Building WorkflowMaker

For WorkflowMaker version **1.0.7** and later
Windows 10 & 11, Ubuntu-based Linux distributions
64-bit only

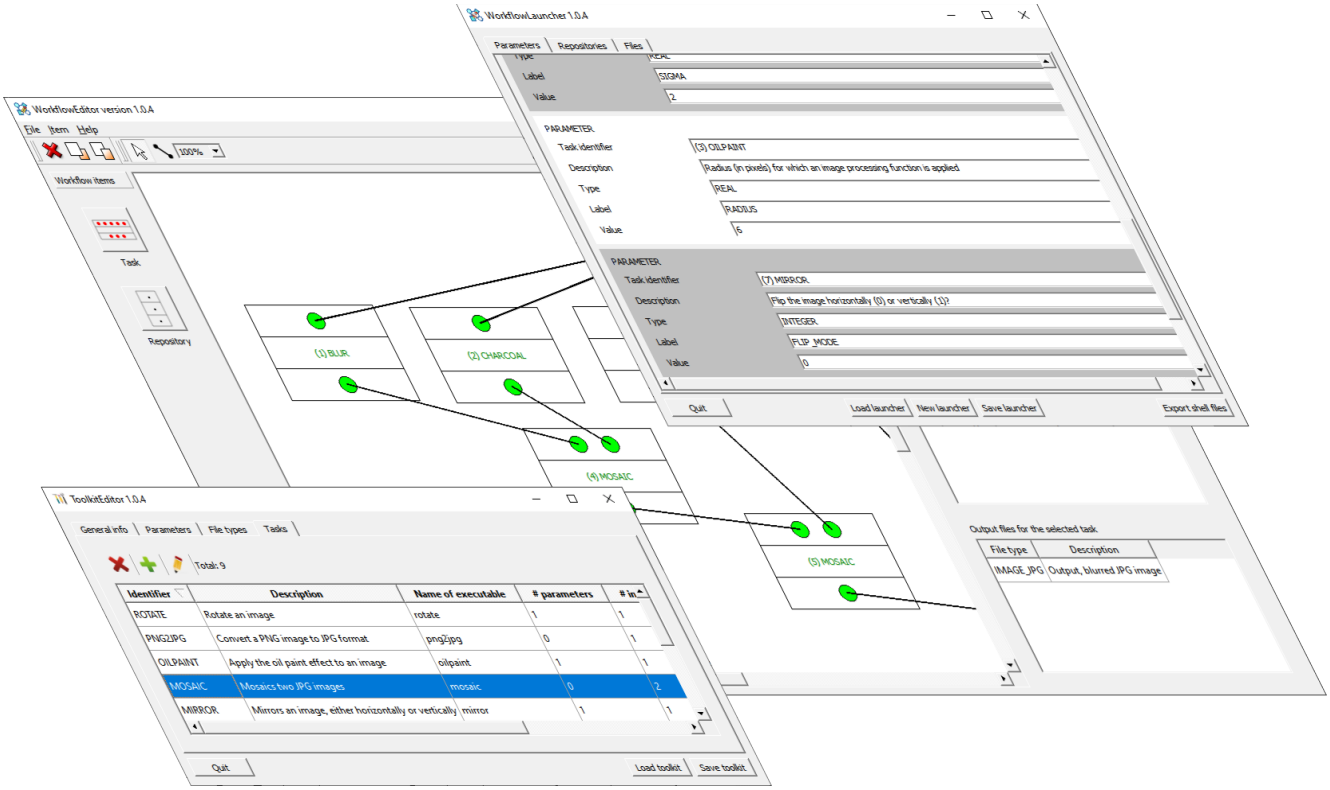

Table of contents

List of figures

1 The tools

To build WorkFlowMaker from source the following tools are required:

- Windows
  - MSVC 2022 Community Edition (https://visualstudio.microsoft.com/downloads/).
  - Qt version 5 (https://www.qt.io/download-open-source). Download the prebuilt binaries for the MSVC 2022 compiler (msvc19_64).
  - The NSIS (Nullsoft Scriptable Install System, https://nsis.sourceforge.io/Download).
  - The HM NIS EDIT editor (https://hmne.sourceforge.net/). This editor is used to build the Windows installer together with NSIS.
  - doxygen (only to build the API documentation, https://www.doxygen.nl/download.html).
- Linux
  - gcc.
  - Qt version 5 (https://www.qt.io/download-open-source).. This document assumes that the source code of the latest available version of Qt 5 is downloaded and then compiled to produce static (not shared) libraries. This makes much easier the distribution of the final WorkflowMaker deb package. Note that this assumption influences the instructions given later to build WorkflowMaker from source for Linux platforms. If a version of Qt consisting of shared libraries instead is used, then such instructions will simply not work.
  - doxygen (only to build the API documentation).

For Linux, all tools except Qt should be available using the built-in package manager.

2 The organization of the source code

The first-level sub-folders of WorkflowMaker's main folder is organized as depicted in Figure 1 on page 4. Their contents is described below;

- `data` – XML schema files defining the syntax of the toolkit, workflow and launcher files.
- `docs` – The documentation, including this guide.
- `installer` – Scripts to build installers, either for Windows or Linux platforms.
- `libraries` – The source code of the libraries developed and used by the tools in WorkflowMaker.
- `samples` – Data and source code samples for two example toolkits (image and text processing).
- `tools` – The source code for the tools (ToolkitEditor, WorkflowEditor, ẆorkflowLauncher).

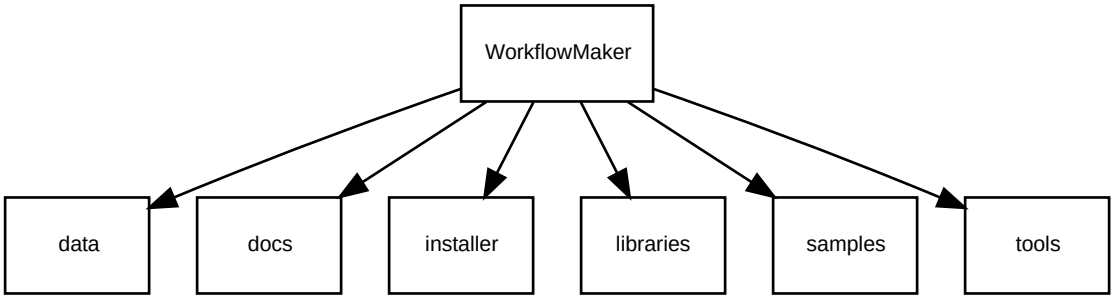

Figure 1: The organization of the source code.

2.1 The installer sub-folder

The structure of the installer sub-folder is shown in Figure 2.

Two different sub-folders exist here.

The first one, `installer_sw` includes all the necessary software and data to build a Windows installer using the Nullsoft Scriptable Install System (NSIS). This includes the NSIS script - the code defining how to build the installer - as well as several images and icons required to build it.

The second sub-folder, `installer_sw_linux`, is targeted at building a package for Debian-based Linux platforms. There, there are three files, namely `control`, `postinst` and `prerm` that will be be copied to the Linux .deb package folder structure, more precisely to the `DEBIAN` sub-folder. These define the characteristics of the package (`control`), the post-build actions to start when the package is copied to the system (`postinst`) and those to execute when uninstalling it (`prerm`). See also section 3.1.

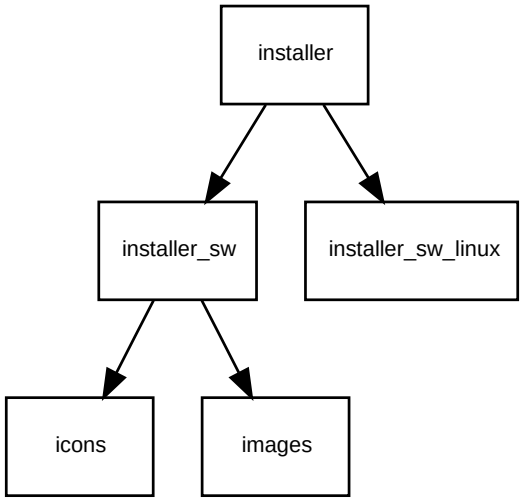

Figure 2: The structure of the installer sub-folder.

2.2 The libraries sub-folder

Figure 3 depicts the structure of the `libraries` sub-folder.

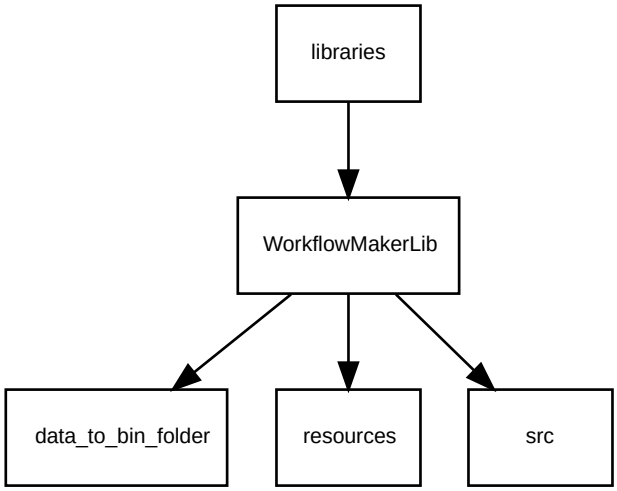

Figure 3: The structure of the libraries sub-folder.

This sub-folder must contain all the libraries used by the tools in the WorkflowMaker package, either self-developed or third-party. Currently, there is only one library, namely WorkflowMakerLib, which has been developed specifically for this project.

The `WorkflowMakerLib` folder is organized as follows;

- `data_to_bin_folder` – A single file is included here, namely `workflowmaker_version.txt`. It contains

the version string for all the applications in WorkflowMaker. Changing this file and rebuilding the library will make all these tools to show a new version string.

- `resources` – Images and HTML files to help build the developer's documentation.
- `src` – The source code for the library.

2.3 The samples sub-folder

Two examples of WorkflowMaker toolkits (as well as workflows and launchers relying on these) have been included. These are very simple image and text processing toolkits. The interest of these samples is twofold: first, it makes possible to test the tools in WorkflowMaker without the hurdle of having to define a toolkit from scratch; but, also, they show the architecture of WorkflowMaker-compliant applications. For instance, they read the values of their keyboard parameters and input and output file names using an options file, which is implemented using a C++ library named simple_options_file_parser included with the samples. The structure of the `samples` sub-folder is shown in Figure 4.

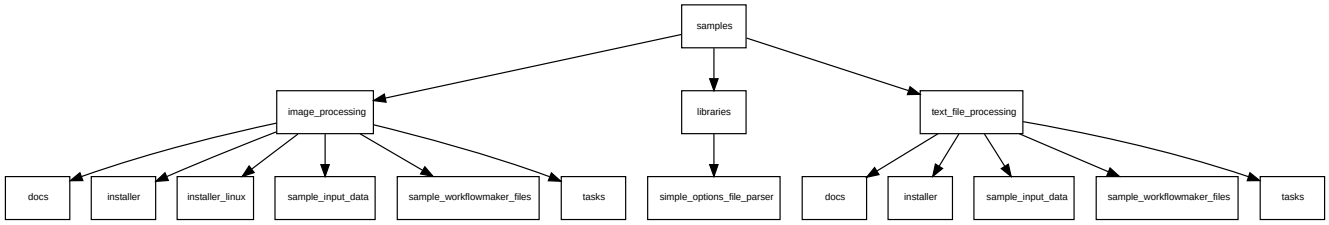

Figure 4: The structure of the samples sub-folder.

The library simple_options_file_parser is organized in the same way than WorkflowMakerLib. Please refer to the description of that library to understand how the sub-folders of simple_options_file_parser are structured (in this case, however, there is no `data_to_bin_folder` sub-folder).

Both sample toolkits are organized as follows:

- `docs` – The user guides for the toolkit.
- `sample_input_data` – Files that may be used as inputs when running the tools in the toolkit (such as images or text files).
- `sample_workflowmaker_files` – Example XML files defining one toolkit, and at least one workflow and launcher relying on the said example toolkit.
- `tasks`- The source code for each of the tools included in the toolkit. These are organized exactly in the same way that the WorkflowMaker tools (ToolkitEditor, WorkflowEditor, WorkflowLauncher) so refer to section 2.4 for details on their structure.

Folders `installer` and `installer_linux` are used to build installers for these samples. Please, refer to the samples documentation for more information.

2.4 The tools sub-folder

The three tools making WorkflowMaker reside in this sub-folder. Figure 5 depicts its structure.

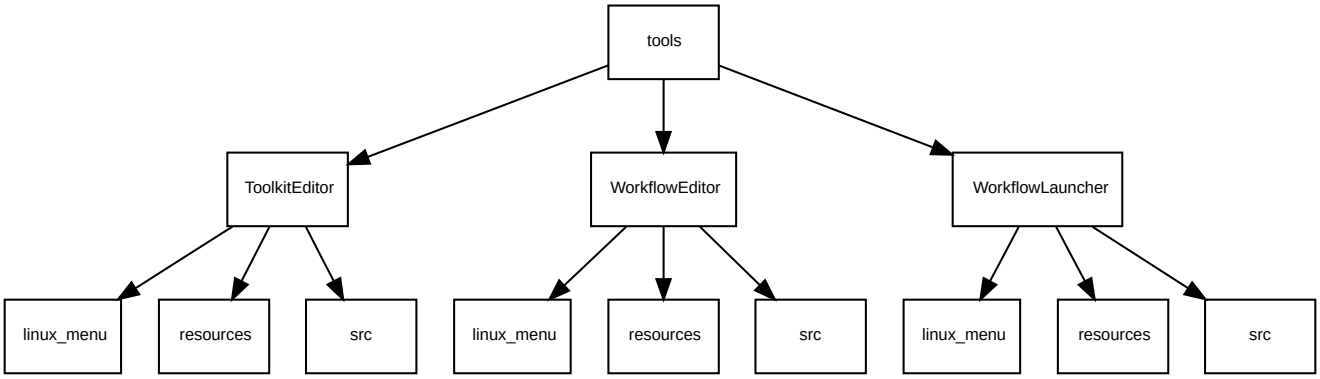

Figure 5: The structure of the tools sub-folder.

All tools have been organized in the same way:

- `linux_menu` – Files to build a menu entry for the tool in those Linux distributions that have a system menu.
- `resources` – Images or other files required to build the application or the documentation.
- `src` – The source code.
-

In the main folder (such as ToolKitEditor) several files are always present:

- *`xxxx`*`.pro` - This is the Qt project file, used to define the components making the application and giving instructions about how to build it.
- *`xxxx`*`_resources.qrc` – Definition of the images used by the application. This is a Qt file too.
- `postbuild.bat` & `postbuild.sh` – Shell files for Windows and Linux respectively, stating the steps to execute once the application has been successfully built. These files take care of creating the installer's appropriate sub-folders, copy files there (such as executable or default option files, etc). These constitute the keystone of the build system, since they prepare everything that is needed to build, later, an installer or .deb package for WorkflowMaker.
- *`xxxx`*`.dox` – Doxygen documentation file for the tool.

In the bullets above, *`xxxx`* stands for the name of each of the tools. For instance, the contents of the ToolkitEditor main folder is: `ToolkitEditor.pro`, `ToolkitEditor_resources.qrc`, `ToolkitEditor.dox`, `postbuild.bat` and `postbuild.sh`.

3 Building WorkFlowMaker

Building WorkflowMaker means first build the base library (WorkflowMakerLib) and then the tools (ToolkitEditor, WorkflowEditor, WorkflowLauncher).

Scripts to automate the whole build process for both for the Linux and the Windows platforms have been provided. In all cases, the result of the build process is copied to some directory(ies) so creating an installer or a deb package for WorkflowMaker is straightforward.

The following sections will discuss how to build WorkflowMaker for Linux and Windows.

In these sections, and to simplify the explanations, the WorkflowMaker folder will be referred to as "`workflowmaker`". Hence, sub-folders like "`data`" or "`libraries`" will be written as "`workflowmaker/data`" or "`workflowmaker/libraries`".

3.1 Building for the Linux platform

3.2 The deb folder structure

The WorkflowMaker build script creates a folder structure holding the necessary files to, later, create a deb package. This folder is always named `workflowmaker_x.y-zz_amd64` and it is placed in the sub-folder `workflowmaker/installer`, that is, its full path is

```
workflowmaker/installer/workflowmaker_x.y-zz_amd64
```

Each element built (the library, each tool) copies one or more elements to this folder structure. When everything is built, the deb package may be created.

Figure 6 depicts the structure of the said folder. The contents of each sub-folder is:

- `DEBIAN` – It contains the `control`, `postinst` and `prerm` files.
  - `control` – Defines the contents of the package, according to the syntax required for this files. It includes the name of the package, its version, the packages on which it depends, for instance. If the software is changed, this file must be modified in order to reflect any changes (such as, of course, the version number or dependencies).
  - `postinst` – This is a shell file that is executed as the last step when the deb package is installed. In the case of WorkflowMaker, it takes care of creating the menu entries for the different tools.
  - `prerm` – It is executed as the first step when the deb package is uninstalled. In this case it takes care of removing the menu entries for the applications.
  - All these files are copied from `workflowmaker/installer/installer_sw_linux`.
- `usr/local/bin` – Here, the executable files for the tools as well as the XML schema definitions

for toolkits, workflows and launchers are copied. The file with the package version string is also stored here. The executable files are produced by building the respective tools; the XML schema definitions are copied from `workflowmaker/data`. The version string file is copied from `workflowmaker/libraries/WorkflowMakerLib/data_to_bin_folder`.

- `usr/share/workflowmaker` – Here, the .desktop and icons for the tools are stored. These are copied, for each tool, from their `linux_menu` subfolders; for instance, the files for ToolkitEditor are copied from `workflowmaker/tools/ToolkitEditor/linux_menu`.

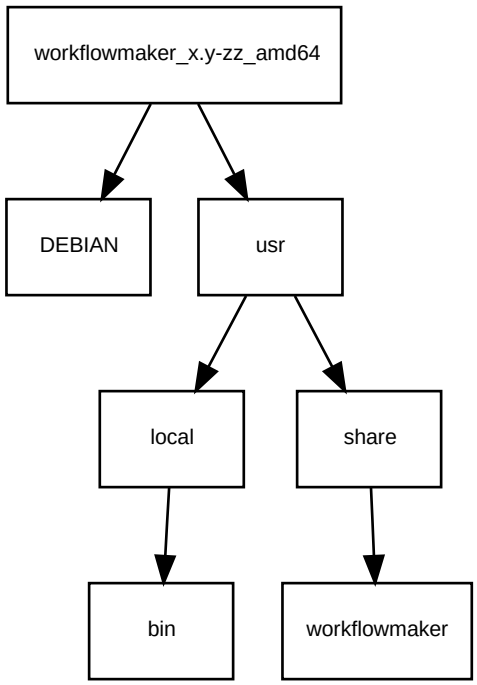

Figure 6: The structure of the folder to build the deb package.

3.2.1 Building

First of all, take care of changing the version string of the software, but **only** if changes have made; this is not necessary when building the code just downloaded from the repository. Version changes must be reported in the following files:

- `workflowmaker/libraries/WorkflowMakerLib/data_to_bin_folder/workflowmaker_version.txt` – The typical version string follows the pattern x.y.z (e.g., 1.0.7). Change this value if necessary and save the file.
- `workflowmaker/installer/installer_sw_linux/control` – Change the line “Version: x.y-z” to match the actual version. **Note the dot-dash pattern** between x and y and then between y and z.

Open a terminal. Change the default directory to `workflowmaker`

```
cd  workflowmaker
```

Make sure that the file build_all.sh has the execution privilege:

```
chmod +x build_all.sh
```

Run the build script:

```
./build_all.sh
```

The result of the execution of the script is that the deb folder structure has been created and populated with all the necessary files to build the deb package.

3.2.2 Building by hand

When changes are made to specific components of WorkflowMaker (such as a single tool, or the library), the general process to build these is:

- Open Qt Creator.
- Select the pro file for the desired target (the library or any of the tools).
- Configure the project (selecting the Qt 5 installed in your computer and the target output).
- Build.

It is worth noting that changing the library implies rebuilding all the tools to guarantee that these changes are conveniently propagated. Also, it is important to remark that building manually any of the components, either library or tools, will also copy the corresponding files to the deb folder structure.

3.2.3 Creating the deb package

Once that the library and the tools have been built, it is possible to create a deb package. To do it, follow the steps below.

Rename the folder `workflowmaker_x.y-zz_amd64` located in `workflowmaker/installer` and name it according to the version it implements. For instance, if the new version is 1.0.7, then the new name should be `workflowmaker_1.0-7_amd64`.

Open a terminal. Change the default directory to `workflowmaker/installer`

```
cd  workflowmaker/installer
```

Run the following command:

```
dpkg-deb --build --root-owner-group <deb-folder>
```

where *<deb-folder>* stands for the deb folder just renamed. For instance, if the original `workflowmaker_x.y-zz_`folder has been renamed to `workflowmaker_1.0-7_amd64`, then the command should be:

```
dpkg-deb --build –root-owner-group workflowmaker_1.0-7_amd64
```

Obviously, the name of the *<deb-folder>* must be changed in the example above to match the actual name of the deb folder.

This will create the deb file, which may be used to install WorkflowMaker in Debian-based Linux distributions.

3.3 Building for Windows platforms

3.3.1 Preparing the redistributables

The first thing to do to build WorkflowMaker from source on Windows platforms is **modifying the folder structure shown in Figure 1 to include two more sub-folders**, namely `QT_redistributables` and `VC_redistributables`. These folders must be added manually, since they are not part of the source code distribution. Figure 7 depicts this new folder structure. The red boxes stand for the changes.

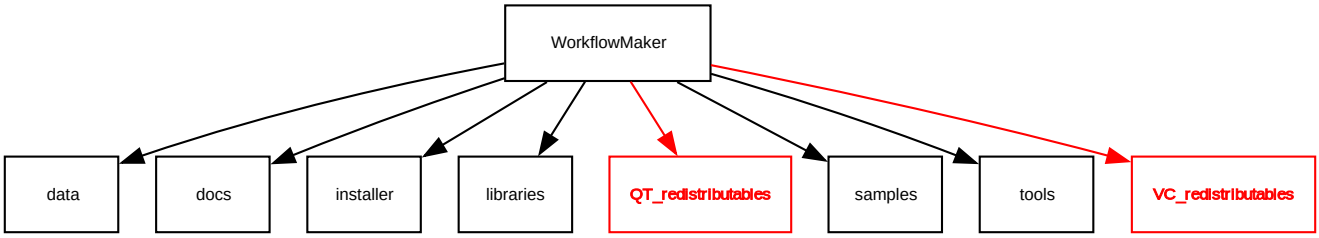

Figure 7: The structure of the source code, extended for Windows builds.

The `QT_redistributables` and `VC_redistributable` folders must be structured as shown in Figure 8.

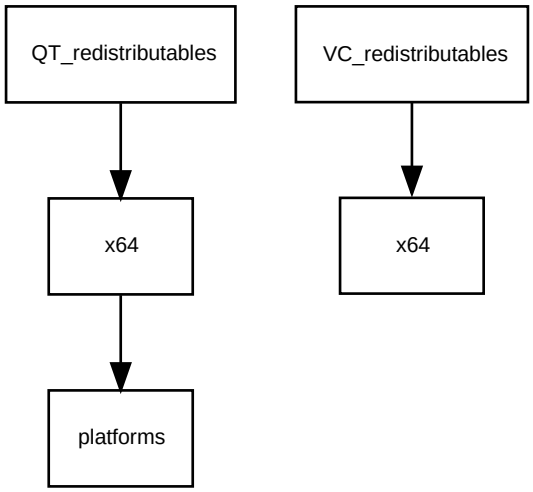

Figure 8: The structure of the QT and VC redistributable folders.

The QT_redistributables folder must contain all the Qt dynamic link libraries and platform code required by the applications. These are:

- Sub-folder x64:
  - Qt5Core.dll
  - Qt5Gui.dll
  - Qt5Network.dll
  - Qt5Widgets.dll

    - Qt5Xml.dll
    - Qt5XmlPatterns.dll
- Sub-folder x64/platform
    - qwindows.dll

All these dlls must be copied from the Qt installation folder. On Windows, this is typically `C:\Qt`. Assuming that this is the root directory, then all the dlls that must be copied to the `x64` sub-folder may be found at `C:\Qt\<qt_version>\<compiler_version>\bin`, while `qwindows.dll`, which is copied to `x64\platforms` must be obtained from `C:\Qt\<qt_version>\<compiler_version>\plugins\platforms`.

In these two paths, *`<qt_version>`* stands for the specific version of Qt installed (such as 5.15.12) and *`<compiler_version>`* for the version of the compiler for which the prebuilt Qt binaries have been installed, such as *`msvc19_64`*.

The VC_redistributables folder must contain the official Microsoft installer for the redistributables for the compiler used to build WorkflowMaker. The URL to download these may change with time, but at the time of writing this guide it could be downloaded from:

https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170#visual-studio-2015-2017-2019-and-2022

The file to download is that for x64 architectures; again, at the time of writing this guide its name is `VC_redist.x64.exe`. This file must be copied to the `x64` sub-folder of the `VC_redistributables` folder.

These redistributables (either Qt or VC) will be copied to the deployment directory (see next section) so the installer will incorporate these to the setup program that will be, finally, created.

3.3.2 The structure of the windows installer folder

When building WorkflowMaker for Windows, the build scripts create automatically a folder where all the files required to make an installer will be copied. This folder, named deployment, is created inside the installer directory, that is, at

```
workflowmaker/installer/deployment
```

Figure 9 depicts the structure of the deployment folder. These are the contents copied there by the build scripts:

- `deployment/bin/x64.`
    - The executable files of the several tools (created by the build process).
    - The XML schema files defining the toolkit, workflow and launcher (`toolkit.xsd, workflow.xsd, launcher.xsd`). Copied by the build process from folder `workflowmaker/data.`
    - The Qt redistributable. Copied `workflowmaker/QT_redistributables.`
    - The MSVC redistributable. Copied from `workflowmaker/VC_redistributables.`

    - The version string file, `workflowmaker_version.txt`, copied from `workflowmaker/libraries/WorkflowMakerLib/data_to_bin_folder`.
- `deployment/data_samples`. A set of files consisting of a toolkit for image processing, a workflow example relying on this toolkit, and an example laucher. All these files are copied from `workflowmaker/samples/image_processing/sample_workflowmaker_files`.
- `deployment/docs` .The documentation folder. All PDFs are copied from `workflowmaker/docs/user guide`.

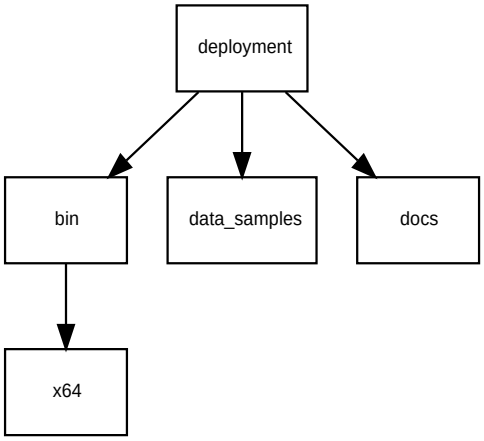

Figure 9: The structure of the Windows (installer) deployment folder.

3.3.3 Building

First of all, take care of changing the version string of the software, but **only** if changes have made; this is not necessary when building the code just downloaded from the repository. Version changes must be reported in the file named:

`workflowmaker/libraries/WorkflowMakerLib/data_to_bin_folder/workflowmaker_version.txt`

The typical version string follows the pattern x.y.z (e.g., 1.0.7). Change this value if necessary and save the file.

Open a MSVC 2022 x64 terminal.

Beware that a regular Windows terminal is not enough to build WorkflowMaker. Instead, a specific MSVC 2022 terminal to build x64 applications must be used. These may be started from the Visual Studio menú (in the Windows start menu). Its precise name is "*x64 Native Tools Command Prompt for VS 2022*".

Change the default directory to `workflowmaker`

```
cd  workflowmaker
```

Run the build script:

```
build_all.bat
```

The result of the execution of the script is that the deployment folder structure has been created and populated with all the necessary files to later build the installer.

3.3.4 Building the installer

Open the installer script.

This is file `workflowmaker/installer/installer_sw/WorkflowMaker.nsi`.

Open this file using the HM NIS EDIT editor (see section 1) **only**, since it is integrated with the NSIS installer system and only from here it will be possible to build the installer.

Change the version of the installer.

Do this only when the version of the installer must reflect a change in the version of the software. If the installer is just being regenerated, it is not necessary to change the installer's version.

To change the version, look for the following line close to the beginning of `WorkflowMaker.nsi`:

```
!define defVersion "1.0.6"
```

(the actual version number may be different) and change the version string as needed. For instance:

```
!define defVersion "1.0.7"
```

Save the installer script. Only if it has been modified by the previous step.

Build the installer.

To do it, click on the build button in the editor's toolbar (see Figure 10).

Close HM NIS Edit.

The installer is created in the folder

```
workflowmaker/installer/installer_sw
```

and it is named

```
WorkflowMaker-x.y.z-setup.exe
```

where x.y.z corresponds to the version string indicated in the installer script (see previous steps). Thus, if such version string is, for instance, 1.0.7, the name of the installer woud be:

```
WorkflowMaker-1.0.7-setup.exe
```

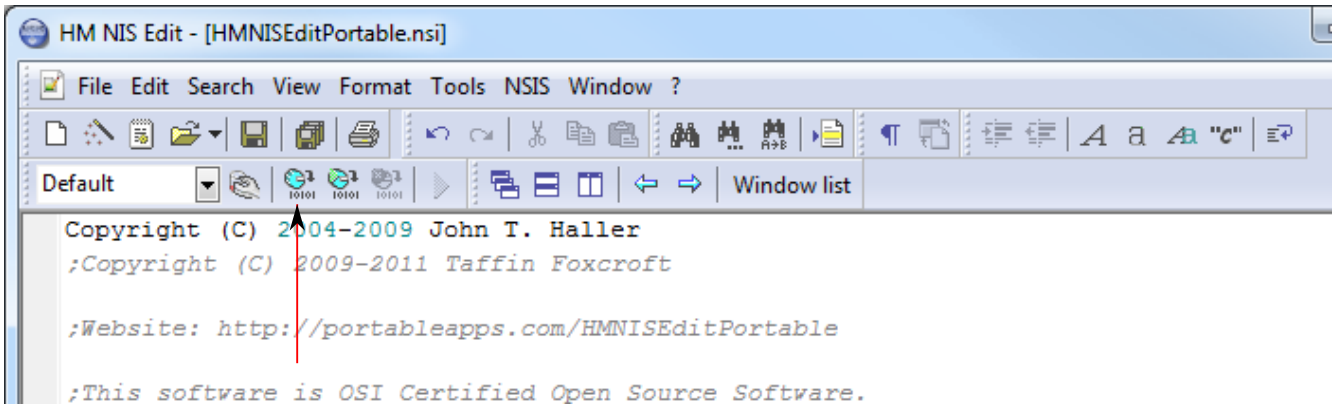

Figure 10: Building the windows installer using HM NIS Edit.

Running this installer will install WorkflowMaker on the target Windows computer.

3.3.5 Building by hand

When changes are made to specific components of WorkflowMaker (such as a single tool, or the library), the general process to build these is:

- Open Qt Creator.
- Select the pro file for the desired target (the library or any of the tools).
- Configure the project (selecting the Qt 5 installed in your computer and the target output).
- Build.

It is worth noting that changing the library implies rebuilding all the tools to guarantee that these changes are conveniently propagated. Also, it is important to remark that building manually any of the components, either library or tools, will also copy the corresponding files to the deployment folder structure.

3.4 Building the API documentation

There is no automated build tools to generate the API documentation yet. Therefore, it must be built by hand.

To do it, run doxygen using the dox file present in each sub-folder (WorkFlowMakerLib, ToolKitEditor, WorkflowEditor, WorkflowLauncher, see Figure 1). This will generate a sub-folder named `html`. Open the `index.html` file inside the `html` sub-folder to see the documentation.