a shared object
- `make altinstall` allows multiple versions of Python to coexist.

Link the new installation of python and check the link

```
$ ln -sf $HOME/BUILD/bin/python2.6 $HOME/bin/python
$ ln -sf $HOME/BUILD/bin/python2.6-config $HOME/bin/python-config
$ ls -l ‘which python‘
```

NOTE: The command `which python` tells you which directory the command python looks for the libraries and binary files. Place the result of this command into the last bash command for above.

Installing Boost from Source

Get the Boost sourcecode from `http://www.boost.org/users/history/` and place into the base directory. These instructions will be using Boost 1.52.0 release. Adjust the commands accordingly to the release required.

```
$ cd ~/BUILD/sources
$ tar --bzip2 -xf boost_1_52_0.tar.bz2
$ rm boost_1_52_0.tar.bz2
$ cd boost_1_52_0
$ ./bootstrap.sh --prefix=$HOME/BUILD --with-libraries=filesystem,
   python,regex,system
$ ./b2
$ ./b2 install
```

NOTE: When configuring ESyS-Particle, add the `--with-boost=x` option where x is your base directory. If your Boost installation results in nonstandard library names (such as the version of the C++ compiler included as part of the Boost library name), the `--with-boost-filesystem=` and `--with-boost-python=` options will also be needed to configure ESyS-Particle. The string following the `=` symbol needs to be the Boost library name without the initial `lib` string and without the final `.so` extension (for example, `--with-boost-python=boost_python-gcc_4_3_3` if the Boost::Python library is called `libboost_python-gcc_4_3_3.so`). If the configure script cannot find the correct Python library, rerun the script with this additional option (including the quotation marks): `LDFLAGS="-Lx/lib"`.

NOTE: When using Povray 3.7 or higher, include the `thread` library when compiling. Please see Section 3.3 for more details.

NOTE: When building with Python 3.2 and above (see Section 3.2), remember to include directory to the python headers in the b2 command. Such as `./b2 include ="~/BUILD/include/python3.xm"`.

Installing VTK from Source

The most up-to-date instructions to install VTK from source can be found at `http://www.vtk.org/Wiki/VTK/Configure_and_Build`.

The following instructions will be using CMake 2.8.12 Release and VTK 6.1.0 Release. Adjust the commands accordingly to the release required.

VTK requires CMake to be installed. So, download the source code at `http://www.cmake.org/cmake/resources/software.html` and place in the base folder. These instructions go through installation of CMake in the root folder. So that it may be shared with other testing installations. If it is equired to be installed in a certain folder, add `--prefix=/install/path` to the configure line.

```
$ cd ~/BUILD/sources
$ tar xfz cmake-2.8.12.2.tar.gz
$ cd cmake-2.8.12.2
$ ./configure
$ make
$ make install
```

NOTE: If there is an error about not being able to find the Curses Libraries, install the curses libraries first.

```
$ sudo apt-get install libncurses5-dev
```

Once CMake is built, we move onto VTK.

```
$ cd ~/BUILD/sources
$ tar xfz VTK-6.1.0.tar.gz
$ mkdir VTK-build
$ cd VTK-build
$ ccmake ../VTK-6.1.0
```

In the CCMake GUI, hit [c] to configure for the first time. Then, toggle to advance view [t] and set these values:

- `CMAKE_BUILD_TYPE` - Release
- `CMAKE_INSTALL_PREFIX` - the base directory to install VTK in (can't use "$HOME", have to use " " or "/home/username/")

- `BUILD_SHARED_LIBS` - ON
- `VTK_WRAP_PYTHON` - ON
- `Module_vtkPython` - ON
- `Module_vtkPythonInterpreter` - ON

Once these options are set, hit [c] again to reconfigure, then check these values and make sure it is using the correct python installation of the build.

- `PYTHON_EXECUTABLE` - your python installation (`~/BUILD/bin/python`)
- `PYTHON_INCLUDE_DIR` - (`~/BUILD/include/python2.6`)
- `PYTHON_LIBRARY` - (`~/BUILD/lib/libpython2.6.so`)
- `VTK_INSTAL_PYTHON_MODULE_DIR` - `lib/python2.6/site-packages`

Hit [c] to reconfigure again and then hit [g] to generate the required files. Lastly, build VTK.

```
$ cd ~/BUILD/VTK-build
$ make
$ make install
```

NOTE: Whilst configuring, there might be an error such as `X11_Xt_LIB could not be found`. To fix this problem, simply install the XT library.

```
$ sudo apt-get install libxt-dev
```

Installing Povray from Source

Get the Povray source from the download page `http://www.povray.org/download/index-3.6.php`. This instructions will be using Povray 3.6.1 release. Adjust the commands accordingly to the release required.

```
$ cd ~/BUILD/sources
$ tar xfz povray-3.6.tar.gz
$ rm povray-3.6.tar.gz
$ cd povray-3.6.1
$ ./configure COMPILED_BY="your name <email@address>" --prefix=
   $HOME/BUILD
$ make
$ make install
```

NOTE: There might be a problem with `libpng12-dev`. If so, remove it and use `libpng10-dev` instead.

NOTE: If you are having problems installing Povray 3.7, please see Section 3.3.

Installing Epydoc from Source

Get the Epydoc source from the website `http://epydoc.sourceforge.net/`. This instructions will be using Epydoc 3.0.1 release. Adjust the commands accordingly to the release required.

```
$ cd ~/BUILD/sources
$ unzip epydoc-3.0.1.zip
$ rm epydoc-3.0.1.zip
$ cd epydoc-3.0.1
```

Then use your preferred text editor to change the default installation path in the `Makefile` to point to where you'd like epydoc to be installed.

```
LIB = $HOME/BUILD
```

Then continue on with the installation using the command `make install`.

1.2.2 Configuring and installing ESyS-Particle

Once all the required dependencies are installed, we can start configuring the esys-particle installation and install it. Firstly, create a directory for the ESyS-Particle source and installation files.

```
$ cd ~/BUILD
$ mkdir esys-particle
$ cd esys-particle
$ bzr branch lp:esys-particle
$ mv esys-particle/ source
$ mkdir install build
$ cd source
$ ./autogen.sh
$ cd ~/BUILD/esys-particle/build
$ ../source/configure CC=mpicc CXX=mpic++ CXXFLAGS="-Wall -Wextra"
  --srcdir=$HOME/BUILD/esys-particle/source --prefix=$HOME/BUILD/
  esys-particle/install --with-boost=$HOME/BUILD --with-povray --
  enable-vtk --with-epydoc --enable-docs
$ make
$ make install
```

NOTE: When compiling for Python 3.x, remember that both vtk and epydoc are not supported and thus remove `--enable-vtk` and `--with-epydoc` in the configure command.

NOTE: Remember to uninstall previous version of ESyS-Particle before installing the new one. As sometimes during make, a few errors come up when the previous installations is conflicting.

```
$ cd old/esys/source/dir
$ sudo make uninstall
$ make distclean
```

And lastly, don't forget to add the ESyS-Particle installation paths to your environment variables.

```
$ export PATH=$HOME/BUILD/esys-particle/install/bin:$PATH
$ export LD_LIBRARY_PATH=$HOME/BUILD/esys-particle/install/lib:
    $LD_LIBRARY_PATH
$ export LIBRARY_PATH=$HOME/BUILD/esys-particle/install/lib:
    $LIBRARY_PATH
$ export PYTHONPATH=$HOME/BUILD/esys-particle/install/lib/pythonx.x
    /site-packages/:$PYTHONPATH
```

Chapter 2

Testing the ESyS-Particle Build

Once ESyS-Particle has been built using all its dependencies, the installation needs to be tested. To do this, run all the tutorial scripts from the ESyS-Particle Tutorial. And insure that it all runs correctly.

These test scripts can all be run together by using a bash script to run one after the other. A few things can be added to the script such as a way to move all the output files into another folder to tidy things up when viewing the results.

An example test script can be seen in Appendix A.

Once the build has been tested, the installation directories can be renamed so that a new build can be tested. To rename the builds, we can use the following script:

```
for name in bin include lib share etc; do
  mv $name "python_2_6_9_"$name;
done
mv esys-particle/install esys-particle/python_2_6_9_install
mv particle_build esys-particle/python_2_6_9_build
```

Once the files are renamed, we can start the testing process again from Section 1.2.1.

2.1 Tested Builds

Shown in this section is all the builds with different versions of Boost and Python that has been tested in different systems by the developers.

Python	Boost	Povray	VTK	Epydoc	Built?	Pass Test?	Notes
2.6.9	1.37	3.6.1	Yes	Yes	Yes	Yes	
2.6.9	1.55	3.6.1	Yes	Yes	Yes	Yes	
2.7.8	1.38	3.6.1	Yes	Yes	Yes	Yes	
2.7.8	1.41	3.6.1	Yes	Yes	Yes	Yes	
2.7.8	1.55	3.6.1	Yes	Yes	Yes	Yes	
3.0.1	1.55	3.6.1	No	No	Yes	Yes	
3.1.5	1.41	3.6.1	No	No	Yes	Yes	
3.1.5	1.55	3.6.1	No	No	Yes	Yes	
3.2.5	1.47	3.6.1	No	No	Yes	Yes	
3.2.5	1.55	3.6.1	No	No	Yes	Yes	wide-unicode
3.2.5	1.55	3.6.1	No	No	Yes	Yes	pymalloc
3.2.5	1.55	3.6.1	No	No	Yes	Yes	pydebug
3.3.5	1.47	3.6.1	No	No	Yes	Yes	
3.3.5	1.52	3.6.1	No	No	Yes	Yes	
3.3.5	1.55	3.6.1	No	No	Yes	Yes	wide-unicode
3.3.5	1.55	3.6.1	No	No	Yes	Yes	pymalloc
3.3.5	1.55	3.6.1	No	No	Yes	Yes	pydebug
3.3.5	1.55	3.6.1	No	No	Yes	Yes	
3.4.1	1.47	3.6.1	No	No	Yes	Yes	
3.4.1	1.55	3.6.1	No	No	Yes	Yes	

Table 2.1: A list of the builds of ESyS-Particle v2.3.1 tested on a Ubuntu 14.0.4 system. (VTK version 6.1.0 and Epydoc version 3.0.1)

Python	Boost	Built?	Pass Test?	Notes
2.6.9	1.37	Yes	Yes	
2.7.8	1.37	Yes	Yes	
2.7.8	1.55	Yes	Yes	
3.0.1	1.41	Yes	Yes	
3.0.1	1.56	Yes	Yes	
3.1.5	1.41	Yes	Yes	
3.1.5	1.56	Yes	Yes	
3.2.5	1.47	Yes	Yes	
3.2.5	1.47	Yes	Yes	wide-unicode
3.2.5	1.47	Yes	Yes	pymalloc
3.2.5	1.47	Yes	Yes	pydebug
3.2.5	1.55	Yes	Yes	
3.3.5	1.47	Yes	Yes	
3.3.5	1.55	Yes	Yes	
3.3.5	1.55	Yes	Yes	wide-unicode
3.3.5	1.55	Yes	Yes	pymalloc
3.3.5	1.55	Yes	Yes	pydebug
3.4.1	1.47	Yes	Yes	
3.4.1	1.55	Yes	Yes	
3.4.1	1.56	Yes	Yes	

Table 2.2: A list of the builds of ESyS-Particle v2.3.1 tested on a Ubuntu 12.0.4 system. (VTK version 6.1.0 and Epydoc version 3.0.1 for installs with Python 2.x and Povray 3.6.1)

Python	Boost	Built?	Pass Test?
2.6.8	1.37	Yes	Yes
2.6.8	1.49	Yes	Yes
2.6.9	1.56	Yes	Yes
2.7.3	1.49	Yes	Yes
2.7.8	1.37	Yes	Yes
2.7.8	1.56	Yes	Yes
3.0.1	1.41	Yes	Yes
3.0.1	1.56	Yes	Yes
3.1.5	1.41	Yes	Yes
3.1.5	1.56	Yes	Yes
3.2.3	1.49	Yes	Yes
3.2.5	1.47	Yes	Yes
3.2.5	1.56	Yes	Yes
3.3.5	1.47	Yes	Yes
3.3.5	1.56	Yes	Yes
3.4.1	1.47	Yes	Yes
3.4.1	1.52	Yes	Yes
3.4.2	1.56	Yes	Yes

Table 2.3: A list of the builds of ESyS-Particle v2.3.1 tested on a Debian 7.2 with system Boost 1.49.0 and Python 2.6.8, 2.7.3 and 3.2.3. (VTK version 6.1.0 and Epydoc version 3.0.1 for installs with Python 2.x and Povray 3.6.1)

Python	Boost	Built?	Pass Test?	Notes
2.6.0	1.36	Yes	Yes	These two are the system versions and so are present when the other Boost and Python modules are loaded.
2.7.3	1.51	Yes	Yes	2-B Unicode
2.7.3	1.55	Yes	Yes	2-B Unicode
3.2.2	1.55	Yes	Yes	2-B Unicode
3.3.5	1.47	Yes	Yes	using icc 12.0.3: ESyS-Particle configuration fell over using newer icc during the Boost compatibility test
3.4.1	1.47	Yes	Yes	using icc 12.0.3: ESyS-Particle configuration fell over using newer icc during the Boost compatibility test

Table 2.4: A list of the builds of ESyS-Particle v2.3.1 tested on a Suse 11 system with system Boost 1.36.0 and Python 2.6.0. (VTK version 6.1.0 and Epydoc version 3.0.1 for installs with Python 2.x and Povray 3.6.1)

Chapter 3

Problems found during build

3.1 Problems with Python

3.1.1 Not installing pip

During installation of Python 3.4, sometimes it comes up with a warning where PIP is not installed.

```
Ignoring ensurepip failure: pip 1.5.4 requires SSL/TLS
```

This can be fixed by installing the SSL library:

```
sudo apt-get install libssl-dev openssl
```

3.2 Problems with Boost

3.2.1 Syntax error on bootstrap

For Python 3.1 and above, there is a syntax error that comes when running `bootstrap.sh`. This is caused by an older version of Python's print function. Details on the patch can be found here: `https://svn.boost.org/trac/boost/ticket/5677`.

3.2.2 Can't find pyconfig.h on b2

During the `./b2` command, if pyconfig.h can't be found. This is caused by Python's naming convention for the include directory from 3.2 and above. To fix it, insure that it is in the path. i.e:

```
$ export CPLUS_INCLUDE_PATH="CPLUS_INCLUDE_PATH:$HOME/BUILD/include
   /python3.xm/"
```

3.3 Problems with Povray

Povray 3.7.0 can only be built with Boost 1.53 or higher, undefined references are found when using Boost 1.52. These undefined references are found in the boost threading library.

3.3.1 Missing Makefile.in

If there is a problem from a missing Makefile.in, follow these instructions to install Povray-3.7. The problem here is caused by how different versions of automake respond to files in sub-directories.

```
$ cd povray-3.7-stable/unix
$ sed 's/automake --w/automake --add-missing --w/g' -i prebuild.sh
$ sed 's/dist-bzip2/dist-bzip2 subdir-objects/g' -i configure.ac
$ ./prebuild.sh
$ cd ..
$ ./bootstrap
$ ./configure COMPILED_BY="your name <email@address>" --prefix=
   $HOME/BUILD
$ make
$ make install
```

3.3.2 Missing thread library

Installation of Povray 3.7 can't be done because of this error:

```
checking for boostlib >= 1.37... yes
checking whether the Boost::Thread library is available... yes
checking whether the boost thread library is usable... no
configure: error: in '/home/jrahardjo/BUILD/povray-3.7-stable':
```

```
configure: error: cannot link with the boost thread library
```

This is found during the configure process, this problem is because of it not being able to find the correct Boost::Thread library. So, firstly insure you have installed the threading library, if not, during bootstrapping of boost, insure that it is included (`--with-libraries=.....,thread`).

If the same error persists after installing the boost_threading library, add the libraries manually to the configuration line of Povray (`LIBS="-lboost_system -lboost_thread "`).

3.4 Problems with ESyS-Particle

3.4.1 Warnings on Subdirectories

** This problem has been solved in rev.1139. **

During `./autogen.sh`, a warning with regards to subdirectories is there.

```
Model/Makefile.am:26: warning: source file '$(top_srcdir)/Fields/
   FieldMaster.cpp' is in a subdirectory,
Model/Makefile.am:26: but option 'subdir-objects' is disabled
automake: warning: possible forward-incompatibility.
automake: At least a source file is in a subdirectory, but the '
   subdir-objects'
automake: automake option hasn't been enabled.  For now, the
   corresponding output
automake: object file(s) will be placed in the top-level directory.
     However,
automake: this behaviour will change in future Automake versions:
   they will
automake: unconditionally cause object files to be placed in the
   same subdirectory
automake: of the corresponding sources.
automake: You are advised to start using 'subdir-objects' option
   throughout your
automake: project, to avoid future incompatibilities.
```

This is not a problem atm, but might be worth looking into eventually.

3.4.2 Python library not found or Permission Errors??

During `make install`, there was a problem where Python was not able to find the `libpython2.6.so.1.0` object file.

This problem seems to be caused by some permission errors. As the install folders seem to be under root instead of the current user. Using `chown -R`, on the install folder, seems to have solved the problem. This was caused by the original command `sudo make install`. Use `make install` instead.

3.4.3 Python 3.x Naming conventions

** This problem has been solved in rev.1146. **

There is a problem with Python 3.2 and above where the naming convention has changed from `pythonx.x` to `pythonx.xm`. And as such, there are problems found in the configure file as well as during make. A fix for the configuration process has been placed and it lets the user go through the config with no problems. But this problem arises again during the build process and as such, a soft-link from `libpython3.x.so` to `libpython3.xm.so` needs to be created to solve this problem temporarily.

```
$ ln -sf $HOME/BUILD/lib/libpython3.xm.so $HOME/BUILD/lib/
   libpython3.x.so
```

3.4.4 Cannot find boost/python.hpp

Another error comes up as:

```
checking for boost/python.hpp... no
configure: error: cannot find boost/python.hpp
```

Despite the message, when looking through the `config.log`, it was found that it actually found the correct file, but then found undefined references in the file itself. This is caused by a similar naming convention problem as mentioned in the previous section. This is solved by including the required path to the correct include folder.

```
$ export CPLUS_INCLUDE_PATH="CPLUS_INCLUDE_PATH:$HOME/BUILD/include
   /python3.xm/"
```

3.4.5 Cannot find the flags to link with Boost Python

** This problem has been solved in rev.1146. **

Another error appears where it is unable to link with Boost Python library.

```
checking for Boost python library... no
configure: error: cannot find the flags to link with Boost python
```

After looking through the config.log, this problem was caused by an incompatibility of Unicode between UCS4 and UCS2 of the python version. To solve this problem, specifically add the version of Python when configuring boost.

This problem was caused by similar reasons to 3.4.3.

```
$ ./bootstrap.sh --prefix=$HOME/BUILD --with-python=$HOME/BUILD/bin
   /python3.xm --with-libraries=filesystem,python,regex,system
$ ./b2 include="$HOME/BUILD/include/python3.xm"
$ ./b2 install
```

3.4.6 *.Plo files not found

During make, sometimes it comes up with an error where it is unable to find .Plo files. The solution to this is to add `--disable-dependecy-tracking` option to the `./configure` command

3.4.7 Error on running script

When running the script using `mpirun`, there has been an error with `_sysconfigdata_m` missing from python.

```
Traceback (most recent call last):
  File "/usr/lib/python3.3/site.py", line 629, in <module>
    main()
  File "/usr/lib/python3.3/site.py", line 614, in main
    known_paths = addusersitepackages(known_paths)
  File "/usr/lib/python3.3/site.py", line 284, in
   addusersitepackages
    user_site = getusersitepackages()
  File "/usr/lib/python3.3/site.py", line 260, in
   getusersitepackages
    user_base = getuserbase() # this will also set USER_BASE
  File "/usr/lib/python3.3/site.py", line 250, in getuserbase
    USER_BASE = get_config_var('userbase')
  File "/usr/lib/python3.3/sysconfig.py", line 610, in
   get_config_var
    return get_config_vars().get(name)
  File "/usr/lib/python3.3/sysconfig.py", line 560, in
   get_config_vars
    _init_posix(_CONFIG_VARS)
  File "/usr/lib/python3.3/sysconfig.py", line 432, in _init_posix
    from _sysconfigdata import build_time_vars
  File "/usr/lib/python3.3/_sysconfigdata.py", line 6, in <module>
    from _sysconfigdata_m import *
ImportError: No module named '_sysconfigdata_m'
```

This is caused because of multiple python installations and the system not being able to find the correct one. Fix the python installation to fix this problem. Another solution is to make sure that the environment variables are pointing to the correct directories (check `~/.bashrc`).

Chapter 4

ESyS-Particle Release Steps

Described in this section is the steps needed to create a new release for ESyS-Particle.

1. Write release notes at `https://wiki.geocomp.uq.edu.au/index.php/Release_Notes_for_ESyS-Particle`

2. Update version information in `configure.ac`, `Doxyfile`, `Doc/Tutorial/paper.tex`, `Foundation/version.h`

3. Update the tutorial if necessary and generate the new PDF file:

   `./render.sh` in `Doc/Tutorial/`

4. Upload the new file to `https://wiki.geocomp.uq.edu.au/index.php/File:ESyS-Particle_Tutorial.pdf`

5. Update the trunk:

   ```
   bzr commit
   ```

6. Generate an updated API by building the code using the `--with-epydoc` option

7. Copy the API to `shake200:/data/www/esys/esys-particle_python_doc/esys-particle-2.3.1`

8. Generate updated source code documentation:

   ```
   doxygen Doxyfile
   ```

9. Copy the source code documentation to `shake200:/data/www/esys/esys-particle_doxygen_doc/esys-particle-2.3.1`

10. Branch from the trunk:

    ```
    bzr branch lp:esys-particle source2.3.1
    ```

11. Push the new branch to Launchpad:

    ```
    bzr push lp:~esys-p-dev/esys-particle/2.3.1
    ```

12. Change the new branch status to "Mature":

    Code page → `lp:~esys-p-dev/esys-particle/2.3.1` → Status

For major version:

13. *Create series 2.3:*
    Project overview page → Register a series
14. *Link the mainline branch for the series: 2.3 Series page*

15. Create a 2.3 series milestone:

    2.3 Series page → Create milestone: Name: 2.3.1

16. Link fixed and committed bugs to the milestone, if any exist:

    Bugs page → Target to milestone

17. Create a 2.3 series release:

    2.3 Series page → Create release

18. Make a new tarball:

    ```
    tar -cavf ESyS-Particle-2.3.1.tar.gz --exclude-vcs *
    ```

19. Create a signature for the tarball:

    ```
    gpg --armor --sign --detach-sig ESyS-Particle-2.3.1.tar.gz
    ```

    - if no keys has been defined in the system, generate the keys by using:

      ```
      gpg --gen-key
      ```

    - and afterwards, the key would need to be uploaded to the server:

      ```
      gpg --keyserver 'hkp://keys.gnupg.net' --send keys <key-ids>
      ```

      the key id can be found using: `gpg --list-keys`

20. Upload the new tarball and signature:

    Milestone page → Add download file

21. Change the 2.3 series status from “Active Development” to “Current Stable Release”: 2.3 Series page

22. Change the status of bugs fixed for the release from “Fix Committed” to “Fix Released”: Bugs page

Chapter 5

Definitions and Concepts of Parameters and How-Tos

A few notes on the definitions and concepts of some parameters used in one or both of ESyS-Particle and GenGeo as well as How-To notes.

5.1 Masks

This is essentially a bitwise-mask that is applied to the particle tags. The default value of -1 means that no mask is applied.

The way bitmasking works is that it masks the positions in the binary value. For example, when the given mask is 1, it only selects those with a value of 1 in the first position of the binary. And as such, will return tags with numbers 1, 3, 5, 7 and so on. Whereas when the given mask is 2, it selects those with value of 1 in

Binary			Roman
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6
1	1	1	7

(a) mask = -1

Binary			Roman
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6
1	1	1	7

(b) mask = 1

Binary			Roman
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6
1	1	1	7

(c) mask = 2

Table 5.1: Binary to Roman table

the second position of the binary, thus returning the tags with numbers 2, 3, 6, 7 and so on. Table 5.1 shows the masked bits and the positions of the binary, giving a clearer description of the process.

5.2 Dump2VTK

Converting ESyS-Particle's checkpointer output files into VTK files for viewing in Paraview.

```
$ dump2vtk -i -o -t tini numsnap deltat [-list] [-bkrlist] [-t0]
    [-sz] [-rxb] [-single_tag] [-rot] [-unwrap]
```

Options and their arguments [1]:

- `i`: setup the `CheckPointer fileNamePrefix`, which should be equal to the fileNamePrefix defined in the CheckPointPrms class.
- `o`: setup the VTK output `fileNamePrefix`
- `t`: define the initial recording time step (`tini`), the total number of snapshots that you want to produce (`numsnap`) and the gap between two recording time steps, i.e., the interval (in time steps) between two CheckPointer files (`deltat`), such that dump2vtk knows which CheckPointer file to convert next.
- `list`: instead of constructing the list of input files starting from the CheckPointer fileNamePrefix and from the arguments of the option -t, it takes as input a list of files
- `brklist`: not yet analyzed
- `t0`: not yet analyzed
- `sz`: take only a slice Z = constant
- `rxb`: not yet analyzed
- `single_tag`: not yet analyzed
- `rot`: flag for indicating that the particles are rotational

[1] `http://scientificandhpcomputing.blogspot.com.au/2009/07/dump2vtk-tips-tricks.html`

For most usecases, with checkpointer files that are in the format of: `<CheckPointer fileNamePrefix>_t=????_ID.txt`

The following command will suffice:

```
$ dump2vtk -i <CheckPointer fileNamePrefix> -o snaps_ -t 0 26 10000
    -rot
```

Appendix A

Test Script

A.1 Main bash script

```
#!/bin/sh
# A script that runs all the scripts to test ESys-Particle
    installation

START=`date +%s`

python out2file.py w $1"-Test.txt" "# Testing build: "$1

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/bingle.py
then python out2file.py a $1"-Test.txt" "bingle.py pass"
else
  python out2file.py a $1"-Test.txt" "bingle.py FAIL"
fi

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/bingle_output.py
then python out2file.py a $1"-Test.txt" "bingle_output.py pass"
else
  python out2file.py a $1"-Test.txt" "bingle_output.py FAIL"
fi

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/bingle_chk.py
then python out2file.py a $1"-Test.txt" "bingle_chk.py pass"
else
  python out2file.py a $1"-Test.txt" "bingle_chk.py FAIL"
fi
```

```
sh move_output_files.sh bingle_chk bingle_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/bingle_vis.py
then python out2file.py a $1"-Test.txt" "bingle_vis.py pass"
else
  python out2file.py a $1"-Test.txt" "bingle_vis.py FAIL"
fi
sh move_output_files.sh bingle_vis bingle_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/gravity.py
then python out2file.py a $1"-Test.txt" "gravity.py pass"
else
  python out2file.py a $1"-Test.txt" "gravity.py FAIL"
fi
sh move_output_files.sh gravity gravity_data

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/gravity_cube.py
then python out2file.py a $1"-Test.txt" "gravity_cube.py pass"
else
  python out2file.py a $1"-Test.txt" "gravity_cube.py FAIL"
fi
sh move_output_files.sh gravity_cube gravity_data

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/slope_fail.py
then python out2file.py a $1"-Test.txt" "slope_fail.py pass"
else
  python out2file.py a $1"-Test.txt" "slope_fail.py FAIL"
fi
sh move_output_files.sh slope_fail slope_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/slope_friction.py
then python out2file.py a $1"-Test.txt" "slope_friction.py pass"
else
  python out2file.py a $1"-Test.txt" "slope_friction.py FAIL"
fi
sh move_output_files.sh slope_friction slope_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
   esysparticle' Scripts/slope_friction_floor.py
then python out2file.py a $1"-Test.txt" "slope_friction_floor.py
   pass"
else
```

```
  python out2file.py a $1"-Test.txt" "slope_friction_floor.py FAIL"
fi
sh move_output_files.sh slope_friction_floor slope_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/slope_friction_walls.py
then python out2file.py a $1"-Test.txt" "slope_friction_walls.py
    pass"
else
  python out2file.py a $1"-Test.txt" "slope_friction_walls.py FAIL"
fi
sh move_output_files.sh slope_friction_walls slope_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/hopper_flow.py
then python out2file.py a $1"-Test.txt" "hopper_flow.py pass"
else
  python out2file.py a $1"-Test.txt" "hopper_flow.py FAIL"
fi
sh move_output_files.sh hopper_flow flow_data_

if mpirun -np 2 '/home/jrahardjo/BUILD/esys-particle/install/bin/
    esysparticle' Scripts/rot_compress.py
then python out2file.py a $1"-Test.txt" "rot_compress.py pass"
else
  python out2file.py a $1"-Test.txt" "rot_compress.py FAIL"
fi
mkdir -p rot_compress
mv *.dat rot_compress

END=`date +%s`
ELAPSED=$(( $END - $START ))

python out2file.py a $1"-Test.txt" "Time consumed by testing: "
    $ELAPSED" seconds"
```

A.2 Move output files to folders

```
#!/bin/sh
# A script to move all the output files of a test script into a new
    folder

mkdir -p $1
mv $2* $1 2>/dev/null
mv snap_* $1 2>/dev/null
```

A.3 Output test result to file

```
#!/usr/bin/env python
# A Python script used to write or append a line to a file given
# the file name.
# Author: J. Rahardjo 2014

import sys
import os

def ensure_file_exists(filename = "output.txt"):
  """ Checks if the file to be appended to exists. If not,
    Create the file in the root location
  """
  if not os.path.exists(filename):
    f = open(filename, 'w')
    f.write("# Test results file\n")
    f.close()

def append_to_file(string, filename = "output.txt"):
  ensure_file_exists(filename)
  f = open(filename, 'a')
  f.write(string + "\n")
  f.close()

def write_to_file(string, filename = "output.txt"):
  ensure_file_exists(filename)
  f = open(filename, 'w')
  f.write(string + "\n")
  f.close()

if __name__ == '__main__':
  command = str(sys.argv[1])
  filename = str(sys.argv[2])
  string = str(sys.argv[3])
  if(command == 'a'):
    append_to_file(string, filename)
  elif(command == 'w'):
    write_to_file(string, filename)
  else:
    sys.exit("Command not recognized:\n" + \
      "--- Please use 'a' to append or 'w' to write")
```