# Introduction

Investigating the use of CMake is part of ‘restructuring the D-Flow Flexible Mesh (D-Flow FM) kernel’. This memo contains the results.

Main issue : <https://issuetracker.deltares.nl/browse/DELFT3D-37453>

Public Wiki page : <https://publicwiki.deltares.nl/display/DSCMEET/CMake>

Branch : https://svn.oss.deltares.nl/repos/delft3d/branches/research/Deltares/20200423\_DELFT3D-37453\_cmake\_compilation

## Goals

1. CMake produces Microsoft project files for D-Flow FM (in the branch) (MUST) and the full OSS tree (SHOULD)
2. CMake produces Makefiles for D-Flow FM (in the branch) (MUST) for Linux (MUST) and Windows (SHOULD). Ideally for the full OSS tree (SHOULD).
3. It compiles on Windows (64-bit only)
4. It compiles on Linux (64-bit only)
5. Compilers on Windows with VisualStudio integration: Microsoft (C), Intel(Fortran)
6. Compilers on Linux: Intel (C and Fortran) and Gnu (C and Fortran)
7. SHOULD: Compilers on Windows via MinGW/Cygwin: Gnu (C and Fortran)
8. Report: memo with instructions (prerequisites (inside/outside Deltares), how to execute it) and recommendations (expected problems on TeamCity, yes/no use it in the trunk, expected problems for external users).
9. (Out of scope for this project) : Deliverability of our products to external users (RPATH usage?)

# Results

With respect to the goals:

1 YES: CMake produces Microsoft project files for D-Flow FM (in the branch) (MUST) NO: and the full OSS tree (SHOULD)

2 YES: CMake produces Makefiles for D-Flow FM (in the branch) (MUST) for Linux (MUST) NO: and Windows (SHOULD). NO: Ideally for the full OSS tree (SHOULD).

3 YES: It compiles on Windows (64-bit only)

4 YES: It compiles on Linux (64-bit only)

5 YES: Compilers on Windows with VisualStudio integration: Microsoft (C), Intel(Fortran)

6 YES: Compilers on Linux: Intel (C and Fortran) NO: and Gnu (C and Fortran)

7 NO: SHOULD: Compilers on Windows via MinGW/Cygwin: Gnu (C and Fortran)

8 YES: Report: memo with instructions (prerequisites (inside/outside Deltares), how to execute it) and recommendations (expected problems on TeamCity, yes/no use it in the trunk, expected problems for external users).

9 NO: (Out of scope for this project) : Deliverability of our products to external users (RPATH usage?)

Documentation:

Usage: See README file in the source code: <https://svn.oss.deltares.nl/repos/delft3d/branches/research/Deltares/20200423_DELFT3D-37453_cmake_compilation/src/cmake/readme/README>

Report: This document “CMake\_for\_OSS\_tree\_ project\_report.docx”: [https://svn.oss.deltares.nl/repos/delft3d/branches/research/Deltares/20200423\_DELFT3D-37453\_cmake\_compilation/doc/CMake\_for\_OSS\_tree\_ project\_report.docx](https://svn.oss.deltares.nl/repos/delft3d/branches/research/Deltares/20200423_DELFT3D-37453_cmake_compilation/doc/CMake_for_OSS_tree_%20project_report.docx)

# Remarks

(+): One config tool for both Windows and Linux

(+): Prepare\_sln.py is not needed anymore

(+): Outdated references are removed

(+): CMake GUI(Windows) and command line work fine

(+): CMake does not require a special IDE or configuration tool in order to define CMakeLists files.

(+): It is not required that a CMakeLists.txt file is present at each level. The current architecture supports that a CMakeLists.txt is defined at the top level, while the CMakeLists.txt for the components are defined as close as possible in their component’s directory. This feature prevents clutter of the repository and reduces maintenance. Additionally, the location of the components is defined at the highest level. This design choice means that the components can easily be moved around with minimal effort as long as the location is updated.

(+): CMake promotes modularity. Instead of having one file which configures every individual component based on the platform and the used compilers, the CMakeLists.txt for each component can configure itself. In the unlikely event that the parameters need to be passed between component configurations, this is possible by setting properties onto the "producing" component and retrieve its property values in the "consumer" component. Furthermore, this modularity has the advantage that every component could be used in a new solution or software project, given that they are decoupled from each other (e.g. keep the interaction between the modules at minimum).

(+): CMake is platform independent and so are the CMakeLists.txt files for the individual components. CMake can generate solutions for various platforms which reduce the maintenance of configuring solutions for Windows and Unix platforms.However, it should be kept in mind that CMake is NOT a compilation tool: CMake only generates solutions in order to compile.

(+): Build-in functions. CMake contains a variety of build in functions to link libraries, gather source files etc. User functions can be defined to reduce code duplication and provide a layer of abstraction throughout the solution when necessary.

(+): (Log) messaging with different information levels to provide the user additional information when CMake is configuring the solution. CMake provides a mechanism to generate log messages with different levels, such as STATUS, WARNING, ERROR.

Drawbacks

(-): CMake works with absolute paths for defining references. This becomes especially relevant when performing file operations (such as excluding or including files for the configuration) and specifying include directories.

(-): CMake sets preconfigured compilation and linker flags, depending on the compilers that are being used. However, it is not always clear what these flags are, and it is recommended to check and adjust these flags accordingly.

(-): Additional compilation and linker flags (for example /check:bounds) are not immediately traceable in Visual Studio. While they are specified in a CMakeLists.txt, they do not always show up in the properties panel at their expected location. However, the additional options are present when checking the "Command Line" item of the "Fortran" or "Linker" item in the properties.

(-): Additional Intel settings are not always configurable by CMake in Windows. Some intel settings (such as "Link Library Dependencies" under the "Linker" tab) cannot be configured directly by a CMake file. It is unclear whether this affects the compilation process.

(-): Differences in configured results when referencing defined variables. As CMake is a string based scripting language, referencing to earlier defined variables can give subtle differences when configuring the solution. Variables can be referenced as:

* By variable name: this substitutes the variable "as is" or gives just the string if the variable was not defined
* By ${variable name} which expands the contents of the variable
* By "${variable name}" which performs a string expansion of the contents of the variable

These differences are mainly noticeable when referencing to variables that define a list.

(-): Variable renames need to be verified across the solution. Depending on how the variable is used, CMake usually does not give errors or warnings when a variable becomes undefined due to a rename. This behavior can lead to unintended side-effects after a variable rename.

(-): ???

# Current state of CMake configuration

CMake is configured for the tests.sln with the test executable test\_deltares\_common to verify that the compilation, linking and the executable is working as intended. The kernel itself is defined in two configurations, namely the tests.sln and the dflow\_open.sln. The dflow\_open.sln also contains the console wrapper dflow-cli and the dynamic library dflowfm\_dll to verify the functionality of the CMake configured dflowfm kernel. Besides the dflowfm kernel, there’s a configuration implemented to configure and compile a dimr\_open.sln. For both dflow\_open.sln and dimr\_open.sln the setting “Link Library Dependencies” needs to be *manually* adjusted as shown by Figure 4.1. This setting is not associated with a linker flag and cannot be set by a CMake configuration file. See also (<https://docs.microsoft.com/en-us/visualstudio/msbuild/link-task?view=vs-2019> under section “LinkLibraryDependencies”) The projects that need this manual adjustment are listed in Table 4.1.

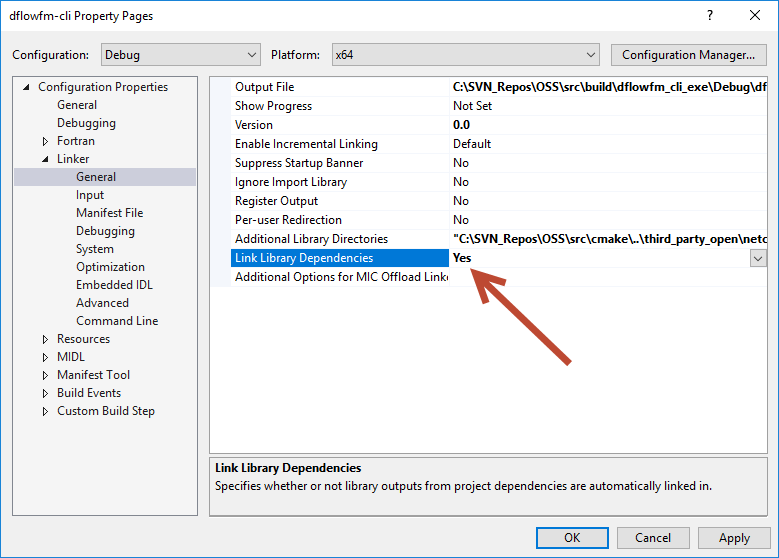


Figure 4.1Change the "Link Library Dependencies" to Yes

Table 4.1 Projects that require manual adjustments

|  |  |
| --- | --- |
| **Solution** | **Project** |
| dflow\_open.sln | dflowfm\_dll |
| dflowfm-cli |
| dimr\_open.sln | dimr |
| (dimr\_lib) (not entirely verified whether this influences compilation) |

The current state of the dflow-cli is that the debug configuration compiles and runs on Windows. However, the functionality is not verified on Unix and its post-build events need to be defined. Additionally, the solution transferred from the generation of tests.sln to dflow\_open.sln which broke the functionality of compiling a working tests.sln.

The dimr solution was verified to be successfully compiling on Windows. However, its functionality still needs verification.

# Next steps

1. In sln-file: group the projects. See the sln-file in the trunk.
2. In \*.cmake: why using "UNIX"/"WIN32" instead of "LNX64"/"WIN64"?
3. In \*.cmake: Replace "dflow" by "dflowfm"
4. Use CMake for the full OSS tree
5. Usage on TeamCity
6. Usage by external customers

# Recommendations

Proceed with using CMake, merge it into the trunk.

## Implementation improvements

* It is recommended to enforce a coding standard/scripting and naming standard for the CMakeLists.txt files. It is recommended to at least generalize:
  + The function naming in functions.cmake. It is also recommended to add comments in what the functions do, what arguments they accept and what values they return (if applicable)
  + Parameter naming, it might be wise to distinct parameter names which have a local scope and parameter names which have a more global scope
* Maintain separation of responsibilities in separate and dedicated files. There are currently the following mechanisms in separating responsibilities, namely:
  + Compiler flags: each compiler can define its own flags in their corresponding CMake file in the oss/src/cmake/compiler\_options directory
  + Scripts that are relevant for just Windows or Unix configurations. These are wrapped accordingly in the functions.cmake when being called
  + Solution configurations: each configuration can define which modules it needs to include in a dedicated configuration.cmake file in oss/src/cmake/compiler\_options
  + Module locations: Each component is categorized according to which directory they belong to (e.g. engines\_gpl, utils\_gpl etc)

To prevent too many responsibilities (and thus reducing its modularity and increasing maintenance) in a single file, it is recommended to split responsibilities as early as possible.

* Investigate the compilation flags that are present in the vfproj files generated by the prepare\_sln.py. Due to time constraints not all settings were incorporated in the CMake configuration files. These settings are mainly present, but not limited to, the items in the "Fortran/Run-time" tab.
* More investigation is required related to compilation flags, linker flags and external/internal dependencies.
* The includes of additional libraries should be improved. The main CMakeLists.txt at oss/src/cmake currently defines some hardcoded paths for the Windows configuration which lower level components use and alter the definitions to configure themselves (which makes them slightly more vulnerable for modifications in file structure, as the lower components have knowledge of the higher-level file hierarchy)
* More investigation is needed with respect to the dependencies (created by the CMakeLists.txt themselves or external dependencies) and linkers.
* A function should be implemented to set the file preprocessing flag (/fpp for Intel in Windows) automatically, instead of manually specifying them for each file.
* It is recommended to investigate how to automatically generate multiple configurations (e.g. tests.sln, dflow\_open.sln etc) in a single or multiple directory. The current recommendation is to run the CMake command for each configuration in a separate target directory.
* Generator expressions should be implemented for different configurations (e.g. release or debug) when they have different compilation settings. Generator expressions could potentially be used when dealing with different (Fortran) compilers.
* Instead of using the functions that operate on directory level, such as include\_directories, add\_dependencies, etc, it is recommended to use target specific functions such as target\_include\_directories, target\_link\_libraries, etc instead. This recommendation is especially relevant when a CMakeLists.txt defines more than one library. The target command explicitly expresses the dependencies to act on a specified target, rather than all the targets that are defined by the CMakeLists.txt.

## Cleanup

* Obsolete files should be removed from the repository as this only causes confusion when defining CMakeLists files.
* Once CMake is fully implemented, vfproj and the automake files are obsolete. These files should be removed.
* The CMakeLists.txt in oss/src/cmake should be cleaned up, placing logical elements in their own respective files.
* A function was implemented to gather the Fortran source files, independent of the case of the file extension (\*.f or \*.f90). This function should be implemented as soon as possible throughout the solution to prevent the overhead when defining new components.

## Process

* The implementation mainly focused on implementing features on Windows and then retrofitting the solution in Unix with adjustments when necessary. The definition of done could be adjusted to also incorporate a check on Unix instead of retrofitting them.
* Standardized QA procedures. Reviewing and testing should preferably incorporate the following aspects:
  + Although the individual components compile, it does not give a guarantee that they work. A test configuration or platform could perhaps be configured to verify that the individual components are compiled correctly and work as intended.
  + The tests verify that the CMake scripts generate the solutions, however it was not documented how this was precisely verified. Formal procedures should be established to keep changes and effects traceable. The procedure should at least verify the behaviour of CMake on a clean checkout, the associated output in the target directory and whether the component compiles.
  + Checking whether all the compiler settings are present in the files generated by the CMake configuration. Examples of these settings are the Fortran compilation and the linker settings.
  + Currently, the CMake solution is configured to generate two solutions, namely the solutions that correspond with dflow\_open.sln and the tests.sln. The test procedure should verify that CMake is able to correctly configure the solutions after changes are made.

# Resources

<https://cmake.org/cmake/help/v3.17/>

<https://cmake.org/cmake/help/latest/guide/tutorial/index.html>

<https://cmake.org/cmake/help/latest/manual/cmake-generator-expressions.7.html>

<https://floooh.github.io/2016/01/12/cmake-dependency-juggling.html>