SKIRT: Installation Guide

Chuizheng Kong

November 8, 2024

1 References

- Installation Guide:
  https://skirt.ugent.be/root/_installation_guide.html
- Install SKIRT using the command line (Unix or macOS):
  https://skirt.ugent.be/root/_install_s_k_i_r_t_unix.html
- Use the SKIRT build options (all platforms):
  https://skirt.ugent.be/root/_install_s_k_i_r_t_options.html
- Enable multi-processing for SKIRT (Unix or macOS):
  https://skirt.ugent.be/root/_install_s_k_i_r_t_m_p_i.html
- Installing PTS (Unix or macOS):
  https://skirt.ugent.be/root/_install_p_t_s.html

2 Install SKIRT Using the Command Line (Unix or macOS)

The SKIRT code is intended to be truly cross-platform. It is written in standard **C++14** and uses the **CMake** build system. The source code for SKIRT is hosted on GitHub and formatted using clang-format.

The default and most basic build configuration includes a fully-functional version of the SKIRT command-line program with support for multiple parallel threads in a single process. In this configuration, the code has no external dependencies other than the C++ compiler and the CMake build tool. Additional capabilities that may depend on external components can be enabled through user-configurable build options. To enable the multi-processing (and hence multi-node) capabilities of SKIRT, the host system must provide an implementation of the standard MPI.

2.1 Install the build tools (skipped)

2.2 Get the source code

To install the SKIRT code you need to copy the source code to your local file system. First create a directory hierarchy that will hold the source code, binaries and runtime information. The top-level directory can have any name (for example `SKIRT`) and can be located anywhere, but the subdirectories should be named as indicated below. Execute the following commands in a Terminal window to create the `SKIRT` directory.

```
cd /home/skirt
mkdir SKIRT
```

```
cd SKIRT
mkdir release run git
```

You now need to create a local repository (*local* meaning "on your own computer") by cloning (i.e. copying) the online public SKIRT GitHub repository. You can clone your local copy directly from the main SKIRT repository. To achieve this, enter the following commands in a Terminal window:

```
cd /home/skirt/SKIRT
git clone https://github.com/SKIRT/SKIRT9.git git
```

To update your local copy of the source code after changes were made to the repository from which you cloned it, use:

```
cd /home/skirt/SKIRT/git
git checkout master
git pull
```

2.3 Configure and build the code

The SKIRT code code includes 2 `bash` shell scripts to help configuring and building SKIRT from the source code in the git directory. Before you use these script for the first time, you may need to make them executable:

```
cd /home/skirt/SKIRT/git
chmod +rx configSKIRT.sh
chmod +rx makeSKIRT.sh
```

To build SKIRT for the first time, using default build options, enter the following commands:

```
cd /home/skirt/SKIRT/git
./makeSKIRT.sh
```

If all goes well, you should see output similar to this:

```
Using /usr/bin/cmake to generate build files

-- The C compiler identification is GNU 11.4.1
-- The CXX compiler identification is GNU 11.4.1
...
[100%] Building CXX object SKIRT/main/CMakeFiles/skirt.dir/SkirtMain.cpp.o
[100%] Linking CXX executable skirt
make[2]: Leaving directory '/home/skirt/SKIRT/release'
[100%] Built target skirt
make[1]: Leaving directory '/home/skirt/SKIRT/release'
make: Leaving directory '/home/skirt/SKIRT/release'
```

By default, the procedure described above builds just the SKIRT command line program. To build optional programs or to enable capabilities such as multi-processing, you need to adjust the corresponding build options through the `configSKIRT.sh` script. For a list of build options, see section 3. For example, to also build the documentation streamliner used by SKIRT developers, enter the following command

(still in the `git` directory):

```
./configSKIRT.sh BUILD_DOX_STYLE=ON
```

In some cases, you need to repeat this process. New options may appear because you enabled the option on which they depend. To adjust such dependent options, you need a 2nd round of performing the `configSKIRT.sh` script.

2.4 Download the SKIRT resource files

Because of size limitations in GitHub repositories, the resource data files needed by the SKIRT code are hosted elsewhere (on the Ghent University science faculty data server) and must be downloaded separately. The resource files are organized in *resource packs*, i.e. ZIP archives containing related resource data. The `Core` resource pack is required for the basic operation of SKIRT; other resource packs are optional and must be installed only if the corresponding SKIRT functionality is actually being used. The SKIRT source code repository *does* contain a list of the names and version numbers of the resource packs that should (or could) be downloaded. The shell script `downloadResources.sh` uses this list to help download and install each of the expected resource packs. The download script can be invoked by entering the following commands:

```
cd /home/skirt/SKIRT/git
./downloadResources.sh
```

The script will ask confirmation before starting the download of a resource pack that has not yet been installed:

```
SKIRT9_Resources_Core is not installed
Do you want to download and install SKIRT9_Resources_Core version 7? [y/n] y
```

Always answer `y` for the `Core` resource pack (you can skip the other resource packs and download them later simply by running the script again). You will then see a log similar to the following:

```
Downloading SKIRT9_Resources_Core_v7.zip ...
-----------------------------------------------
...
Archive:  SKIRT9_Resources_Core_v7.zip
  inflating: SKIRT9_Resources_Core/version.txt
  ...
  inflating: SKIRT9_Resources_Core/SingleGrain/MeanPascucciBenchmarkOpticalProps.stab
-----------------------------------------------
SKIRT9_Resources_BPASS is not installed
Do you want to download and install SKIRT9_Resources_BPASS version 1? [y/n] n
...
Do you want to download and install SKIRT9_Resources_AtomsMolecules version 7? [y/n] n
Done.
```

After downloading, the resource files are extracted from the archive and placed in a subdirectory of the `SKIRT/resources` directory, next to your `SKIRT/git` directory. The file `history.txt` inside that subdirectory offers brief historical release notes for the corresponding resource pack.

2.5 Finalize the installation

To provide easy access to the executables in the SKIRT code, edit your login script (`~/.bashrc` or equivalent if you are using a shell other than `bash`) to add the appropriate SKIRT executable paths to your system path. For example, add the following line:

```
export PATH="${HOME}/SKIRT/release/SKIRT/main:${PATH}"
```

To verify your installation of the SKIRT code, enter the command `skirt` without any command-line arguments. If the SKIRT code has been successfully installed, you should see output similar to this:

```
[skirt@lingshan git]$ skirt
07/11/2024 16:45:35.045   Welcome to SKIRT v9.0 (git 8765014 built on 07/11/2024 at 16:14:31)
07/11/2024 16:45:35.045   Running on lingshan for skirt
07/11/2024 16:45:35.050   Interactively constructing a simulation...
07/11/2024 16:45:35.050 ? Enter the name of the ski file to be created:
```

You can follow the instructions in the Terminal window to create a SKIRT parameter file or press [CTRL] + [C] to abort the program.

3 Use the SKIRT build options (all platforms)

The SKIRT build process is driven by the CMake utility. To this end, CMake manages a set of "variables" that control the build process. In the context of the SKIRT code, we recognize 3 categories of CMake variables:

- Primary build options to include additional programs or to enable extra capabilities. The variable names for these options start with "BUILD_".
- Secondary build options for which CMake usually finds the appropriate values, but which can be adjusted by the user if needed.
- Advanced variables which are almost never to be touched by a user, and which are displayed by CMake only upon special request.

Table 1 includes the primary and secondary build options relevant for building the SKIRT code.

4 Enable multi-processing for SKIRT (Unix or macOS)

The SKIRT command line program can always run multiple execution threads within a single process. The information below is relevant only if you want to run multiple parallel processes, possibly on multiple compute nodes. To enable the multi-processing capabilities of the SKIRT command line program, the host operating system must provide an implementation of the standard MPI, and the SKIRT code must be (re)built with the corresponding build option enabled.

4.1 Install OpenMPI (skipped)

4.2 Enable the MPI build option

Once you have verified that the host system provides an MPI implementation, you need to enable the `BUILD_WITH_MPI` build option and rebuild the SKIRT code.

Name	Default	Relevance	Description
BUILD_DOX_STYLE	OFF		If ON, build the documentation streamliner `doxstyle` (for developers)
BUILD_MAKE_UP	OFF		If ON, build the `MakeUp` desktop utility (for freshmen)
BUILD_SKIRT	ON		If ON, build the SKIRT command line program
BUILD_SMILE_SHAPES	OFF		If ON, build the SMILE `shapes` example (for developers)
BUILD_SMILE_TOOL	OFF		If ON, build the `smiletool` command-line utility (see The smiletool command-line utility)
BUILD_WITH_MPI	OFF		If ON, use the MPI to enable multi-processing
CMAKE_BUILD_TYPE	Release		The type of build, usually Release or Debug
CMAKE_CXX_COMPILER	*auto*		The path to the C++ compiler used for the code
CMAKE_C_COMPILER	*auto*		The path to the "plain" C compiler used for the code
CMAKE_INSTALL_PREFIX	*auto*		Not used.
GIT_EXECUTABLE	*auto*		The path to the executable for the `git` command line tool.
MPI_CXX_COMPILER	*auto*	BUILD_WITH_MPI	The path to the C++ compiler used for compiling files that refer to MPI headers.
MPI_CXX_INCLUDE_PATH	*auto*	BUILD_WITH_MPI	The path to the include directory(ies) for the MPI headers.
MPI_CXX_LIBRARIES	*auto*	BUILD_WITH_MPI	The path to the library directory(ies) for the MPI libraries.
Qt5Core_DIR	*auto*	BUILD_MAKE_UP	The path to the directory for the Qt5Core module.
Qt5Gui_DIR	*auto*	BUILD_MAKE_UP	The path to the directory for the Qt5Gui module.
Qt5Widgets_DIR	*auto*	BUILD_MAKE_UP	The path to the directory for the Qt5Widgets module.

Table 1: Alphabetical list of build options. Here *auto* means that CMake usually finds the appropriate path without user intervention. In case CMake fails (e.g., the proposed path is empty, contains a string such as "NOTFOUND", or points to the wrong executable or library), you can manually override the value of the variable with the correct path.

5 Installing Python Toolkit for SKIRT (PTS) (Unix or macOS)

PTS is written in Python 3 and requires a Python distribution for language version 3.7 or later to be installed on the host computer. In addition to the functionality offered by the Python standard library packages, PTS also depends on some non-standard but commonly available packages. Usually, these packages can be easily obtained through the Python package manager included with the installed Python distribution. The table below lists the non-standard packages that are used at the time of writing. Note that each of these packages may have additional dependencies, requiring other packages to be installed as well.

Package	**Description**
python	Python language environment
numpy	General-purpose array-processing and math
scipy	Mathematics and scientific library
matplotlib	Plotting
astropy	Community python library for astronomy
lxml	Support for XML and XSLT
pillow (PIL)	Basic image processing
reportlab	Direct PDF file generator
ipywidgets	Interactive widgets for Jupyter notebook

Table 2: Non-standard Python packages used in PTS

5.1 Getting the source code

To work with PTS you need to copy the Python source code to your local file system. First create a directory hierarchy that will hold the PTS source code and run-time information. The top-level directory can have any name (for example `PTS`) and can be located anywhere (for example in `/home/skirt`), but the subdirectories should be named as indicated below. Execute the following commands in a Terminal window to create the PTS directory.

```
cd /home/skirt
cd
cd PTS
mkdir run pts
```

The PTS source code is available from the public PTS GitHub repository. To obtain the code, simply type the following commands in a Terminal window:

```
cd /home/skirt/PTS
git clone https://github.com/SKIRT/PTS9.git pts
```

To *update* your local copy of the source code after changes were made to the repository from which you cloned it, use:

```
cd /home/skirt/PTS/pts
git pull
```

5.2 Configuring PTS paths and aliases

It is handy to provide an alias so that you can easily access PTS from the command line. To accomplish this, you will have to add the following lines to your login script (`~/.bash_profile` or equivalent if you are using a shell other than `bash`):

```
export PYTHONPATH=/home/skirt/PTS
alias pts="python -m pts.do"
```

5.3 Test the PTS installation

You can run a basic test of your PTS installation by entering the following command line:

```
pts try me
```

which should produce a response similar to the following:

```
07/11/2024 21:02:48.463   Starting admin/try_do...
07/11/2024 21:02:48.463   Command line arguments are:
07/11/2024 21:02:48.463     Fixed string:    me
07/11/2024 21:02:48.463     Optional string: PTS is great
07/11/2024 21:02:48.463     Float number:    3.14
07/11/2024 21:02:48.463     Integer number:  7
07/11/2024 21:02:48.463   Finished admin/try_do.
```

5.4 List Python package dependencies

You may need to install additional 3rd-party Python packages on which the PTS code depends. To assist with this process, the `list_dependencies` command script lists all packages referred to by the PTS code, including an indication of whether each package is already installed or not.

```
pts list_dependencies
```

The list includes both standard and 3rd-party packages. Because the standard packages are built into the regular Python distribution, they will always be marked as "installed".

```
07/11/2024 21:03:59.976   Starting admin/list_dependencies...
07/11/2024 21:04:00.084   PTS depends on 33 packages:
07/11/2024 21:04:00.084     PIL -- installed
07/11/2024 21:04:00.084     argparse -- installed
07/11/2024 21:04:00.084 !   astropy -- NOT INSTALLED
07/11/2024 21:04:00.084     datetime -- installed
07/11/2024 21:04:00.084     filecmp -- installed
07/11/2024 21:04:00.084 !   fsps -- NOT INSTALLED
07/11/2024 21:04:00.084     functools -- installed
07/11/2024 21:04:00.084     getpass -- installed
07/11/2024 21:04:00.084     glob -- installed
07/11/2024 21:04:00.084     gzip -- installed
07/11/2024 21:04:00.084     importlib -- installed
07/11/2024 21:04:00.084     inspect -- installed
07/11/2024 21:04:00.084 !   ipywidgets -- NOT INSTALLED
```

```
07/11/2024 21:04:00.084     logging -- installed
07/11/2024 21:04:00.084 !   lxml -- NOT INSTALLED
07/11/2024 21:04:00.084     matplotlib -- installed
07/11/2024 21:04:00.084     multiprocessing -- installed
07/11/2024 21:04:00.084     numpy -- installed
07/11/2024 21:04:00.084     os -- installed
07/11/2024 21:04:00.084     pathlib -- installed
07/11/2024 21:04:00.084     pkgutil -- installed
07/11/2024 21:04:00.084     re -- installed
07/11/2024 21:04:00.084 !   reportlab -- NOT INSTALLED
07/11/2024 21:04:00.084     scipy -- installed
07/11/2024 21:04:00.084     shutil -- installed
07/11/2024 21:04:00.084     socket -- installed
07/11/2024 21:04:00.084     struct -- installed
07/11/2024 21:04:00.084     subprocess -- installed
07/11/2024 21:04:00.084     sys -- installed
07/11/2024 21:04:00.084     time -- installed
07/11/2024 21:04:00.084     warnings -- installed
07/11/2024 21:04:00.084     xml -- installed
07/11/2024 21:04:00.084     zipfile -- installed
07/11/2024 21:04:00.084   Finished admin/list_dependencies.
```