# EC Module Interfacing

Version: 0.1

Author: Erwin Mulder erwin.mulder@vortech.nl, erwin.mulder@deltares.nl

Date: 18-1-2024

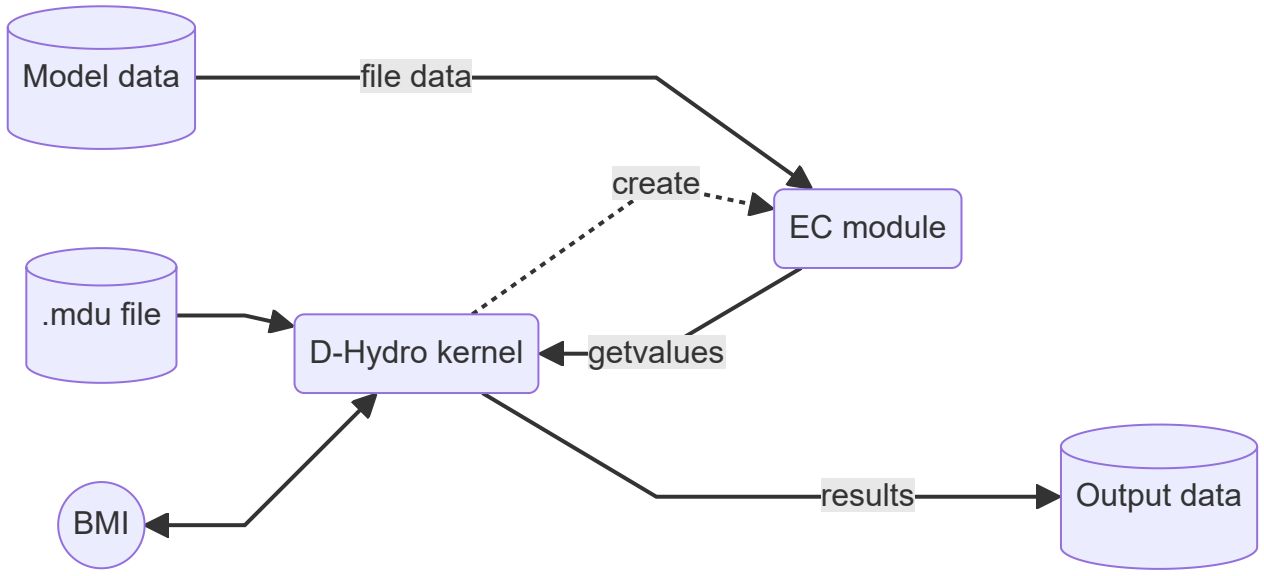
## Introduction

This document describes the preliminary redesign of the EC module API. This document is a living document, so it will be updated regularly.

## General overview

### Current setup

In the current (2024) implementation of D-HYDRO, the kernel is a central component.




**The good**: In a way this is good to have such a central role, because it allows a specific kernel to be implemented and developed further rather independently from other kernels.

**The bad**: The downside is that the kernel can also contain functionalities that are not what one would consider a core functionality. Ideally a kernel would implement just the code required to calculate model results from given input values, but it is easy to contaminate it with logistical concerns. This is especially true because in this current implementation we have different external interfaces that are directly linked to the kernel, which immediately imposes a low-key responsibility on the kernel to somehow translate between the required formats used in these interfaces. Especially the position of the kernel between the configuration interface (.mdu file) and the input module (ec-module) forces the kernel to pass on configuration items that in fact ec-module specific: What input files do we need and how should they be combined.

**The ugly**: Since the configuration for each kernel has it own (historical) implementation, *both* the configuration specifications *and* the ec-module interfacing has grown to become non-uniform in shape: Each kernel has been free to link up these two items in their own specific and unique ways and there is nothing in place to enforce uniformity and compliance.

Figuring out the logistics surrounding the input data is a cross cutting concern over many of the kernels and should ideally *not* be re-implemented all over the place. The goal should be to make the EC module the one-stop-shop for setting up the data pipeline. As mention before, failing to do so has lead to very poor kernel code that has to set up a lot of complicated logistics which are not at all specific to the core problem the kernel is made for. This is not limited to the data sources, but also other configuration items (like the geometry setup) seem to suffer from this.

## The EC Module

Where the current setup for data sources fails is the fact that the EC module has very little control over its own consistency: Kernels can poke away at the internal structures of the EC module and are allowed to call a host of functions at any time in any order.

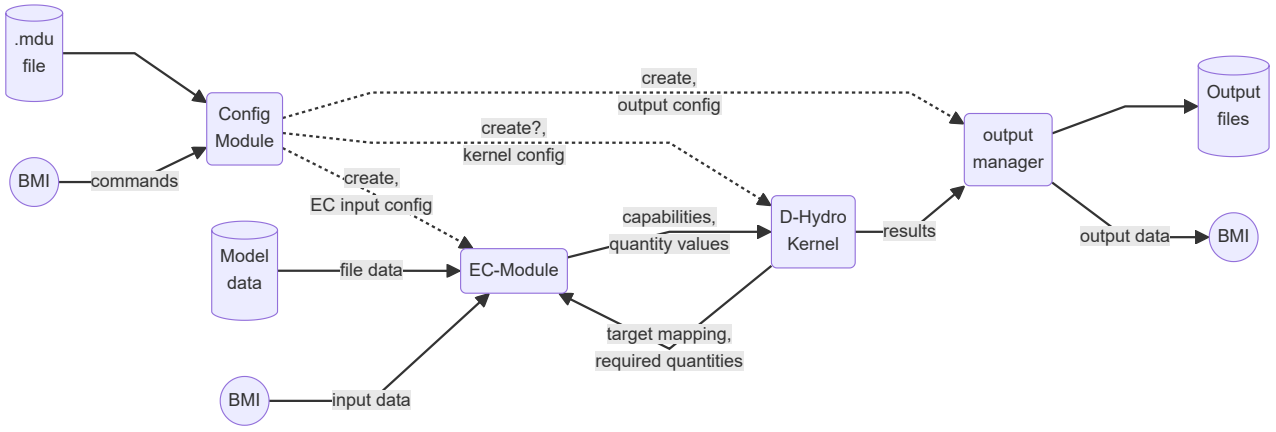
In order to make the EC module not break down under these conditions, there are currently a lot of checks (if-statements) to check for a lot of special cases caused by the external callers of each of these functions. This makes the code of those functions bloated and fragmented and in the end very hard to maintain and extend.

All these checks are rather unnecessary and mostly exist to function as a backstop for things that the kernel forgot to properly handle given the configuration items, which means they are in place to catch instances of bad programming and a waste of time to be checking at runtime.

## The aim

The aim is to remove responsibility from the kernel when it comes to data source logistics to the point that the only thing the kernel has to do is declare what kind of quantities and geometry it would like to consume.

All the other concerns should be handled by the EC-module and a (to be created) configuration module. This allows us to implement configuration logistics in a single place and redirect the part of the config that does not concern the kernel directly to the EC module without the kernel having to implement the handling of it at all.

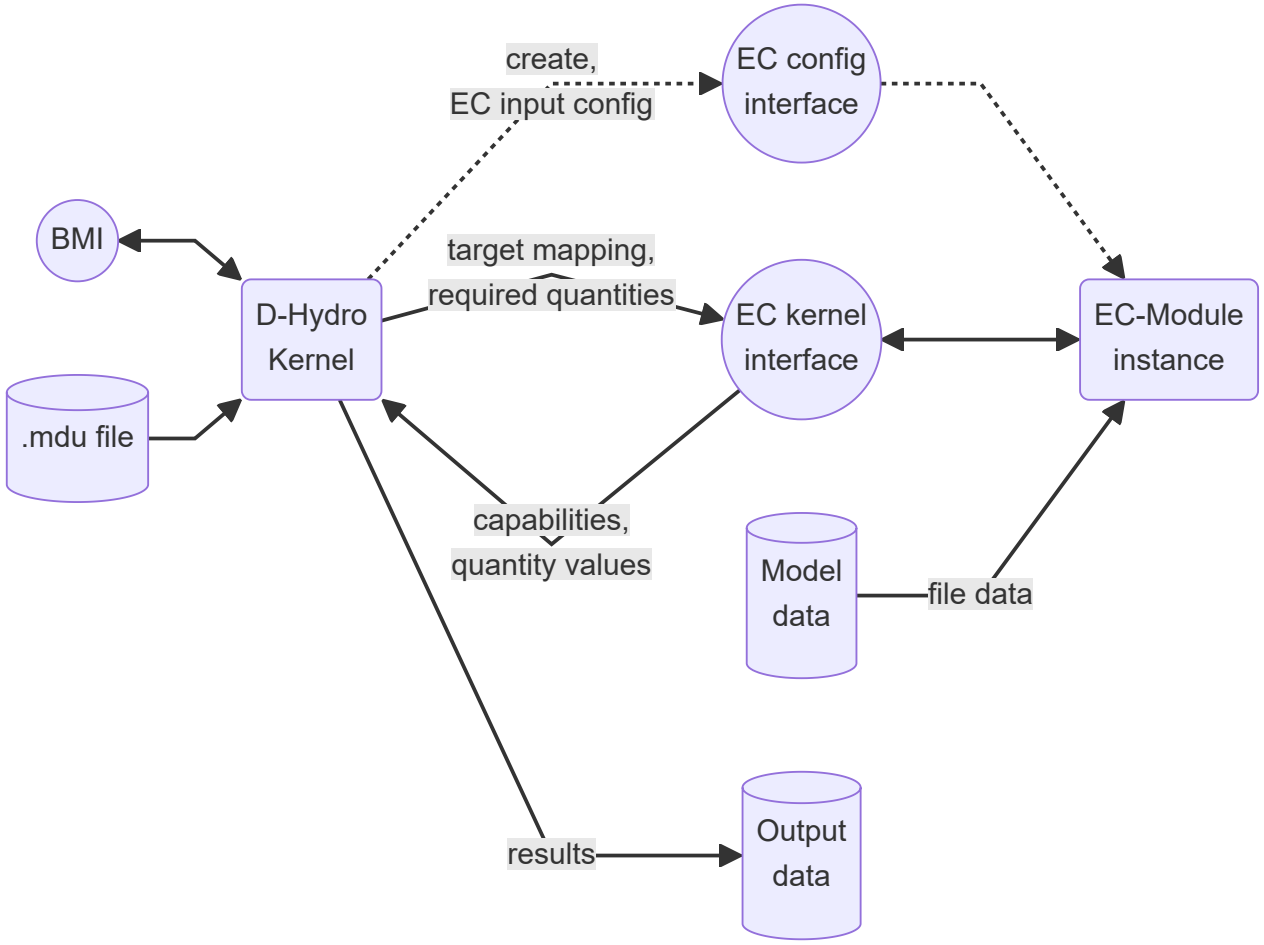



On top of creating simpler kernels and easier to maintain modules, it will also gain the benefit of more flexibility with respect to parallelization and orchestrating more elaborate calculations which would not be possible from the level of an individual kernel. An example of this is a parallel job where some kernels can consume the output of other kernels in a pipeline fashion.

Note that on top of a redesigned EC-module we can already identify a similar need to create an output manager. Knowledge on the output(format) or destination should not be a concern in the scope of the kernel and should be able to bypass the kernel entirely.

## The focus

In order to limit the scope of the initial work, we will *not* be focusing on creating the additional config, nor an output module. Instead, we will try and focus on the EC-module interfacing first. In this effort, we will try and make the distinction between the two different interfaces for '**config**' and '**kernel**' concerns. While initially both these interfaces will be called from the kernel, this will open an avenue to remove config related functionality from the kernel(s) and move it into a central config module in the future.




## Interactions

We identify 3 stages:

1. Config phase [config interface]
    a. source/file/BMI/?? + Q + operator(s) [ add, merge, avg, replace, … ]
2. Inquire phase [kernel interface]
    a. Which quantities Q are available? (maybe also allow a simple availability yes/no query for specific Q's)
    b. What is the spatial domain for specific Q?
    c. What is the time domain for a specific Q?
    d. Set a target domain (both in space/time) for a specific Q.
3. Calculate phase [kernel interface]
    a. getvalues(Q, domain, t, target)
    b. setvalues(BMI)

…

TODO: Add sequence diagrams.