Table of Contents

Version: 08/22/12

Proposal for a ITM library layout standard

This documents outlines a directory structure for software libraries released to the ITM platform. It aims to support to differentiate the library releases by:

- ITM data version number
- possibly multiple (sub-)versions of a library for every data version
- optimized production and non-optimized debugging binaries
- separate binaries and include files for different compilers and compiler versions where required

Distribution of libraries is ideally done through the module system.

1 Directory structure

- **Root Library Folder** (e.g. /afs/efda-itm.eu/project/switm/lib)
  - **Library name** (e.g. itmggd)
    - **Version name** (e.g. r587, production, development) (optional)
      - **Data version** (e.g. 4.09a, 4.09a-dev, see below) (optional)

        An environment variable ITMLIB_<LIBNAME>_DIR pointing here should be defined by the module system

        - **include**
          - compiler-independent include files (e.g. C header files)
        - **lib**
          - OBJECTCODE (see below)
            - compiler-dependent files
            - *.a, *.so library archive files (e.g. libitmggd.a)
            - *.mod Fortran module files (e.g. itm_types.mod)
          - pythonX.Y
            - Python modules for Python version X.Y (see below)
          - compiler-independent library files (e.g. libUALLowLevel.a)
        - **libdebug**
          - same as lib, but with debug symbols and no optimization
        - **bin**
          - executables & scripts
        - **man**
          - manpages
        - **doc**
          - documentation files

1.1 Access rights

In order for projects to maintain their library releases themselves, the responsible users require access rights to the library folder on the directory level below the root folder. This is probably best done via user groups on the file system level.

1.2 Library version & data version

Below the library name, there can be an optional level in the hierarchy specifying the library version (e.g. production, development, SVN release, …).

For libraries that depend on data version, there is an optional level specifying the data version.

Not every library version must provide a release for every data version (this has to be taken into account when setting up the module file for the library!).

Example:

- /afs/efda-itm.eu/project/switm/lib
  - itmassert
    - lib
    - include
  - amns
    - r52
      - 4.09a
        - lib
        - …
      - 4.09b
      - 4.10a
    - r67
      - 4.10a
  - ual
    - production (current production version)
      - 4.09a
      - 4.10a
    - production-2012.05.27 (backup of previous production version)
      - 4.09a
      - 4.10a
    - memorycache-test (development version)
      - 4.09a

Environment variables defined by modules (e.g. for DATAVERSION = 4.09a):

ITMLIB_ITMASSERT_DIR = /afs/efda-itm.eu/project/switm/lib/itmassert

ITMLIB_AMNS_DIR = /afs/efda-itm.eu/project/switm/lib/amns/r67/4.09a

ITMLIB_UAL_DIR = /afs/efda-itm.eu/project/switm/lib/ual/production/4.09a

2 Library module files

Libraries should be made available through module files. A library module file should provide the following:

- explicitly state prerequisites
- check availability of the library for given versions of prerequisites, with error reporting if loaded for unsupported versions
  - DATAVERSION
  - PYTHONVERSION
  - …
- export environment variables pointing to the library location for use in build systems

3 OBJECTCODE constants

The purpose of an OBJECTCODE constant is to identify a specific compiler and the version of the compiler (if applicable). This is required for libraries containing files that are compiler-dependent, as is typically the case for binaries generated by Fortran compilers.

3.1 Objectcode on the (first) Gateway

Proposed OBJECTCODE variables for the Gateway computer at ENEA Portici:

- amd64_g95_0.91
- amd64_g95_0.92
- amd64_gnu_4.6
- amd64_gnu_4.7
- amd64_gfortran_4.7
- amd64_pgi_8
- amd64_pgi_10
- amd64_intel_11
- amd64_intel_12
-

3.2 Automatic definition of OBJECTCODE_* variables by the module system

To allow automatic selection of the correct version of a library in a build system (e.g. Makefile), the currently selected versions of all compilers should be made available through environment variables.

This can be done trough the module system:

Example:

module load compilers/intel/11.0

sets OBJECTCODE_INTEL = amd64_ifort_11

module load compilers/intel/12.0.2

sets OBJECTCODE_INTEL = amd64_ifort_12

Proposed environment variable names:

OBJECTCODE_G95	G95 Fortran compiler
OBJECTCODE_GNU	GNU compilers (gfortran, gcc, g++)
OBJECTCODE_PGI	Portland group compilers (pgf90, pgcc, pgcpp
OBJECTCODE_INTEL	Intel compilers (ifort, icc, icpc)

3.3 Python packages

Python packages are included in the structure similar to the convention for UNIX described here: http://docs.python.org/install/index.html#alternate-installation-the-user-scheme

- **Root Library Folder** (e.g. /afs/efda-itm.eu/project/switm/lib)
  - **Library name** (e.g. itmggd)
    - **Version** (optional)
      - **Data version** (optional)
        - **include**
          - pythonX.Y/library name
        - **lib**
          - pythonX.Y
            - Python modules for Python version X.Y
        - **bin**
          - scripts
  - data files are placed in the root folder