Installing AMUSE

Release 8.0

The AMUSE Team

May 23, 2013

Contents

1 Obtaining AMUSE

1.1 Download

Go to the downloads page.

1.2 Getting started

The first step in getting AMUSE to work is obtaining the AMUSE source code. We advice you to do this even before installation of the prerequisite software (*Installation of the prerequisite software*). In the following installation instructions we assume that you will install AMUSE in a directory `/amuse`.

1.3 Releases

For the official releases we provide tarballs and subversion repository access (you need an account for the latter).

Need an account?

You can find us on google groups, http://groups.google.com/group/amusecode or on IRC at the #amuse channel on irc.freenode.net.

Tarball

Obtain the tarball (e.g. amuse-4.0.tar.gz) from the download-site and unpack it in the amuse directory using:

```
> tar -xf amuse-4.0.tar.gz
```

this will make an amuse sub-directory `amuse-4.0`, which we will be referring to as the AMUSE root directory, e.g.:

```
./amuse
+-- amuse-4.0
    |-- bin
    |-- build.py
    |-- configure
    |-- cuda_self_help
    +-- data
    |-- doc
    |-- lib
    |-- Makefile
    |-- MANIFEST.in
    |-- README.txt
    |-- sandbox
    |-- setup.py
    |-- slowtests
    |-- src
    |-- support
    |-- test
    |-- test_results
```

From here proceed by reading the *Installation of the prerequisite software* section.

1.4 Bleeding edge

The current development version is available via subversion repository access by issuing the following command:

```
> svn co http://www.amusecode.org/svn/trunk amuse-svn
```

This will make an AMUSE root directory with the name "amuse-svn".

2 Installation of the prerequisite software

2.1 Installing on Ubuntu

Installing on Ubuntu version > 10.10

In this section we asume a default Ubuntu desktop installation.

All

The prerequisites can be installed with a couple of commands on Ubuntu. The only choice to make is between openmpi and mpich2. Most of our testing is done with MPICH2 but openmpi should also work.

For openmpi do:

```
> sudo apt-get install build-essential gfortran python-dev \
  libopenmpi-dev openmpi-bin \
  libgsl0-dev cmake libfftw3-3 libfftw3-dev \
  libgmp3-dev libmpfr4 libmpfr-dev \
  libhdf5-serial-dev hdf5-tools \
  python-nose python-numpy python-setuptools python-docutils \
  python-h5py python-setuptools

> sudo easy_install mpi4py
```

For mpich2 do:

```
> sudo apt-get install build-essential gfortran python-dev \
  mpich2 libmpich2-dev \
  libgsl0-dev cmake libfftw3-3 libfftw3-dev \
  libgmp3-dev libmpfr4 libmpfr-dev \
  libhdf5-serial-dev hdf5-tools \
  python-nose python-numpy python-setuptools python-docutils \
  python-h5py python-setuptools

> sudo easy_install mpi4py
```

Note: Please make sure not to install mpich2 and openmpi together. When both openmpi and mpich2 are installed strange errors will occur and AMUSE will not work. If you see both installed please remove both and install one.

Installing on Ubuntu 9.04

In this section we asume a default Ubuntu desktop installation.

Python

Ubuntu comes with python2.6 pre-installed, you can check if python is installed by doing:

```
> python --version
Python 2.6.2
```

If this failes with an error or a version before 2.6, please install python first(the package is called `python2.6`). You also need the `python2.6-dev` development package. To install it, do:

```
> sudo apt-get install python2.6-dev
```

GCC

By default, Ubuntu does not install a fortran 90 or a C++ compiler. We suggest using gfortran and g++. These compilers are installed with the `build-essential` and the `gfortran` package. To install these, do:

```
> sudo apt-get install build-essential gfortran
```

MPI2

Ubuntu does not provide installation packages for MPICH2. You can build MPICH2 by hand (a good HOWTO can be found at https://wiki.ubuntu.com/MpichCluster). Or, you can download and install pre-build packages from the MPICH2 site (http://www.mcs.anl.gov/research/projects/mpich2/index.php).

If you prefer OpenMpi over MPICH2, you can install openmpi from the Ubuntu packages. To install the openmpi packages, do:

```
> sudo apt-get install libopenmpi-dev openmpi-bin
```

HDF5

Amuse can work with HDF5 versions 1.6.* and 1.8.3. Ubuntu 9.04 comes with HDF5 version 1.6.6. To install it, do:

```
> sudo apt-get install libhdf5-serial-dev hdf5-tools
```

FFTW

On Ubuntu, FFTW can be installed with:

```
> sudo apt-get install libfftw3 libfftw3-dev libfftw3-doc
```

GSL

On Ubuntu, GSL can be installed with:

```
> sudo apt-get install libgsl0 libgsl0-dev
```

CMake

CMake is used to build EVTwin. On Ubuntu, CMake can be installed with:

```
> sudo apt-get install cmake
```

GMP

GMP is required for Adaptb. On Ubuntu, GMP can be installed with:

```
> sudo apt-get install libgmp3 libgmp3-dev
```

MPFR

MPFR is required for Adaptb. On Ubuntu, MPFR can be installed with:

```
> sudo apt-get install libmpfr4 libmpfr-dev
```

Python packages in Ubuntu

Ubuntu comes with python packages for nose and numpy. You also need the setuptools package to be able to install the `mpi4py` and `h5py` software. To install these , do:

```
> sudo apt-get install python-nose python-numpy python-setuptools python-docutils
```

Python packages with easy_install

The `mpi4py` and `h5py` can be installed with the `easy_install` command:

```
> sudo easy_install mpi4py
> sudo easy_install h5py
```

Installing on Ubuntu 9.10

In this section we asume a default Ubuntu desktop installation. This installation is for the most part the same as for Ubuntu 9.04, see previous section.

The development packages of python are needed, to install these do:

```
> sudo apt-get install python-dev
```

FFTW

For 9.10 the FFTW package name is fftw3 and not libfftw3, FFTW can be installed with:

```
> sudo apt-get install fftw3 fftw3-dev fftw3-doc
```

2.2 Installing on OS X

Installing on MAC OS X with MacPorts

In this section we asume a clean MacPorts installation. The MacPorts build system will build every package from source so installation will be slow. The packages in MacPorts support different *variants*, each *variant* is build differently. The default *variant* of most packages does not support fortran and AMUSE needs fortran and fortran enabled packages. Below, with all installation commands we will specify the variant. AMUSE is tested with the gcc45 variant, gcc43 and gcc44 have been known to work also. Below, we will use gcc45.

Note: If you want to use a different fortran compiler (ifort), you are better of using the **install.py** script in the **doc/install** directory.

Note: Make sure you have a recent MacPorts installation. You need at least an up to date MacPorts 1.8.6 or later.

Note: If you are unsure of your installation you can uninstall and clear the packages with:

```
port uninstall py26-docutils py26-nose py26-mpi4py py26-h5py py26-numpy hdf5-18 fftw-3 gsl openmp
```

or, for python 2.7:

```
port uninstall py27-docutils py27-nose py27-mpi4py py27-h5py py27-numpy hdf5-18 fftw-3 gsl openmp
```

To make a clean install of MacPorts, please remove the MacPorts directory and read the guide at: http://guide.macports.org/

All in one

You can install all the packages described below in one go with:

```
> sudo port install gcc45

> sudo port install python27
> sudo port install openmpi +gcc45
> sudo port install fftw-3 +gcc45
> sudo port install hdf5-18 gsl cmake gmp mpfr
> sudo port install py27-numpy py27-h5py py27-nose py27-docutils +gcc45

> sudo port install py27-mpi4py +openmpi
```

```
> sudo port install py27-matplotlib +gcc45
```

After installing you will need to configure the code with the following line:

```
> ./configure --with-fftw=/opt/local \
              MPICXX=openmpicxx MPICC=openmpicc MPIFC=openmpif90 \
              FC=gfortran-mp-4.5 PYTHON=python2.7 \
              PREFIX=/opt/local
```

Note: The `--with-fftw` option will ensure that fftw is found in `/opt/local` and not in any other location on OS X. Often, incompatible versions of fftw will be installed in `/usr/include` or `/usr/local/inlude`. These versions may have the wrong processor type (32 vs 64bits) or not contain a fortran API. For both cases compiling `Fi` will fail. In case the configure script does not pick the wanted fftw directories, you can edit the `config.mk` file to point to the right version.

Note: The `PREFIX` variable will make sure some support libraries for community codes (gsl, gmp and mpfr) are found in `/opt/local`.

Note: Please, make sure you have no other compile variables specified (like CC or CXX or CFLAGS), unless you have customized MacPorts in any way. Some variable settings will likely give compile errors for the community codes.

For example, BHTree is compiled with openmpicxx and $CXX. The command in the CXX variable must be compatible with openmpicxx (you can do `openmpicxx --show` to get the command openmpicxx is using)

Note: The version of openmpi in macports (1.5.5 and before) does not handle MPI_IN_PLACE calls correctly for Fortran codes. Unfortunately, this means that **Capreole**, **PhiGrape** and **Mocassin** will not work with more than one worker on the mac. Also, **MpiAmrVac**, will give incorrect results in any case.

If you want to run these codes on OS X, please use the amuse install scripts and make sure macports is not on your path.

Warning: With macports, the nose package will not install `nosetests` under it's standard name. It will be named `nosetests-<python-version>`. So for python2.7 you'll need to use *nosetests-2.7*

```
> nosetests-2.7
........................................

OK
```

GCC

By default MacPorts uses the XCode compilers, these compilers have no support for fortran, a MacPorts gcc compiler set needs to be installed. We suggest installing gcc 4.5:

```
> sudo port install gcc45
```

Note: If you have installed a different version of gcc, you need to select a different variant of the packages below. To select a different variant replace **+gcc44** with **+gcc43**, **+gcc42** or any other version matching your gcc installation. Note, apple-gcc versions will not work, these do not support fortran.

Python

MacPorts supports several python versions in different variants, we will install the python27 versions

```
> sudo port install python27 +gcc45
```

MPI2

MacPorts provides packages for mpich2 and openmpi. Although you can probably install both, this is not recommended. We suggest you install openmpi.

To install openmpi, do:

```
> sudo port install openmpi +gcc45
```

HDF5

Amuse can work with HDF5 versions 1.6.* and 1.8.3. MacPorts comes with HDF5 version 1.8.*. To install it, do:

```
> sudo port install hdf5-18 +gcc45
```

FFTW-3

MacPorts comes with a FFTW and FFTW-3 package, for AMUSE we need FFTW-3. FFTW-3 can be installed with:

```
> sudo port install fftw-3 +gcc45
```

GSL

GSL is used to build Gadget2, GSL can be installed with:

```
> sudo port install gsl +gcc45
```

CMake

CMake is used to build EVTwin, CMake can be installed with:

```
> sudo port install cmake
```

GMP

GMP is required for Adaptb. With MacPorts, GMP can be installed with:

```
> sudo port install gmp
```

MPFR

MPFR is required for Adaptb. With MacPorts, MPFR can be installed with:

```
> sudo port install mpfr
```

Python packages

By this point all libraries and frameworks are installed. We can now install python packages (some depend on the installed libraries):

```
> sudo port install py27-numpy py27-h5py py27-nose py27-docutils +gcc45
```

If you installed openmpi in the MPI2 step you need to set the "openmpi" variant for "py27-mpi4py":

```
> sudo port install py27-mpi4py +openmpi
```

Matplotlib

Matplotlib is not required but is very useful for creating graphics, you can install it with:

```
> sudo port install py27-matplotlib +gcc45
```

Note: Macports will install the compilers under non standard names. To use the right compilers you need to specify these during the configure stage of AMUSE.

See the output for `configure --help` for a list of all environment variables you can set.

If you installed openmpi you need to specify the mpi compilers like so:

```
./configure MPICXX=openmpicxx MPICC=openmpicc MPIFC=openmpif90
```

2.3 Installing on Arch Linux

In this section we asume a default Arch Linux installation.

All

The prerequisites can be installed with a couple of commands on Arch Linux.

To install the prerequisites do (for base-devel select *all* members):

```
> sudo pacman -Syu base-devel curl gcc-fortran gettext zlib
```

Install python and dependencies:

```
> sudo pacman -Syu python2 python2-numpy \
  hdf5 docutils openmpi
  python2-mpi4py python2-nose\
  fftw gsl cmake gmp mpfr
```

To install h5py, first install distribute and then run easy_install:

```
> sudo pacman -Syu python2-distribute
```

```
> sudo easy_install-2.7 h5py
```

2.4 Installing on Fedora 18

In this section we asume a basic install of Fedora 18 installation.

All in One

The prerequisites can be installed with a couple of commands on Fedora.

For mpich2 do:

```
> sudo yum install make gcc gcc-c++ gcc-gfortran\
        cmake zlib-devel\
        mpich2 mpich2-devel\
        hdf5 hdf5-devel\
        fftw fftw-devel\
        gsl gsl-devel\
        gmp gmp-devel\
        mpfr mpfr-devel\
        python-nose numpy numpy-f2py\
        h5py\
        python-setuptools python-setuptools-devel\
        mpi4py-mpich2\
        python-matplotlib
```

Note: This line will also install *matplotlib*, this package is used for all plotting in AMUSE. If you do not need any plotting you can leave it out.

After installing mpich2, you need to activate it using the 'module' command:

```
> module load mpi/mpich2-$(uname -i)
```

Note: We recommend to put the module activation script in your .bashrc or .cshrc file.

For openmpi do:

```
> sudo yum install make gcc gcc-c++ gcc-gfortran\
        cmake zlib-devel\
        openmpi openmpi-devel\
        hdf5 hdf5-devel\
        fftw fftw-devel\
        gsl gsl-devel\
        gmp gmp-devel\
        mpfr mpfr-devel\
        python-nose numpy numpy-f2py\
        h5py\
        python-setuptools python-setuptools-devel\
        mpi4py-openmpi\
        python-matplotlib
```

After installing openmpi, you need to activate it using the 'module' command:

```
> module load mpi/openmpi-$(uname -i)
```

Note: On Fedora you can install both mpich2 and openmpi, the module command will keep manage these separate installation, so no conflict will exists. If you change between implementation, you will need to recompile the amuse community codes with:

```
> make clean; make
```

2.5 Installing on Fedora 11

In this section we asume a live-cd install of Fedora 11 installation.

Python

Fedora comes with python2.6 pre-installed, you can check if python is installed by doing:

```
> python --version
Python 2.6.2
```

If this failes with an error or a version before 2.6, please install python first(the package is called `python`). You also need the `python-devel` development package. To install it, do:

```
> sudo yum install python-devel
```

GCC

By default, Fedora does not install a fortran 90 or a C++ compiler. We suggest using gfortran and g++. These compilers are installed with the `gcc`, `gcc-c++` and the `gcc-gfortran` packages. To install these, do:

```
> sudo yum install gcc gcc-c++ gcc-gfortran
```

MPI2

Fedora comes with packages for MPICH2 and Openmpi.

To install MPICH2, do:

```
> sudo yum install mpich2 mpich2-devel
```

If you prefer OpenMpi over MPICH2, you can install openmpi from the Fedora yum database. To install the openmpi packages, do:

```
> sudo yum install openmpi openmpi-devel
```

HDF5

Amuse can work with HDF5 versions 1.6.* and 1.8.3. Fedora 11 has a package with HDF5 version 1.8.3. To install it, do:

```
> sudo yum install hdf5 hdf5-devel
```

FFTW

On Fedora, FFTW can be installed with:

```
> sudo yum install fftw fftw-devel
```

GSL

On Fedora, GSL can be installed with:

```
> sudo yum install gsl gsl-devel
```

CMake

CMake is used to build EVTwin. On Fedora, CMake can be installed with:

```
> sudo yum install cmake
```

GMP

GMP is required for Adaptb. On Fedora, GMP can be installed with:

```
> sudo yum install gmp
```

MPFR

MPFR is required for Adaptb. On Fedora, MPFR is currently included in the gmp package. So, if you have not already done so, MPFR can be installed with:

```
> sudo yum install gmp
```

Python packages in Fedora

Fedora comes with python packages for nose and numpy. You also need the setuptools package to be able to install the `mpi4py` and `h5py` software. To install these , do:

```
> sudo yum install python-nose numpy numpy-f2py \
    python-setuptools python-setuptools-devel
```

Python packages with easy_install

The `mpi4py`, `h5py` and `docutils` can be installed with the `easy_install` command:

```
> sudo easy_install mpi4py
> sudo easy_install h5py
> sudo easy_install docutils
```

2.6 Installing on RedHat (CentOS)

Installing on CentOS 6

In this section we asume a minimal CentOS 6 installation.

All

The prerequisites can be installed with a couple of commands on CentOS 6.

To install the prerequisites do (for base-devel select *all* members):

```
> sudo yum install make gcc gcc-c++ gcc-gfortran \
    cmake zlib-devel\
    openmpi openmpi-devel \
    fftw fftw-devel \
    gsl gsl-devel gmp
```

After installing openmpi, you need to activate it using the 'module' command:

```
> module load openmpi-$(uname -i)
```

Note: We recommend to put the openmpi module activation script in your .bashrc or .cshrc file.

Install python and dependencies:

```
> sudo yum install python-devel \
    docutils python-nose \
    numpy numpy-f2py\
    python-docutils
```

To install hdf5 and docutils first install an additional rpm forge. For documentation see http://wiki.centos.org/AdditionalResources/Repositories/RPMForge

After installing an rpm forge do:

```
> sudo yum install hdf5 hdf5-devel
```

To install h5py do:

```
> sudo easy_install h5py
```

Last, you need to install mpi4py with:

```
> su -
> module load openmpi-$(uname -i)
> easy_install mpi4py
```

Note: The default CentOS sudo policy resets the environments variables and thereby removes the openmpi settings. So for the last step you cannot use `'sudo easy_install mpi4py'` but must install under root directly.

2.7 Installing on Suse Linux

Installing on OpenSuse 11

In this section we asume a normal desktop install of OpenSuse 11. Not all packages are available in the default OpenSuse package repository. We recommend to add the **Packman Repository** to the list of configured sofware reporistories (To do so, open Yast and go to *Software Repositories*).

Python

OpenSuse comes with python2.6 pre-installed, you can check if python is installed by doing:

```
> python --version
Python 2.6
```

If this failes with an error or a version before 2.6, please install python first(the package is called `python`). You also need the `python-devel` development package. To install it, do:

```
> sudo zypper install python-devel
```

GCC

By default, OpenSuse does not install a fortran 90 or a C++ compiler. We suggest using gfortran and g++. These compilers are installed with the `gcc`, `gcc-c++` and the `gcc-fortran` packages. To install these, do:

```
> sudo zypper install gcc gcc-c++ gcc-fortran
```

MPI2

The Packman Repository provides an OpenMPI package. To install the openmpi packages, do:

```
> sudo zypper install openmpi openmpi-devel
```

Unfortunately the openmpi installation does not work out of the box, you need to set the **LD_LIBRARY_PATH** variable and edit a configuration file first.

Setting the LD_LIBRARY_PATH The LD_LIBRARY_PATH must be set so that mpi4py can find the openmpi libraries. To set the variable we must first find out where the openmpi libs can be found, to do so execute:

```
> mpicxx -showme:link
-pthread -L/usr/lib/mpi/gcc/openmpi/lib -lmpi_cxx -lmpi
-lopen-rte -lopen-pal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

We need to set LD_LIBRARY_PATH variable to the path after the **-L** in the output (so in this example case '/usr/lib/mpi/gcc/openmpi/lib', this may be a different path if you system is 64-bits or if the opensuse version is different).

In bash do:

```
> export LD_LIBRARY_PATH=/usr/lib/mpi/gcc/openmpi/lib
```

We recommend you add this line to your '.bashrc' file so that the variable is set correctly for all sessions. If you have a C shell you need to do a *setenv* and edit the .cshrc file.

Editing the configuration file It seems that the default openmpi installation has some problems with loading an LDAP library. To check if your installation has this problem do:

```
> python -c "from mpi4py import MPI; print MPI.get_vendor()"
...
WARNING: ....
...
DAT: library load failure: libdaplscm.so.2: cannot open shared object file: No such file or direct
...
```

If you get a long list of warings about DAT providers not found, you need to edit the configuration file and turn off ldap. To do so, open an editor (as root) on the file **/etc/openmpi-mca-params.conf** and add this line to the bottom of the file:

```
btl = ^udapl
```

After saving the file, you can rerun the python statement:

```
> python -c "from mpi4py import MPI; print MPI.get_vendor()"
('Open MPI', (1, 2, 8))
```

HDF5

Amuse can work with HDF5 versions 1.6.* and 1.8.*. The Packman Repository has a package with HDF5 version 1.8.1. To install it, do:

```
> sudo zypper install hdf5 hdf5-devel
```

FFTW

Some codes in AMUSE need FFTW 3, FFTW can be installed with:

```
> sudo zypper install fftw3 fftw3-devel
```

GSL

On OpenSuse (10.2 and newer), GSL can be installed with:

```
> sudo zypper install gsl gsl-devel
```

CMake

CMake is used to build EVTwin. On OpenSuse, CMake can be installed with:

```
> sudo zypper install cmake
```

GMP

GMP is required for Adaptb. On OpenSuse, GMP can be installed with:

```
> sudo zypper install gmp-devel
```

MPFR

MPFR is required for Adaptb. On OpenSuse, MPFR can be installed with:

```
> sudo zypper install libmpfr4 mpfr-devel
```

Python packages in Fedora

Fedora comes with python packages for numpy. You also need the setuptools package to be able to install the other python packages. To install these, do:

```
> sudo zypper install python-numpy \
    python-setuptools python-setuptools-devel
```

Python packages with easy_install

The `nose`, `mpi4py`, `h5py` and `docutils` can be installed with the `easy_install` command:

```
> sudo easy_install nose
> sudo easy_install mpi4py
> sudo easy_install h5py
> sudo easy_install docutils
```

Before installing AMUSE several software packages must be installed. These software packages can be installed manually or with two prepared installation scripts. The installation scripts will install python and the other pre-requisites in a user directory. No “root” access is required.

These are the packages AMUSE needs:

- Python (version >= 2.6)

- Numpy (version >= 1.3.0)
- HDF (version 1.6.5 - 1.8.x)
- h5py (version >= 1.2.0)
- MPI (OpenMPI or MPICH2)
- mpi4py (version >= 1.0)
- nose (version >= 0.11)
- docutils (version >= 0.6)
- FFTW (version >= 3.0)
- GSL
- CMake (version >= 2.4)
- GMP (version >= 4.2.1)
- MPFR (version >= 2.3.1)

In the first two sections (compilers and installation_scripts) we explain how to use the two installation scripts to install AMUSE. In the last section (manual) we have specified the required packages with the needed version for each.

2.8 Compilers

To build AMUSE from source you need to have a working build environment. The AMUSE build system needs a C++ and fortan 90 compiler. Please check first if you have a working build environment on your system.

In Ubuntu you can setup the environment with (as root):

```
apt-get install build-essential curl g++ gfortran gettext zlib1g-dev
```

In Fedora you can setup the environment with (as root):

```
yum groupinstall "Development Tools" "Development Libraries"
```

Installation scripts

We have created two installation scripts to automate the installation of the required packages on a LINUX and OS.X system. These scripts will install these packages in a user directory. One script downloads and installs python while the other script downloads and installs the libraries and python packages. As everything is installed in a user directory these packages can be installed even if a version of the software is already installed on your system.

The scripts will download and install the software in a user directory. This user directory must be specified with the `PREFIX` environment variable. Before running the installation scripts you must set the `PREFIX` environment variable and update the path and library path. For shell (bash) you need to do:

```
export PREFIX=~/amuse/prerequisites
export PATH=${PREFIX}/bin:${PATH}
export LD_LIBRARY_PATH=${PREFIX}/lib:${LD_LIBRARY_PATH}
```

One script will download, build and install python on your system. The other script is written in Python and will download and install the other packages. Both scripts can be found in the `doc/install` directory.

To start the installation do:

```
# 1. Open a shell and go to the <doc/install> directory
>

# 2. Set the PREFIX, PATH and LD_LIBRARY_PATH environment variables:
```

```
> export PREFIX=~/amuse/prerequisites
> export PATH=${PREFIX}/bin:${PATH}
> export LD_LIBRARY_PATH=${PREFIX}/lib:${LD_LIBRARY_PATH}

# 3. Start the installation script for python
> ./install-python.sh

# 4. Start the installation script for the prerequisite packages
> ./install.py download
> ./install.py install

# 5. Update your PATH variable in your profile.
# Make sure the `${PREFIX}/bin` directory is the first entry in the PATH!
```

You should now be able to install AMUSE.

Using the installation scripts on OS X

For OS.X you need to install XCode and a gfortran compiler first. The XCode development package is available on the Apple developers site or for Lion on the Apple Store application.

The standard XCode release does not come with a gfortran compiler. Go to the HPC Mac OS X site for a recent gfortran compiler, compatible with the XCode tools.

After installing XCode and gfortan, follow the steps described in the previous paragraph.

Manually installing the prerequisites

Python

Python is probably already installed on your system. To check the version of python do:

```
> python --version
Python 2.6.2
```

You can download python from http://www.python.org.

Numpy

To check if numpy is installed on your system do:

```
> python -c 'import numpy; print numpy.version.version'
1.3.0
```

If this fails with an error or a version before 1.3 you need to install numpy. You can download numpy from http://www.scipy.org/NumPy.

HDF5 library

HDF5 is a data format specification. The HDF group provides a C library to write and access HDF files.

To check if the HDF library is installed on your system do:

```
> h5ls -V
h5ls: Version 1.8.3
```

If this fails with an error or a version before 1.6.5 you need to install the HDF library. You can download HDF from http://www.hdfgroup.org/.

h5py

To access HDF5 files from python we use the `h5py` library.

To check if the h5py library is installed on your system do:

```
> python -c 'import h5py; print h5py.version.version'
1.2.0
```

If this fails with an error or a version before 1.2.0 you need to install h5py. You can download h5py from http://code.google.com/p/h5py/.

docutils

To check if the python docutils are installed on your system do:

```
> python -c 'import docutils; print docutils.__version__'
0.6
```

If this fails with an error or a version before 0.6 you need to install docutils. You can download docutils from http://docutils.sourceforge.net/

MPI

The installed MPI framework must be MPI 2 compatible. AMUSE will work with MPICH2 or OpenMPI

MPICH2 MPICH2 is a portable implementation of the MPI 2 standard.

To check if MPICH2 is installed on your system do:

```
> mpdhelp

The following mpd commands are available.  For usage of any specific one,
invoke it with the single argument --help .

mpd           start an mpd daemon
mpdtrace      show all mpd's in ring
mpdboot       start a ring of daemons all at once
mpdringtest   test how long it takes
...
```

If this fails with an error you need to install MPICH2 or check for OpenMPI support. You can download MPICH2 from http://www.mcs.anl.gov/research/projects/mpich2/.

OpenMPI OpenMPI is another portable implementation of the MPI 2 standard

To check if OpenMPI is installed on your system do:

```
> mpicxx -v
```

If this fails with an error you need to install MPICH2 or OpenMPI support. Most examples in the dopcumentation assume OpenMPI. You can download OpenMPI from http://www.open-mpi.org/.

MPI4PY

To access MPI from python we use the `mpi4py` software. To check if the mpi4py library is installed on your system do:

```
> python -c 'import mpi4py; print mpi4py.__version__'
1.0.0
```

If this fails with an error or a version before 1.0 you need to install mpi4py. You can download mpi4py from http://code.google.com/p/mpi4py/.

Nose

Nose is an extension of the python testing framework. It is used for all unit testing in AMUSE.

To check if Nose is installed on your system do:

```
> nosetests --version
nosetests version 0.11.1
...
```

If this fails with an error or a version before 0.11 you need to install nose. You can download nose from http://somethingaboutorange.com/mrl/projects/nose/.

FFTW

FFTW is a C subroutine library for computing discrete Fourier transforms. To check for the availability of fftw on your system, you can use `fftw-wisdom`:

```
> fftw-wisdom --version
fftw-wisdom tool for FFTW version 3.2.1.
```

You can download the FFTW library from http://www.fftw.org.

GSL

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License. To check for the availability of GSL on your system, you can use `gsl-config`:

```
> gsl-config --version
1.14
```

You can download GSL from http://www.gnu.org/software/gsl/.

CMake

CMake is a cross-platform, open-source build system. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice. CMake is used to build EVTwin. To check whether you have CMake installed on your system:

```
> cmake --version
cmake version 2.8.2
```

You can download CMake from http://www.cmake.org/cmake/resources/software.html.

GMP

GNU MP is a library for arbitrary precision arithmetic (ie, a bignum package). It can operate on signed integer, rational, and floating point numeric types. GMP is required for Adaptb (Accurate Dynamics with Arbitrary Precision by Tjarda Boekholt). The best way to check whether you have the right version of GMP installed on your

system depends on the package manager you use, but this should always work (note that the library numbers do not match the release version):

```
> locate libgmp
/usr/lib64/libgmp.so
/usr/lib64/libgmp.so.10
/usr/lib64/libgmp.so.10.0.3

> locate gmp.h
/usr/include/gmp.h

> grep GNU_MP_VERSION /usr/include/gmp.h
#define __GNU_MP_VERSION 5
#define __GNU_MP_VERSION_MINOR 0
#define __GNU_MP_VERSION_PATCHLEVEL 3
```

You can download GMP from http://gmplib.org.

MPFR

The MPFR library is a C library for multiple-precision floating-point computations with correct rounding. MPFR is required for Adaptb (Accurate Dynamics with Arbitrary Precision by Tjarda Boekholt). The best way to check whether you have the right version of MPFR installed on your system depends on the package manager you use, but this should always work (note that the library numbers do not match the release version):

```
> locate libmpfr
/usr/lib64/libmpfr.so
/usr/lib64/libmpfr.so.4
/usr/lib64/libmpfr.so.4.1.0

> locate mpfr.h
/usr/include/mpfr.h

> grep MPFR_VERSION /usr/include/mpfr.h
#define MPFR_VERSION_MAJOR 3
#define MPFR_VERSION_MINOR 1
#define MPFR_VERSION_PATCHLEVEL 0
```

You can download MPFR from http://www.mpfr.org.

author: Arjen van Elteren (vanelteren@strw.leidenuniv.nl) date: 2010/09/22

3 Installation of the AMUSE software

Before installing AMUSE the prerequisite software must be downloaded and installed, see *Installation of the prerequisite software*.

In the current stage of development AMUSE will not be installed in the python `site-packages` library. Instead, all code is build in the AMUSE source directories. With this setup we can easily edit the code and run it, without the need for an extra installation step.

3.1 Configuring the code

The code is configured using the `configure` command. Before building the code, run 'configure' in the AMUSE root directory.

```
> ./configure
```

The 'configure' script will check for all prerequisite software and report if any are missing.

3.2 Building the code

The code is build using a `Makefile`. To build the code run 'make' in the AMUSE root directory.

```
> make clean
> make
...
legacy codes build
==================
* sse
* hermite0
* bhtree
* phiGRAPE
running generate_main
```

If everything goes well all legacy codes will be build (sse, hermite0, bhtree and phiGRAPE).

In order to use either MESA, SEBA or ATHENA the codes must be downloaded additionally. This is done automatically after setting the environment variable DOWNLOAD_CODES to 1. Alternatively, instead of a plain 'make' like in the example above you could do:

```
> make DOWNLOAD_CODES=1
```

or

```
> make mesa.code DOWNLOAD_CODES=1
> make seba.code DOWNLOAD_CODES=1
> make athena.code DOWNLOAD_CODES=1
```

3.3 Testing the build

Warning: For MPICH2 installations, the *mpd* process daemon must be started befor testing the code. The *mpd* application manages the creation of MPI processes. If this is the first time the MPICH2 daemon is run it will complain about a missing `.mpd.conf` file. Please follow the instructions printed by the mpd daemon.

```
> mpd &
```

If the mpd deamon only complains with 'no mpd.conf', these are the steps to take, to create a mpd.conf file:

```
> echo 'MPD_SECRETWORD=secret' > ~/.mpd.conf
> chmod 600 ~/.mpd.conf
```

Please make sure to replace ''secret''.
After starting *mpd* we can start the tests.

The tests are run using the nosetests program.

```
> nosetests
.........................................
Ran 91 tests in 12.013s

OK
```

Warning: If you have an MPICH2 installation but no mpd program your MPICH2 installation has been configured for the Hydra process manager. To run amuse scripts with the hydra process manager you must start every command with `mpiexec`:

```
> mpiexec nosetests -v
```

If you do not run under mpiexec you get an error with a usage statement. The error starts like this:

```
unable to parse user arguments

Usage: ./mpiexec [global opts] [exec1 local opts] : [exec2 local opts] : ...
```

Warning: On some laptops the hostname will not point to the correct internet address. For these laptops you can start the mpd daemon on the localhost ip. To do so, you need to set the `--ifhn` option:

```
> mpd --ifhn=localhost &
```

Warning: On OS X, when you install the prerequisites with macports, `nosetests` will not have a standard name. It will be named `nosetests-<python-version>`. So for python2.7 you'll need to use *nosetests-2.7*

```
> nosetests-2.7
.........................................

OK
```

Real-time testing

The code includes support for real-time testing. The real-time testing application monitors the files in the source directories ('src' and 'test'). Every time a file is changed it will run most of the tests. After each test a report is created, this report can be viewed with a web browser.

```
# go to the AMUSE root directory
# display help information of the realtime_test script
>  python -m support.realtime_test --help
Usage: realtime_test.py [options]

Options:
  -h, --help            show this help message and exit
  -p PORT, --port=PORT  start serving on PORT

# start the python realtime_test script on port 9080
> python -m support.realtime_test -p 9080
starting server on port:  9080
start test run
...
# open a browser to view the results
> firefox http://localhost:9080/
```

3.4 Running the code

A python script will not find the AMUSE code as the code is not installed into the python 'site-packages' directory or any other directory that can be found by python automatically.

During a build a shell script is created to run the AMUSE code. To use this script you first have to copy it to a directory in your PATH. The script is called ''amuse.sh''. After copying this script you can run amuse code from

anywhere on your disk by starting 'amuse.sh'. This script has exactly the same command line parameters as the normal python application.

```
> amuse.sh
Python 2.6.2 (r262:71600, Sep  1 2009, 16:14:27)
[GCC 4.3.2 20081105 (Red Hat 4.3.2-7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from amuse.units import units
>>> units.m
unit<m>
```

4 Getting started with AMUSE

4.1 Introduction

At this point you should have built and tested AMUSE, as described in the previous sections, and are probably wondering "What can AMUSE do for me?". This section will get you started with AMUSE.

AMUSE is based on python, so if you're new to Python, you'll find the official Python documentation a valuable resource. Like with Python, there are basically two ways to use AMUSE. Firstly, directly via the interactive (Python) command line:

```
> amuse.sh
Python 2.6.4 (r264:75706, Feb 17 2010, 12:05:36)
[GCC 4.4.3 20100127 (Red Hat 4.4.3-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> quit()
```

Secondly, by writing (Python) scripts. Suppose you wrote the following script *myscript.py*, and saved it in the current working directory:

```
from amuse.units.units import *
from amuse.units import constants

def convert_to_freq(wavelengths = [355.1, 468.6, 616.5, 748.1, 893.1] | nano(m)):
    """
    This function converts wavelength to frequency, using the speed of light in vacuum.
    """
    print "The speed of light in vacuum:", constants.c
    print "wavelength -->  frequency"
    for wavelength in wavelengths:
        print wavelength, "  --> ", (constants.c/wavelength).as_quantity_in(giga(Hz))
```

Then this script can be executed from the AMUSE interactive command line:

```
>>> import myscript
>>> help(myscript) # Tells you what myscript can do, ...
>>>     # ... for example that it has a function to convert wavelength to frequency.
>>> myscript.convert_to_freq()
The speed of light in vacuum: 299792458.0 m * s**-1
wavelength -->  frequency
355.1 nm   -->  844247.98085 GHz
468.6 nm   -->  639761.967563 GHz
616.5 nm   -->  486281.359286 GHz
748.1 nm   -->  400738.481486 GHz
893.1 nm   -->  335676.24902 GHz
>>> from amuse.units.units import *
>>> myscript.convert_to_freq([21.0, 18.0, 6.0] | cm)
The speed of light in vacuum: 299792458.0 m * s**-1
wavelength -->  frequency
```

```
21.0 cm    -->  1.42758313333 GHz
18.0 cm    -->  1.66551365556 GHz
6.0 cm    -->  4.99654096667 GHz
>>> quit()
```

You can also run scripts directly from the terminal prompt. Calling *amuse.sh* with a file name argument will make AMUSE execute the file. For this you need to add the following line to your script, telling the script which of its functions to call when executed:

```
if __name__ == '__main__':
    convert_to_freq()
```

Your script can now be executed directly from the terminal prompt:

```
 > amuse.sh myscript.py
The speed of light in vacuum: 299792458.0 m * s**-1
wavelength -->  frequency
355.1 nm    -->  844247.98085 GHz
468.6 nm    -->  639761.967563 GHz
616.5 nm    -->  486281.359286 GHz
748.1 nm    -->  400738.481486 GHz
893.1 nm    -->  335676.24902 GHz
```

4.2 Example interactive session

This is an example of an interactive session with AMUSE, showing how the interface to a typical (gravitational dynamics) legacy code works. Using the Barnes & Hut Tree code, the dynamics of the Sun-Earth system is solved. This two-body problem is chosen for simplicity, and is, of course, not exactly what a Tree code normally is used for. First we import the necessary AMUSE modules.

```
>>> from amuse.community.bhtree.interface import BHTree
>>> from amuse.support.data import core
>>> from amuse.units import nbody_system
>>> from amuse.units import units
```

Gravitational dynamics legacy codes usually work with N-body units internally. We have to tell the code how to convert these to the natural units of the specific system, when creating an instance of the legacy code class.

```
>>> convert_nbody = nbody_system.nbody_to_si(1.0 | units.MSun, 149.5e6 | units.km)
>>> instance = BHTree(convert_nbody)
```

Now we can tell the instance to change one of its parameters, before it initializes itself:

```
>>> instance.parameters.epsilon_squared = 0.001 | units.AU**2
```

Then we create two particles, with properties set to those of the Sun and the Earth, and hand them over to the BHTree instance.

```
>>> stars = data.Particles(2)
>>> sun = stars[0]
>>> sun.mass = 1.0 | units.MSun
>>> sun.position = [0.0,0.0,0.0] | units.m
>>> sun.velocity = [0.0,0.0,0.0] | units.m / units.s
>>> sun.radius = 1.0 | units.RSun
>>> earth = stars[1]
>>> earth.mass = 5.9736e24 | units.kg
>>> earth.radius = 6371.0 | units.km
>>> earth.position = [1.0, 0.0, 0.0] | units.AU
>>> earth.velocity = [0.0, 29783, 0.0] | units.ms
>>> instance.particles.add_particles(stars)
```

We need to setup a channel to copy values from the code to our model in python:

```
>>> channel = instance.particles.new_channel_to(stars)
```

Now the model can be evolved up to a specified end time. The current values of the particles are retieved from the legacy code by using *copy* from the channel.

```
>>> print earth.position[0]
149597870691.0 m
>>> print earth.position.as_quantity_in(units.AU)[0]
1.0 AU
>>> instance.evolve_model(1.0 | units.yr)
>>> print earth.position.as_quantity_in(units.AU)[0] # This is the outdated value! (should update
1.0 AU
>>> channel.copy()
>>> print earth.position.as_quantity_in(units.AU)[0]
0.999843742682 AU
>>> instance.evolve_model(1.5 | units.yr)
>>> channel.copy()
>>> print earth.position.as_quantity_in(units.AU)[0]
-1.0024037469 AU
```

It's always a good idea to clean up after you're finished:

```
>>> instance.stop()
```

4.3 Example scripts

In the test/examples subdirectory several example scripts are included. They show how the different legacy codes can be used. One such example is test_HRdiagram_cluster.py. It has several optional arguments. The example script can be executed from the AMUSE command line as well as from the terminal prompt (in the latter case use -h to get a list of the available command line options):

```
>>> import test_HRdiagram_cluster
>>> test_HRdiagram_cluster.simulate_stellar_evolution()
The evolution of  1000  stars will be  simulated until t= 1000.0 Myr ...
Using SSE legacy code for stellar evolution.
Deriving a set of  1000  random masses following a Salpeter IMF between 0.1 and 125 MSun (alpha =
Initializing the particles
Start evolving...
Evolved model successfully.
Plotting the data...
All done!
>>> from amuse.units.units import *
>>> test_HRdiagram_cluster.simulate_stellar_evolution(end_time=5000 | Myr)
The evolution of  1000  stars will be  simulated until t= 5000 Myr ...
...

> amuse.sh test_HRdiagram_cluster.py -h
Usage: test_HRdiagram_cluster.py [options]

This script will generate HR diagram for an
evolved cluster of stars with a Salpeter mass
distribution.

Options:
  -h, --help            show this help message and exit
...
> amuse.sh test_HRdiagram_cluster.py
The evolution of  1000  stars will be  simulated until t= 1000.0 Myr ...
...
```

If instead of "Plotting the data..." the script printed "Unable to produce plot: couldn't find matplotlib.", this probably means you do not have Matplotlib installed. See the subsection on Matplotlib below.

Matplotlib

Matplotlib is a python plotting library which produces publication quality figures. Many of the AMUSE example scripts use this library to produce graphical output. If you would like to take advantage of this library, get it from http://matplotlib.sourceforge.net/ and install it in the Python site-packages directory. For your own work, it is of course also possible to print the required output to the terminal and use your favourite plotting tool to make the figures, or use gnuplot, as described in the next section.

Gnuplot

Another plotting utility that can be used from Python and AMUSE scripts is gnuplot. Gnuplot can be downloaded from http://www.gnuplot.info/. If you have gnuplot, you can install the **gnuplot-py** package to control gnuplot directly from your script.

To install **gnuplot-py**, open a shell and do:

```
easy_install gnuplot-py
```

4.4 Further documentation

I hope this got you started with AMUSE. To further explore the possibilities with AMUSE, take a look at the other example scripts, and the available:

- *tutorials-label*
- *reference-label*
- *design-label*

5 Configuring AMUSE

5.1 Introduction

In AMUSE a configuration script is used to for two purposes; to run on different operating systems and to set compile time options. The AMUSE framework has been built on Linux, AIX and OS X systems, it also runs on Windows. AMUSE can be configured to run with or without MPI, GPU (CUDA) and openmp. In this document we will provide a short overview of the configuration options and their effects.

5.2 Basic

The basic configuration of AMUSE uses MPI as the communication channel, does not build any GPU enabled codes (or GPU enabled versions) and uses openmp if available. The configuration script can be run as:

```
> ./configure
```

To get a list of options and important environment variablse run `'configure'` with the help flag:

```
> ./configure --help
```

A very important variable for the configuration script is the location of the python executable. The python executable is searched for in the PATH and you can override it by setting the `'PYTHON'` environment variable:

```
> ./configure PYTHON=/path/to/libraries/python
```

The configuration script will look for dependent libraries in default locations of the system and, if defined, also in directories under the `'PREFIX'` environment variable. If you installed the prerequisites with the AMUSE installion scripts (see *Installation of the prerequisite software*), the configuration script should find all the packages

installed. For most libraries the `'PREFIX/lib'` or `'PREFIX/lib64'` is searched before the system path. You can override the `'PREFIX'` environment variable:

```
> ./configure PREFIX=/path/to/libraries/root
```

5.3 GPU

Currently all codes in AMUSE capable of using the GPU are based on CUDA. To run these codes you will need CUDA libraries and drivers. Once these have been installed you can run configure like so:

```
> ./configure --enable-cuda
```

The configuration script will look for the `'nvcc'` compiler and the cuda libraries, in well known paths. Unfortunately it often will not find the cuda tools and you have to specify the some environment variables or configuration options.

If the configuration script cannot find the nvcc compiler (or if it finds the wrong one) you can specify the nvcc compiler with the `'NVCC'` environment variable:

```
> ./configure --enable-cuda NVCC=/path/to/nvcc
```

The configure script also searches for the nvcc compiler in the `'$CUDA_TK/bin'` directory:

```
> ./configure --enable-cuda CUDA_TK=/opt/nvidia
```

The configure script looks for cuda and cudart libraries in `'$NVCC/../lib'` or `'$NVCC/../lib64'`, if your libraries cannot be found there you can override the library path with:

```
> ./configure --enable-cuda --with-cuda-libdirs=/path/to/cuda/lib
```

Using `'--with-cuda-libdirs'` will always override the local search paths and should also work if you have an old version of cuda in `'/usr/lib'`.

Finally, if all else fails, you can edit the `'config.mk'` file after configure has finished. The important variables in the file are:

- CUDA_ENABLED, valid values are “yes” or “no”.
- NVCC, absolute path to the nvcc executable.
- CUDA_TK, directory of the cuda toolkit installation
- CUDA_LIBS, library flags the add in the linking stage (-L/path -lcuda -lcudart)

Please remember that the `'config.mk'` file is overwritten every time configure is run.

Sapporo library version

The Sapporo library will be build when CUDA is enabled. The Sapporo library implements the GRAPE6 API on GPU hardware. AMUSE is shipped with two versions of the Sapporo library:

- An older version `'sapporo_light'` that runs on most CUDA devices but is not maintained any longer
- The latests version `'sapporo'` that runs on modern GPU harware. This version should also run on OpenCL devices but this is still a work in progress.

By default AMUSE will use the older `'sapporo_light'` version, to enable the latests version do:

```
> ./configure --enable-cuda --enable-sapporo2
```

6 Writing documentation

6.1 Getting started

The documentation for AMUSE is generated from ReStructured Text using the Sphinx documentation generation tool. Sphinx version 1.0 or later is required. You might still run into problems, so most developers work from the sphinx source repository (Mercurial based) because it is a rapidly evolving project:

```
> hg clone http://bitbucket.org/birkenfeld/sphinx/
> cd sphinx
> python setup.py install
```

The documentation sources are found in the `doc/` directory in the trunk. To build the AMUSE documentation in html format, cd into `doc/` and do:

```
make html
```

you can also pass a `pdflatex` flag to make to build a pdf, or pass no arguments to show help information.

The output produced by Sphinx can be configured by editing the `conf.py` file located in the `doc/`.

6.2 Organization of the AMUSE documentation

The actual ReStructured Text files are kept in `doc/install`, `doc/design`, `doc/tutorial`. The main entry point is `doc/index.txt`. The documentation suite is built as a single document in order to make the most effective use of cross referencing, we want to make navigating the AMUSE documentation as easy as possible.

Additional files can be added to the various sections by including their base file name (the .txt extension is not necessary) in the table of contents.