Water quality and aquatic ecology modelling suite

D-Water Quality

Dutch Delta Systems

Open Processes Library

User Manual

Open Processes Library

User Manual

Released for:
Delft3D FM Suite 2023
D-HYDRO Suite 2023
SOBEK Suite 3.7
WAQ Suite 2023

Version: 0.99
Revision: 78577

25 April 2024

Open Processes Library, User Manual

Published and printed by:
Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone:	+31 88 335 82 73
e-mail:	Information
www:	Deltares

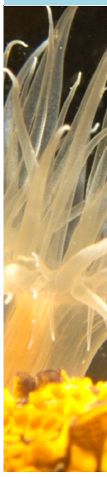
For sales contact:

telephone:	+31 88 335 81 88
e-mail:	Sales
www:	Sales & Support

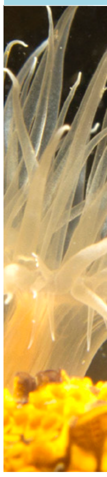
For support contact:

telephone:	+31 88 335 81 00
e-mail:	Support
www:	Sales & Support

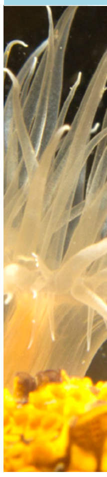
Contents



List of Tables



List of Figures



1 A guide to this manual

1.1 Introduction

The Open Processes Library is an extensible collection of subroutines used by the water quality module in Delft3D and SOBEK. To support this extensibility the tool that gives access to the contents of the processes library has been adjusted. It now essentially has two modes of operation:

- ◇ Selecting the substances and water quality processes for a particular computation
- ◇ Extending the library itself with new substances and processes

It is this latter mode that is the subject of the current document.

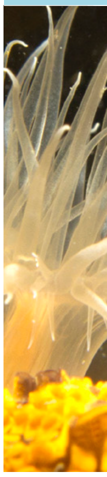
1.2 Manual version and revisions

This manual describes the functionality of the Open Processes Library Tool, version 2.99.00 and later versions with minor revisions.

1.3 Typographical conventions

Throughout this manual, the following conventions help you to distinguish between different styles of text used to describe various aspects of the graphical user interface.

Example	**Description**
Module **Project**	Title of a window or a sub-window are in given in **bold**. Sub-windows are displayed in the **Module** window and cannot be moved. Windows can be moved independently from the **Module** window, such as the **Visualisation Area** window.
Save	Item from a menu, title of a push button or the name of a user interface input field. Upon selecting this item (click or in some cases double click with the left mouse button on it) a related action will be executed; in most cases it will result in displaying some other (sub-)window. In case of an input field you are supposed to enter input data of the required format and in the required domain.
<\tutorial\wave\swan-curvi> <siu.mdw>	Directory names, filenames, and path names are expressed between angle brackets, <>. For Linux environments a forward slash (/) is used instead of the backward slash (\) for Windows environments.
"27 08 1999"	Data to be typed by you into the input fields are displayed between double quotes. Selections of menu items, option boxes etc. are described as such: for instance 'select *Save* and go to the next window'.



Example	Description
`delft3d-menu`	Commands to be typed by you are given in the font Courier New, 10 points.
➢	In this User manual, user actions are indicated with this arrow.
[m s^{-1}] [–]	Units are given between square brackets when used next to the formulae. Leaving them out might result in misinterpretation. Most units will be in SI notation. [m AD] stands for 'meter Above Datum', which denotes a level relative to the vertical reference system in the model.

2 Introduction to Open PLCT

2.1 Background

As important as grid generation for the hydrodynamicist

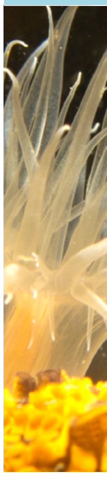
For hydrodynamic models the schematisation of a river basin, a lake, an estuary or a sea is of utmost importance. The model grid; the correct bottom elevation on that grid, external forcing etc. are the scope of work of the hydrodynamic modeller. The mathematical modelling software solves the model equations on the prescribed grid, with the prescribed external forcing. For water quality modelling the spatial resolution generally consists of the resolution of the underlying flow field as generated by the hydrodynamic model itself or of flows on integer multiples of those hydrodynamic grid cells. For water quality modelling there also is external forcing in the form of waste loads, meteorology, open boundary concentrations etc. Important scope of work for the water quality modeller however is the selection of substances and organisms to be included in the model and the nature of their mutual interactions. This selection of substances and their mutual interactions is perhaps as important for the water quality modeller as grid generation is for the hydrodynamic modeller. This documentation deals exactly with this activity of the water quality modeller.

Closed software

Many water quality models consist of a piece of software that computes the spatial concentration patterns and their evolution over time for a certain set of substances and their interactions. Users are able to insert waste load information and open boundary conditions and other forcing like meteorology. Furthermore, a number of reaction coefficients, playing a role in the implemented formulae of interactions, can be modified to calibrate the model behaviour to available measurement data. If, for a new application, the setting of substances and/or their interactions need to be changed, the computer code of the water quality model needs to be changed. If you use closed "as-is" software, you cannot change the computer code and you must ask the maintainer to do so. The only thing you can furthermore do is fiddle around a bit with the value of coefficients.

Open software

The first versions of Deltares water quality software, in use from the early '80s to the early '90s of the past century had the option of "user programmable water quality". You wrote a Fortran subroutine with water quality kinetics. That routine was linked with the main system and you were able in this way to change your water quality model equations at need. A number of sample kinetics was delivered to give you a 'flying start'. We used this procedure also for our own advisory work from 1983 to 1993. This resulted over time in a number of in-house programs or subroutines all doing partly the same for common substances and partly different things for substances and interactions that were specific for the problem at hand. This procedure was very prone to errors, especially because for a somewhat realistic water quality model the amount of code for the substances and their interactions could become substantial and also the amount of different projects and thus different versions of the 'user-defined code' was substantial at Deltares.



Software library approach

That is why Deltares developed in the early ’90s of the past century its processes library. This is an object oriented approach where each substance is an object. Each “arrow between substances” in the commonly used flow charts of water quality kinetics (called process) is an object as well. Finally each steering coefficient or steering function is also an object. The library includes all substances we could think of being relevant and includes all interactions between the substances we could think of being relevant. At the moment some 223 substances/organisms and fractions of substances (e.g. dissolved and particulate, and also different algal species) are contained for the standard WAQ module alone (there are also ECOlogical and CHEMistry extensions). The amount of interactions between the substances, the “arrows between the boxes” in common water quality graphs, stands now at some 425 processes for the WAQ module alone. The number of coefficients to steer the processes is several thousands. The 425 processes are computed by some 89 library software functions (Fortran subroutines). Because several processes for e.g. different heavy metals or different organic micro-pollutants essentially behave the same, although with different coefficients, they could be computed by the same library function, fed with different input. This is why the library functions are re-used on average 425/89 = 4.8 times. Although with careful design probably even a somewhat higher re-use rate could be obtained, a rate of almost 5 seems not bad at all. Because you could not add substances or processes Fortran source code to this library himself, Deltares felt committed to add all user requests free of charge and in due time during the past decade.

Success through multi-level user interface

This huge amount of substances and organisms and their processes can only be managed by a multi-level user interface. We made a six levels deep hyperlink user interface. In this user interface you select substances/organism from the library by switching them on. For all selected substances, you can subsequently switch on processes at will. Most of the coefficients and steering functions have default values. You select which of these many coefficients and functions you want to make editable by you rather than to apply these default values. This was a great success. Since the mid ’90s all Deltares advisory work on water quality was conducted with this process library and the many hundreds of users of the water quality module that use it world-wide have benefitted from this environment. The most important advantage was that it ensured error-less functioning.

Error-less because all code and administration on e.g.:

◇ which coefficients enter where in the system
◇ are the coefficients spatially constant or heterogeneous
◇ are the coefficients constant in time or a time function
◇ how do your results enter into graphs

was taken care of by the system. The only care for you (and that is also your core job, requiring your expertise and experience) is to:

◇ select the appropriate combinations of substances/organisms
◇ select the processes that should be active
◇ to select the appropriate setting of coefficients for the problem at hand.

You could save your set-up for one project and use it with slightly different selections for another project, without stumbling around with different versions of source code that may mutually conflict.

Why is there a need for an open processes library?

Why then at the first decade of this century must we set a next step? Both staff at engineering firms and institutes who are using our software, and our own staff want now and then new substances/organisms, processes and coefficients to be introduced in the library. Up to now they had to contact our software support group who do this fast.

This works fine but some users would prefer to obtain that ability in-house. This is especially true for the university community. They want to be able to program their own water quality kinetics. Prof. Joseph Lee of the old and respected place Hong Kong University once said that students should not only just apply commercial software, but also be able to create the computer code that represents their scientific topic of study. Since his students are not programming their own word processor before they are writing their thesis, we saw possibilities for the processes library software. It could act as a kind of word processor for the water quality processes that his students would write during their study. If we could deliver the library empty and allow the students to program the substances/organisms and processes themselves, they could focus on the water quality processes without having to deal with a lot of non-relevant computer programming and data structures and administration. Subsequently, students then could add their result to the library of the scientific institute as well. In this way their work would be saved for the institute that automatically obtains a software knowledge base that steadily increases like the institutes library of normal books. This is quite a different approach from the sometimes current practice that the computer program of the student is archived as a dedicated piece of software that becomes obsolete after the student finalises his/her study.

Why is the “Open Processes Library” more than just a computer program

The *Open Processes Library* can best be compared with a real library with books on the shelves, one book for each substance, one book for each process, one book for each steering coefficient or function. Thousands of books that are already contained in the existing processes library produced at Deltares. If a new substance, a new process or a new steering function is needed, you only need to make this ‘new book’, put it on the shelf and insert its name and reference in the online catalogue. From that time on the existing library is expanded with your substance, process or steering function. Whether you (or others) will use this functionality in subsequent studies will depend on the study at hand. Sometimes you will, sometimes you won’t. The feature is however contained in the library and you can select it (borrow it) any time you want through the multi-level user interface, like all other already existing features. No software errors are possible once you have added your book to the library. If your *book* says that *temperature* is required for the process in the book, the system will take care that *temperature* is there at the right place and in the right form, without any coding from your side. If your *book* produces a result in the form of a flux from one organism to another organism, the system takes care that the flux actually operates on the organisms, without any further coding from your side.

This manual describes the functioning of the processes library itself to start with a good understanding. It then shows with a tutorial how to add your own substances/organisms and processes to an initially empty processes library. Then a number of advanced programming topics is explained. Finally, in tutorial style, new European Community so called Priority Substances for the Water Framework Directive will be added to the existing Deltares pre-WFD version of the library.

With this functionality we feel that we can face the next 10 years of water quality modelling with confidence. What will come after that period is still to be guessed. The specialist on artificial intelligence and formula generation Vladan Babovic already expects that probably by that time the process formulas in the library could be generated by formula generators that are using intelligent inferences on the data.

Teamwork

The Deltares team that works on the Processes Library consists of

◇ Jos van Gils who developed the library approach in the early ‘90s of the past century and who was the teamleader of the further development of the library since.
◇ Pascal Boderie who developed the first comprehensive set of process routines.
◇ Jan van Beek, who developed the whole data structure and administrative programs calling the process routines, is responsible for the whole core simulation interface with the process routines of which he also developed quite a number himself.
◇ Arjen Markus, Stef Hummel and Antoon Koster were involved in its user interface that runs both on MS-Windows and Linux.
◇ Johannes Smits recently reviewed the whole set of processes and coefficients and expanded the library approach to a layered bottom underneath the water phase.
◇ Leo Postma saw that the others did a splendid job, was responsible for the ‘opening up’ of the process library for external users and wrote this manual.

2.2 Concepts

The processes library is an extensible library of water quality components. The following water quality components are distinguished:

◇ *Substances*
These are all constituents where a concentration or density can be computed for by the water quality model. The concentration/density is computed by the time integration of the reaction equation ($dC/dt = \ldots$) together with the advection diffusion equation (ADE) if the substance is carried with the water or without the ADE for substances that are not carried with the water (e.g. those that lay on the bottom as particulates). Also organisms (bacteria, algae, filter feeders etc.) are called substances. Also phenomena (e.g. BOD, salinity, pH) are called substances although they are not in strict chemical or biological sense. Because the list of available substances in the user interface could become long, they are divided into substance groups. This is just for the ease of reference and has no specific further meaning in the computation.
◇ *Processes*
These are the water quality kinetics. It is the right-hand side of $dC/dt = \ldots$ In flow charts of water quality models processes generally represent the *arrows* or fluxes between the substances. Each process is computed by a tiny computer program, a Fortran subroutine. Each process has its input coefficients and produces one or more *arrows* in the flow chart of water quality kinetics. The arrow itself is called *Flux*, so the process produces fluxes.
The process may also produce output variables. These variables may be used as input for other processes. If e.g. mortality of bacteria depends on *salinity*, but the modelled substance is *chloride*, then a process may take chloride and other ions as input and compute salinity as output that is used again as input for the mortality of bacteria.
Output variables may also be used for analysis purposes. If you e.g. let the process deliver some intermediate results (like the nutrient limitation factor for algal growth) as output, then it can be displayed in graphs and statistics just like a normal substance. Several processes can use the same Fortran subroutine for their computation but with different input, output and flux variables. If the resuspension and sedimentation of suspended sediment has to be computed for 3 size fractions of suspended sediments in an identical way but with different coefficients, they can use the same Fortran subroutine. The same holds for all those heavy metals that partition in the same way but with different partitioning coefficients or isotherms.
◇ *Items*

These are the input variables for the processes, but also the output variables (that may be input to other processes). Also the fluxes between substances are ‘items’, but they are special in the sense that they contain a description of the substances that they affect. Also the concentrations can appear as items, if they are input to a process. A process may behave e.g. temperature dependent. temperature can be a *substance* if it is modelled as a state variable of the model. It can also act as ordinary input item if it is prescribed from outside by you. The same items can be input to several different processes (*temperature* for instance is an input parameter for many processes). Input items for a process can have a default value. Per process, however, you can specify whether you want the system to use this default value or not. If an item is modelled, the system generally automatically takes the model result as input for all processes that require the item. You can overrule this by specifying the item externally.
Items can be:

- constant in time and for the whole model area
- a time function that is uniform over the whole model area
- a spatially heterogeneous factor that is constant in time
- a spatially and temporally varying item.

A model result generally is in the last form. You do not need to care. You can use different approaches for the same item in different studies:

- temperature as a prescribed constant for a short term simulation
- as a prescribed time function for a one year simulation of a coastal zone or river
- temperature modelled with input from solar radiation, cloudiness and humidity.

The system automatically resolves all references.

◇ *Fluxes*
A special class of items consists of the *fluxes* between substances. They have an associated *stoichiometry*. That is a table with the name of the substances where they act upon and the factor with which they act. The *nitrification* flux e.g. acts with a factor -1.0 on ammoniacal nitrogen and with a factor +1.0 on oxidised nitrogen, but it also acts with a factor of -4.57 on oxygen. This 4.57 represents the amount of oxygen in gram needed to nitrify 1 gram of ammoniacal nitrogen to oxidized nitrogen.

What services does the processes library offer you?

◇ *Make the model as large as you specify*
You can switch on substances, switch on processes that influence the substances and you can define which items you want to be editable through the user interface and where a default value would do for you. You furthermore can specify which items you want to be added to the output set (the modelled substances are by default members of the output set, but you can add any other intermediate computational result). Once you have finished, the model is exactly as large as you have specified, so there is no overhead by the non-used part of the processes library.

◇ *Administrate all data flows*
The system takes care that it acquires all input for the processes, that the processes are invoked and produce output and fluxes and that the fluxes act between the substances. You cannot influence this and that is why you also cannot make errors in this respect. The system automatically takes your modelled substance *temperature* if you switched it on. The system also automatically switches on all processes that produce output that is required as input for the processes that you have switched on yourself manually. At set-up time the system prompts you already to take action if not all required input for a simulation is resolved yet.

◇ *Intelligent data administration*
The system identifies for all items that you have specified to be editable by you, whether

they are provided as a constant, as a single time function for the whole area, as a spatially distributed constant or as a spatially distributed time function.

◇ *Archive of the set-up*
The total configuration of your water quality model consists of the choices you made. This total configuration is saved for archive purposes. This means that you can always reproduce your selection from a previous study by importing its setting. Your new study may consist of a slight modification of this setting. You save the new setting under a different name and both settings remain available for future use.

How do you turn your saved set-up into an actual water quality model?

The saved set-up is imported either in the user interface of the Deltares 1D-2D system SOBEK or in the Delft3D user interface. In this user interface you will see that all items appear that you have specified as being 'user-editable'. In this SOBEK or Delft3D user interface you define whether you will specify this input as a constant that you enter into the corresponding field (like is done with parameters for e.g. a sensitivity test).

You may also specify it as a time function (like solar radiation above your model area throughout the year) or as a spatially distributed function (like the bottom-organic matter content). Both SOBEK and Delft3D have tools to conveniently make such a spatially distributed function from e.g. measured data or just by specifying them yourself in a number of points. Finally, you may specify it as a time-series of spatially distributed fields. In a number of cases the latter can come from the hydrodynamic model result (water depth, salinity, temperature, bottom shear stress, water velocity etc. if switched on there). In those cases the link with the hydrodynamic model result is made automatically.

In the user interface you will also see the selected substances as potential output items. Also all those process input and output items that you clicked on as being available for output appear as candidate for output in the user interface. In the user interface you specify which of all these items you want actually be saved into the output file and thus be available for graphical post-processing. It is very convenient to have spatial graphs and/or animations also for the *nutrient limitation of algal growth* or the *local bottom shear stress* besides that of your modelled substances.

Finally you can specify online statistics, like averages (moving and total), exceedence frequencies and percentiles of all these variables. Those statistics also appear in normal output files that can be processed in a standard way by the graphical post-processing tools.

Different levels of users

The afore-mentioned approach distinguishes the following different levels of use:

◇ *The model end-user*
Through the SOBEK or Delft3D user interface scenario computations and corresponding graphics are made by adjusting e.g. wasteloads, boundaries or coefficients for an existing model through the user interface. This can be done by a technician.
◇ *The modeller*
Using the SOBEK or Delft3D productivity tools, a model is set up for a certain area. An existing saved file with a selected set of substances and processes is imported.
◇ *The model developer*
Using the PLCT, Processes Library Configuration Tool, a selection of existing substances processes and items from the Library is made and saved in a file.
◇ *The water quality/ecological modelling scientist*
Using the Open Processes Library and a Fortran compiler, substances, processes and/or

items are added to the existing library.

The last level is the only level that requires computer programming effort and may thus be conducted by a scientist together with a computer programmer. The procedures for this last level are explained further in Chapter 6. Please be aware that this is for each substance/process a ‘one of’ activity comparable with the purchase of new books for the collection by a real librarian. The borrowing of books from the library is a one level higher activity and the reading of the borrowed books is done by the top two levels.

It is of course up to the procedures within your organisation whether you want to distinguish all of these levels by assigning them to different persons or not. If you do all 4 levels yourself, you will nevertheless recognize these different levels as separate actions to be taken during your work.

3 Getting started

3.1 Starting Delft3D

The Delft3D suite is a range of modules that can be run independently of one another. The main menu of Delft3D provides access to each of the main modules. Delft3D-MENU can be started either:

◇ In Microsoft®Windows, select Delft3D in the Programs Menu or click on the Delft3D icon on the desk-top
◇ In Linux, type `delft3d-menu` on the command line.

The main window of Delft3D-MENU is shown in Figure 3.1.

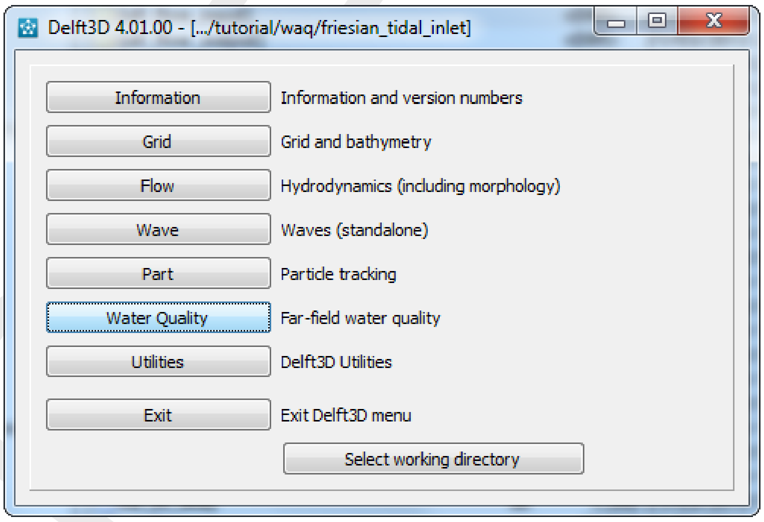


Figure 3.1: *Main window of Delft3D-MENU*

Whether or not you may use specific Delft3D modules and features depends on the license file you have (delft3d_<hostid>.lic in your DS_Flex directory).

There are two ways to exit the Delft3D-MENU:

◇ Click *Exit*.
◇ Click the 'close' button (×) in the top-right corner of the menu bar.

3.2 Getting into the OpenPLCT

Before continuing with any of the selections, you must select the directory in which you are going to prepare a scenario:

◇ Click the *Select working directory* button, see Figure 3.2 for the window displayed.

A standard file selection window is opened and you can navigate to the required directory.

◇ Browse to the desired directory, and enter this working directory.
◇ Confirm your selection by clicking *OK*.

Remark:

◇ In case you want to create a new directory, click and specify a name. Enter the new

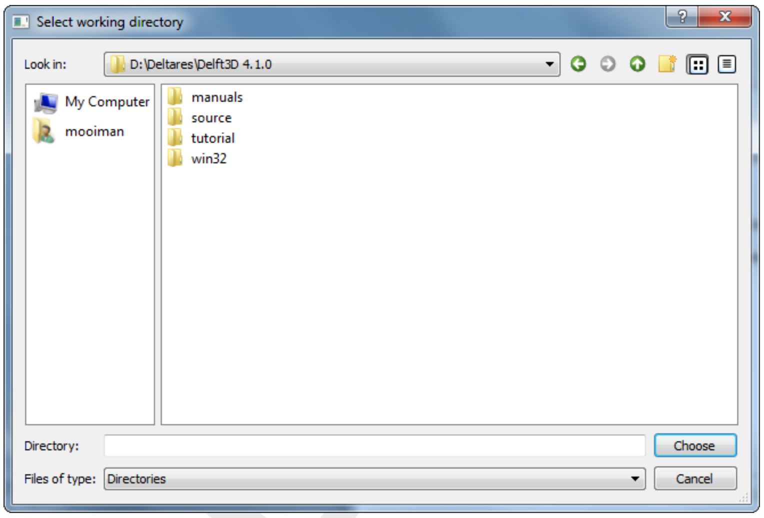


***Figure 3.2: Select working directory** window to set the directory to for saving files*

directory and click *OK* to confirm your selection.

Now we are back in the main water quality menu and we can define new substances and processes.

To start the OpenPLCT for WAQ (general water quality) press first th button *Water Quality* (Figure 3.1, followed by *General* and *Tools* and *OpenPLCT* (now you get a window similar to Figure 3.3) and *Define Input*. The OpenPLCT will start, showing three different windows, see Figure 3.4.

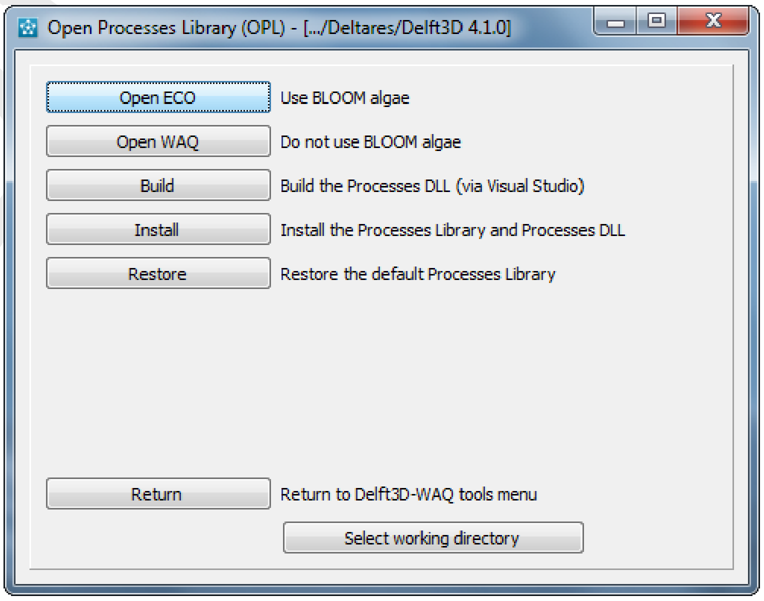
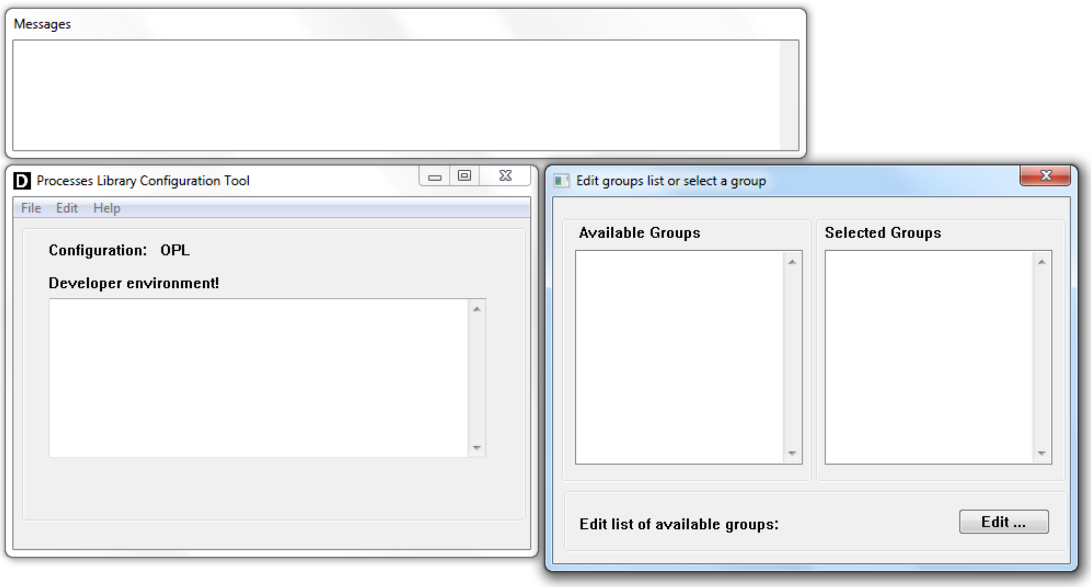

Open Processes Library (OPL) - [.../Deltares/Delft3D 4.1.0]
Open ECO
Use BLOOM algae
Open WAQ
Do not use BLOOM algae
Build
Build the Processes DLL (via Visual Studio)
Install
Install the Processes Library and Processes DLL
Restore
Restore the default Processes Library
Return
Return to Delft3D-WAQ tools menu
Select working directory

***Figure 3.3: Open Processes Library (OPL)** window to start the OpenPLCT*

To start the OpenPLCT for WAQ (simple algae) a similar path need to be followed.


Messages
Processes Library Configuration Tool
File Edit Help
Edit groups list or select a group
Available Groups
Selected Groups
Edit ...

Configuration: OPL

Developer environment!

Edit list of available groups:

Figure 3.4: *Open Processes Library Configuration Tool start windows (empty database)*

3.3 Exiting OpenPLCT

To exit the OpenPLCT:

◇ Click *File* $\longrightarrow$ *Exit.*

You will be back in the **Open Processes Library (OPL)** selection window, see Figure 3.3.

Now, ignore the other options and just:

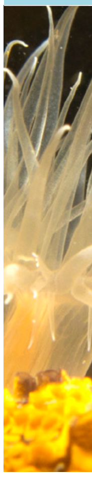
◇ Click several times *Return* to return to the main window of Delft3D-MENU, see Figure 3.1.
◇ Click *Exit.*

The window is closed and the control is returned to the desktop or the command line.

This Getting Started session will have given you the general idea of how to access the Open Processes Library Configuration Tool (OpenPLCT).

4 Tutorial

The *Open Processes Library* is the developer environment of the existing Deltares Processes Library Configuration Tool PLCT. Every licensed user who is using an existing processes library (like the library that is delivered together with the Deltares software), uses the Processes Library Configuration Tool (PLCT) to select substances for modelling, to select processes for the interactions and to select parameters for user input and user output. It is easier to understand the developer environment if you already know the functioning of this PLCT. It is however not mandatory. You can use this tutorial also as a novice user.



In the tutorial two things will be done step by step to illustrate the functioning:

◇ starting with a fully empty processes library we will set up a complete Streeter-Phelps BOD set of processes for the library
◇ starting with the full WAQ license from Deltares, we will add an additional substance that is considered a priority substance for the European Water Framework Directive and that was not contained yet in the library at the time of writing the WAQ manual.

4.1 How to start the developer environment

The developer environment is incorporated in the Processes Library Configuration Tool. You start the PLCT from the SOBEK or Delft3D user interface using the corresponding tool button. In developer mode, the PLCT takes several extra parameters in a Windows INI-file:

◇ *configuration*
The configuration parameter gives the configuration where you start the system with. Common configurations are:
   - ECO - the ecology configuration.
   - WAQ - the general water quality configuration.

◇ *application_type*
Indicating in which environment the PLCT is running, Delft3D or SOBEK
◇ *nefis_data_file and nefis_definition_file*
The configuration tables of the processes library are contained in a database file. These used to be two files, <proc_def.dat> and <proc_def.def>, but a more recent version of the underlying library combines these two files into one. If these parameters are missing the PLCT will prompt you to select them.
If you have an empty processes library, then these files are simply missing. At the prompt you just push the *Cancel* button and the PLCT starts with an empty processes library.
Once you have edited things and you save your work in a processes database, your own processes library, then you can specify this file at the next startup and you can continue where you have saved the work in the previous session. The file you save is created as a single file.
◇ *defaulthome*
This is the directory containing any selected substances, processes and/or items with normal use of the PLCT. In general you do not need this directory when you are adding new items to the library. Usually you will start the PLCT from the SOBEK or Delft3D user interface at normal use, and then it is set by the respective user interfaces.
Your configuration should match the license file that you have got. If started with an empty processes library, your configuration will become USER.

Example:

```
[General]
```

```
configuration     = ECO
application_type = Delft3D
mode              = Developer

[System]
nefis_data_file=D:\Deltares\Delft3D 4.1.0\win32\waq\default\proc_def.dat
nefis_definition_file=D:\Deltares\Delft3D 4.1.0\win32\waq\default\proc_def.def
algaedb=D:\Deltares\Delft3D 4.1.0\win32\waq\default\bloom.spe
defaulthome=D:\Deltares\Delft3D 4.1.0\win32\waq
```

4.2 Create substance groups

After start-up the PLCT consists of three windows (Figure 4.1):

1	**Messages**	Window displays messages when errors occur, mostly related to missing information.
2	**Processes Library Configuration Tool**	File managing window for opening and saving files and exiting the PLCT. The window also gives an overview of the substances selected so far.
3	**Select Groups**	Overview of available substances groups, and entry point to select substances. Also, entry point to Extra processes

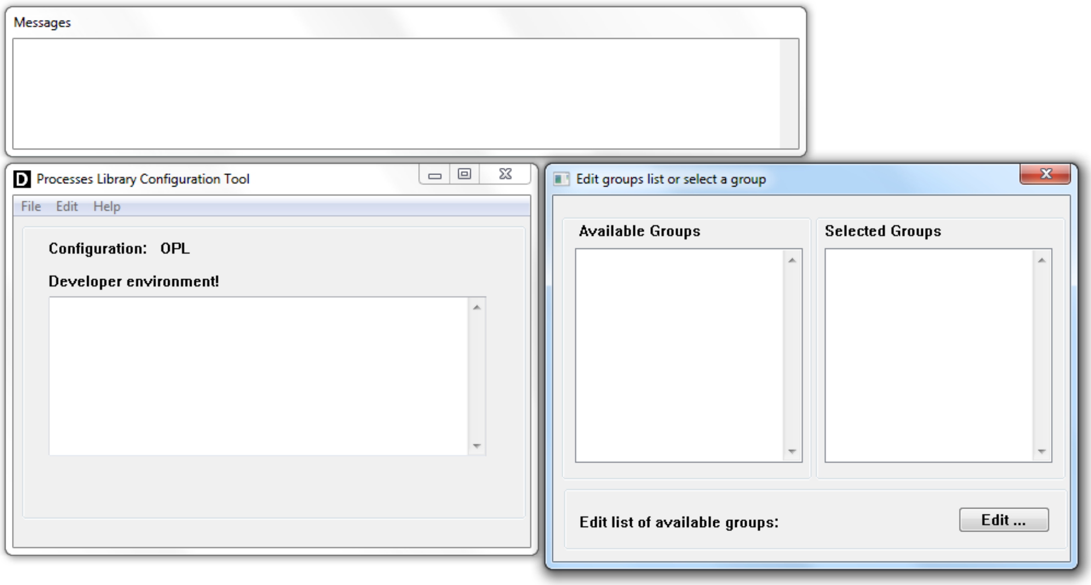


Figure 4.1: *First PLCT screens with an empty Processes Library*

The menu bar has the following menu bar options (**Processes Library Configuration Tool** window):

File	*New*	Start a new scenario — clear all input.
	Open...	Open an existing PLCT <∗.0> file.
	Save	Save the current PLCT <∗.0> file using the previously given name, or the default name <substate.0>. A substance file <∗.sub> is saved simultaneously.
	Save as...	Save the current PLCT <∗.0> file and specify its filename. A substance file <∗.sub> is saved simultaneously.
	Exit	Exit the PLCT.

Edit	*Merge*	Merge the current data with an existing processes library, the new processes library will be available after saving this session.
	Tables	Export the data to several <*.cvs> files on the current directory (Expets).
Help	*About*	When clicking *About* a window will display the current version number of PLCT

If you press the *Edit* button a screen opens and you can enter the name of a substance group. We will enter the name "BOD-DO" (Figure 4.2) , because our first additions to the library will consist of the classic BOD-DO kinetics by Streeter and Phelps (1925). We press the *Add* and *Save* buttons to create the group.

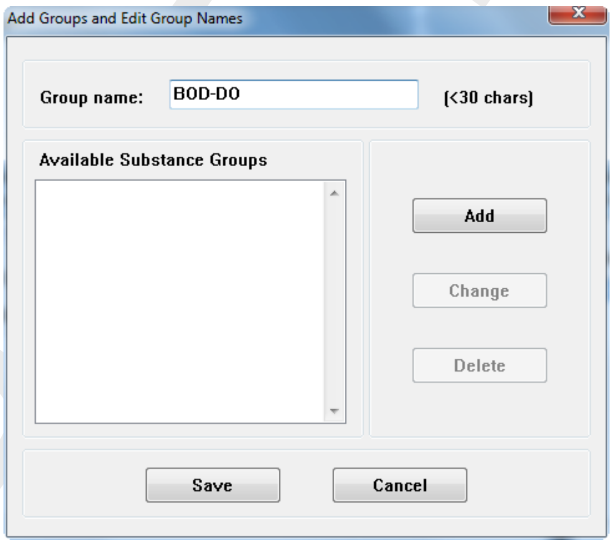


Figure 4.2: ***Add groups and Edit Group names*** *window after entering 'BOD-DO', before pressing* Add*.*

The *Change* button in Figure 4.2 will just change the name of the selected group to the string that you entered in the edit-field.

The *Delete* button only deletes the group if it does not contain any substances any more. So you need to move substances to other groups or delete substances first to make a group empty, before you can delete the group itself. This procedure for deleting a group is exemplary for all such actions in the developer environment: this way you avoid accidentally losing data.

Once you press the *Save* button, the modifications are irreversibly updated into your session. To save your modifications to disk, you have to select the *Files* $\rightarrow$ *Save As. . .* menu item of the main window.

Remark:

◇ It is good practice to save your work some now and again. You can then always start again from a previous saved state if something goes wrong. Then you can exit the software and restart it with the intermediately saved database as input. There is no facility equivalent to an *undo* button in the developer environment of the PLCT.

After you have added and saved the 'BOD-DO' group, it is inserted in the **Available Groups** canvas of the **Edit groups list or select a group** window and you may select the group by clicking it. Your screen will look like Figure 4.3.

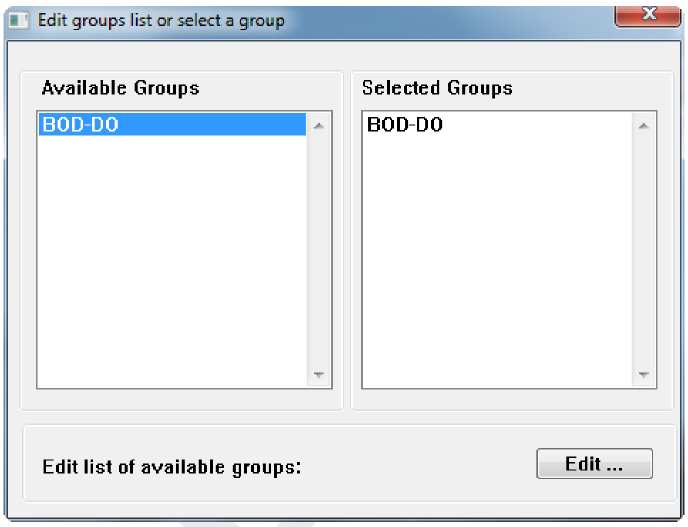

Edit groups list or select a group
Available Groups
BOD-DO
Selected Groups
BOD-DO
Edit list of available groups:
Edit ...

Figure 4.3: *The newly created group 'BOD-DO' is selected*

4.3 Create substances

If you select the 'BOD-DO' group also in the right hand side list of the **Edit groups list or select a group** window (Figure 4.3) then the **Edit substances list or select a substance** window of Figure 4.4 opens. We want to add substances to the group so we press the *Edit* button to edit the substances list. We now see the window of Figure 4.5. Because you use this window to display, add or edit substances in a substances group, the name of the group is displayed in the window title, "BOD-DO" in our case.

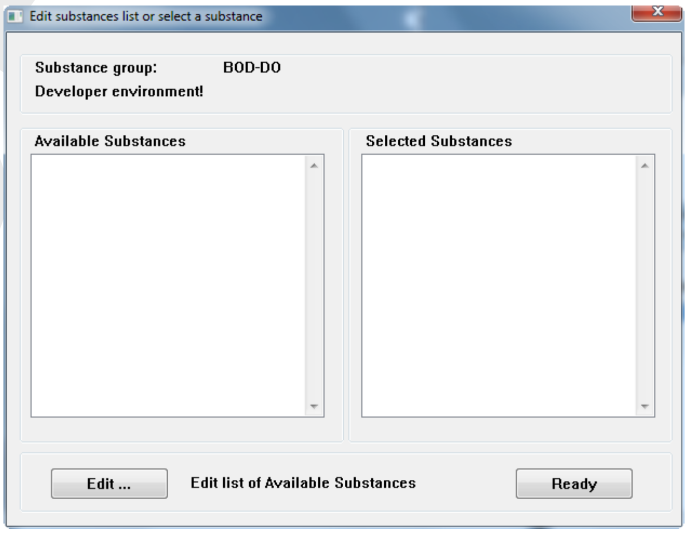


Figure 4.4: Edit substances list or select a substance *window before our new group*

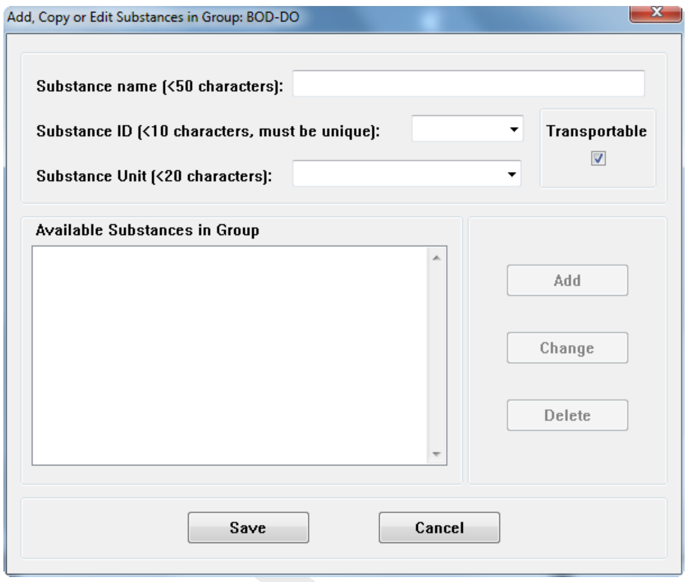

Add, Copy or Edit Substances in Group: BOD-DO
Substance name (<50 characters):
Substance ID (<10 characters, must be unique):
Substance Unit (<20 characters):
Transportable
Available Substances in Group
Add
Change
Delete
Save
Cancel

***Figure 4.5: Add, Copy or Edit Substances in Group** window*

The window contains a number of input fields:

◇ The *substance name*, which can extend to 50 characters and is meant for a meaningful description. It does not need to be unique, but it is easier if it is. It is displayed in many windows.
◇ The unique *substance ID*, by which each substance is identified, is limited to 10 characters only and gives little opportunity for a meaningful description. Nevertheless you should try to make the ID as explanatory as possible. The end-user will see the ID for instance as a name in the output files.
◇ The *unit* is part of the explanatory information. The system does nothing do with the unit. It is neither parsed nor automatically converted, although features of that type may come in future. Nevertheless it is important to think over the units carefully.

The WAQ system uses SI-units like meter [m] for lengths and distances. If you want to have your substances represented by the [mg/l] unit, then you need to realise that this is equivalent with [g/m^3] and you have to release waste loads of the substance in [g/*unit of time*] to obtain [mg/l] as concentration unit. This seems trivial, but the concentrations of heavy metals and organic micropollutants are generally expressed in [μg/l] or [mg/m^3]. So here the waste load needs to be in [mg/*unit of time*].

More complicated is the common unit for faecal bacteria: MPN/100 ml. If you release MPN per unit of time, then you get MPN/m^3 and that is 10^4 MPN/100 ml. This means that you have to divide your waste load by 10^4 to get the required concentration unit as output.

Errors are easily made in this respect, so a careful check of your settings is advised. As an exercise you may try to identify how you have to scale your thermal waste load given in megawatt to produce the substance ‘excess-temperature’ in °C in the water[1].

For your convenience the *unit* field is a drop-down box, containing all units that have previously been defined. This allows you to select one from previously entered substances to avoid

[1] 1 megawatt = 1 MW = 10^6 J/s. 1 calorie heats 1 cm^3 of water 1°C. 1 calorie is 4.19 J. This means that 1 MW = 10^6/4.19 [cal/s]. or 1/4.19 [°C m^3/s]. So you divide the number of megawatts by 4.19 to get the load per second that corresponds with the state variable “temperature” expressed in °C.

typographical differences such as differences in case and/or additional spaces.

The 10 character *unit* field is in some applications added to the 10 character *ID* field of a substance. For those situations it is convenient to distinguish the unit format from the ID by placing the unit between brackets, e.g. "[g/m3]".

Remark:

◇ The check-box *Transportable* may seem strange. If you realise however that material as settled on the bottom of your computational element can also be distinguished as a substance, you realise that not all substances are transported by the water. Transportable is the default state. If a substance is attached to the bottom or walls, you uncheck this box and the substance is excluded from the advection-diffusion equation solver, but the water quality processes still act.

Once you start typing a substance name, ID and unit, the *Add* button becomes active. You add the substance to the list by the *Add* button.

If you select a substance in the list, the fields for name, ID and unit are filled in with the information of that substance. Now also the *Delete* button becomes active. The *Delete* button deletes a substance. To do so however, all processes associated with that substance should also be removed. Even then the substance is not fully removed from the system, but it is converted into an input-item. This input-item can later on be deleted according to the rules for those items. This very conservative approach is used to avoid accidental loss of data.

You will have noticed that the title of the *Add* button changed to *Copy*. That reflects the fact that you cannot add a substance with the same specification as an existing one, but you can copy it under a different name. This copy feature is discussed in more detail in Section 5 of this manual.

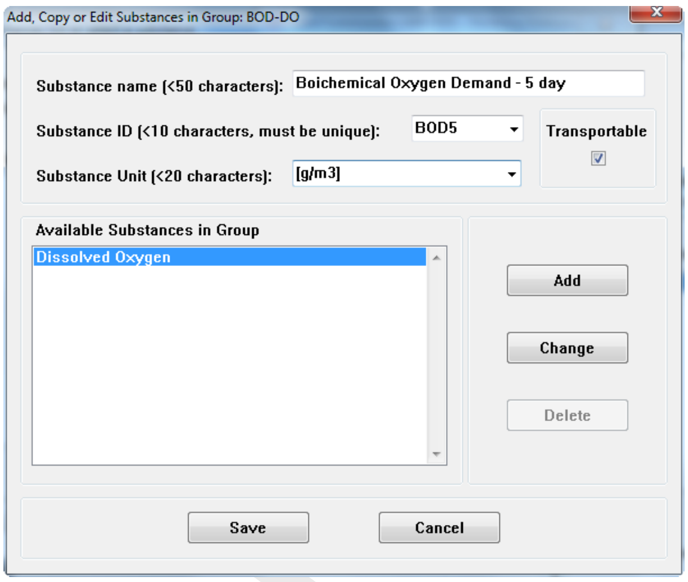
If you change the information in name, ID or unit field, the buttons will be labelled *Add* and *Change*, reflecting the fact that you can add a substance with the modified name, ID and unit, or change the selected substance towards the new values in the fields.

A special situation occurs if you specify a substance-ID using the drop-down list. You can then select a substance that already exists in another group. If you now press the *Add* button, you are asked whether you want to move the substance from the other group to this group. If you answer "yes", the substance is moved from the other group to this group. You only need to specify the ID for that action. All ID's are case-insensitive, so Oxygen is the same substance as OXYGEN. It is however good policy to adopt a more or less uniform typographical convention.

We will insert the substances "Dissolved Oxygen" with ID "DO" and unit "[g/l]" and "Biochemical Oxygen Demand - 5 day" with ID "BOD5" and unit "[g/m3]" in this new group. Before the *Add* button is pressed for the last substance, the window may look like Figure 4.6.

Note: that at this point the program does not know whether you want to change the selected Dissolved Oxygen substance ID and rename it to BOD5, or whether you want to add BOD5.

Press the *Add* button. A new substance is added and becomes selected in its turn. Once you press the *Save* button, the substance will appear in the list of the **Edit Substances list or select a substance** window (See Figure 4.7).


Add, Copy or Edit Substances in Group: BOD-DO

Substance name (<50 characters): Boichemical Oxygen Demand - 5 day

Substance ID (<10 characters, must be unique): BOD5

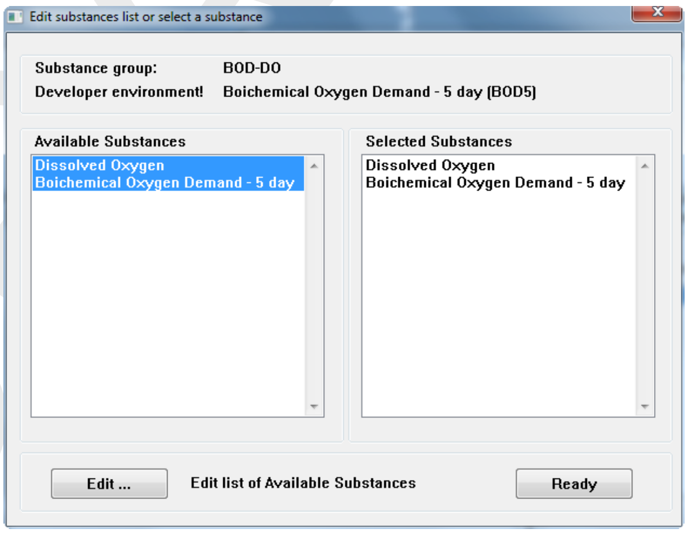
Transportable ✓

Substance Unit (<20 characters): [g/m3]

Available Substances in Group

Dissolved Oxygen

__Figure 4.6: Add, Copy or Edit Substances in Group__ window before BOD5 is added


Edit substances list or select a substance

Substance group: BOD-DO

Developer environment! Boichemical Oxygen Demand - 5 day (BOD5)

Available Substances

Dissolved Oxygen
Boichemical Oxygen Demand - 5 day

Selected Substances

Dissolved Oxygen
Boichemical Oxygen Demand - 5 day

__Figure 4.7: Edit Substances list or select a substance__ window with the new substances selected.

4.4 Create processes

Now we have substances and we need to specify interactions between the substances. The interactions are made by processes. In the **Edit Substances list or select a substance** window we first click *Biochemical Oxygen Demand - 5 day* in the list at the right hand side. Then the (empty) **Select Processes** window (Figure 4.8) appears. If we click the *Add/Edit. . .* button there, the **Add new and Edit existing Processes** window (Figure 4.9) appears.

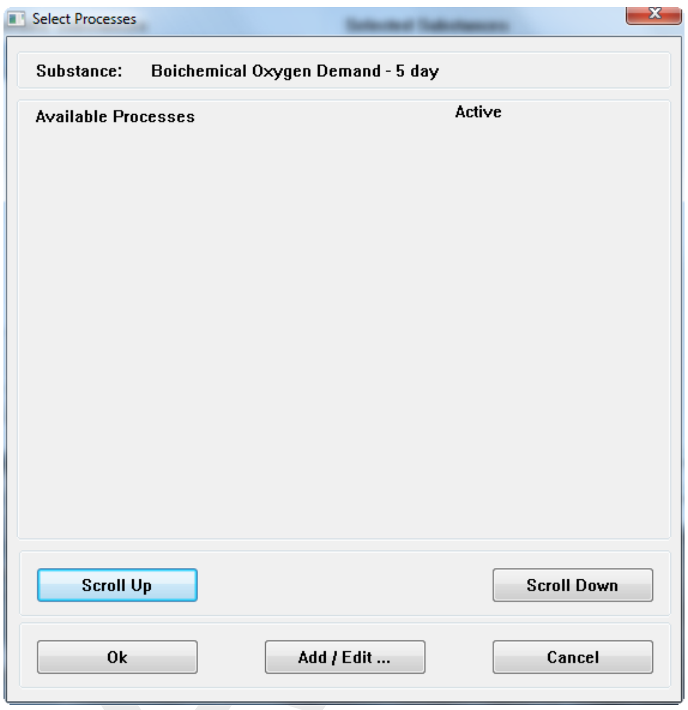


Figure 4.8: *The empty **Select Processes** window for "Biochemical Oxygen Demand - 5 day"*

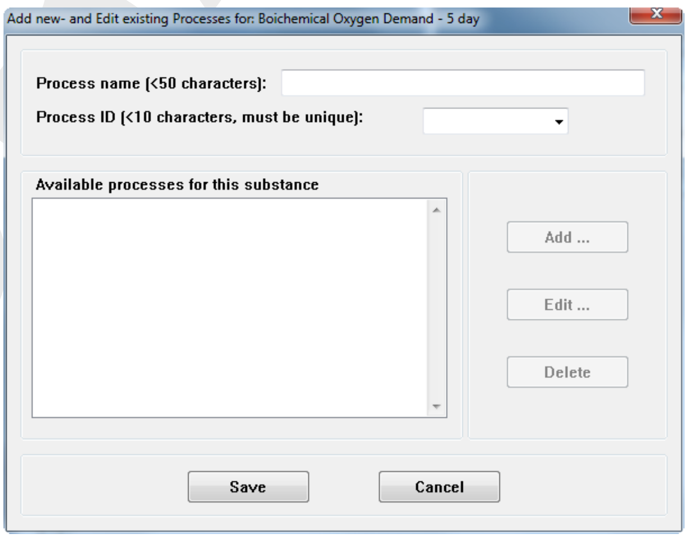
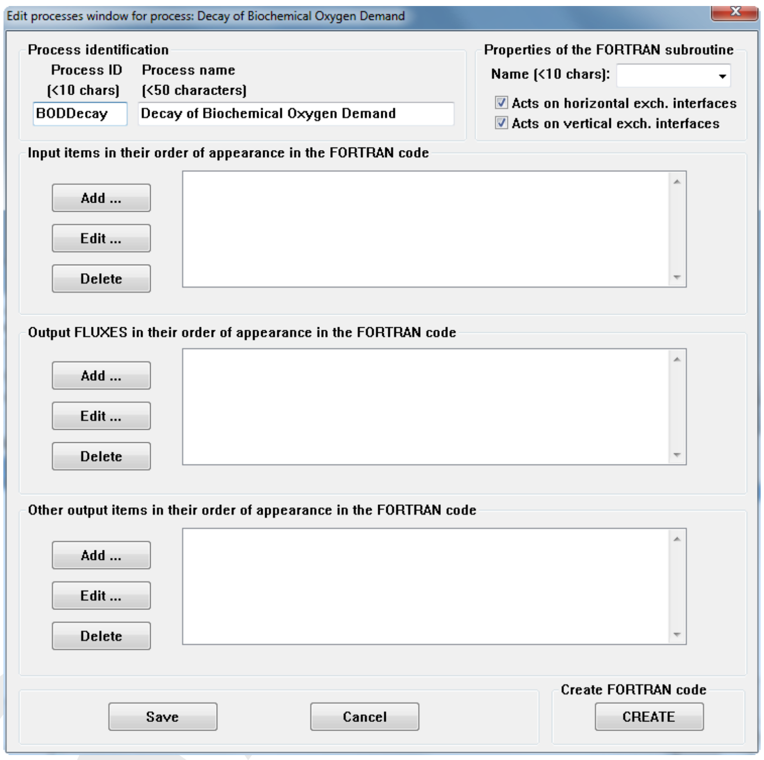

Add new- and Edit existing Processes for: Boichemical Oxygen Demand - 5 day
Process name (<50 characters):
Process ID (<10 characters, must be unique):
Available processes for this substance
Add ...
Edit ...
Delete
Save
Cancel

***Figure 4.9: Add new and Edit existing Processes** window*

Now we will specify the process "Decay of Biochemical Oxygen Demand" and give it the ID "BODDecay".

Pressing the *Add* button causes several things to happen. First, a warning is displayed. It says that this process is now in the list, but that it only stays there if the influences the substance BOD5. This reflects the fact that a process computes fluxes between substances and the system does not know yet between which substances you will create a flux with this process. (You could, for instance, create a process that does a lot, but nothing with BOD5. In that case the process will later on appear only in the list of the substances that are affected by this process and not in the list for BOD).

If we press the *OK* button for the message, we automatically enter the **Edit processes** window for this new process (Figure 4.10).


Edit processes window for process: Decay of Biochemical Oxygen Demand

Process identification

Process ID (<10 chars)	Process name (<50 characters)
BODDecay	Decay of Biochemical Oxygen Demand

Properties of the FORTRAN subroutine

Name (<10 chars):

[✓] Acts on horizontal exch. interfaces
[✓] Acts on vertical exch. interfaces

Input items in their order of appearance in the FORTRAN code

Output FLUXES in their order of appearance in the FORTRAN code

Other output items in their order of appearance in the FORTRAN code

Create FORTRAN code

Figure 4.10: Edit processes window for process *window for the newly created 'BOD-Decay' process.*

This is an important window: it is your link with the computer code. We have to specify the name of a Fortran subroutine. We will fill in the name of a subroutine that we will program ourselves, to compute the 'BODDecay' 'process.

Note: that you can use a generic approach here. If you have three types of bacteria for which the mathematical description of the processes (decay or growth) differ only by their coefficients, they may share the same Fortran routine. The same is true for ten heavy metals or fifteen organic micro-pollutants that mutually react in the same way but only with differences in reaction rates. Furthermore, the name of the subroutine is converted to uppercase (Fortran standard) and is used in uppercase throughout.

We could call our subroutine "FIRSTORDER" and use it for all first order decay processes, like coliform bacteria as well. This would be a very generic routine. This has a drawback: We would have to include an additional process that computes the coliform decay rate as a function of temperature, salinity and UV radiation. For BOD we have to include an additional process that computes the BOD decay rate as a function of temperature and of available oxygen. Instead of doing so, we will join these two steps into one subroutine for the decay of BOD, that we will call "BOD_DECAY".

The above reasoning shows the type of decisions you have to make to keep your library manageable. It is a matter of taste whether you prefer a few, large subroutines or many small subroutines. In general, smaller routines make the system more flexible but, like with everything in life, exaggeration will do no good.

(The two checkboxes *act on horizontal/vertical exchanges* in the upper right corner are reserved for future use.)

We see three empty lists:

◇ **Input items**
This is the list of all items that the process needs to do its computation
◇ **Output Fluxes**
The list of all 'arrows-between-substances' that the process will compute
◇ **Other output items**
The list of all output items that the process will produce.

Input items

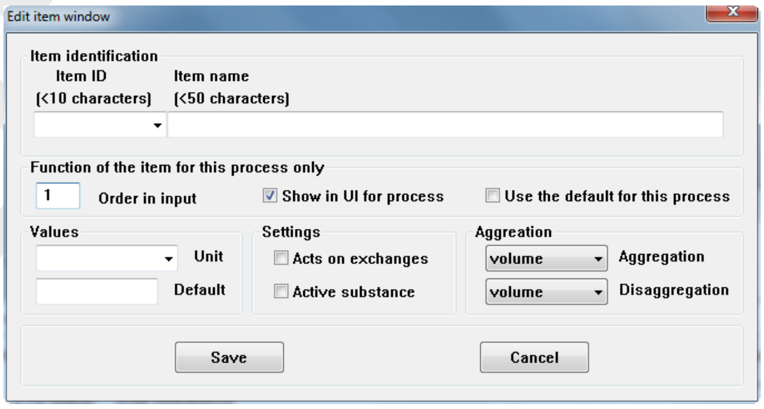
We need the decay rate k_1 for BOD as input parameter for the simple BOD decay formula:

$$\frac{\partial BOD}{\partial t} = -k_1 \cdot BOD \tag{4.1}$$

In general however this decay rate is temperature-dependent according to:

$$k_1 = k_{20} \cdot T_{coef}^{(T-20)} \tag{4.2}$$

The rate equals the rate at 20 °C multiplied with a temperature coefficient to the power $T-20$. This means that we now have three input items: temperature, the temperature coefficient and the decay coefficient at 20 °C. We press the *Add* button besides the **Input Items** section and the **Edit Item** window appears (Figure 4.11).


Edit item window

Item identification
Item ID (<10 characters)
Item name (<50 characters)

Function of the item for this process only
1 Order in input
☑ Show in UI for process
☐ Use the default for this process

Values
Unit
Default

Settings
☐ Acts on exchanges
☐ Active substance

Aggreation
volume Aggregation
volume Disaggregation

Save
Cancel

***Figure 4.11: Edit Item** window*

Here we insert the ID "BOD_Dr_20" with the name "Decay rate at 20 C for Biochemical Oxygen Demand" (this is 48 characters, so about the maximum for this field).

Remark:

◇ ID's are directly used in the computer-generated source code. So they should be valid Fortran names. Use underscores in stead of minus signs. That is also the reason why Oxygen should be preferred as ID rather that DO, because it then would be similar to the Fortran DO-loop instruction.

We see that it is assumed to be the first item in the input sequence for this subroutine. You can change that number to a higher number, but that will leave some entries with lower numbers blank. If you mention here a lower number, that already exists, the system asks you whether you want to replace the existing item or to move the existing item. This allows you to change the order of items for your Fortran program.

The *Show in UI for process* checkbox is checked by default. If you uncheck this, your item will be hidden and the user will not be able to specify it. You may do so if you would define e.g. an item Pi, that you want to use consistently throughout the system, but you do not want to show it to the users. Be aware that the item then needs to have a default value (e.g. 3.14 for the item Pi), because it cannot be resolved otherwise. This option also allows you to hide some constants for the user that you do not want them to change. The only one who then can change the value of the constant is you, through this developer environment.

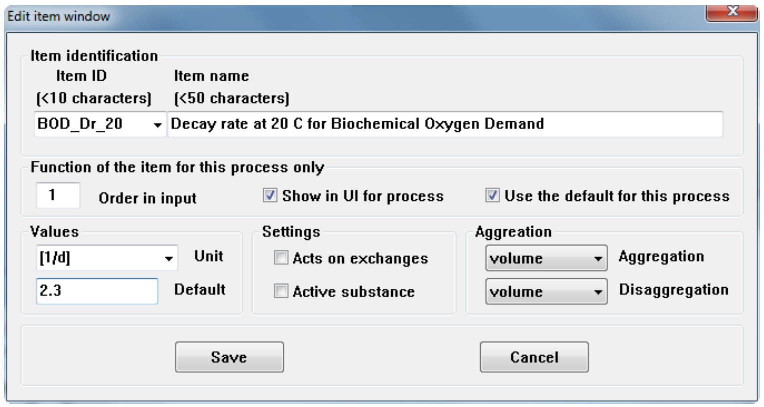
You can specify the unit of the item either directly or select a previously entered unit from the list, here we specify "[1/d]".

An item specified this way can appear in many processes (like temperature does in general). It has one default value, but per process you can specify whether you allow this default value to be used, here "2.3". The consequences of not allowing a default value is, that the item must be specified as a modelled substance, the output of some process or as input provided by the user. This is a method to force an entry by the user for the item for certain processes. As soon as you modify the default value, the checkbox is checked automatically. You can however uncheck it again.

Remark:

◇ If you try to check *Act on exchanges*, the system produces a warning and refuses to do so. This feature only can be checked for output items that are made on computational element interfaces (e.g. a settling velocity) rather than on computational elements itself. Once such an output item is made this way, you can select the existing item from the ID listbox as an input item for other processes (e.g. the settling of suspended sediment itself).

Also if you want to check the checkbox *Active substance*, the system refuses with a warning. This checkbox is only meant for existing items that you selected from the ID listbox. If they are a transportable substance (like Oxygen in our example) then this box will show up as checked. Your entry could look like in Figure 4.12.


Edit item window

Item identification

Item ID (<10 characters): BOD_Dr_20

Item name (<50 characters): Decay rate at 20 C for Biochemical Oxygen Demand

Function of the item for this process only

1 Order in input

☑ Show in UI for process

☑ Use the default for this process

Values: Unit [1/d]; Default 2.3

Settings: ☐ Acts on exchanges; ☐ Active substance

Aggreation: Aggregation volume; Disaggregation volume

***Figure 4.12: Edit Item** window with the BOD Decay Rate filled in*

We save this new item and see it appear in the **Input Items** list of the **Edit Process** window.

We continue the same way with temperature and with the temperature coefficient.

1	Id	"Temp_C",
	name	"Temperature in degrees Celsius",
	unit	"[oC]"
2	Id	"Tcoef_BOD",
	name	"Temperature coefficient for BOD - Decay",
	unit	"[-]"

We continue the same way for BOD5. The formula, Equation 4.2, shows that we need not only the decay rate, but also the actual BOD concentration to determine the decay flux.

1	Id	"BOD5",

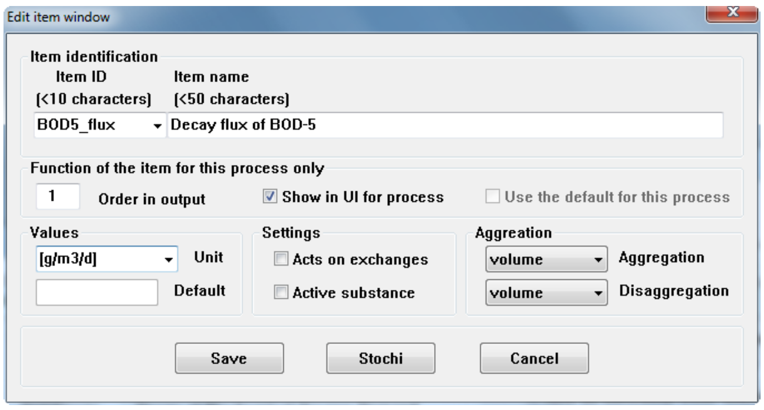
When we leave the ID field and click another field, the system prompts that it already knows a *Biochemical Oxygen Demand - 5 day* item (that was the substance that we defined earlier) and asks permission to include it here. You press on *Yes* and all fields are filled in with the settings for this substance. The checkbox *Active substance* is checked, since BOD is an active, transportable substance. Now you can immediately press the *Save* button to add BOD as input for your process.

Output fluxes

Now we press *Add* besides the **Output Fluxes** list, because we must influence the substances DO and BOD with our process. We define a flux with (see Equation (4.2))

1	Id	"BOD5_flux",
	name	"Decay flux of BOD-5",
	unit	"[g/m3/d]"

The **Edit item** window looks like Figure 4.13. We see that there is also a *Stochi* button available, that was not available for input items. This button will allow us to link our 'arrow' to the substances between which the arrow works.


Edit item window

Item identification
Item ID (<10 characters): BOD5_flux
Item name (<50 characters): Decay flux of BOD-5

Function of the item for this process only
1 Order in output
[✓] Show in UI for process
[ ] Use the default for this process

Values
[g/m3/d] Unit
Default

Settings
[ ] Acts on exchanges
[ ] Active substance

Aggregation
volume Aggregation
volume Disaggregation

Save | Stochi | Cancel

*Figure 4.13: **Edit Item** window with specified flux and* Stochi *button.*

4.5 Setting up the stoichiometry

After pressing the *Stochi* button a message appears. The message says that the Item settings will be saved first to ensure consistency. That is because you are about to connect substances to each other with a *flux item* and if you would save the links in the **Edit stochiometry of flux** window but cancel the **Edit Item** session of the stochi, an inconsistency results. Once you have confirmed the save action, the **Edit stochiometry of flux** window appears (Figure 4.14).

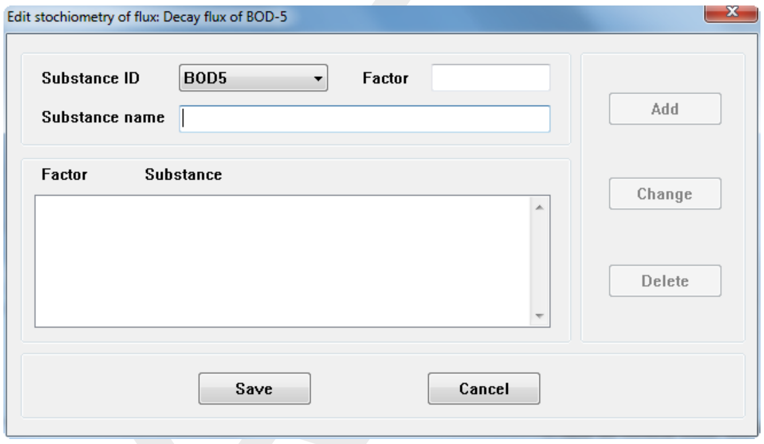
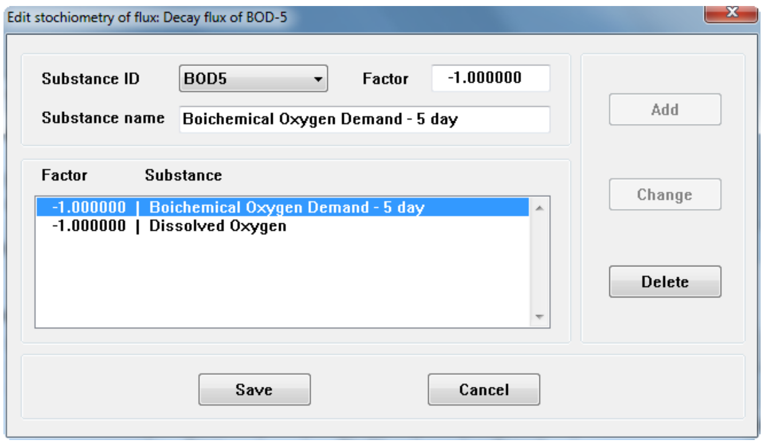


***Figure 4.14: Edit Stochiometry of flux** window*

The flux you are editing is displayed in the title of the window. Fluxes can only act on substances that are already defined, so you should select the substance from the drop-down list.

Now we must take some care: The formula says that the decay is $-k_1 \cdot BOD5$. If we specify k_1 and $BOD5$ with positive values, then a negative sign is required somewhere. If we compute the flux as $-k_1 \cdot BOD5$ in our subroutine, then we should use a *positive* factor of +1.0 to let it work correctly on the substance $BOD5$. If we compute the flux as $k_1 \cdot BOD5$, on the other hand, we should use -1.0 as factor.

We will do the latter. So we choose “BOD5” (explicitly) for substance and “-1.0” as the factor and we press Add. Our flux however also diminishes the Oxygen concentration in the same way, so we also add “DO” as stoichiometry and “-1.0” as factor and add this. The result looks like in Figure 4.15.



Factor: -1.000000
Substance name: Boichemical Oxygen Demand - 5 day

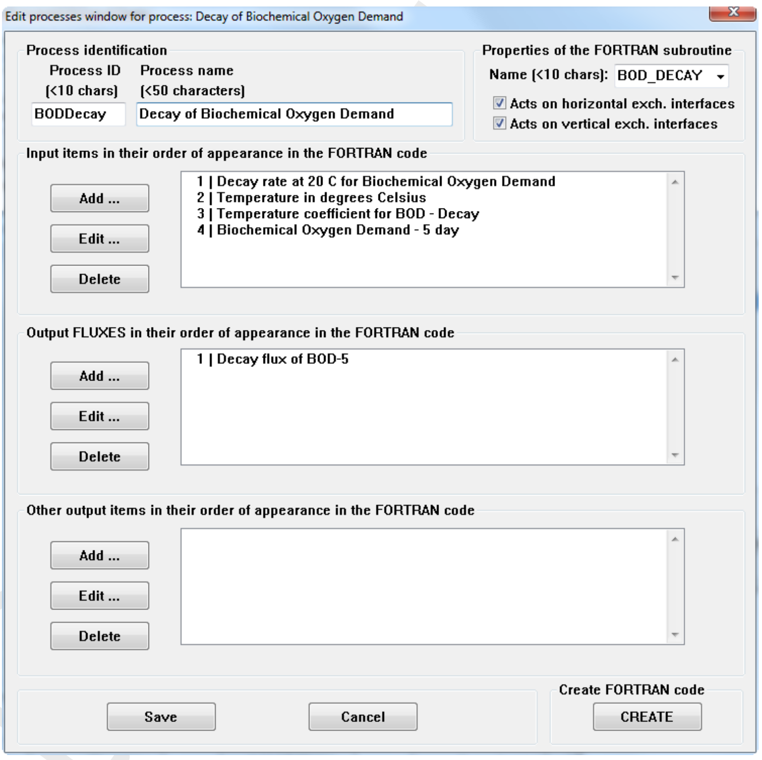
```
-1.000000 | Boichemical Oxygen Demand - 5 day
-1.000000 | Dissolved Oxygen
```

***Figure 4.15: Edit Stochiometry of flux** window after stoichiometry has been entered*

Now we save the stoichiometry and we save the flux.

4.6 Generating computer code

The **Edit Process** window now shows the input items and the flux item in its lists, see Figure 4.16.


Edit processes window for process: Decay of Biochemical Oxygen Demand

Process identification
Process ID (<10 chars): BODDecay
Process name (<50 characters): Decay of Biochemical Oxygen Demand

Properties of the FORTRAN subroutine
Name (<10 chars): BOD_DECAY
☑ Acts on horizontal exch. interfaces
☑ Acts on vertical exch. interfaces

Input items in their order of appearance in the FORTRAN code
1 | Decay rate at 20 C for Biochemical Oxygen Demand
2 | Temperature in degrees Celsius
3 | Temperature coefficient for BOD - Decay
4 | Biochemical Oxygen Demand - 5 day

Output FLUXES in their order of appearance in the FORTRAN code
1 | Decay flux of BOD-5

Other output items in their order of appearance in the FORTRAN code

Create FORTRAN code

***Figure 4.16: Edit Process** with input items*

We now have to create the subroutine that will do the actual work:

◇ Fill in the name of the subroutine, “BOD_DECAY”
◇ Press the *CREATE* button
◇ The system generates a skeleton subroutine file.

We will call our subroutine “BOD_DECAY” and we press the *CREATE* button for the creation of a Fortran file. The system tells that it successfully wrote the skeleton file and indeed we see that a file named <BOD_DECAY.f90> is created in our working directory. The code is standard Fortran90 except for the question marks where you are supposed to insert your water quality computations. Inspection shows something like:

```
      subroutine BOD_DECAY  ( pmsa    , fl      , ipoint , increm, noseg , &
                              noflux , iexpnt , iknmrk , noq1   , noq2   , &
                              noq3    , noq4    )
!DEC$ ATTRIBUTES DLLEXPORT, ALIAS: 'BOD_DECAY' :: BOD_DECAY
!
!*******************************************************************************
!
      IMPLICIT NONE
!
!     Type     Name          I/O Description
!
```

```
      real(4) pmsa(*)     !I/O Process Manager System Array, window of routine to process libr
      real(4) fl(*)       ! O  Array of fluxes made by this process in mass/volume/time
      integer ipoint(  4) ! I  Array of pointers in pmsa to get and store the data
      integer increm(  4) ! I  Increments in ipoint for segment loop, 0=constant, 1=spatially
      integer noseg       ! I  Number of computational elements in the whole model schematisat
      integer noflux      ! I  Number of fluxes, increment in the fl array
      integer iexpnt(4,*) ! I  From, To, From-1 and To+1 segment numbers of the exchange surfa
      integer iknmrk(*)   ! I  Active-Inactive, Surface-water-bottom, see manual for use
      integer noq1        ! I  Nr of exchanges in 1st direction (the horizontal dir if irregul
      integer noq2        ! I  Nr of exchanges in 2nd direction, noq1+noq2 gives hor. dir. reg
      integer noq3        ! I  Nr of exchanges in 3rd direction, vertical direction, pos. down
      integer noq4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use on
      integer ipnt(  4)   !    Local work array for the pointering
      integer iseg        !    Local loop counter for computational element loop
!
!*******************************************************************************
!
!     Type    Name         I/O Description                                        Unit
!
      real(4) Temp_C      ! I  Temperature in degrees Celsius                     [oC]
      real(4) BOD5        ! I  Biochemical Oxygen Demand - 5 day                  [g/m3]
      real(4) BOD_Dr_20   ! I  Decay rate at 20 C for Biochemical Oxygen Demand   [1/d]
      real(4) Tcoef_BOD   ! I  Temperature coefficient for BOD - Decay            [-]
      real(4) BOD5_flux   ! F  Decay flux of BOD-5                                [g/m3/d]
      integer IBOD5_flux  !    Pointer to the Decay flux of BOD-5
!
!*******************************************************************************
!
      ipnt        = ipoint
      IBOD5_flux  = 1
!
      do 9000 iseg = 1 , noseg
!
         Temp_C      = pmsa( ipnt(  2) )
         BOD5        = pmsa( ipnt(  4) )
         BOD_Dr_20   = pmsa( ipnt(  1) )
         Tcoef_BOD   = pmsa( ipnt(  3) )
!
!   *****     Insert your code here  *****
!
         BOD5_flux   = ??????
!
!   *****     End of your code       *****
!
         fl  ( IBOD5_flux  ) = BOD5_flux
!
         IBOD5_flux  = IBOD5_flux  + noflux
         ipnt        = ipnt        + increm
!
 9000 continue
!
      return
      end subroutine
```

The skeleton code shows:

- ◇ The decay rate at 20 °C is input parameter 1
- ◇ The temperature in degrees Celsius is input parameter 2
- ◇ The temperature coefficient is nr. 3
- ◇ The concentration of BOD is nr 4.
- ◇ The flux (the amount of BOD that is decayed) is saved in the FL array.

At the location of the question marks we just insert: our process code. The code for our sample case is:

```
BOD5_flux = BOD_Dr_20 * TCoefBOD**(Temp_C-20.0) * BOD5
```

With this we have completed our computer code and we can compile it to run our water quality model.

Note: that if we specified a stoichiometric factor of +1.0 rather than -1.0, we would have had to specify the flux as -1.0 times the flux shown above.

Save your subroutine at a safe location, because every time you press the Fortran button for this process, this subroutine may be overwritten. Note also that the whole linkage to data and other substances is resolved by the system itself. We just need to insert our process formula in Fortran style.

Note: also the compiler directive in line 4. It is used by the Intel Fortran compiler to properly prepare the library (that is, the subroutine can be used from outside the DLL that should be produced). You may need to change this to the conventions of your own Fortran compiler.

4.7 Make a Windows DLL from your source files

Once you have created one or more subroutines for use in the water quality module, you need to compile them and link them into a dynamic link library, called *D3Dwaq_OpenPL.dll*[2]. This library must then be found by the water quality module.

While the procedure varies for each compiler, we describe it for the Microsoft Visual Studio and Intel Fortran:

◇ Store the relevant source files in a separate directory, for easy reference
◇ Start the Visual Studio and create in this directory a project of the type “Fortran Dynamic Link Library”. Set the name of the project to “D3Dwaq_OpenPL” or make sure that the resulting DLL is called “D3Dwaq_OpenPL.dll”
◇ Add the source files to the Visual Studio project
◇ Starting from version 4.02.00 of Delft3D, you have to build a 64-bit DLL. When you are using a seperate Delwaq version (e.g. built from the open source code), you will have to make sure to use a 32-bit DLL (Windows) or so (Linux) with a 32-bits Delwaq executable, and a 64-bit DLL or so (Linux) with a 64-bits Delwaq executable. In Visual Studio you can add a 64-bit configuration through the menu via Build - Configuration Manager... .
◇ Build the DLL

When this is done, the DLL and the process definition file must be copied to a spot where the water quality module can find them. This is described in the next section.

Note: On Windows, you can check wheter executables and dll’s are 32-bit or 64-bit using tools like Dependency Walker (http://www.dependencywalker.com/). Here you can also see if your fortran subroutine is actually available in the DLL.

Note: in the next section a semi-automatic procedure is described to help you with the build process.

[2]On Linux is this D3Dwaq_OpenPL.so – note the mixture of upper and lower case!

4.8 Incorporation of your new process library database

The database file that you created must be copied to the location in your Deltares software environment where the file <proc_def.dat> file is located. For the Delft3D environment this is e.g. the directory <C:\Program Files\Deltares\Delft3D 4.1.0\win32\waq\default\> if you installed the system in the <C:\Program Files\Deltares\Delft3D 4.1.0> directory. If the existing <proc_def.dat> file is not empty (you got a license for the Deltares processes library as well), this file should be saved for future use. Now you copy your file to overwrite the existing <proc_def.dat> in the directory.

As these actions are somewhat tedious and error-prone, we provide automatic procedures to do this.

4.8.1 Delft3D

The menu by which you start the PLCT in developer mode, also facilitates the building and installation of the new library, see Figure 4.17.

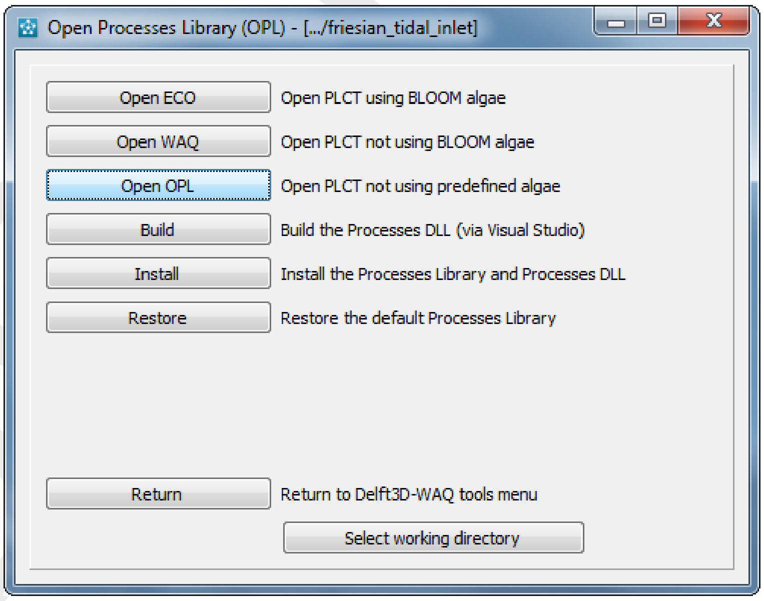

Open Processes Library (OPL) - [.../friesian_tidal_inlet]
Open ECO
Open PLCT using BLOOM algae
Open WAQ
Open PLCT not using BLOOM algae
Open OPL
Open PLCT not using predefined algae
Build
Build the Processes DLL (via Visual Studio)
Install
Install the Processes Library and Processes DLL
Restore
Restore the default Processes Library
Return
Return to Delft3D-WAQ tools menu
Select working directory

***Figure 4.17: Open Processes Library (OPL)** window*

- ◇ The assumption is that the *working directory* is the directory that contains or will contain the sources, the files required for building the library and the library itself.
- ◇ The button marked *Build* starts the Visual Studio for either a select you select or the only solution in that directory. This way you can adjust the source files you just generated and build the DLL without leaving Delft3D.
- ◇ If there is no solution file yet, an empty one will be copied, with the right settings. All you need to do is, add the source files and edit them.
- ◇ The button marked *Install* copies the various files (that is, the database file, which must be called <proc_def.dat> and the DLL, called <D3Dwaq_OpenPL.dll> from either the debug or the release directory, whichever is newer, into the Delft3D installation). The original files will be saved, so that you can restore them whenever you want.
- ◇ The button marked *Restore* will restore the original Delft3D processes database and it will remove the extra DLL.

The most important thing to keep in mind is that the procedures behind these buttons assume fixed file and directory names, so that you do not need to select files over and over again.

4.8.2 SOBEK

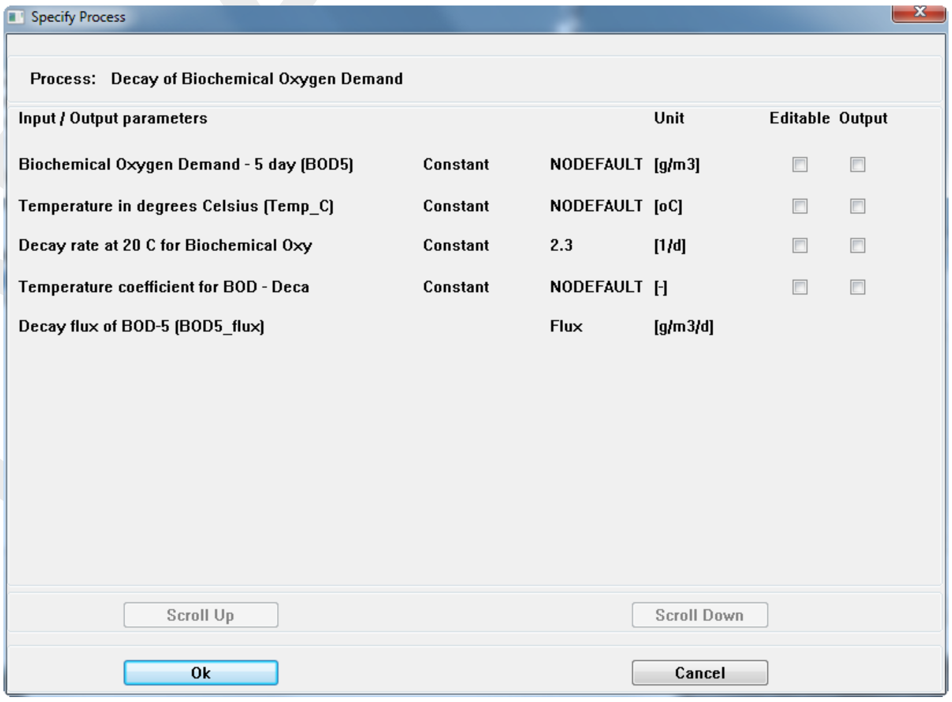
In development

4.9 Modify an existing process library

Now it is also time to save our work. We press the *Save* buttons in the various windows until we reach the **Processes Library Configuration Tool** window again and we press *Save* from the *File*. Specify a meaningful name and your very first and small process library is created. You exit the computer program and you restart it using the freshly created process library.

You now see that the ‘BOD-DO’ group exists in the **Select Groups** window. If you select it and click the right list of this window, the substances in this group are shown. Here you find ’Dissolved Oxygen’ and ’Boichemical Oxygen Demand - 5 day’. If you now select *Dissolved Oxygen*, you see that the ’Decay of Biochemical Oxygen Demand (BODDecay)’ process is listed as process for this substance. That is because the flux made by the BOD_Decay process also affects Dissolved Oxygen through its stoichiometry.

If you make the process active, you enter immediately into a specification window (see Figure 4.18) because some of the items are not specified: neither BOD5 or Temp_C have a default.

[htb] 

Specify Process

Process: Decay of Biochemical Oxygen Demand

Input / Output parameters			Unit	Editable	Output
Biochemical Oxygen Demand - 5 day (BOD5)	Constant	NODEFAULT	[g/m3]	☐	☐
Temperature in degrees Celsius (Temp_C)	Constant	NODEFAULT	[oC]	☐	☐
Decay rate at 20 C for Biochemical Oxy	Constant	2.3	[1/d]	☐	☐
Temperature coefficient for BOD - Deca	Constant	NODEFAULT	[-]	☐	☐
Decay flux of BOD-5 (BOD5_flux)		Flux	[g/m3/d]		

Figure 4.18: ***Specify processes*** *window for process:* Decay of Biochemical Oxygen Demand

If you press *Cancel* in this window and also *Cancel* in the window by which you made the process active, you can switch on ’Dissolved Oxygen’ together ’Biochemical Oxygen Demand - 5 day’. After selecting ’Dissolved Oxygen’ in right hand side listbox the window **Select Processes** appears. Press the button *Specify...* and nearly the same window as in Figure 4.18 appears but you will see that the BOD5 item is now considered to be ‘modelled’.

The item Temp_C, temperature in $^\circ$C, still has no value and if you click that it is *Editable*, it will appear in the Delft3D and SOBEK user interface for specification. You should be able to use this model set-up for an actual computation. The result will be something like Figure 4.19 for a river that flows with 1 m/s.

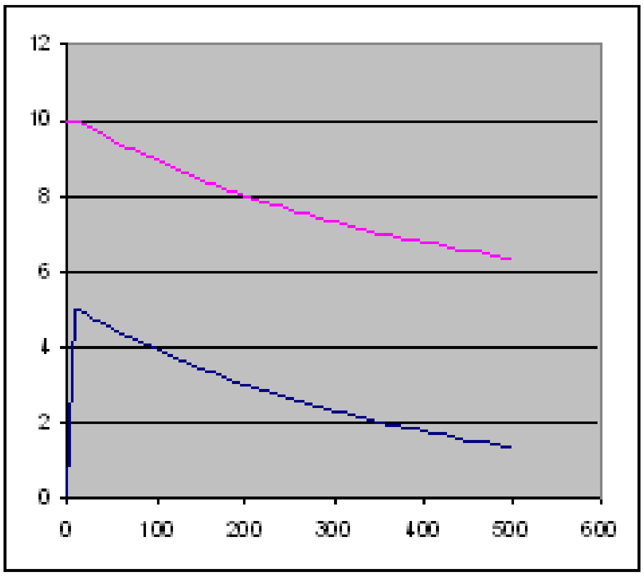
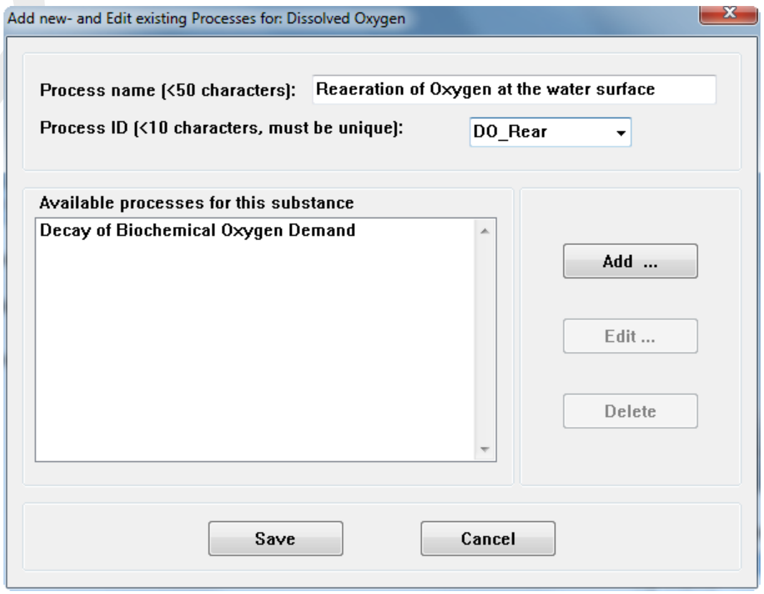


Figure 4.19: *BOD-DO curve for BOD-Decay. x-axis in km. y-axis BOD (lower graph) and Oxygen (upper graph).*

Our BOD-DO model is not ready yet, because in this classical model there is also reaeration of oxygen at the water surface that we should incorporate as a process. We need to enter the developer mode again.

In the *Selected Substances* list box, we select 'Dissolved Oxygen' to add a process (see Figure 4.20).


Add new- and Edit existing Processes for: Dissolved Oxygen

Process name (<50 characters): Reaeration of Oxygen at the water surface

Process ID (<10 characters, must be unique): DO_Rear

Available processes for this substance

Decay of Biochemical Oxygen Demand

Figure 4.20: *Adding substance 'Dissolved Oxygen reaeration'*

We will implement the well-known O'Connor-Dobbins reaeration formula (O' Connor and Dobbins, 1956):

$$\text{Rearation rate (1/day)} = \frac{3.95 \cdot v^{0.5}}{\text{Depth}^{1.5}} \tag{4.3}$$

Some remarks must be made here:

Remarks:

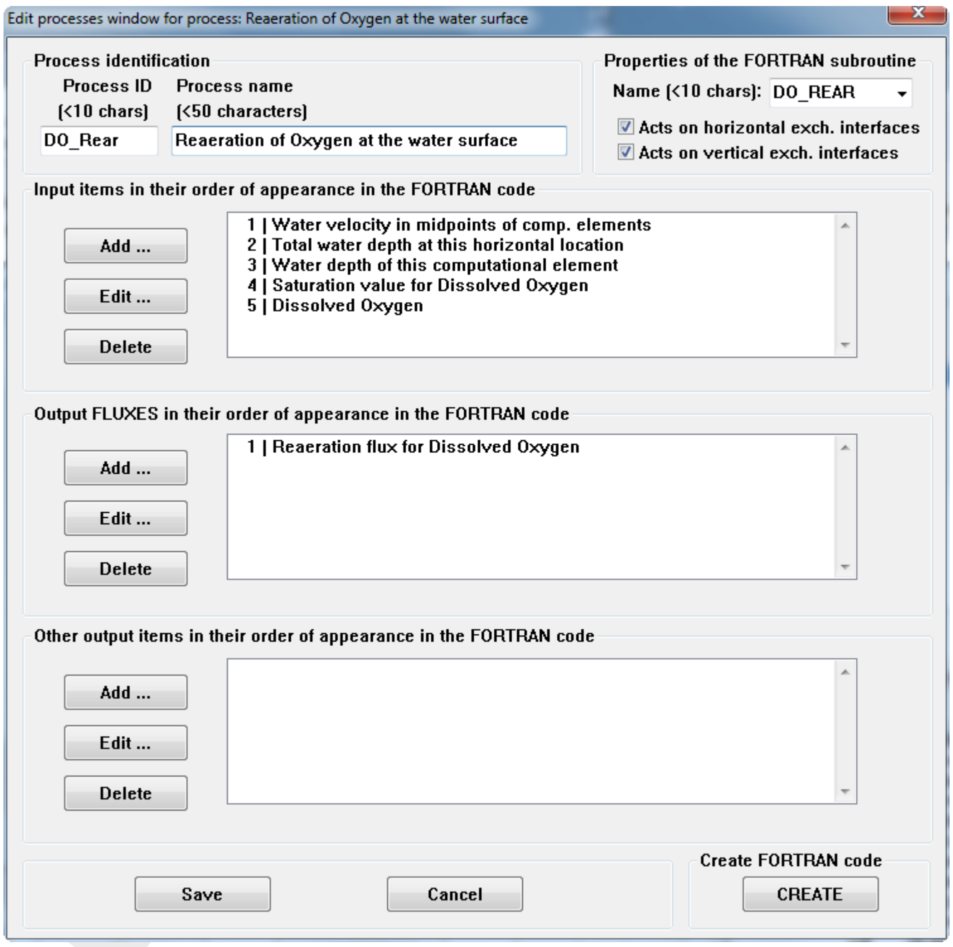
- ◇ O'Connor derived his formula for the concentration equation and got a rate of 1/day. The correct dimension in a numerical model is m/day. The 1/day rate is obtained after multiplication with the water surface area [m^2] and division by the underlying water volume [m^3]. This means the reaeration coefficient consists of a $3.95 \cdot (v/Depth)^{0.5}$ part [m/day] and a 1/Depth part [1/m] from *surface area divided by volume*. The last term reflects that surface reaeration influences shallow water more pronounced than deep water.
- ◇ Because O'Connor derived his formula for depth-averaged modelling, both factors containing the depth were equal for him. In a 3D model you should use the total water depth in the factor $3.95 \cdot (v/Depth)^{0.5}$ and the depth of a computational element in the factor 1/Depth.

To get the oxygen flux, we have to multiply the O'Connor-Dobbins rate with the difference between the oxygen saturation value and the actual oxygen concentration. So we define a reaeration process with five input items and one output flux item. Five input items:

1	Id	"Velocity",
	name	"Water velocity in midpoints of comp. elements",
	unit	"[m/s]"
2	Id	"Tot_Depth",
	name	"Total water depth at this horizontal location",
	unit	"[m]"
3	Id	"Depth",
	name	"Water depth of this computational element",
	unit	"[m]"
4	id	"DO_Sat",
	name	"Saturation value for Dissolved Oxygen",
	unit	"[g/m3]"
5	id	"Oxygen",
	name	"Dissolved Oxygen",
	unit	"[g/m3]"

and one output flux item:

1	id	"DO_R_flux",
	name	"Reaeration flux for Dissolved Oxygen",
	unit	"[g/m3/d]"


Edit processes window for process: Reaeration of Oxygen at the water surface

Process identification

Process ID (<10 chars)	Process name (<50 characters)
DO_Rear	Reaeration of Oxygen at the water surface

Properties of the FORTRAN subroutine

Name (<10 chars): DO_REAR

[✓] Acts on horizontal exch. interfaces
[✓] Acts on vertical exch. interfaces

Input items in their order of appearance in the FORTRAN code

1 | Water velocity in midpoints of comp. elements
2 | Total water depth at this horizontal location
3 | Water depth of this computational element
4 | Saturation value for Dissolved Oxygen
5 | Dissolved Oxygen

Output FLUXES in their order of appearance in the FORTRAN code

1 | Reaeration flux for Dissolved Oxygen

Other output items in their order of appearance in the FORTRAN code

***Figure 4.21: Edit Processes** window for the DO-reaeration process*

The process produces the Reaeration flux with dimension $gO_2/m^3/day$. If the influx of Oxygen through the water surface is considered positive, then we add +1 for DO in the stoichiometry window. We insert the O'Connor-Dobbins reaeration rate formula in the skeleton Fortran file and multiply it with (Saturation value minus Dissolved Oxygen concentration) to obtain the reaeration flux.

Figure 4.21 shows how the reaeration process would look like in the **Edit Processes** window after you followed similar procedure as described for the set up of decay of BOD5.

```
      .......
      integer noq4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use on
      integer ipnt(  5)   !    Local work array for the pointering
      integer iseg        !    Local loop counter for computational element loop
!
!*******************************************************************************
!
!     Type    Name         I/O Description                                        Unit
!
      real(4) Velocity    ! I  Water velocity in midpoints of comp. elements      [m/s]
      real(4) Tot_Depth   ! I  Total water depth at this horizontal location      [m]
      real(4) Depth       ! I  Water depth of this computational element          [m]
      real(4) DO_Sat      ! I  Saturation value for Dissolved Oxygen              [g/m3]
      real(4) Oxygen      ! I  Dissolved Oxygen                                   [g/m3]
      real(4) DO_R_flux   ! F  Reaeration flux for Dissolved Oxygen               [g/m3/d]
      integer IDO_R_flux  !    Pointer to the Reaeration flux for Dissolved Oxygen
!
!*******************************************************************************
!
      ipnt        = ipoint
      IDO_R_flux  = 1
!
      do 9000 iseg = 1 , noseg
!
         Velocity   = pmsa( ipnt(  1) )
         Tot_Depth  = pmsa( ipnt(  2) )
         Depth      = pmsa( ipnt(  3) )
         DO_Sat     = pmsa( ipnt(  4) )
         Oxygen     = pmsa( ipnt(  5) )
!
!   *****     Insert your code here  *****
!
         DO_R_flux  = ??????
!
!   *****     End of your code       *****
!
         fl  ( IDO_R_flux  ) = DO_R_flux
!
         IDO_R_flux  = IDO_R_flux  + noflux
         ipnt        = ipnt        + increm
!
 9000 continue
!
      return
      end subroutine
```

Figure 4.22: *Source code of the Oxygen Reaeration process*

Figure 4.22 shows the corresponding source code of the reaeration process and Figure 4.23 shows the result that would be obtained with a saturation value of 10 mg/l, a water depth of 10 m and a river stream velocity of 1 m/s. A minimum concentration of oxygen can be seen after about 100 km, although it is not striking for this relatively fast flowing large river.

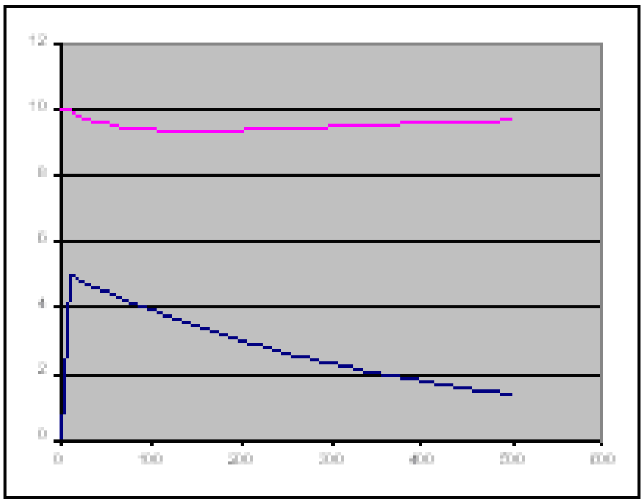


***Figure 4.23:** BOD-DO curve for BOD-Decay as before, but now with reaeration*

4.9.1 Towards advanced features

The addition of reaeration brings up some questions:

- ◇ Why consider the water velocity as ‘input’, it should be known by the model.
- ◇ Why consider the total water depth and the depth of a computational element as input, it should be known by the model.
- ◇ The oxygen saturation value depends on temperature and salinity.

The last item is easiest to deal with. It is analogous to the temperature dependence of BOD-Decay. It can be incorporated in this process (and then temperature and salinity should be added to the set of input variables) or you can compute the DO-saturation value by a separate process, that is a matter of taste. In any case it is useful to let your Fortran routine also compute the percentage of saturation as output variable, because that can be a useful item for graphical presentation.

The flow velocity and the total depth and depth of computational elements need to be calculated by separate processes. The water quality module only has flow velocities at the *interfaces* of computational elements and the total depth and element depth are no state variables at all. There are several solutions to this issue. You can:

- ◇ Use the pre-programmed processes in the standard processes library that take care of velocity and both depth values. You do so in the standard processes library by selecting the items ‘Velocity’, ‘Depth’ and ‘TotalDepth’ that are already available there. They are calculated by the processes ‘Veloc’, ‘DynDepth‘ and ‘TotDepth’ respectively.
- ◇ Or you can also program them yourself, as will be explained in the Chapter 5. If you do the last, you need to do so only once, because once the processes are on your bookshelf, you can use them any time you want.

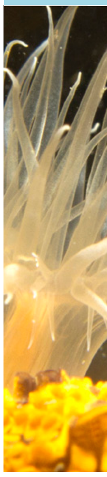
If you follow the last line of working, you will get more familiar with the advanced topics of process library programming. This basically should allow you in the end to rebuild Deltares pre-programmed processes library yourself from scratch, using our technical reference manual, and program anything additional that comes to your mind. We will follow the second line in this tutorial to show you examples of more advanced programming efforts.

5 Advanced programming

In Chapter 4 the basics of setting up a process library have been shown. It may seem somewhat complicated to conduct so many actions to set up the classical Streeter-Phelps BOD-DO model (Streeter and Phelps, 1925). The advantage however is, that you do this only once for each substance or process that you want to include. Then they are in the library and you can easily decide to switch substances on or off and processes on or off, without having to bother about IT things as "what input is connected to what piece of the software".

Once you have included correct pieces of Fortran in the tiny objects called process routines, you cannot make any programming errors anymore. You add substances and processes by adding new routines in the same way to the existing library. You use only what you switch on. Not fully tested new developments (like the products of students in an academic environment or results from still ongoing in-house research) can be added without disturbing the functioning of the existing system as long as they are not switched on. This is a systematic and consistent way to preserve your investments in research and developments without having to bother about versions and releases of source code and merging of source code.

The topics mentioned in this *Advanced features* section may seem even more cumbersome than the first introduction of the BOD-DO model, but they also need to be done only once and then future developments can easily benefit from their results.



5.1 Use hydrodynamic results to compute depth

The first advanced program is the use of hydrodynamic model results 'on the fly' to compute the water depth of a computational element. The procedure is given pointwise with explanation added to several points, so it is more or less cook-book style, but you should understand the reasons while doing it.

1 Open the PLCT, using the newly saved BOD-DO kinetics in developer mode.
2 Select e.g. Dissolved Oxygen and press the 'Add/Edit' button in the 'Select Processes' window
3 Make a new process called 'Depth'; 'Computation of depths of computational elements'
4 Add the process and see the 'Edit processes' window open. Call the Fortran routine DEPTH and press *Add* at the *Input items* section.
5 Press the list button for Item ID's. And see that there is an item VOLUME and SURF in this list. You did not define this items, they were defined already, because they are available from the hydrodynamic computation. We will add both as input item.
6 In the Fortran routine we will compute the depth of a computational element by dividing the volume through the horizontal surface area. We will not produce a 'Flux', but we will produce an output. This will be called 'Depth' and will be the corresponding input item for the reaeration formula. So press *Add* in the *Output Item* section and add the existing item 'Depth' to this Output Item section.
7 Now we create the skeleton Fortran file as <DEPTH.f90>
8 We insert `Depth = VOLUME/SURF` in the user-code section of the Fortran routine. If you feel not sure, you could insert: `Depth = 1.0` and then `if (SURF > 1.0E-10) Depth = VOLUME/SURF`. This is safer in case the horizontal surface could have a value of zero.
9 You save your newly created version of the process library.

We will check what happened:

a. We open the PLCT again and choose our newly created processes file. *Do not enter the developer mode.*

b. We press the *Edit* button for *Extra processes* and see that the window is empty.
c. We select the group BOD-DO
d. We select the substance *Dissolved Oxygen* and we switch the *Reaeration* process on.
e. We enter immediately into the **Specify Process** window, because some of the input is not resolved (velocity, total depth and saturation value of dissolved oxygen, if you did not give these items a default value).
f. You also see that the depth of the computational element is now resolved and will be computed by the process called 'Depth'.
g. For all items without a default you check *Editable* and you press *OK*
h. Now you will see in the **Edit Extra processes** window, that the process 'Depth' has appeared.

This means that the computation of depth is switched on automatically as soon as you switch on the 'Reaeration' process that needs the depth. If it is not needed, the depth is not computed. If it is needed several times in different processes, it is only computed once and used everywhere where needed. It is now 'on the shelf' of your processes library.

5.2 Use hydrodynamic results to compute the velocity

The second advanced programming effort is in the computation of the flow velocity. As stated before, you can use our standard process for that aim, but the development of a routine of your own gives valuable experience and insight in how to deal with these topics.

You must realise that the water quality module uses the so called *finite volume approach*. For the mathematicians: this means that the advection diffusion equation is integrated over all individual computational elements. Gauss's theorem furthermore says that the volume integral of the advection and diffusion terms equal the surface integral of the advective and diffusive fluxes through the outer surface of each volume. This ensures mass conservation, but it also explains why in the water quality module the velocities are known at the segment interfaces only. Even they are not known as velocities, but as flows and cross-sectional areas, but that is a minor detail, since velocity = flow/area.

Now you have to decide how to compute the velocity that is representative in a computational element in e.g. a 1D-branched river basin network. That is a modelling decision. It has nothing to do with the concept of an open processes library. Any model using a so-called staggered grid (velocities defined at the interfaces of computational elements) has to incorporate a procedure to assess velocities that are representative for the computational elements itself. The open processes library, however, allows you to develop your own ideas on this topic. For a 1D branched river, you could consider to average velocities proportional to the areas after a split or before a confluence and to average the upstream and downstream value. This would create the following formula for a split or confluence:

$$v = 0.5\frac{Q_i}{A_i} + 0.5\left(\frac{\frac{Q^1_{i+1}}{A^1_{i+1}}A^1_{i+1} + \frac{Q^2_{i+1}}{A^2_{i+1}}A^2_{i+1}}{A^1_{i+1} + A^2_{i+1}}\right) \approx \frac{Q_i + Q^1_{i+1} + Q^2_{i+1}}{A_i + A^1_{i+1} + A^2_{i+1}} \tag{5.1}$$

The approximation is exact if the sum of the two surface areas after the split equals the surface area before the split. The resulting formula is simple to implement. You sum all flows and areas associated with a computational element separately and divide in the end.

To compute the horizontal velocity in a 2D-horizontal model (or in one layer of a layered 3D model), you could use the same procedure for each of the 2 main directions if main directions can be distinguished. Then however you need to use Pythagoras law to compute the velocity as the root of sum of squares of the velocities in the two main directions. If the grid is irregular, it is difficult to propose an approach. Finite element models may deliver you the velocities in the nodes, for finite volume models on irregular grids you must make 'the best of it'.

How can your process routine see that it is operating in a 1D, 2D or 3D environment? When you inspect the skeleton Fortran routine in detail, you will see a NOQ1, NOQ2, NOQ3 and a NOQ4 in the parameter list. They reflect the number of interfaces between computational elements in four directions. The number in the first horizontal direction, NOQ1, generally always differs from 0. The number in the second horizontal direction, NOQ2, generally does so only if you have a horizontal or layered 3D model with two distinct directions. The number in the third direction, NOQ3, reflects the number of vertical interfaces. It determines whether your model is horizontally only (NOQ3=0), 2D-vertically (NOQ1 and NOQ3 non zero, NOQ2=0) or 3D (NOQ1, NOQ2 and NOQ3 are non zero). In a 1D-vertical model NOQ3 may even be the only non-zero of the three. NOQ4 is the number of interfaces between computational elements within the water bottom.

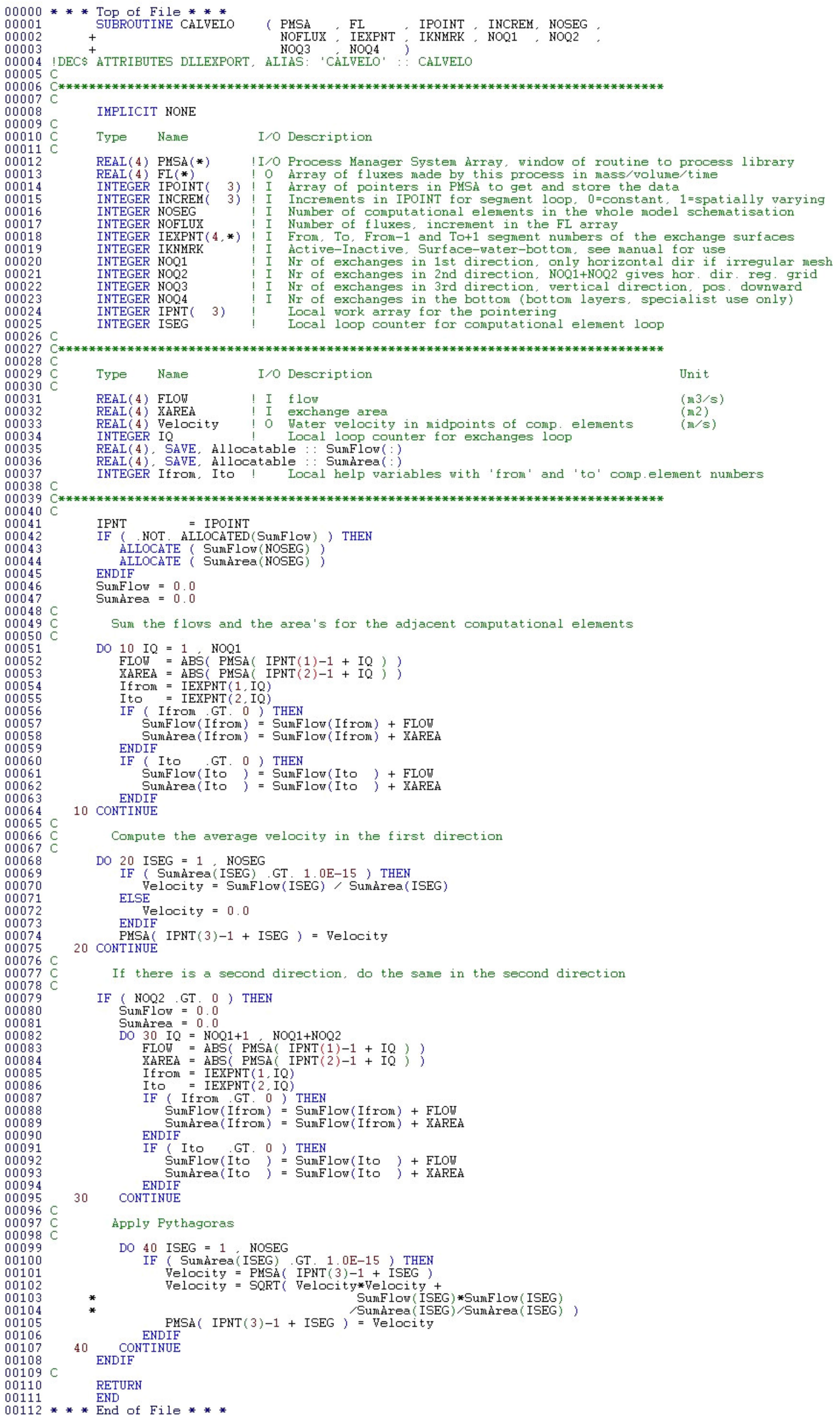
A further inspection of the subroutine header gives you an integer variable IEXPNT. This is not a single variable. It is an array with dimension (4, NOQ1+NOQ2+NOQ3+NOQ4) . It stores for each of the interfaces between computational elements the 'from' element number, the 'to' element number, the 'from-1' element number and the 'to+1' element number respectively. The last two may be zero is the interface with the hydrodynamic model does not support them. They allow for higher order approximations along the directions if present.

The cookbook style procedure to make the velocity now becomes:

1 Open the PLCT, using the just saved BOD-DO kinetics
2 Switch 'Developer mode' on and select e.g. Dissolved Oxygen and press the *Add/Edit* button in the **Select Processes** window
3 Make a new process called "Calvelo"; "Computation of horizontal velocities of computational elements"
4 Add the process and see the **Edit processes** window open. Name the Fortran file to "CALVELO" and press *Add* at the *Input items* section.
5 Press the list button for *Item IDs*. And see that there is an item FLOW and XAREA in this list. You did not define this items, they were defined already, because they are available from the hydrodynamic computation. We will add both as input item.
6 In the Fortran routine we will compute the velocity in a computational element by the earlier described procedure. We will not produce a 'Flux', but we will produce an output. This will be called 'Velocity' and will be the corresponding input item for the reaeration formula. So press *Add* in the *Output Item* section and add the existing item "Velocity" to this *Output Item* section.
7 Now we create the skeleton Fortran file as CALVELO.f
8 We insert the earlier described computational procedure in the user-code section of the Fortran routine. If you feel not sure, you could insert precautions for the case where the XAREA is zero. Your will look like in Figure 5.1. See also the explanatory text following the figure.
9 You save your newly created version of the process library.

We could not lean that much any more on the skeleton Fortran file that was produced by the software.

◇ In the declaration section we had to add a loop-counter over exchanges, called IQ and two convenience variables "Ifrom" and "Ito".
◇ We furthermore had to declare working space in the form of two allocatable arrays: "SumFlow" and "SumArea". With the 'SAVE' attributed they are allocated only once and reused



```
* * * Top of File * * *
      SUBROUTINE CALVELO     ( PMSA    , FL      , IPOINT , INCREM, NOSEG ,
     +                         NOFLUX , IEXPNT , IKNMRK , NOQ1   , NOQ2   ,
     +                         NOQ3    , NOQ4    )
!DEC$ ATTRIBUTES DLLEXPORT, ALIAS: 'CALVELO' :: CALVELO
C
C*******************************************************************************
C
      IMPLICIT NONE
C
C     Type    Name         I/O Description
C
      REAL(4) PMSA(*)     !I/O Process Manager System Array, window of routine to process library
      REAL(4) FL(*)       ! O  Array of fluxes made by this process in mass/volume/time
      INTEGER IPOINT(  3) ! I  Array of pointers in PMSA to get and store the data
      INTEGER INCREM(  3) ! I  Increments in IPOINT for segment loop, 0=constant, 1=spatially varying
      INTEGER NOSEG       ! I  Number of computational elements in the whole model schematisation
      INTEGER NOFLUX      ! I  Number of fluxes, increment in the FL array
      INTEGER IEXPNT(4,*) ! I  From, To, From-1 and To+1 segment numbers of the exchange surfaces
      INTEGER IKNMRK      ! I  Active-Inactive, Surface-water-bottom, see manual for use
      INTEGER NOQ1        ! I  Nr of exchanges in 1st direction, only horizontal dir if irregular mesh
      INTEGER NOQ2        ! I  Nr of exchanges in 2nd direction, NOQ1+NOQ2 gives hor. dir. reg. grid
      INTEGER NOQ3        ! I  Nr of exchanges in 3rd direction, vertical direction, pos. downward
      INTEGER NOQ4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use only)
      INTEGER IPNT(  3)   !    Local work array for the pointering
      INTEGER ISEG        !    Local loop counter for computational element loop
C
C*******************************************************************************
C
C     Type    Name         I/O Description                                        Unit
C
      REAL(4) FLOW        ! I  flow                                               (m3/s)
      REAL(4) XAREA       ! I  exchange area                                      (m2)
      REAL(4) Velocity    ! O  Water velocity in midpoints of comp. elements      (m/s)
      INTEGER IQ          !    Local loop counter for exchanges loop
      REAL(4), SAVE, Allocatable :: SumFlow(:)
      REAL(4), SAVE, Allocatable :: SumArea(:)
      INTEGER Ifrom, Ito  !    Local help variables with 'from' and 'to' comp.element numbers
C
C*******************************************************************************
C
      IPNT        = IPOINT
      IF ( .NOT. ALLOCATED(SumFlow) ) THEN
         ALLOCATE ( SumFlow(NOSEG) )
         ALLOCATE ( SumArea(NOSEG) )
      ENDIF
      SumFlow = 0.0
      SumArea = 0.0
C
C       Sum the flows and the area's for the adjacent computational elements
C
      DO 10 IQ = 1 , NOQ1
         FLOW  = ABS( PMSA( IPNT(1)-1 + IQ ) )
         XAREA = ABS( PMSA( IPNT(2)-1 + IQ ) )
         Ifrom = IEXPNT(1,IQ)
         Ito   = IEXPNT(2,IQ)
         IF ( Ifrom .GT. 0 ) THEN
            SumFlow(Ifrom) = SumFlow(Ifrom) + FLOW
            SumArea(Ifrom) = SumFlow(Ifrom) + XAREA
         ENDIF
         IF ( Ito   .GT. 0 ) THEN
            SumFlow(Ito  ) = SumFlow(Ito  ) + FLOW
            SumArea(Ito  ) = SumFlow(Ito  ) + XAREA
         ENDIF
   10 CONTINUE
C
C       Compute the average velocity in the first direction
C
      DO 20 ISEG = 1 , NOSEG
         IF ( SumArea(ISEG) .GT. 1.0E-15 ) THEN
            Velocity = SumFlow(ISEG) / SumArea(ISEG)
         ELSE
            Velocity = 0.0
         ENDIF
         PMSA( IPNT(3)-1 + ISEG ) = Velocity
   20 CONTINUE
C
C       If there is a second direction, do the same in the second direction
C
      IF ( NOQ2 .GT. 0 ) THEN
         SumFlow = 0.0
         SumArea = 0.0
         DO 30 IQ = NOQ1+1 , NOQ1+NOQ2
            FLOW  = ABS( PMSA( IPNT(1)-1 + IQ ) )
            XAREA = ABS( PMSA( IPNT(2)-1 + IQ ) )
            Ifrom = IEXPNT(1,IQ)
            Ito   = IEXPNT(2,IQ)
            IF ( Ifrom .GT. 0 ) THEN
               SumFlow(Ifrom) = SumFlow(Ifrom) + FLOW
               SumArea(Ifrom) = SumFlow(Ifrom) + XAREA
            ENDIF
            IF ( Ito   .GT. 0 ) THEN
               SumFlow(Ito  ) = SumFlow(Ito  ) + FLOW
               SumArea(Ito  ) = SumFlow(Ito  ) + XAREA
            ENDIF
   30    CONTINUE
C
C       Apply Pythagoras
C
         DO 40 ISEG = 1 , NOSEG
            IF ( SumArea(ISEG) .GT. 1.0E-15 ) THEN
               Velocity = PMSA( IPNT(3)-1 + ISEG )
               Velocity = SQRT( Velocity*Velocity +
     *                          SumFlow(ISEG)*SumFlow(ISEG)
     *                         /SumArea(ISEG)/SumArea(ISEG) )
               PMSA( IPNT(3)-1 + ISEG ) = Velocity
            ENDIF
   40    CONTINUE
      ENDIF
C
      RETURN
      END
* * * End of File * * *
```

Figure 5.1: *Computer code to compute horizontal velocities*

every timestep, that is better for performance.

- Then we constructed a loop over NOQ1 exchanges and picked the FLOW and AREA variables directly from the PMSA array, without incrementing the IPNT pointer like in previous code.
- We got the values for the Ifrom and the Ito variables from the exchange pointers IEXPNT as explained and we summed Flows and Areas.
- Then we included a segment loop that is somewhat comparable with previous code examples and store the velocity directly in the PMSA array without incrementing IPNT
- If there is a second direction, we do the same in the second direction and apply Pythagoras. Note how we picked Ifrom and Ito from IEXPNT for this second direction.

Although this code is already somewhat more complicated, it should be easily understood by an average Fortran programmer. Again, you do this only once, you add the routine to the DLL and your process producing 'Velocity' is "in the library" and is even switched on automatically if some other process needs this 'Velocity'.

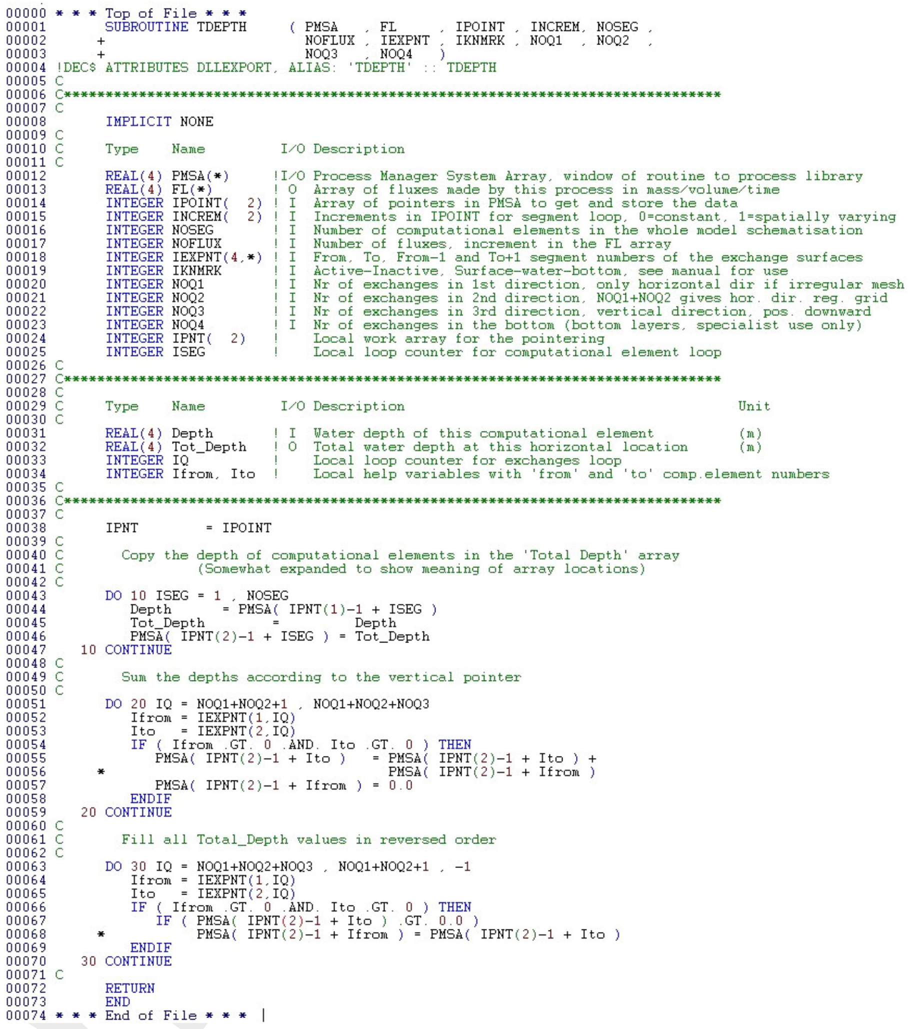
5.3 Compute the total water depth

The third advanced programming effort is the computation of total water depth. This variable may be produced by the hydrodynamic model and may be available as parameter specified through an external file from the hydrodynamic model. You may also use our pre-programmed code for that aim. In this tutorial however we use the subject to show how the vertical is addressed in the processes library.

The 'depth of computational elements' as computed in 5.1 is equal to the 'total water depth' for depth averaged models. For layered models however the 'total water depth' is the sum of all depths of all layers on top of each other.

We must make some restrictions here. The first is that it is only possible for a process routine to compute total depth in a layered system. The water quality module allows for arbitrarily shaped and stacked computational elements, but with a free vertical topology (e.g. one element having more elements on top of it and covering only part of another element that also expands to the side of it) it is impossible to program the intelligence to compute total water depth from individual depths. Secondly, although not impossible, it is difficult to program intelligence to compute total water depth if the (exchange pointers between the) layers are not stored in either a systematic order form top to bottom or from the bottom to the top. WAQ at the moment generally uses the order that the first layer is the surface water layer and the last layer is the bottom layer and also stores its pointers that way, but the routine that we will develop will also work for the other way around. The routine that we will develop will also work for incomplete layers (e.g. strictly horizontal layers that may disappear partly at shallow locations, so called z-coordinates contrary to σ-coordinates). The procedure will be developed along the following lines:

- The depth of each computational element will be copied to the 'total depth' array. If there is no vertical in the model, the procedure stops.
- The vertical 'from' and 'to' exchange pointers will be used to sum the depth of the 'from' element to that of the 'to' element and set the depth of the 'from' element to zero. After this step is completed, either at the top or at the bottom now the total water depth is accumulated and at the remainder of the water column the depth is zero. For dry parts of incomplete layers, it is assumed that zero depth was stored in the array locations (so zero volume of the computational element). This requirement can be relaxed later on, because then the options to check whether a computational element is dry or wet etc. will be discussed.
- The vertical 'from' and 'to' pointer table is travelled in reversed order and all depth value



```
* * * Top of File * * *
      SUBROUTINE TDEPTH     ( PMSA   , FL     , IPOINT , INCREM, NOSEG ,
     +                        NOFLUX , IEXPNT , IKNMRK , NOQ1   , NOQ2  ,
     +                        NOQ3   , NOQ4   )
!DEC$ ATTRIBUTES DLLEXPORT, ALIAS: 'TDEPTH' :: TDEPTH
C
C*******************************************************************************
C
      IMPLICIT NONE
C
C     Type    Name         I/O Description
C
      REAL(4) PMSA(*)     !I/O Process Manager System Array, window of routine to process library
      REAL(4) FL(*)       ! O  Array of fluxes made by this process in mass/volume/time
      INTEGER IPOINT(  2) ! I  Array of pointers in PMSA to get and store the data
      INTEGER INCREM(  2) ! I  Increments in IPOINT for segment loop, 0=constant, 1=spatially varying
      INTEGER NOSEG       ! I  Number of computational elements in the whole model schematisation
      INTEGER NOFLUX      ! I  Number of fluxes, increment in the FL array
      INTEGER IEXPNT(4,*) ! I  From, To, From-1 and To+1 segment numbers of the exchange surfaces
      INTEGER IKNMRK      ! I  Active-Inactive, Surface-water-bottom, see manual for use
      INTEGER NOQ1        ! I  Nr of exchanges in 1st direction, only horizontal dir if irregular mesh
      INTEGER NOQ2        ! I  Nr of exchanges in 2nd direction, NOQ1+NOQ2 gives hor. dir. reg. grid
      INTEGER NOQ3        ! I  Nr of exchanges in 3rd direction, vertical direction, pos. downward
      INTEGER NOQ4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use only)
      INTEGER IPNT(  2)   !    Local work array for the pointering
      INTEGER ISEG        !    Local loop counter for computational element loop
C
C*******************************************************************************
C
C     Type    Name         I/O Description                                        Unit
C
      REAL(4) Depth       ! I  Water depth of this computational element          (m)
      REAL(4) Tot_Depth   ! O  Total water depth at this horizontal location      (m)
      INTEGER IQ          !    Local loop counter for exchanges loop
      INTEGER Ifrom, Ito  !    Local help variables with 'from' and 'to' comp.element numbers
C
C*******************************************************************************
C
      IPNT        = IPOINT
C
C       Copy the depth of computational elements in the 'Total Depth' array
C                (Somewhat expanded to show meaning of array locations)
C
      DO 10 ISEG = 1 , NOSEG
         Depth      = PMSA( IPNT(1)-1 + ISEG )
         Tot_Depth        =          Depth
         PMSA( IPNT(2)-1 + ISEG ) = Tot_Depth
   10 CONTINUE
C
C       Sum the depths according to the vertical pointer
C
      DO 20 IQ = NOQ1+NOQ2+1 , NOQ1+NOQ2+NOQ3
         Ifrom = IEXPNT(1,IQ)
         Ito   = IEXPNT(2,IQ)
         IF ( Ifrom .GT. 0 .AND. Ito .GT. 0 ) THEN
            PMSA( IPNT(2)-1 + Ito )   = PMSA( IPNT(2)-1 + Ito ) +
     *                                  PMSA( IPNT(2)-1 + Ifrom )
            PMSA( IPNT(2)-1 + Ifrom ) = 0.0
         ENDIF
   20 CONTINUE
C
C       Fill all Total_Depth values in reversed order
C
      DO 30 IQ = NOQ1+NOQ2+NOQ3 , NOQ1+NOQ2+1 , -1
         Ifrom = IEXPNT(1,IQ)
         Ito   = IEXPNT(2,IQ)
         IF ( Ifrom .GT. 0 .AND. Ito .GT. 0 ) THEN
            IF ( PMSA( IPNT(2)-1 + Ito ) .GT. 0.0 )
     *           PMSA( IPNT(2)-1 + Ifrom ) = PMSA( IPNT(2)-1 + Ito )
         ENDIF
   30 CONTINUE
C
      RETURN
      END
* * * End of File * * * |
```

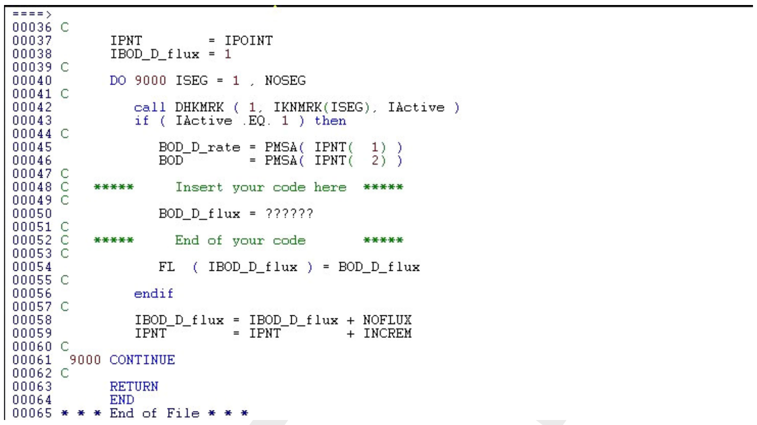
Figure 5.2: *Code to compute the total water depth at all locations*

larger than zero in a 'to' element is copied to the location for the 'from' element. The result of this is that in all locations along the water column the same 'total depth' of that water column is stored.

We may call our Fortran routine for this process TDEPTH and call the process TotDepth. We will only have one input variable, the already existing item 'Depth'. We will produce one output variable, the already existing item 'TotDepth'. We will produce no fluxes. The code looks like in Figure 5.2.

Remarks:

- ◇ The loops with labels 20 and 30 are automatically skipped if NOQ3 equals zero (so if no vertical is present).
- ◇ Here (and also in 5.2 to compute the velocities) the construction with the IPNT and the INCREM array are not used any more. That construction is aimed to deal with spatially constant values (INCREM = 0) and spatially varying values (INCREM = 1) in the same routine. This allows for process routines that operate equally well for spatially constant steering (as may be used in some applications) as for spatially variable steering (as may be used in other applications). The fact that this construction is not used in sections 5.2 and 5.3 reflects the fact that it is assumed that velocities, flows, exchange areas and water depths will always be spatially distributed variables.



```
====>
00036 C
00037       IPNT        = IPOINT
00038       IBOD_D_flux = 1
00039 C
00040       DO 9000 ISEG = 1 , NOSEG
00041 C
00042          call DHKMRK ( 1, IKNMRK(ISEG), IActive )
00043          if ( IActive .EQ. 1 ) then
00044 C
00045             BOD_D_rate = PMSA( IPNT(  1) )
00046             BOD        = PMSA( IPNT(  2) )
00047 C
00048 C   *****     Insert your code here  *****
00049 C
00050             BOD_D_flux = ??????
00051 C
00052 C   *****     End of your code       *****
00053 C
00054             FL  ( IBOD_D_flux ) = BOD_D_flux
00055 C
00056          endif
00057 C
00058          IBOD_D_flux = IBOD_D_flux + NOFLUX
00059          IPNT        = IPNT        + INCREM
00060 C
00061  9000 CONTINUE
00062 C
00063       RETURN
00064       END
00065 * * * End of File * * *
```

Figure 5.3: *Code modified with lines for computation at active segments only*

5.4 The feature or segment property

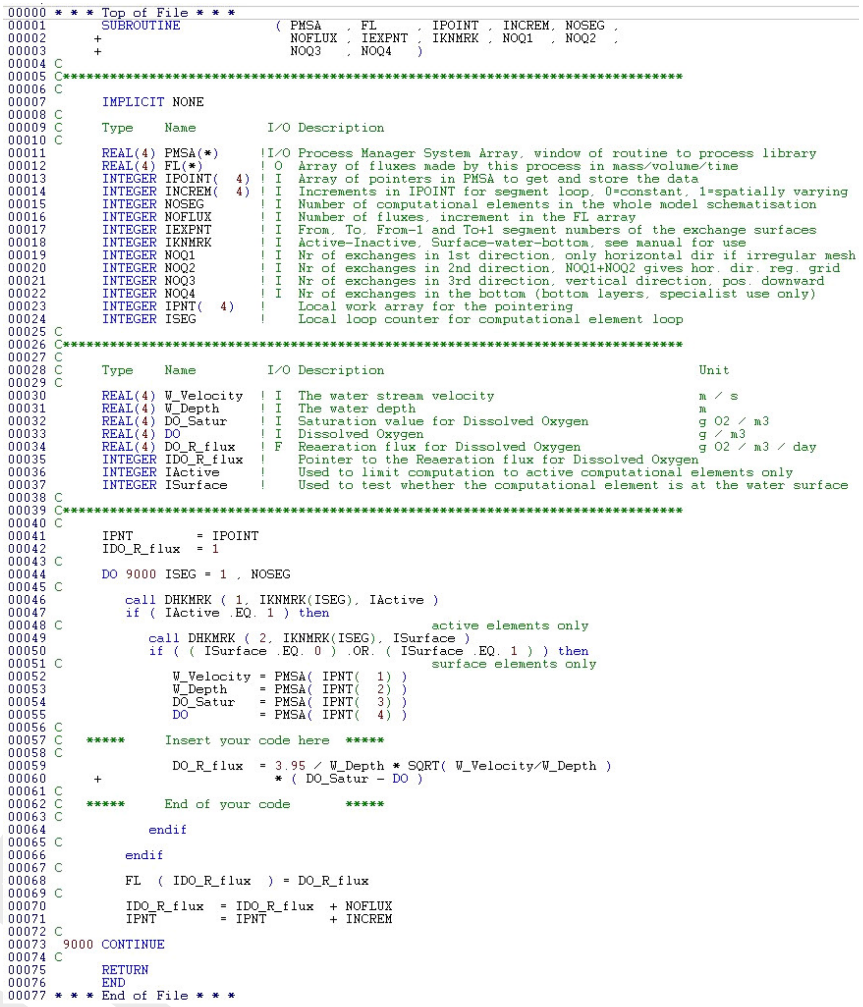
Up to now all computational elements were considered the same. They however may not be the same. In some Delft3D schematisations a full matrix is used with roughly half of the computational elements being active and the others being inactive. One may want to limit water quality computations to active elements only. Speaking about Delft3D, the reaeration as introduced in previous BOD-DO example only works at the water surface. This means that we want to 'see' whether our computational element is a water surface element (and equally for any bottom oxygen demand that you may whish to introduce) you want to see whether a computational element is at the bottom. There are generic approaches for this, allowing your model to automatically work correct both in depth averaged mode and in modes with a schematisation between the surface and the bottom like in 2D-vertical and in 3D models.

Together with the software of you developer version, a set of Fortran subroutines is delivered (in source code). You must compile them together with your process routines to the Windows 'dll' or to the Linux 'run-time library' that you will use for modelling. One of the routines is the DHKMRK routine. It looks into a 'segment property'- or 'feature'-array for values. This IKNMRK array is one of the arguments to your own process routine. You invoke the routine with: CALL DHKMRK (feature number, IKNMRK(ISEG), result). The middle argument comes from the argument and the integer ISEG is the linear segment number like in the BOD-DO example.

Active or inactive computational elements

A call to DHKMRK with feature number 1 returns integer one (1) for active computational elements and zero (0) for elements that are not active at that time. So the computation is limited to active computational elements only by the insertion of code like in Figure 5.3.

Be aware! The 'endif' of this 'if' statement should be placed before all pointers are incremented (lines 58 and 59 in Figure 5.3). The issue was to skip inactive computational elements. Those computational elements are there in storage and in the linear numbering, so you should skip the inactive elements by incrementing the pointers only.



```
* * * Top of File * * *
      SUBROUTINE            ( PMSA   , FL     , IPOINT , INCREM, NOSEG ,
     +                        NOFLUX , IEXPNT , IKNMRK , NOQ1   , NOQ2   ,
     +                        NOQ3   , NOQ4   )
C
C*******************************************************************************
C
      IMPLICIT NONE
C
C     Type    Name         I/O Description
C
      REAL(4) PMSA(*)     !I/O Process Manager System Array, window of routine to process library
      REAL(4) FL(*)       ! O  Array of fluxes made by this process in mass/volume/time
      INTEGER IPOINT(  4) ! I  Array of pointers in PMSA to get and store the data
      INTEGER INCREM(  4) ! I  Increments in IPOINT for segment loop, 0=constant, 1=spatially varying
      INTEGER NOSEG       ! I  Number of computational elements in the whole model schematisation
      INTEGER NOFLUX      ! I  Number of fluxes, increment in the FL array
      INTEGER IEXPNT      ! I  From, To, From-1 and To+1 segment numbers of the exchange surfaces
      INTEGER IKNMRK      ! I  Active-Inactive, Surface-water-bottom, see manual for use
      INTEGER NOQ1        ! I  Nr of exchanges in 1st direction, only horizontal dir if irregular mesh
      INTEGER NOQ2        ! I  Nr of exchanges in 2nd direction, NOQ1+NOQ2 gives hor. dir. reg. grid
      INTEGER NOQ3        ! I  Nr of exchanges in 3rd direction, vertical direction, pos. downward
      INTEGER NOQ4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use only)
      INTEGER IPNT(  4)   !    Local work array for the pointering
      INTEGER ISEG        !    Local loop counter for computational element loop
C
C*******************************************************************************
C
C     Type    Name         I/O Description                                         Unit
C
      REAL(4) W_Velocity  ! I  The water stream velocity                           m / s
      REAL(4) W_Depth     ! I  The water depth                                     m
      REAL(4) DO_Satur    ! I  Saturation value for Dissolved Oxygen               g O2 / m3
      REAL(4) DO          ! I  Dissolved Oxygen                                    g / m3
      REAL(4) DO_R_flux   ! F  Reaeration flux for Dissolved Oxygen                g O2 / m3 / day
      INTEGER IDO_R_flux  !    Pointer to the Reaeration flux for Dissolved Oxygen
      INTEGER IActive     !    Used to limit computation to active computational elements only
      INTEGER ISurface    !    Used to test whether the computational element is at the water surface
C
C*******************************************************************************
C
      IPNT        = IPOINT
      IDO_R_flux  = 1
C
      DO 9000 ISEG = 1 , NOSEG
C
         call DHKMRK ( 1, IKNMRK(ISEG), IActive )
         if ( IActive .EQ. 1 ) then
C                                               active elements only
            call DHKMRK ( 2, IKNMRK(ISEG), ISurface )
            if ( ( ISurface .EQ. 0 ) .OR. ( ISurface .EQ. 1 ) ) then
C                                               surface elements only
               W_Velocity = PMSA( IPNT(  1) )
               W_Depth    = PMSA( IPNT(  2) )
               DO_Satur   = PMSA( IPNT(  3) )
               DO         = PMSA( IPNT(  4) )
C
C   *****     Insert your code here  *****
C
               DO_R_flux  = 3.95 / W_Depth * SQRT( W_Velocity/W_Depth )
     +                    * ( DO_Satur - DO )
C
C   *****     End of your code       *****
C
            endif
C
         endif
C
         FL  ( IDO_R_flux  ) = DO_R_flux
C
         IDO_R_flux  = IDO_R_flux  + NOFLUX
         IPNT        = IPNT        + INCREM
C
 9000 CONTINUE
C
      RETURN
      END
* * * End of File * * *
```

Figure 5.4: *Sample Fortran file for reaeration of Dissolved Oxygen*

Computational elements at the water surface or at the water bottom

The call to DHKMRK with feature number 2 as argument returns an integer that can be tested for its value as follows:

◇ Value is 0; the computational element has both a water surface an a bottom (typical for depth averaged modelling)
◇ Value is 1; the computational element has a water surface but no water bottom
◇ Value is 2; the computational element has no water surface and no water bottom
◇ Value is 3; the computational element has no water surface, but it has a water bottom.

The process routine for reaeration that was not furthermore elaborated in detail in Chapter 4, could look like in Figure 5.4.

If you carefully proceed along these lines of computer code generation, you make your modelling library functioning equally correct for vertically average modelling and for layered modelling of e.g. stratified lakes or 3D estuaries. The standard Deltares processes library e.g. is developed in this way, so exactly the same library is used in the Delft1D/2D - SOBEK system and in the Delft3D system, making water quality modelling completely consistent.

5.5 Settling from the water column towards the bottom

The BOD-DO example up to now was limited to the water column either as one depth averaged computational element or as layered system of computational elements. Part of the water quality substances may not be dissolved but may be particulate. Especially in BOD as raw sewage a substantial part may be released as particulate fraction. The settling tanks in a treatment plant may reduce the release of particulates, but may not fully eliminate them. And why wouldn't you want to apply the water quality model to a settling tank? Roughly there are two approaches to the settling of substances:

- ◇ You define additional substances for the settled substances. Those substances are NOT transported with the water and are only found in the bottom layer of a 3D system. Settling then means a migration from the water-bound normal substance to the settled-substance. This is the way the simple sediment model in Deltares processes library works. All particulate substances in the water have their S1 and S2 equivalent for the first and the second sediment layers. But also all sediment models with own, often analytical, solutions of the behaviour IN the bottom work this way. In dedicated versions of the Deltares processes library this holds for the European ERSEM model and for the US-HydroQual Sediment Flux Model (SFM).
- ◇ You define bottom layers underneath the water phase. Now no additional substances are defined, but the same substances also exist in bottom layers. The settling from the water column now is just a transport process from a water bound computational element to a computational element in the bottom (and through benthic mixing by worms or other invertebrates the substance can reach lower bottom layers as well). This approach is used in the 'Layered bottom' version of the Deltares processes library

It is to be expected that the second approach of a layered bottom will replace the first approach of separated bottom models with own substances in the bottom, within several years. Up to now however most models behave with the first approach so this tutorial will focus on the first approach. There is another reason to do so. Often an approach with one or two bottom layers is simpler to implement and adequate for the problem at hand, while the layered bottom approach requires additional schematisation of the water bottom and of the transport processes in the water bottom. The basic reason to include the water bottom in Dissolved Oxygen models is twofold:

- ◇ You want to model the disappearance of organic Carbon and Nitrogen towards the water bottom in such a way that it does not influence the water phase any more. This is the case if part of the organic matter will migrate to the confined bottom to become oil or natural gas in the end.
- ◇ You want to include a reduced decomposition of the material that settled to the bottom. In such a way the pool of dead organic matter increases rapidly at the die off of algae blooms and reduces gradually over the months to follow. This is the origin of the so called 'bottom oxygen demand'.

Oxygen models exist where a bottom-oxygen-demand is included as a parameter, as a coefficient that can be set to e.g. 1 $gO_2/m^2/day$ or so. It must be noted that the only source of organic matter that can cause such a 'bottom-oxygen-demand' is the organic matter that settled towards the bottom at earlier days. That is why this approach is not recommended. It allows you to let the bottom demand more oxygen than would be justified by the settling of organic matter in the past. It also allows you accumulate dead organic matter on the bottom without knowing it if the bottom oxygen demand is much lower than would be justified by the settling of organic matter in the past. It basically is a violation of the law of conservation of mass of organic matter.

The settling discussed here can be included as a process by the following steps:

1 Define a new substance being the BOD at the water bottom called BOD_Sed. Take care that you specify that it is NOT transported by the water (see section 4.3)
2 Add the computation of the shear stress τ to the process that computes the water velocity (see section 5.2). The formula can e.g. be

$$\tau = \frac{\rho \cdot g \cdot \text{Velocity}^2}{\text{Chezy}^2} \tag{5.2}$$

More advanced approaches are possible with additions of the effects of wind generated waves and the influences of shipping. It is also possible to make a separate process to compute τ with the water velocity as an input. All up to you.
3 Make a process and a coorsponding Fortran source code that computes the settling of BOD e.g. according to the following formula:

$$fSed = v_{sed} \cdot \max\left(0, 1 - \frac{\tau}{\tau_c}\right) \cdot BOD \tag{5.3}$$

with τ_c is the critical shear stress for sedimentation and vsed is the settling velocity in still water.
4 Modify the process such that it only works for those computational elements that have a water bottom (see section 5.5). This makes your model working equally well for 1D, 2D and 3D applications.
5 Set the stoichimetry (see section 4.5) such that the settling acts with factor -1 on BOD and with a factor of $+1$ on BOD_Sed.

In the same way a resuspension process can be constructed that takes material from the BOD_Sed pool towards the BOD pool as soon as the shear stress is so high that resuspension occurs.

5.6 The vertical velocity in 3D models

In section 5.5 we discussed the settling of particulates towards the water bottom. It should however be noted that particulates generally also have a vertical gravity velocity in the water column that must be taken into account especially in 3D and in vertical 2D or 1D models. This is special, because all processes and items up to now were computed per computational element whereas transport velocities and flows are defined on interfaces between computational elements. A first encounter of this phenomena was in section 5.2 where we computed the segment velocity using the flows and the cross-sectional areas at the interfaces that came from the Hydrodynamic model. Now we will produce ourselves an additional settling velocity in the water column. The procedure is as following:

1 Define a process that will compute the settling velocity, with all input that you might need to do so. In its simplest form this input it is just one coefficient, the settling velocity of that particulate. However for e.g. cohesive sediments many formulations are known that relate the settling velocity to the concentration of the sediment, either positively through cohesion of single particles or negatively through hindered settling.
2 Define as output item for this process the additional settling velocity at the interfaces (see section 4.4). But now you specify that this output item should act on exchanges (see Figure 4.12). Now something special happens. You are asked whether you want to specify a velocity type term or a diffusion (mixing) type term at the interfaces or no specific term at all. You select 'velocity'.
3 Make a Fortran routine that computes this velocity. For our vertical velocity, you set the value at 0.0 for the first NOQ1+NOQ2 entries in the array (those are the horizontal exchanges) and you fill in the computed vertical velocity in the NOQ3 entries. Be sure that

you use the correct sign (if the first layer is on top and the NOQ3 pointers are administrated positively from top to bottom (as is common in the standard processes library of Deltares (WL | Delft Hydraulics)) then a positive settling velocity acts towards the bottom (and a negative velocity would e.g. account for the floatation like with algae and oil)).

4 You see that a stochiometry button appeared. You press it and you select all substances that you want to be subject to this settling velocity and you give as weight coefficient 1.0.

Remarks:

◇ The array-space that is available for the result of your process Fortran routine generally is equal to the amount of computational elements. By pressing the button *acts on exchanges*, the array space becomes as large as the number of exchanges.

◇ By the selection of 'velocity' in step 2, you cause this term to be added to the water velocity in the advection diffusion solver for this substance. This means that the term is subject to all numerics of the selected advection diffusion solver. This is especially important for settling velocities, since they may be so high that explicit schemes along the vertical would require a much to small computational time step. You solve this problem by selection for you simulation of one of the numerical scheme's that deal with the vertical in an implicit way.

◇ Had you selected a 'diffusion' type term, the result would be added to the diffusion term in the advection diffusion equation and again the term was taken into account by the advection diffusion solver. If you selected neither a velocity nor a diffusion type of term, then the term will not be added to the advection diffusion equation, but it will just be there, to serve as input for other processes. The latter situation can e.g. be the case for an item as 'the density gradient at a cross-sectional area'. This density gradient can then be used in commonly used formula's for reduced vertical diffusion due to stratification with as net result a vertical diffusion type of term. It is however also possible to specify a velocity of salmon towards a freshwater source if you first computed the salinity gradient as cross-sections.

5.7 Further performance improvement

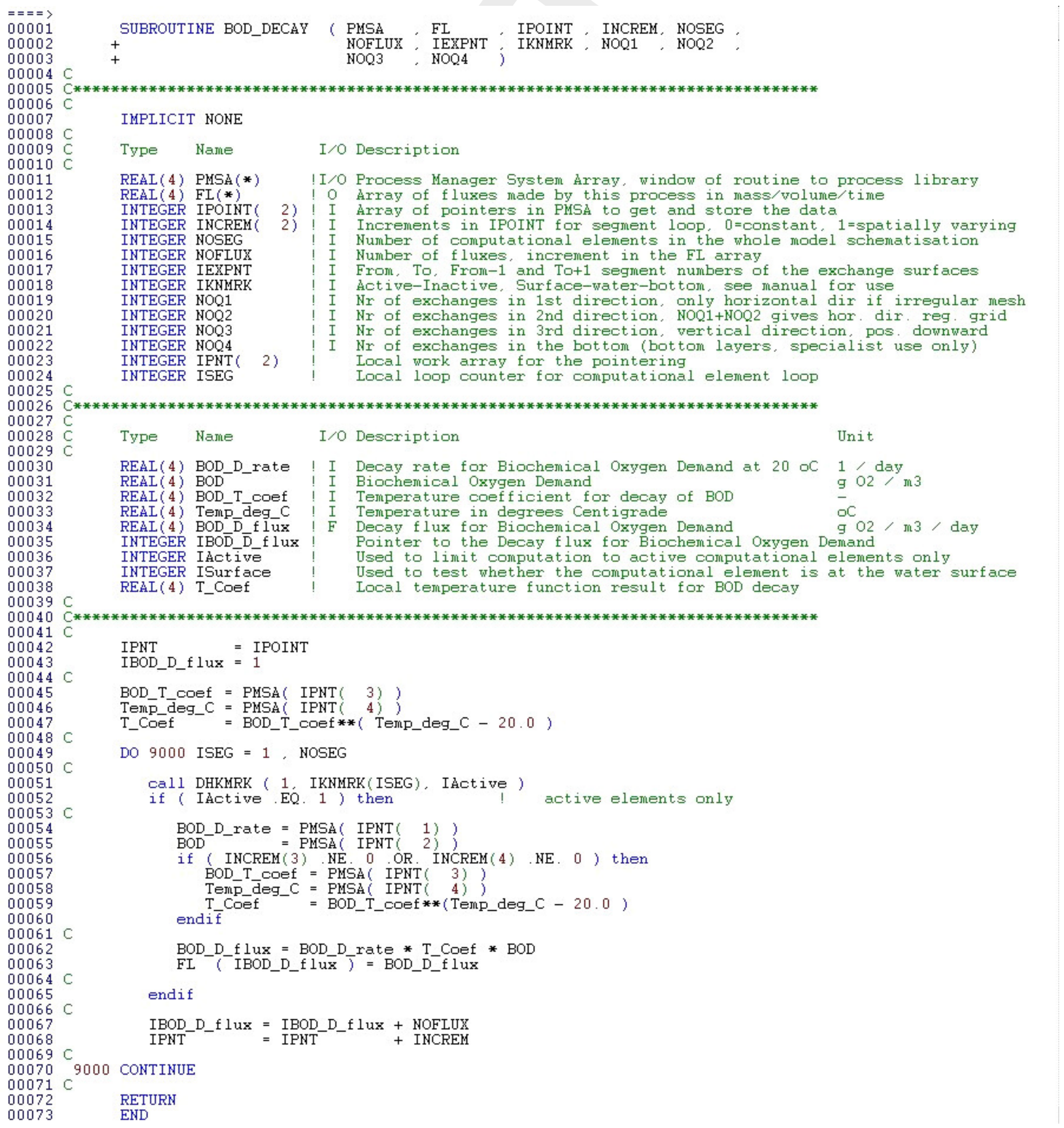
If we look at the BOD-Decay formula

$$\frac{\partial BOD}{\partial t} = -k_1 \cdot BOD \tag{5.4}$$

then the decay constant k_1 is a constant. In reality it is a bacterial process that is temperature dependent with

$$k_1 = k_{20} \cdot T_{coef}^{(T-20)}. \tag{5.5}$$

Here T_{coef} is the temperature dependency (generally 1.047 for decay of organic matter), T the temperature in degrees centigrade and k_{20} the decay constant at 20 °C. This formulation is easily included in the source code with help of the previously given examples. If the temperature T is not modelled, but specified as a constant for the whole river or estuary, then straight forward programming will result in the evaluation of the same $T_{coef}^{(T-20)}$ for every computational element and powers with real numbers are computation time consuming. In Figure 5.5 a code example of temperature dependent BOD decay is given that evaluates the temperature function only once for the whole area if the temperature is the same everywhere. It makes use of the property that the INCREM entry for items is 0 if an item is specified as a spatially constant (e.g. as a single time function for the whole area). The code also works if T is different per location. That is important, never lean on limitations, always insert general and generic code into your library, the you will benefit most from its generality.



```
====>
      SUBROUTINE BOD_DECAY  ( PMSA   , FL     , IPOINT , INCREM, NOSEG ,
     +                        NOFLUX , IEXPNT , IKNMRK , NOQ1   , NOQ2  ,
     +                        NOQ3   , NOQ4   )
C
C*******************************************************************************
C
      IMPLICIT NONE
C
C     Type    Name         I/O Description
C
      REAL(4) PMSA(*)     !I/O Process Manager System Array, window of routine to process library
      REAL(4) FL(*)       ! O  Array of fluxes made by this process in mass/volume/time
      INTEGER IPOINT(  2) ! I  Array of pointers in PMSA to get and store the data
      INTEGER INCREM(  2) ! I  Increments in IPOINT for segment loop, 0=constant, 1=spatially varying
      INTEGER NOSEG       ! I  Number of computational elements in the whole model schematisation
      INTEGER NOFLUX      ! I  Number of fluxes, increment in the FL array
      INTEGER IEXPNT      ! I  From, To, From-1 and To+1 segment numbers of the exchange surfaces
      INTEGER IKNMRK      ! I  Active-Inactive, Surface-water-bottom, see manual for use
      INTEGER NOQ1        ! I  Nr of exchanges in 1st direction, only horizontal dir if irregular mesh
      INTEGER NOQ2        ! I  Nr of exchanges in 2nd direction, NOQ1+NOQ2 gives hor. dir. reg. grid
      INTEGER NOQ3        ! I  Nr of exchanges in 3rd direction, vertical direction, pos. downward
      INTEGER NOQ4        ! I  Nr of exchanges in the bottom (bottom layers, specialist use only)
      INTEGER IPNT(  2)   !    Local work array for the pointering
      INTEGER ISEG        !    Local loop counter for computational element loop
C
C*******************************************************************************
C
C     Type    Name         I/O Description                                        Unit
C
      REAL(4) BOD_D_rate  ! I  Decay rate for Biochemical Oxygen Demand at 20 oC  1 / day
      REAL(4) BOD         ! I  Biochemical Oxygen Demand                          g O2 / m3
      REAL(4) BOD_T_coef  ! I  Temperature coefficient for decay of BOD           -
      REAL(4) Temp_deg_C  ! I  Temperature in degrees Centigrade                  oC
      REAL(4) BOD_D_flux  ! F  Decay flux for Biochemical Oxygen Demand           g O2 / m3 / day
      INTEGER IBOD_D_flux !    Pointer to the Decay flux for Biochemical Oxygen Demand
      INTEGER IActive     !    Used to limit computation to active computational elements only
      INTEGER ISurface    !    Used to test whether the computational element is at the water surface
      REAL(4) T_Coef      !    Local temperature function result for BOD decay
C
C*******************************************************************************
C
      IPNT        = IPOINT
      IBOD_D_flux = 1
C
      BOD_T_coef  = PMSA( IPNT(  3) )
      Temp_deg_C  = PMSA( IPNT(  4) )
      T_Coef      = BOD_T_coef**( Temp_deg_C - 20.0 )
C
      DO 9000 ISEG = 1 , NOSEG
C
         call DHKMRK ( 1, IKNMRK(ISEG), IActive )
         if ( IActive .EQ. 1 ) then           !    active elements only
C
            BOD_D_rate = PMSA( IPNT(  1) )
            BOD        = PMSA( IPNT(  2) )
            if ( INCREM(3) .NE. 0 .OR. INCREM(4) .NE. 0 ) then
               BOD_T_coef = PMSA( IPNT(  3) )
               Temp_deg_C = PMSA( IPNT(  4) )
               T_Coef     = BOD_T_coef**(Temp_deg_C - 20.0 )
            endif
C
            BOD_D_flux = BOD_D_rate * T_Coef * BOD
            FL  ( IBOD_D_flux ) = BOD_D_flux
C
         endif
C
         IBOD_D_flux = IBOD_D_flux + NOFLUX
         IPNT        = IPNT        + INCREM
C
 9000 CONTINUE
C
      RETURN
      END
```

Figure 5.5: *Subroutine for temperature dependent BOD Decay with optimisations*

5.8 Retrieving information from the hydrodynamics

In earlier sections of this chapter we showed how we could derive the segment velocity, the segment depth and the total water depth from the terms that were delivered by the hydrodynamic model. Here an overview is given of all the terms that are available from the hydrodynamic model. There are 11 items in the processes library that are always present and that can be used. They are listed in Table 5.1.

Table 5.1: *Items that are available to the process library from the system*

Item	Dimension	Meaning
VOLUME	NOSEG	The volumes of the computational elements [m^3]
FLOW	NOQ	The flows between the computational elements [m^3/s]
XAREA	NOQ	The exchange surfaces between computational elements [m^2]
XLENFROM	NOQ	The distance of mid of ‘from’ element to exchange surface [m]
XLENTO	NOQ	The distance of mid of ‘to’ element to exchange surface [m]
SURF	NOSEG	The horizontal surface area of the computational elements
IDT	1	The system time step for the Advection Diffusion solver [s]
DELT	1	The system time step fro water quality processes [d]
ITIME	1	The time in simulation from reference [s]
ITSTOP	1	The simulation start time from reference [s]
ITSTRT	1	The simulation stop time from reference [s]

If one or more of the items of Table 5.1 are specified as input for a process the system considers them as resolved by the model. So you can assume them to be available. It is also possible to obtain other information from the hydrodynamic modelling part. If you want to use e.g. the bottom shear stress in the Delft3D environment, then you define an input to your process with this name and at modelling time you select the item to be specified as user input. In the Delft3D user interface you couple the datafile containing this bottom shear stress to the item with this name of your processes library and you can use the item. The difference with the items in Table 5.1 is that those in Table 5.1 are always present, both in the 1D-2D SOBEK environment and in the Delft3D environment. The availability of other items depends on the setting of your hydrodynamic run. If you model e.g. thermal stratification with your Delft3D hydrodynamic model, then you can specify the temperature result of that model as item for your Delft3D water quality simulation. If you do not model temperature with the hydrodynamic model, then you will have to model temperature with your water quality model, or just specify a temperature time function for the area in the user interface.

We however need to obtain the velocity and the depth and Figures 5.1 and 5.2 give routines that do that job.

5.9 Derived entities

Processes and their input and output are only displayed in the user interface if they affect substances in the model. There however also may be derived quantities (e.g. Secchi-depth as result of available particulates in the water or Chlorophyll as result of Algae-Carbon in the water) that are not used in that form by the substances. In that case they would be 'unreachable' for the user in the user interface. You can prevent this by specifying those derived properties as a flux and let them act on all substances where they may be associated with. As weighing factor for the flux you use however 0.0, so no actual flux takes place. The process however now appears in the list of processes that can be switched on for a substance and the Secchi Depth and the Chlorophyll can be selected as output variable for graphical post-processing. It is one of the very few tricks that is required in the furthermore very straight forward definition of substances, processes, fluxes and input and output items.

6 Using waqpb export and import tools

6.1 Background

An alternative method to specify processes

As described in earlier chapters OpenPLCT can be used to define new processes, based on a DLL created from Fortran code and a process definition file. This method is ideally suited for less experienced users, but only allows small interventions and simpler processes.

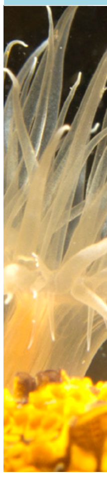
An alternative method is to regenerate the process definition files including the new or modified process(es). This also allows for changing input parameters of existing processes. The waqpb_export and waqpb_import tools allow the user to do this. These tools have been made generally available (versions of 2022 and later). This method is recommended for experienced users only. This functionality is currently only supported under Windows. Note that the process definition files which are generated under Windows can be used on Linux, and vice versa.

6.2 Workflow

The data for the process definition files are contained in the <*.csv> files, which are present in the x64/dwaq/default/csvFiles folder. The process definition files are located inside the x64/dwaq/default folder.

The workflow in generating a new process definition file is:

1 navigate to the x64/dwaq/scripts folder.
2 open the <export_procdef_csvfiles.bat> script in a text editor
3 change the variable *serial* to a new value. Usually this is the date in the format yyyymmdd, possible followed by an iteration number.
4 run the script <export_procdef_csvfiles.bat>, either in a terminal or command prompt, which calls the waqpb_export tool. This script will export the <proc_def> files and create an ASCII file (<procesm.asc>) containing a list of all the processes and associated parameters.
5 copy the ASCII file <procesm.asc> to a new file, called <proces.asc>.
6 edit the new file <proces.asc> (add the desired new process, modify an existing process, etc.).
7 run the script <import_procesasc_changes.bat>, either in a terminal or command prompt, which calls the waqpb_import tool. This script will import the changes made in the <proces.asc> file.
8 run the script <export_procdef_csvfiles.bat> again, either in a terminal or command prompt. This will export the <proc_def> files and create a new <procesm.asc> ASCII file. The <proc_def> file in the x64/dwaq/default folder is directly updated and ready to be used for a new simulation.
9 compare the <procesm.asc> and <proces.asc> files to see if the changes are correctly included.



What if you make a mistake?

In the folder x64/dwaq/installation_default contains the original <*.csv> and <proc_def> files as backup. If you make a mistake while adapting the process definition files to your requirements, you can always return to the original state by copying all the files from the x64/dwaq/installation_default folder to the x64/dwaq/default folder. Note that then all changes will be lost.

The installation_default folder can also be used for comparing the adapted <.csv> files with the original <.csv> files, to see how the changes made in the <proces.asc> file are reflected in the process definition database.

Other remarks

It is also possible to edit the <*.csv> files directly. This is not for the faint-hearted, since the database can be rather ambiguous, and its workings mysterious. It does allow for even greater control.

Note that some changes in the <proces.asc> may not be not incorporated in the new version of process definition files by the waqpb_export and waqpb_import tools, as the <.csv> files will updated, rather than rewritten from scratch. An example of this is changing a default value to a different value. Comparing the <procesm.asc> and <proces.asc> files after rerunning the <export_procdef_csvfiles.bat> script will in such as case show clear differences, since the modified values are not reflected in the <procesm.asc> file. In such a case it is necessary to edit the <.csv> files directly.

References

O' Connor, D. and W. Dobbins, 1956. "The mechanism of reaeration in natural streams. ASCE. 82 (no. SA6): 469." *J.San.Eng. ASCE* 82 (no. SA6): 469.

Streeter, H. and E. Phelps, 1925. *Study of the pollution and natural purification of the Ohio river. III. Factors Concerned in the Phenomena of Oxidation and Reaeration.* Tech. Rep. U.S. Public Health Serv., Pub. Health Bulletin No. 146, U.S. Public Health Service, Washington D.C. (reprinted 1958).

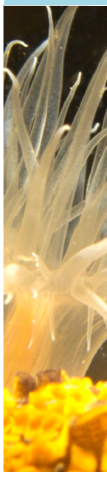


Photo by: Mathilde Matthijsse, www.zeelandonderwater.nl

Rijkswaterstaat
Ministry of Infrastructure and Water Management

PO Box 177
2600 MH Delft
Boussinesqweg 1
2629 HV Delft
The Netherlands

Deltares