WallyTutor

Kompanion Environment

Walter Dal'Maz Silva

2026-08-18

This it the *Kompanion companion* notes!

Kompanion is a portable setup toolbox for working in Scientific Computing under Windows. It is designed to be used in a laptop with no or minor administrator rights, and it allows you to install and use a wide range of tools and libraries for scientific computing without the need for a full installation.

For more details, please check our repository at GitHub. You might also be interested in Majordome, a project closely related to Kompanion (which acts as its build environment and provides installation of several packages it interacts with). Contributions and corrections are welcome, please reach me by mail.

Important: before using this, please be aware that your company/university probably forbids the use of portable tools outside sandboxed systems. It is your responsability to use this toolbox only in systems where you are allowed to. The developper does not hold any resposibility on your failure to respect your local regulations.

Foreword

For many years I have been trying to *work*... well, when you are in scientific computing you quickly discover that IT knows nothing about it and even in academic settings these guys are unaware of your needs. Those who are supposed to help you thrive are actually stopping you from even trying to do things. The *corporate* rules are too restrictive for any creative mind; *governance* means failure.

As a natural rebel, I learned everything to by-pass rules. Early in my career, when I was working only in Linux systems, that was pretty easy as you can compile your code and that is not against the rules at all in most settings. That allowed me to develop a solid background in compiling the code of others, what is not always trivial. When reading the *README* and *INSTALL* projects of major scientific computing projects, you quickly realize the inconsistencies in procedures and the available flags for compilation. When faced many and many times with this sort of issue, you are obliged to master at least `Makefiles` and `CMakefiles`. The classical `Makefiles` may be obscure

with some macro expansions doing weird stuff, while `CMakefiles` can be so huge that it takes years to fully create a grasp of them.

Next, when moving into the corporate world, you will at some point be forced to go into Windows, and that was the case for me too. About 10 years ago, before the *big-bullshit era* of AI, the scientific computing for Windows was pretty poor. Even Python was quite limited, you most certainly needed to use Anaconda at that time. Most scientific libraries are harder to compile under Windows and you will probably learn about Cygwin and MSYS2 to try to overcome the issues, as getting a proper license of Windows compiler will probably be difficult unless that is your core job as a developer.

The other part of the nightmare concerns *PortableApps*: you can't simply trust those packages, and you don't want to get in trouble by bringing some viruses home. To be avoided at all costs. That's when you learn that many installers are simply zip packages and that you can extract them somewhat. It was that that gave rise to *Kompanion* (it had so many names and repositories over the years, but now we will stick with it): allow people to do scientific computing in a laptop with no or minor (needed to run codes in parallel) administrator rights. I discuss about that in more detail in a dedicated section below, and I strongly encourage you to read that to understand a bit what is happening under `kompanion.ps1` script without reading its thousands lines of *copy-paste* code.

Installation

Before you start

The minimal Kompanion setup includes:

- Visual Studio Code
- LiteXL
- Git
- 7-Zip
- Curl
- Quarto
- uv

Some programming languages are also included by default and always:

- .NET (C#, F#, VB.NET)
- MingW64 (C/C++, Fortran)
- NodeJS
- Rust
- Python (WinPython)

If that's all you need, simply open a PowerShell terminal (preferred over the legacy *Windows PowerShell*) at this directory and run `.\kompanion.ps1` (or `.\kompanion.ps1 -RebuildOnStart` if the first run fails) to install the defaults. In most cases this will work out of the box, as the links for these tools are from well known sources and are generally not filtered by corporate firewalls. If that fails, you might need to build Kompanion elsewhere and drop it in a USB drive to transfer to your workstation.

Installing the basic setup before tweaking the packages you need is a good choice, as generally you will have no proper code editor (other than the trashy Notepad) to configure `kompanion.json` file, as described in what follows. Also, if there are any errors, a minimal configuration will produce less messages and it will be easier to debug.

After succeding the preliminary installation, you can start tweaking the packages you need. For that, run `.\kompanion.ps1 -RunVsCode` from this directory. That will start VSCode with the current project open. Under `.kompanion` directory you will find a `kompanion-sample.json` which allows one to configure packages to be installed. Copy this file and rename it to `kompanion.json`, where you can play with the `true/false` flags for selecting packages. After saving the file run `.\kompanion.ps1 -RebuildOnStart` once again and wait for completion.

- General purpose tools:

Tool	Install
drawio	False
ffmpeg	False
imagemagick	False
inkscape	False
jabref	False
lessmsi	False
logseq	False
miktex	False
nvim	False
pandoc	False
poppler	False
tabby	False
zettlr	False

- Programming languages:

Tool	Install
coq	False
elm	False
erlang	False
haskell	False
julia	False
octave	False
perl	False
racket	False
rlang	False

- Scientific libraries:

Tool	Install
blender	False
dwsim	False
elmer	False
freecad	False
freefem	False
gmsh	False
gnuplot	False
meshlab	False
ollama	False
opencascade	False
paraview	False
prepomax	False
radcal	False
su2	False
tesseract	False

Note: please notice that if some package fails to download, *e.g.* firewall issues, you might need to manually clean the respective folder under `.kompanion\bin` before running again. Simply delete the directory named after the failed package. You may also need to delete the file named after that package under `.kompanion\temp` as there be a partial download there.

If you want to save some important disk space, after installation you can delete the whole `.kompanion\temp`.

Now you should be good to go!

Automatic sourcing

This section describes how to get Kompanion automatically sourced in your user profile for all PowerShell sessions. Start by creating an environment variable `KOMPANION_SOURCE` pointing to the *full-path* of file `kompanion.ps1`.

From Windows menu, search for *Environment variables for your account* and open it. Click on *New* create a variable named `KOMPANION_SOURCE`; use the *Browse file...* button to select the `kompanion.ps1` file to get the right path.

For instance, if you are called like me and cloned this repository at `GitHub\kompanion` under your profile, then its value should be `C:\Users\walter\GitHub\kompanion\kompanion.ps1`

Next, start PowerShell and identify the path to your user profile:

```
echo $PROFILE
```

If that file does not exist, create it with:

```
New-Item -ItemType File -Path $PROFILE -Force
```

Now you can run `notepad $PROFILE` to open it and append the following lines:

```
if (Test-Path $env:KOMPANION_SOURCE) {
    . $env:KOMPANION_SOURCE

    # Uncomment the following if you want the terminal to start at
    # the root of this repository; useful for development.
    # cd "$env:KOMPANION_DIR"
} else {
    Write-Host "KOMPANION_SOURCE not found at '$env:KOMPANION_SOURCE'"
}
```

> **i** About Antigravity
>
> For some reason, when running Google Antigravity from within a shell with Kompanion sourced, it *believes* the override of home directory performed by Kompanion and variable `$PROFILE` points to Kompanion's root directory. You can create a directory `Documents\PowerShell` located at Kompanion root and place the configuration file there. For getting the right path/file name, run `echo $PROFILE` from a terminal running from Antigravity.

Now all PowerShell sessions will automatically source Kompanion, and you can start using it right away. You can run `Kode` from any directory to open the root of this repository in VSCode.

Graphical user interface

Kompanion includes a self-contained WPF application for Windows that scans `$env:KOMPANION_REPO` directory for Git repositories, and lets you launch VS Code, pull, or push each one with a single click. You can find the source code in the `app` directory of this repository. To build the application you need to enable .NET support in Kompanion (see `kompanion-sample.json`) and have added `$env:KOMPANION_SOURCE` to your user environment variables as described in the previous section. Then you can follow instructions in the `README.md` file of that directory to build and run the application. You can pin the resulting `KompanionUI.exe` to your taskbar for easy access.

User configuration

Other than the `kompanion.json` file, you can also create a `.kompanion/psrc.ps1` file to add your own custom functions and aliases. This file will be automatically sourced after the main `kompanion.ps1` script, so you can use it to customize your environment without modifying the main script. Also notice that upon start, Kompanion will dump all its generated environment variables to `.kompanion/envs.ps1`. You can use that file as a reference to know which variables are available for your own customizations.

Known limitations

- Windows MPI support (as required by SU2 in their distributed version) still requires administrator rights, as you need to install the Microsoft MPI.
- You cannot pin the editor to the taskbar (because of how sourcing environment works), but you can create a shortcut to `kode.vbs` in your desktop.

How does it work?

Kompanion is intended to be used as a portable environment with *almost* all batteries included. Applications are placed under `.kompanion/bin/` directory and all configuration of paths and other required environment variables are provided by *kompanion.ps1* script.

It mostly revolves around VS Code. You launch the editor through a dedicated script and everything else is available in its terminal. That said, the whole of the ecosystem is command line based - but this is scientific computing, if you don't know command line you are in the wrong job. If you followed the instructions to source Kompanion from $PROFILE, then you can also launch the terminal from the start menu and everything will be available there as well.

It is intended to leave minimal track on host system (probably a few files and directories on your user home directory or under *AppData*), but this is not enforced in its development, so take care if you are not allowed to execute some software in a given computer.

Other possibly supported applications

The following packages are not currently included in Kompanion, but they are known to be somewhat portable and they might be added in the future. Those displayed in italics cannot be currently automated, but they can be installed in a machine then copied to Kompanion (ongoing investigation on how to automate that).

- DualSPHysics
- Dyssol
- Fiji
- FileZilla
- *FreeFEM++*
- gnuplot
- Graphviz
- iperf
- Ipopt
- Jamovi
- JASP
- lazarus
- lite-xl
- LAMMPS
- MFiX
- MSYS2
- MUSEN
- Notepad++
- Octave
- OpenCALPHAD
- OpenModelica
- Orange3
- Portable LISP

- puTTy
- Ruby
- Rufus (requires admin privileges to be used though)
- SALOME (needs to be manually downloaded)
- *Scilab*
- strawberry-perl
- VMD
- ZeroBraneStudio
- Zettlr
- Xpra

Software specific instructions

DualSPHysics

For DualSPHysics one might also want to install this FreeCAD addon and this Blender addon.

gmsh

Because it was chosen that SDK version of gmsh was more adapted to integrate Kompanion, if you want to be able to pin the executable to your taskbar you need to copy the DLL from its `lib/` directory to `bin/`. If you think you will not need Python/Julia interfaces of gmsh, then you can move the DLL instead to save ~80MB of disk (and you can always install gmsh with pip for those cases).

Graphviz

If the first time you run `dot` you get a message as

```
There is no layout engine support for "dot"
Perhaps "dot -c" needs to be run (with installer's privileges) to register
the plugins?
```

Simply run `dot -c` as suggested and it should work fine (without admin privileges).

MFiX

This package is installed in a separate environment and not added to the toolbox path; that is because other Python environments could interact with each other and lead to unpredictable behavior. The following instructions are provided:

- Install miniforge3 under a dedicated directory.
- Add a script with the following content to launch miniforge console:

```
%windir%\system32\cmd.exe "/K" ^
    %~dp0\miniforge3\Scripts\activate.bat ^
    %~dp0\miniforge3
```

- Go to the downloads page, login and copy the personal installation command that looks like the following:

```
conda create -n mfix-<version> ^
    mfix==<version>            ^
    mfix-doc==<version>        ^
    mfix-gui==<version>        ^
    mfix-solver==<version>     ^
    mfix-src==<version>        ^
    -c conda-forge -c          ^
    https://mfix.netl.doe.gov/s3/<personal-token>//conda/dist
```

- To run the software activate the created environment and call its executable:

```
conda activate mfix-<version>
mfix
```

Creating a portable environment

This section illustrates the generalities of how do create a portable software toolbox. Before proceding with any installation, make sure to read all the elements provided below; simply trying to follow instructions in a step-by-step fashion will certainly lead to failure and frustration.

For each of the applications you will install, make sure to perform the following generic steps:

1. Download the portable version of the application, generally a `.zip` compressed file, and save it to some temporary location. Sometimes the normal installer of an application already supports portable mode.
2. Extract the compressed file to a dedicated directory. From there, you can move the content to the final location. See the notes below for details about compression practices.
3. Configure the application in a dedicated script or add a routine to the application script that manages all the paths and settings. The latter approach is recommended for putting everything together and is how Kompanion currently works.

Important: You need to be aware that there are two common practices of compressing (zip) a portable software: (1) some editors place everything in a directory and compress the directory,

while others (2) pack everything under the root of the compressed file. Windows built-in system allows your to navigate compressed files as part of the filesystem, so inspect which of the above methods was used. If the later is detected, take care to select the right option to ensure the contents are extracted to a new directory. Several packages are stored directly at *zip* root and that may be messy to clean if extracted without the above precautions.

Important: since setting up the environment may require editing batch scripts, be aware that to open them you cannot *click* the file, but *right-click* and *edit*, as Windows see these as executables. In these files, lines starting by `@REM` are seem as comments and generally explain something about what follow them. If you are starting from scratch and do not master batch scripting, prefer learning PowerShell instead, as it is more modern and powerful, and you can easily call PowerShell scripts from batch scripts. It is how the current version of Kompanion is implemented and it is the recommended way to go.

Initial setup

These are the required steps to get your system working for the first time:

1. Go to VS Code download page and get the `.zip` version for `x64` system (`Arm64` is not supported here). Extract it to the binaries directory and copy the path of VS Code directory the set the variable `VSCODE_DIR`.
2. Go to Git download page and select *64-bit Git for Windows Portable*, download it. Notice that this is not a compressed file *per se*, but it is desguised as an executable. Double-click it and accept the default `PortableGit` installation directory. After extraction finishes, move it to binaries directory. Add it to the path by setting the variable `GIT_DIR` to the path of the directory.

Important: After installing `VS Code`, do **NOT** enable its portable mode because the VBS file for launching it will point to a specific user-data folder.

Important: these notes are quite general and skip details. Refer to `kompanion.ps1` and look for the functions `Invoke-Configure*` and `Invoke-Install*` for the exact steps to configure each application.

Programming languages

The following provides instructions for some languages you might wish to have fully integrated to your environment. For lower level languages, please consider using MSYS2 as described in its dedicated section.

Python

Go to WinPython download page and find the latest version tagged with a *Latest* green flag (avoid *Pre-release* versions on top). Expand the *Assets* and download the `.exe` version (also a

disguised compressed file). Follow steps similar to Git above. See `Invoke-ConfigurePython` in `kompanion.ps1` for the configuration of paths and environment variables.

Julia
Go to Julia download page and select the latest stable Windows 64-bit portable version. See `Invoke-ConfigureJulia` in `kompanion.ps1` for the configuration of paths and environment variables.

Rust
Download rustup-init following the installation. See `Invoke-ConfigureRust` in `kompanion.ps1` for the configuration of paths and environment variables.

Octave
Go to Octave download page and identify the `.zip` package for `w64`. As the main use of Octave is through its user interface, you can extract the content directly to `bin/octave`.

Ruby
Go to Ruby download page and identify the `Devkit` package for `x64`. During installation change the path to point to binaries and unselect the options to associate extensions and adding to the path.

LISP
LISP may be installed through Portacle. Instead of right-clicking and installing the executable, Shift-click and select to extract the contents; place the resulting `portacle` directory. You may choose not to add it to the path if using through its EMACs interface.

LaTeX and related
Although LaTeX is not mandatory, it is highly encouraged; otherwise, what is the point of doing any scientific computing and not publishing its results?

- Start by following the instructions provided here to install MikTeX. Point the installation to a directory `miktex-portable` under `apps` and select *on-the-fly* installation of new packages.
- Consider installing the following additional software:
  - JabRef
  - pandoc
  - inkscape
- Globally install `pip install Pygments` for enabling syntax highlight in LaTeX using `minted`; that is the most flexible highlighting method for adding code snippets to your documents.
- Also consider installing extension LaTeX Workshop for automatic compilation in editor with built-in PDF visualization.

To append to `TEXMF` variable one can use the MiKTeX Console graphical interface and under `Settings > Directories` navigate and select the local path.

If you prefer a dedicated LaTeX editor, you may wish to install texstudio; some configuration of paths with the application may be required as it is not directly integrated to this environment.

Warning: for now having both Rust and Octave fully operational in command-line is not working. This should not be a problem as Octave can be launched with its user-interface and that remain the most popular/recommended way of using it.

Warning: For a fully operational *Jupyter notebook* environment, other the LaTeX support discussed in its dedicatet section, you also need both `pandoc` and `inkscape`.

Important: It is strongly recommended to install Quarto as a supporting tool for publishing reports from Julia and Python.

MSYS2

MSYS2 environment is useful for developing native Windows applications and programming in languages as C, C++, Fortran, and Rust. Kompanion VS Code configuration will automatically include this environment to its available terminal lists, what might fail if you do not install it. To get it working do the following:

- Go to MSYS2 page and save the installer to `downloads`.
- Run the installer and change installation path to point to a local `msys64` directory instead of the default `C:/msys64` (requiring admin rights).
- Once finished, launch an UCRT64 environment as proposed in the installation guide. Install at least the following:

```
pacman -S \
    mingw-w64-ucrt-x86_64-toolchain \
    mingw-w64-ucrt-x86_64-binutils \
    mingw-w64-ucrt-x86_64-gcc \
    mingw-w64-ucrt-x86_64-gcc-fortran
```

- Full package list is provided here.

Creating a portable launcher

A simple way to create a portable launcher requiring to source extra variables is by writing a simple batch script exporting or calling another script with the definitions:

```
@echo off

@REM Add variables to be sourced here such as
@REM set PATH="/path/to/some/dir";%PATH%
@REM ... or call another shared script doing so.
@REM call %~dp0\env

MyCode.exe
```

Because a batch script will keep a console window open, create a VB file with the following

```
Set oShell = CreateObject ("Wscript.Shell")
Dim strArgs
strArgs = "cmd /c MyCode.bat"
oShell.Run strArgs, 0, false
```

In the example we assume the program is called `MyCode.exe` and the batch script has been named in an analogous way `MyCode.bat`.