

Elements of DELWAQ programmer's guide

Jos van Gils
Leo Postma

Title

Elements of DELWAQ programmer's guide

Client

Deltares

Pages

1

Keywords

Place keywords here

Summary

Place summary here

References

Place references here

Version	Date	Author	Initials	Review	Initials	Approval	Initials
	dec. 2012	Jos van Gils					
		Leo Postma					

State

draft

This is a draft report, intended for discussion purposes only. No part of this report may be relied upon by either principals or third parties.

Contents

1 Introduction	1
1.1 General	Error! Bookmark not defined.
1.2 A guide to this manual	Error! Bookmark not defined.
2 DELWAQ Flow Charts	3
2.1 DELWAQ as a module of Delft3D	3
2.2 DELWAQ as a module of Sobek2.xx	4
2.3 DELWAQ stand-alone	5
2.4 DELWAQ in single step mode	5
2.5 File names	6
3 Software structure	7
3.1 Key concepts, arrays and dimension variables	7
3.2 DELWAQ1.EXE	8
3.3 Real numbers memory management in DELWAQ2	9
3.4 DELWAQ2.EXE	10
3.5 Parallel processing	14
4 The Processes Library	15
4.1 General	15
4.1.1 Dynamic solvers	15
4.1.2 Steady state solvers	15
4.2 What is a process?	16
4.2.1 Process definition	17
4.2.2 FORTRAN code	18
4.3 The Open Processes Library (OPL)	20
4.4 Managing the processes library	20
4.4.1 Use the OPL GUI	20
4.4.2 Use the waqpb tools	22
4.5 The waqpb_import and waqpb_export tools	22
4.5.1 How to use waqPB_import	22
4.5.2 How to use waqPB_export	23
4.5.3 Definition of the ASCII proces definition file (new format)	24
4.5.4 Specific functionality for DUPROL	26

Appendices

A A-1

1 Introduction

Early 2013, the computational kernel called “DELWAQ” became open source. This programme is the water quality modelling programme of the Sobek and Delft3D software suites. This document discusses some elements of a programmer’s guide for DELWAQ. It has been drafted based on the presentations during the “D-WAQ Open Source” workshop organised during the Delft Software Days 2012.

In stand-alone mode, the programme DELWAQ consists of:

- DELWAQ1.exe: which reads the user input, provides feedback to the user and prepares the necessary files for the simulation;
- DELWAQ2.exe, including the processes library: which carries out the simulation and prepares output files.

For further backgrounds we refer to:

- the D-WAQ User Manual;
- the Open Processes Library User Manual;
- the description of the D-WAQ Input File;
- the description of the mass balances utility.

2 DELWAQ Flow Charts

2.1 DELWAQ as a module of Delft3D

Figure 2.1 shows the operation of DELWAQ as a part of Delft3D. This figure shows the following elements:

- the Graphical User Interface (GUI) of the Processes Library Configuration Tool (PLCT), which allows the user:
 - to select substances, processes, input items and output items from the processes library,
 - to store the selection in a file ("wq-model-a" etc. in Figure 2.1);
- the Graphical User Interface (GUI) of Delft3D-WAQ ("GUI" in Figure 2.1), which allows the user:
 - to specify a Delft3D-WAQ simulation;
 - to import results from a hydrodynamic model simulation (optionally more than one result);
 - to import the selections from the Processes Library (a "wq-model-a" file);
 - to import additional files with input data ("bulk model steering" etc. in Figure 2.1);
 - to store this information and save a DELWAQ input file;
- the processing of the input by DELWAQ1.exe;
- the simulation proper by DELWAQ2.exe, optionally using user defined dynamic link libraries (dll's) for additional waste load definitions and open processes library routines;
- the postprocessing of the output, for example with QuickPlot.

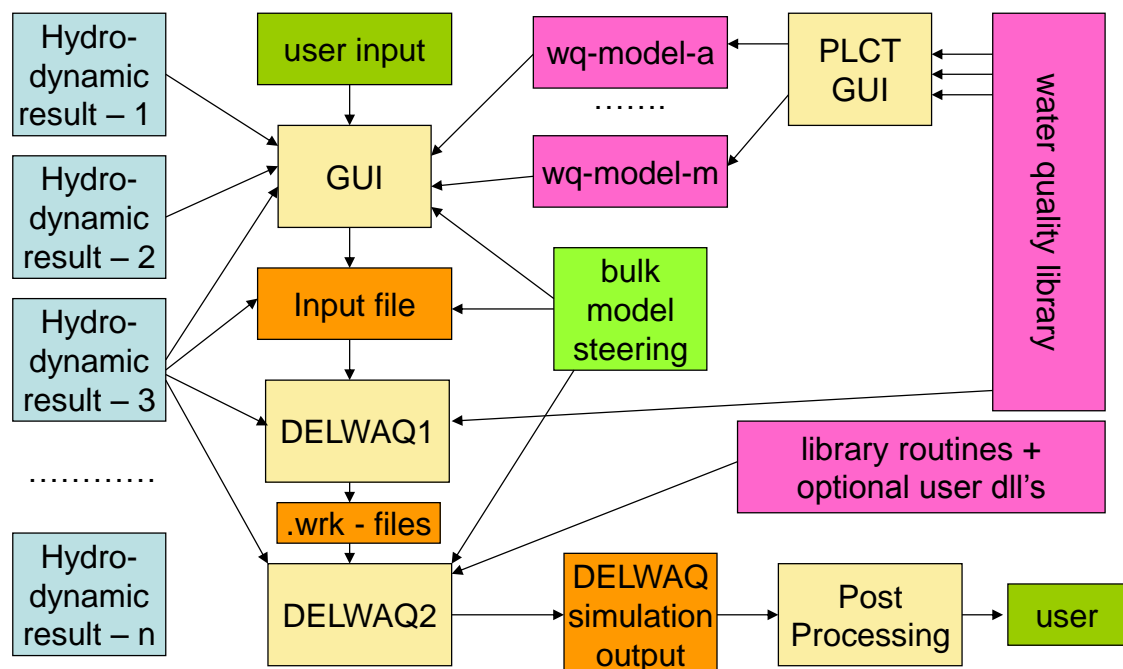


Figure 2.1 Operation flow chart of DELWAQ as a module of Delft3D

2.2 DELWAQ as a module of Sobek2.xx

Figure 2.2 shows the operation of DELWAQ as a part of Sobek. This figure shows the following elements:

- the Graphical User Interface (GUI) of the Processes Library Configuration Tool (PLCT), which allows the user:
 - to select substances, processes, input items and output items from the processes library,
 - to store the selection in a file (“wq-model-a” in Figure 2.2);
- the SOBEK hydrodynamic model which produces some parts of the DELWAQ input file;
- the Graphical User Interface (GUI) of Sobek (“GUI” in Figure 2.2), which allows the user:
 - to input various elements of the SOBEK-WQ simulation;
 - to store this information as parts of the DELWAQ input file;
- a static “template” input file for DELWAQ, where all simulation dependent parts are included;
- the processing of the input by DELWAQ1.exe;
- the simulation proper by DELWAQ2.exe, optionally using user defined dynamic link libraries (dll's) for additional waste load definitions and open processes library routines;
- the postprocessing of the output, for example with OdsView or the map-based user interface of SOBEK (Netter).

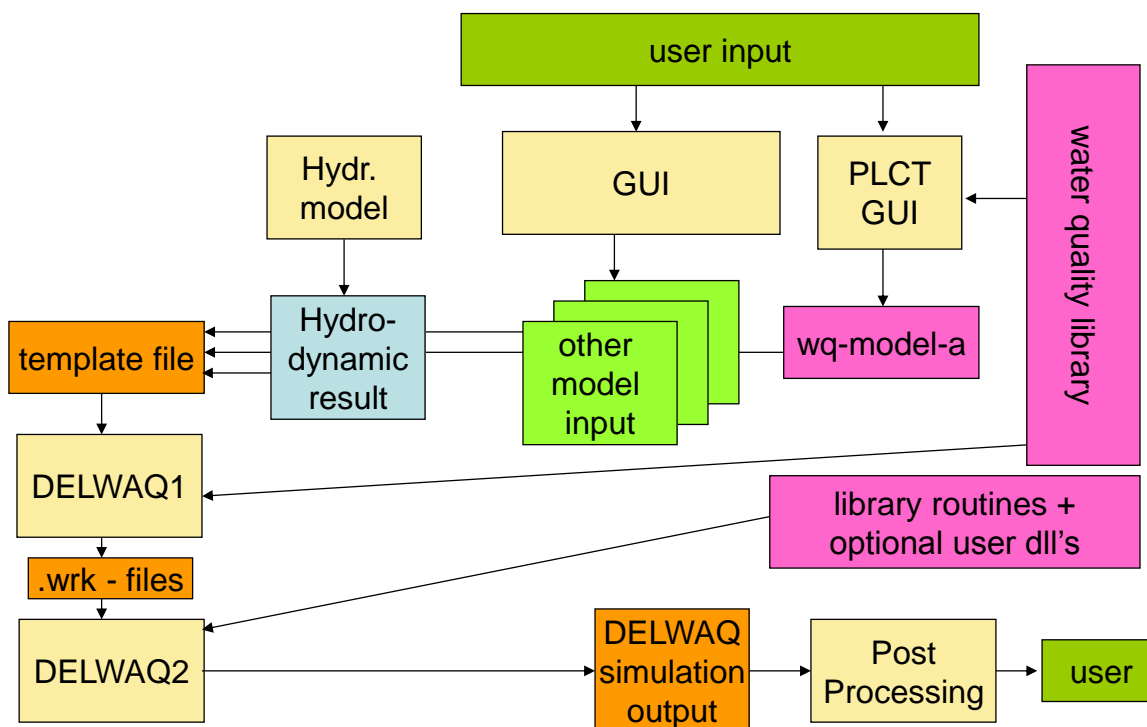


Figure 2.2 Operation flow chart of DELWAQ as a module of Sobek

2.3 DELWAQ stand-alone

Figure 2.3 shows the operation of DELWAQ as a stand alone simulation programme. This figure shows the following elements:

- the manual preparation of an input file by the user:
 - including results from a hydrodynamic model simulation;
 - including information to control the Processes Library;
 - including additional files with input data (“bulk model steering”);
- the processing of the input by DELWAQ1.exe;
- the simulation proper by DELWAQ2.exe, optionally using user defined dynamic link libraries (dll's) for additional waste load definitions and open processes library routines;
- the postprocessing of the output.

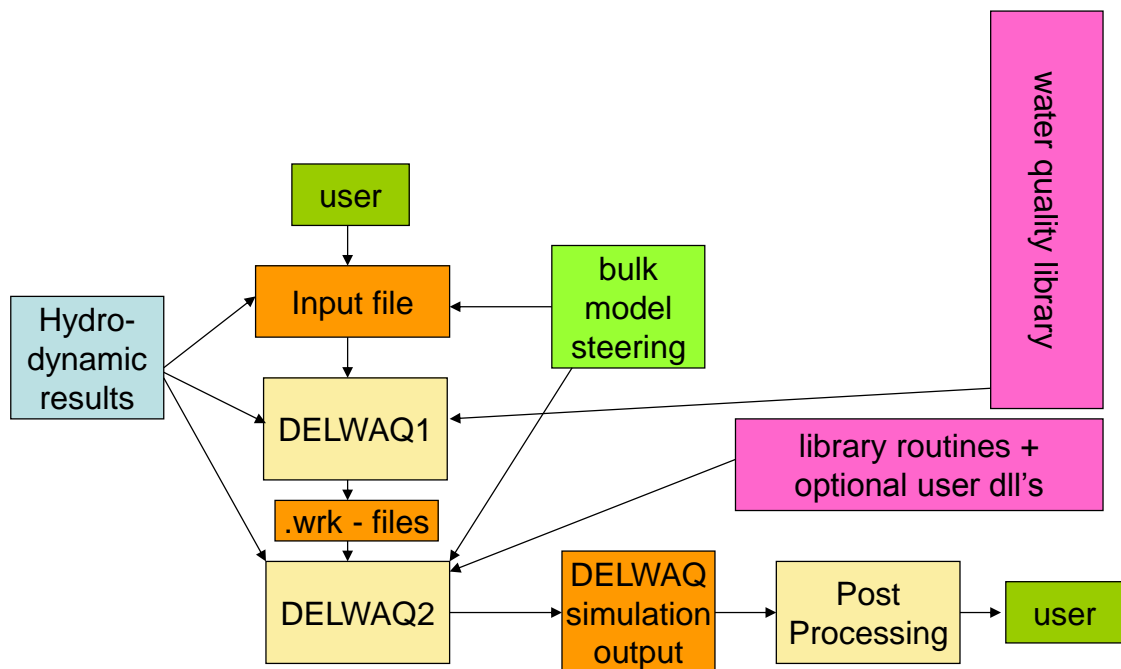


Figure 2.3 Operation flow chart of DELWAQ as a stand-alone programme

2.4 DELWAQ in single step mode

DELWAQ also supports the operation in single step mode, which implies that the simulation is carried out step by step, with a dynamic coupling to another programme (a hydrodynamic model in the example of Figure 2.4), which calls a DELWAQ2 dynamic link library.

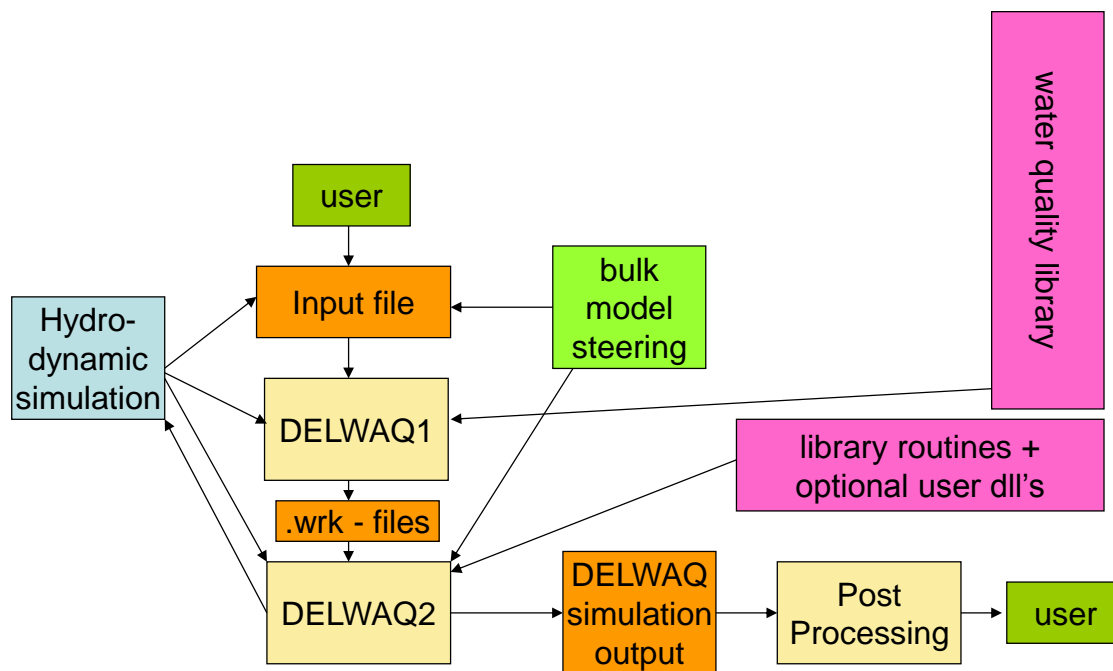


Figure 2.4 Operation flow chart of DELWAQ, with the simulation dynamically linked to a another programme

2.5 File names

Most of the files associated to DELWAQ are named using a run identification string ("runid") and a standard file extension. The run identification string is derived from the input file name by removing the extension ".inp". For further information we refer to the D-WAQ User Manual.

3 Software structure

3.1 Key concepts, arrays and dimension variables

DELWAQ simulates one or more state variables or “substances”. These substances can be “active”, which means that they are transported between grid cells or “inactive”, which means that they are not transported.

DELWAQ operates on an unstructured calculation grid or “schematisation”. The basic schematisation elements are:

- 1 a “segment” has the following key properties:
 - a number;
 - the masses of the simulated variables or “substances” (unit M);
 - a volume (unit L^3);
 - the concentrations of the substances follow by dividing mass and volume (unit $M.L^{-3}$);
- 2 an “exchange” between two adjacent segments has the following key properties:
 - the “from” pointer (segment or boundary on one side of the exchange);
 - the “to” pointer (segment or boundary on the other side of the exchange);
 - a discharge or water flux (unit $L^3.T^{-1}$);
 - a contact surface area (unit L^2);
 - a dispersion coefficient (unit $L^2.T^{-1}$);
 - the distance between the centre point of the “from” segment and the contact surface (“from length”);
 - the distance between the centre point of the “to” segment and the contact surface (“to length”).
- 3 a “boundary” has the following key properties:
 - a number (preceded by a minus sign, to distinguish boundaries from segments);
 - the concentration of the simulated active substances (unit $M.L^{-3}$).
- 4 a “waste load” has the following key properties:
 - the segment where the load is;
 - the released mass flux for simulated substances (unit $M.T^{-1}$).

Table 3.1 Key variables in DELWAQ code

Variable	Meaning	Variable	Meaning
noseg	nr of segments	flow(nog)	exchange flows
noq	nr of exchanges	area(nog)	exchange surface areas
notot	nr of substances	leng(2,noq)	exchange lengths
nosys	nr of active substances	conc(notot,noseg)	substance concentrations
nobnd	nr of boundaries	amass(notot,noseg)	substance masses
nowst	nr of waste loads	deriv(notot,noseg)	substance time derivatives
ipoint(4,noq)	pointer table	bset(nosys,nobnd)	boundary conditions
idt	time step	waste(notot,nowst)	waste loads
volume(noseg)	segment volumes		

3.2 DELWAQ1.EXE

DELWAQ1.EXE reads the input file, which consists of the following input groups:

- 1 Model identification and substances IDs
- 2 Integration options and timers, output timers
- 3 Cells, grids, cell properties, volumes
- 4 Exchanges between cells, flows, dispersions, areas, lengths, velocities
- 5 Open boundary conditions
- 6 Discharges and withdrawals
- 7 Water quality constants, parameters and functions
- 8 Initial conditions
- 9 Output specification
- 10 Statistical output specification

Each group ends with the “#n” token (e.g. “#5” for the end of the group 5 input section). After reading the input file, DELWAQ1 resolves all specifications of the input for the processes library (substances, processes, steering items, output items etc.).

DELWAQ1 generates the following output:

- a <runid.lst> file that echoes the input;
- a <runid.lsp> file that provides feedback on the processes library input specification;
- a <memory_map.out> file that shows the required length of the array work space during execution of DELWAQ2.

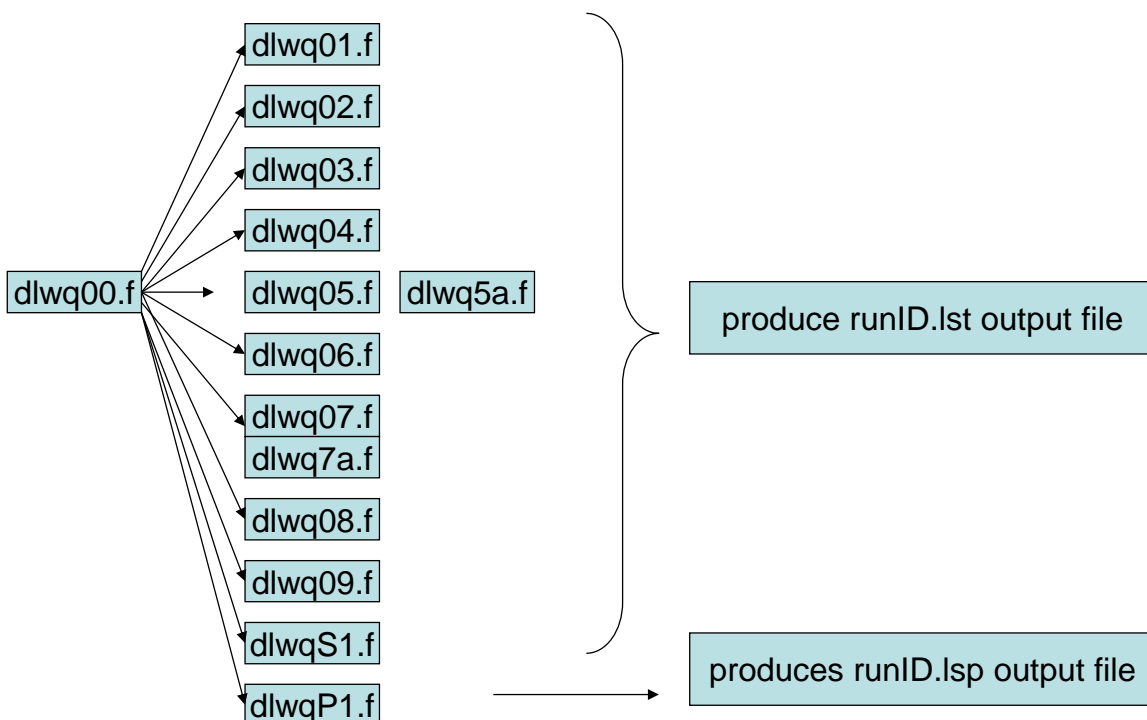


Figure 3.1 Schematic representation of DELWAQ1 source code

The DELWAQ1 input processing software code is located in:

- the <waq-io> folder for the reading of groups 1 – 9:
 - contains dlwq01 – dlwq09 subroutines;
 - contains 56 additional source files with supporting subroutines;
- the <proc_preprocess> folder:
 - for reading of group 10 (statistics are implemented as water quality processes);
 - for the evaluation of process information;
 - contains 65 additional source code files with supporting subroutines;
- the folder <waq_utils_f> contains 99 files:
 - that are more widely used within DELWAQ (also at run time);
 - that are also used outside of DELWAQ (like for particle tracking);
 - only some of them are also used for input processing.

Among the 56 routines in the <waq-io> folder there are:

- the 7 dlwq5a – dlwq5h routines for intelligent processing of loads and bounds;
- the 8 opt0 – opt3, rdpoint, fmread, rwfunc, dmatrix routines for old style input;
- the 13 read_.... routines for multigrid driven reading of process steering;
- the 3 point.... routines for reading and deriving model schematizations;
- the 3 dlwq0t, timer and cnvtim routines for dealing with time tokens.

Among the routines in the <waq_utils_f> folder are:

- the 4 files rd_token, dlwqj1, gettok, mestok for tokenized free formatted reading;
- the 1 file timers.f for all performance timers in DELWAQ.

3.3 Real numbers memory management in DELWAQ2

The real arrays used are listed in the <memory_map.out> file written by DELWAQ1. The first 78 real arrays listed in this file are stored in one long array called “a”. The variables stored in these arrays can be addressed by:

- the starting point in “a”.
- the increment to find the value for the next segment in “a” (for example “notot” in the concentration array “conc”).

We note that this is exactly the way how these items are passed to the processes in the processes library: a starting point array (called “ipoint” inside an individual process subroutine) and an increment (“incrm”) for all input and output items (see example in Figure 3.2).

There is a similar array “j” for integer arrays and an array “c” for character arrays.

```

====>
00001      subroutine CHLOR      ( pmsa , fl , ipoint , increm, nseg , &
00002                          noflux , iexpnt , iknmrk , noq1 , noq2 , &
00003                          noq3 , noq4 )
00004      $DEC$ ATTRIBUTES DLLEXPORT, ALIAS: 'CHLOR' :: CHLOR
00005      !
00006      !*****
00007      !
00008      IMPLICIT NONE
00009      !
00010      !      Type      Name      I/O Description
00011      !
00012      real(4) pmsa(*)      ! I/O Process Manager System Array, window of routine to process library
00013      real(4) fl(*)        ! O Array of fluxes made by this process in mass/volume/time
00014      integer ipoint( 6 ) ! I Array of pointers in pmsa to get and store the data
00015      integer increm( 6 ) ! I Increments in ipoint for segment loop, 0=constant, 1=spatially varying
00016      integer nseg        ! I Number of computational elements in the whole model schematisation
00017      integer noflux       ! I Number of fluxes, increment in the fl array
00018      integer iexpnt(4,*)  ! I From, To, From-1 and To+1 segment numbers of the exchange surfaces
00019      integer iknmrk(*)    ! I Active-Inactive, Surface-water-bottom, see manual for use
00020      integer noq1         ! I Nr of exchanges in 1st direction (the horizontal dir if irregular mesh)
00021      integer noq2         ! I Nr of exchanges in 2nd direction, noq1+noq2 gives hor. dir. reg. grid
00022      integer noq3         ! I Nr of exchanges in 3rd direction, vertical direction, pos. downward
00023      integer noq4         ! I Nr of exchanges in the bottom (bottom layers, specialist use only)
00024      integer ipnt( 6 )    ! Local work array for the pointering
00025      integer iseg         ! Local loop counter for computational element loop
00026      !
00027      !*****
00028      !
00029      !      Type      Name      I/O Description      Unit
00030      !
00031      real(4) Salinity     ! I Salinity (g/kg)
00032      real(4) GtCl         ! I Salinity:Chloride ratio in sea water (l/kg)
00033      real(4) Temp         ! I ambient water temperature (oC)
00034      real(4) Sal0         ! I salinity at zero chloride concentration (g/kg)
00035      real(4) RhoWater     ! O density of water (kg/m3)
00036      real(4) Cl           ! O Chloride (g/m3)
00037      !
00038      !*****
00039      !
00040      ipnt = ipoint
00041      !
00042      do 9000 iseg = 1 , nseg
00043      !
00044          Salinity = pmsa( ipnt( 1 ) )
00045          GtCl     = pmsa( ipnt( 2 ) )
00046          Temp     = pmsa( ipnt( 3 ) )
00047          Sal0     = pmsa( ipnt( 4 ) )
00048      !
00049      ! ***** Insert your code here *****
00050      !
00051          RhoWater = ??????
00052          Cl       = ??????
00053      !
00054      ! ***** End of your code *****
00055      !
00056          pmsa( ipnt( 5 ) ) = RhoWater
00057          pmsa( ipnt( 6 ) ) = Cl
00058      !
00059          ipnt = ipnt + increm
00060      !
00061      9000 continue
00062      !
00063      return
00064      end subroutine

```

Figure 3.2 Example process subroutine

3.4 DELWAQ2.EXE

The main code of DELWAQ2 is structured as in Figure 3.3. After initialisation, DELWAQ2 proceeds to one of the more than 20 supported numerical methods and passes a cycle of subroutines shown in Figure 3.4.

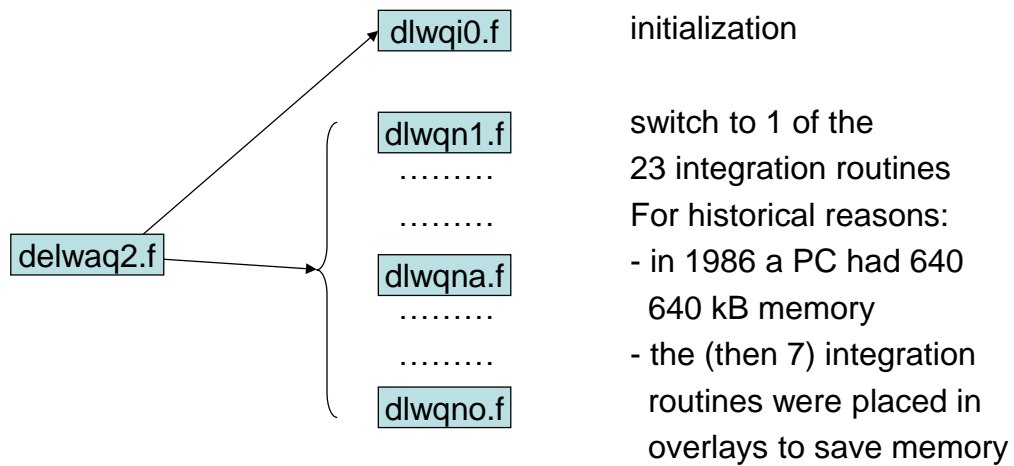


Figure 3.3 Schematic representation of DELWAQ2 source code

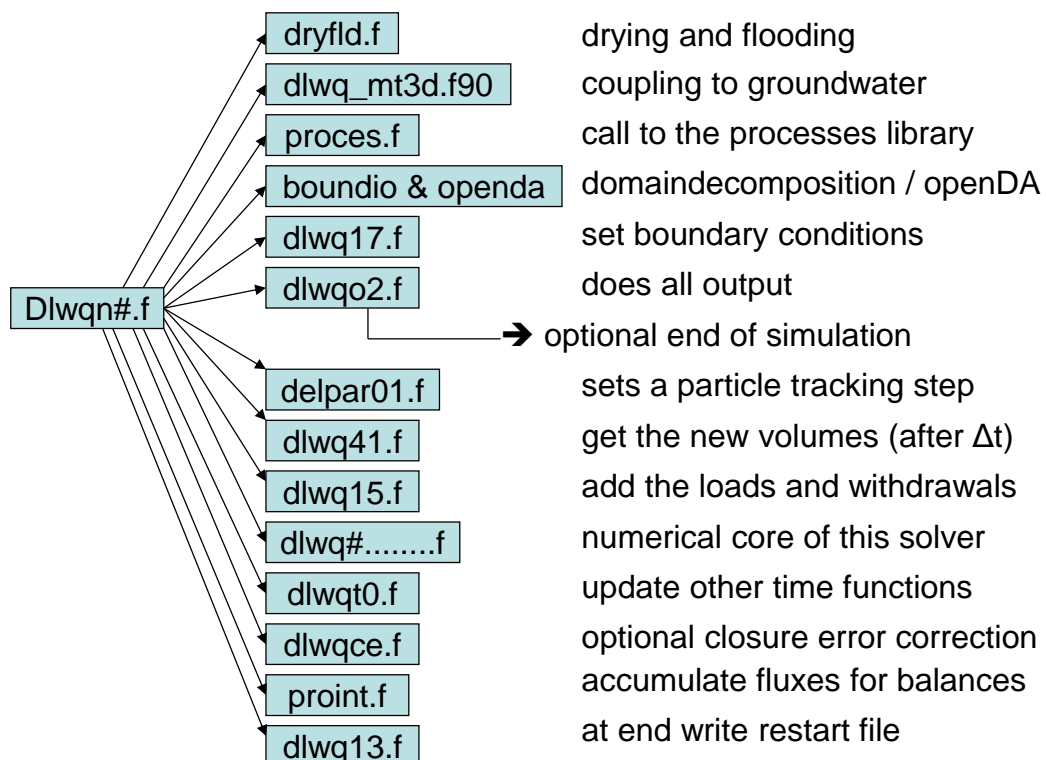



Figure 3.4 Schematic representation of DELWAQ2 source code for a single integration method

```

00445         else                                ! read new volumes from files
00446             call dlwq41 ( lun      , itime  , itime1  , a(iharm), a(ifarr),
00447             &             j(inrha), j(inrh2), j(inrft), nolog  , a(ivol2),
00448             &             j(ibulk), lchar  , ftype  , isflag , ivflag ,
00449             &             update , j(inisp), a(inrsp), j(intyp), j(iwork),
00450             &             lstrec , lrewin , a(ivoll), mypart , dlwqd  )
00451             if ( update ) updatr = .true.
00452 ----- 6 line(s) not displayed -----
00458
00459 ! add the waste loads
00460
00461             call dlwq15 ( nosys    , notot    , nolog    , noq    , nowst    ,
00462             &             nowtyp   , ndmps    , intopt   , idt     , itime    ,
00463             &             iaflag   , c(isnam) , a(iconc) , a(ivol) , a(ivol2) ,
00464             &             a(iflow) , j(ixpnt) , c(iwsid) , c(iwnam) , c(iwtyp) ,
00465             &             j(inwtyp) , j(iwast) , iwstkind , a(iwste) , a(iderv) ,
00466             &             iknmkv   , nopa     , c(ipnam) , a(iparm) , nosfun   ,
00467             &             c(isfna) , a(isfun) , j(isdmp) , a(idmps) , a(imas2) ,
00468             &             a(iwdmp) , 1        , notot    , j(iowns) , mypart  )
00469 ----- 12 line(s) not displayed -----
00470
00482             call dlwqm7 ( noq      , noq1     , noq2     , a(iarea), a(iflow),
00483             &             a(ileng), ilflag , intopt , j(ixpnt), mixlen ,
00484             &             iknmkv   )
00485
00486             if ( timon ) call timstrt ( "ADE solver", ithand1 )
00487             timon_old = timon
00488             noth = OMP_GET_MAX_THREADS()
00489             if ( noth .gt. 1 ) timon = .false.
00490 !$OMP PARALLEL
00491 !$OMP DO PRIVATE(ith)
00492 ! start of loop over substances
00493
00494             do 40 isys = 1, nosys
00495
00496                 ith = OMP_GET_THREAD_NUM()+1
00497
00498 ! make flow and dispersion arrays
00499             call dlwqm0 ( isys      , nosys    , noq      , noq1     , noq2     ,
00500             &             a(iarea) , a(iflow) , flowtot(1,ith), nvdim    , j(ivpnw) ,
00501             &             a(ivnew) , a(idisp) , disptot(1,ith), nddim    , j(idpnw) ,
00502             &             a(idnew) , mixlen   )

```

Figure 3.5 Example of DELWAQ2 source code for integration method 21.

This looks like  Figure 3.5 for numerical method 21. We note that the “a”, “j” and “c” arrays are visible at this level. Within the subroutines called, the arrays are declared by their own name (“conc” in stead of “a(iconc)” in the call to dlwq15).

The example in Figure 3.5 also shows the provisions made for parallel processing. This will be further discussed in Section 3.5).

The DELWAQ2 routines are mainly located in:

- the <waq_kernel> folder:
 - contains the 21 dlwqn1 – dlwqno solver main subroutines;
 - contains about 175 additional source files with supporting subroutines;
- the <waq_process> folder:
 - contains the water quality process routines;
- the folder <waq_utils_f> contains 99 files that are more widely used within DELWAQ, including:
 - routines for memory management and allocation;
 - routines for questioning the arrays with substance, parameter etc. ID’s.

DELWAQ produces the output that was requested in the input file:

- the history (<runid.his>) and map (<runid.map>) files for model results;
- the balances files for mass balance information (<runid-bal.his> and <runid-bal.prn>);
- the monitoring file (<runid.mon>) with all run-time messages and global output as shown in Figure 3.6.

```

00207 MESSAGE: Bloom fractional step switched on
00208 partit: nolog,nolay=      120400      43 , nseghr=      2800
00209 partit: npart= 1, assigning all segments to part "npart", none to other parts
00210
00211 CONSTANT ON UNIT:      26, READING: e:/Bodensee/ZLayer43_1min/com-BCO_dt1min_0708.len
00212 SIMULATION TIME : 2557D 0H 0M 0S !
00213 TIME IN FILE : 4D 3H 33M 20S !
00214 Closure error correction enabled
00215
00216 Initialising numerical options for method 15...18
00217
00218 Preconditioner switch not found, using default
00219 switch = 3, corrections based on lower and upper
00220
00221 Maximum number of iterations found in input
00222 Maximum number of iterations set to : 100
00223 Maximum number of vectors is: 50
00224
00225 Relative tolerance found in input
00226 Relative tolerance set to :0.100E-03
00227
00228 Scaling switch not found, using default
00229 switch = 1, scaling is switched on
00230
00231 Iteration report switch found in input
00232 switch = 0, iteration report is switched off
00233
00234 Extra functionality DLWQTR
00235 Dispersion length in third dir. will be calculated
00236 End extra functionality DLWQTR

```

Figure 3.6 Part of the <runid.mon> file

```

02459 DUMP OF INTERMEDIATE RESULTS IN SELECTED SEGMENTS
02460 TIME = 7Y 3D 0H 0M 0S .
02461
02462 TOTAL MASS IN SYSTEM 0.0000E+00 5.9520E+10 1.2353E+10 1.2353E+09 1.2353E+09 1.2487E+08 6.2455E+10 0.0000E+00 1.2353E+10 0.0000E+00
02463 CHANGES BY PROCESSES 0.0000E+00-2.2526E+09 0.0000E+00 0.0000E+00 0.0000E+00-6.6990E+05 6.6990E+05 0.0000E+00 0.0000E+00 0.0000E+00
02464 CHANGES BY LOADS 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.6955E+06 2.3462E+07 0.0000E+00 0.0000E+00 0.0000E+00
02465 BOUNDARY INFLOWS 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
02466 BOUNDARY OUTFLOWS 0.0000E+00 9.4265E+06 1.9256E+06 1.9254E+05 1.9254E+05 1.9383E+04 9.8062E+06 0.0000E+00 1.9373E+06 0.0000E+00
02467 AAP DetC DetN DetP DetS1 NH4 NO3 OOC OON OOP

```

Figure 3.7 Part of the <runid.mon> file; model-wide balances

```

00023808
00023809 REWIND ON UNIT: 20, READING: com-GB-Wet3.vol
00023810 AT SIMULATION TIME: 31D 12H 0M 0S !
00023811 TIME IN FILE: 31D 12H 0M 0S !
00023812 Performing closure error correction
00023813 Total volume before rewind: 0.3247E+10
00023814 Total volume after rewind : 0.3290E+10
00023815 Total correction factor : 0.1013E+01
00023816
00023817 REWIND ON UNIT: 23, READING: com-GB-Wet3.are
00023818 AT SIMULATION TIME: 31D 12H 0M 0S !
00023819 TIME IN FILE: 31D 12H 0M 0S !
00023820
00023821 REWIND ON UNIT: 24, READING: com-GB-Wet3.flo
00023822 AT SIMULATION TIME: 31D 12H 0M 0S !
00023823 TIME IN FILE: 31D 12H 0M 0S !
00023824
00023825 REWIND ON UNIT: 0, READING: com-GB-Wet3.vdf
00023826 AT SIMULATION TIME: 31D 12H 0M 0S !
00023827 TIME IN FILE: 31D 12H 0M 0S !

```

Figure 3.8 Part of the <runid.mon> file; input file rewind and closure error correction

The monitoring file also contains information about the model-wide mass balances (Figure 3.7) and about model input rewinding and the applied closure error correction (Figure 3.8).

There is an additional output file <runid-timers.out> that provides information on where the cpu time is spent, see Figure 3.9.

00000001	nr.	times called	cpu time in seconds	cpu %	wall clock in seconds	wc %	level 2	3	4	5	6	7	8	routine name
00000002														
00000003	1	1	0.251672D+03	100.00	0.254297D+03	100.00								delwaq2
00000004	2	6	0.000000D+00	0.00	0.000000D+00		0.00							getcom
00000005	3	1	0.100000D+01	0.40	0.187500D+01		0.74							unlock
00000006	4	1	0.100000D+01	0.40	0.187500D+01			0.74						dhlic
00000007	5	4	0.000000D+00	0.00	0.000000D+00				0.00					getcom
00000008	6	3	0.000000D+00	0.00	0.000000D+00				0.00					zoek
00000009	7	1	0.500000D+00	0.20	0.578000D+00		0.23							dlwqi0
00000010	8	1	0.000000D+00	0.00	0.000000D+00			0.00						dlwqi2
00000011	9	1	0.000000D+00	0.00	0.000000D+00			0.00						dlwqiv
00000012	10	1	0.000000D+00	0.00	0.000000D+00			0.00						dlwqio
00000013	11	1	0.000000D+00	0.00	0.000000D+00			0.00						partit
00000014	12	1	0.000000D+00	0.00	0.000000D+00			0.00						ixsets
00000015	13	1	0.000000D+00	0.00	0.000000D+00			0.00						chkmmr
00000016	14	1	0.109375D+00	0.04	0.188000D+00			0.07						dlwqt0
00000017	15	10	0.937500D-01	0.04	0.172000D+00				0.07					dlwqt1
00000018	16	3	0.000000D+00	0.00	0.000000D+00					0.00				dlwqt3
00000019	17	3	0.781250D-01	0.03	0.790000D-01					0.03				dlwqt4
00000020	18	3	0.000000D+00	0.00	0.000000D+00						0.00			filefind
00000021	19	6	0.000000D+00	0.00	0.000000D+00							0.00		flinterpol
00000022	20	1	0.000000D+00	0.00	0.780001D-01					0.03				dlwqt2
00000023	21	2	0.000000D+00	0.00	0.000000D+00					0.00				dlwqib
00000024	22	2	0.000000D+00	0.00	0.000000D+00					0.00				dlwqtb
00000025	23	1	0.156250D-01	0.01	0.160000D-01				0.01					dlwqta
00000026	24	1	0.000000D+00	0.00	0.000000D+00					0.00				dlwqia
00000027	25	1	0.000000D+00	0.00	0.000000D+00						0.00			dlwqt5
00000028	26	1	0.156250D-01	0.01	0.160000D-01					0.01				dlwqt6
00000029	27	1	0.156250D-01	0.01	0.160000D-01						0.01			dlwqt4
00000030	28	1	0.000000D+00	0.00	0.000000D+00							0.00		filefind
00000031	29	2	0.000000D+00	0.00	0.000000D+00								0.00	flinterpol
00000032	30	1	0.000000D+00	0.00	0.000000D+00					0.00				dlwqtk
00000033	31	1	0.000000D+00	0.00	0.000000D+00				0.00					zlayer
00000034	32	2	0.000000D+00	0.00	0.000000D+00					0.00				zoek
00000035	33	1	0.000000D+00	0.00	0.000000D+00				0.00					zoek
00000036	34	1	0.000000D+00	0.00	0.000000D+00		0.00							dlwqf5
00000037	35	5	0.000000D+00	0.00	0.000000D+00				0.00					zoek
00000038	36	1	0.156250D-01	0.01	0.160000D-01		0.01							dlwqf1
00000039	37	1	0.250156D+03	99.40	0.251828D+03		99.03							dlwqng
00000040	38	361	0.781250D-01	0.03	0.450001D-01				0.02					dryfld
00000041	39	722	0.000000D+00	0.00	0.000000D+00					0.00				zoek
00000042	40	1	0.000000D+00	0.00	0.000000D+00					0.00				zoek
00000043	41	361	0.000000D+00	0.00	0.000000D+00					0.00				setset
00000044	42	361	0.000000D+00	0.00	0.000000D+00					0.00				help1
00000045	43	361	0.000000D+00	0.00	0.000000D+00					0.00				proces
00000046	44	361	0.000000D+00	0.00	0.000000D+00					0.00				help2
00000047	45	361	0.000000D+00	0.00	0.000000D+00					0.00				dlwq_boundio
00000048	46	2	0.000000D+00	0.00	0.000000D+00					0.00				getcom
00000049	47	361	0.000000D+00	0.00	0.000000D+00					0.00				dlwq17
00000050	48	361	0.125359D+03	49.81	0.128842D+03		50.67							dlwqo2
00000051	49	4332	0.156250D-01	0.01	0.150001D-01					0.01				stepyn
00000052	50	2529	0.156250D-01	0.01	0.160000D-01					0.01				actloc
00000053	51	1445	0.000000D+00	0.00	0.000000D+00					0.00				fiosub
00000054	52	361	0.125000D+00	0.05	0.107000D+00					0.04				outmo3

Figure 3.9 Output file <runid-timers.out>

3.5 Parallel processing

Parallel processing is implemented in two places. First, the applicable water quality processes routines are distributed over the available processors, taking into account the output-input couplings between different processes. Second, for the numerical methods involving the implicit GMRES solver (methods 15/16 and 21/22), the modelled substances are distributed over the available processors, see Figure 3.5. This implies that the positive effect on performance of parallel processing not only depends on the number of processors, but also on the selected numerical method, on the processes and on the number of modelled substances.

4 The Processes Library

4.1 General

The basic equation solved by DELWAQ can be expressed as follows:

$\text{Change of mass per time} = \text{transport} + \text{loads} + \text{"processes"}$

The "processes" term is substance specific and it includes:

- sources: one substance appearing;
- sinks: one substance disappearing;
- transformations: one substance being converted into another substance.

The implementation of the processes term is different for the steady state and for the dynamic solvers.

4.1.1 Dynamic solvers

The processes are defined as fluxes (unit $M.T^{-1}$) acting on one or more substances. The processes term is solved explicitly:

$$\frac{Mass^{t+\Delta t} - Mass^t}{\Delta t} = Flux^t$$

This implies that the integration is potentially unstable. If we express the flux by means of a first order rate constant k (unit T^{-1}), the associated stability criterion reads $\Delta t < 1/k$.

The Processes Library provides DELWAQ with a series of subroutines to calculate processes. It was created to be able to re-use process formulations from previous applications. To ascertain flexibility, all processes subroutines have a standard interface which makes them independent of (a) whether a relevant substance is modelled or forced, and (b) the way of specification of any relevant input item.

The structured way of defining fluxes in the Processes Library also allows their automatic specification in DELWAQ's standard mass balance output.

The Processes Library provides a standard collection of processes that can be optionally expanded with your own set of processes (open PL).

Individual processes are activated from the input file by including a constant input item (keyword CONSTANTS) with the id "ACTIVE_xxxx" where xxxx is the process id. Processes not activated are not affecting the simulation.

4.1.2 Steady state solvers

The steady state solvers (6-9, 17-18) allow the specification of a processes term from inside the DLWQWQ subroutine only. An example is shown below.

<pre>SUBROUTINE DLWQWQ (NOTOT , NOSYS , NOSEG , NOPA , NOSFUN ,</pre>
--


```

*          VOLUME , CONC  , CONS  , PARAM  , FUNC  ,
*          SEGFUN , DERIV  , ITIME  , IDT    , ASMASS ,
*          IBFLAG , SYNAME , NOCONS , NOFUN  , CONAME ,
*          PANAME , FUNAME , SFNAME , NODUMP , IDUMP  )
.....

      ISYS = ITIME
      DO ISEG = 1, NOSEG
C ---      Zero order term in mass/time
            DERIV (ISYS, ISEG) = 0.0
C ---      First order term in volume/time
            CONC (ISYS, ISEG ) = 0.0
      ENDDO

      RETURN
      END

```

Note:

- the processes are defined substance by substance, where the number of the substance is passed as the 13th argument (variable “itime” in dynamic solvers);
- a zero order contribution (unit $M.T^{-1}$) can be defined in the “deriv” array;
- a first order contribution (unit $L^3.T^{-1}$) can be defined in the “conc” array.

The processes library can not be used in steady state solvers. The remainder of this chapter concerns the processes library and dynamic solvers only.

4.2 What is a process?

Figure 4.1 shows a schematic functional representation of a process in the Processes Library. The purpose of a process can be:

- to produce one or more fluxes, that affect the model equation for one or more substances;
- to produce output intended to be input for another process;
- to produce output intended to be inspected by the user.

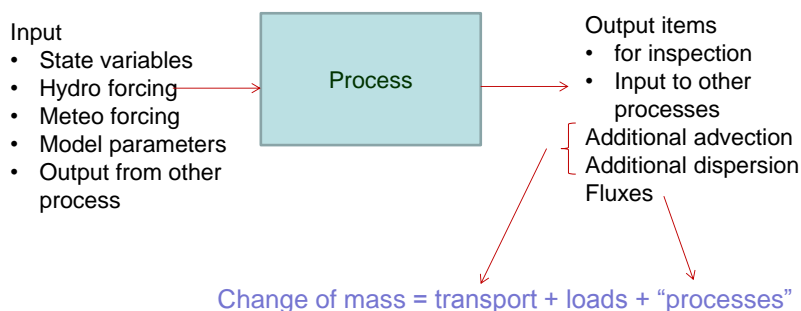


Figure 4.1 Schematic functional representation of a process in the Processes Library

We note that a process can not only produce fluxes that affect the model equations, but also additional transport terms of an advection or dispersion nature. This is typically the case for transport processes that are substance dependent, for example settling.

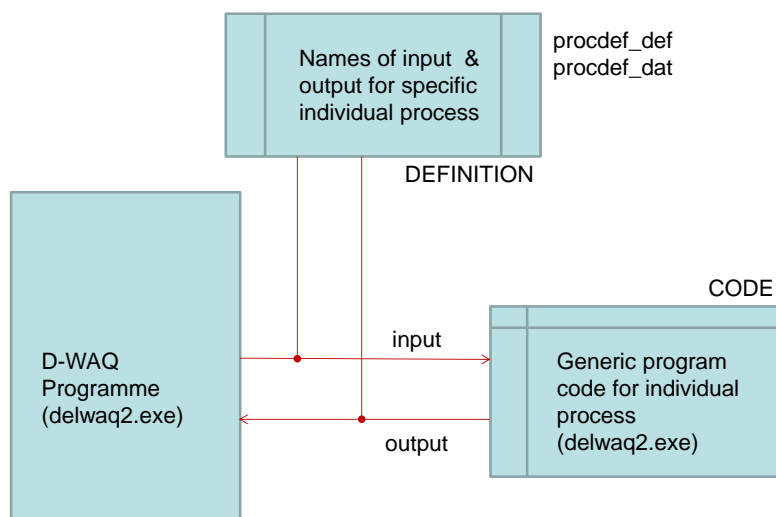


Figure 4.2 Schematic technical representation of a process in the Processes Library

Figure 4.1 shows a schematic technical representation of a process in the Processes Library. This figure shows that the process code is linked to the remainder of the DELWAQ2 programme through its standard subroutine call. The information passed via this call is derived from the process definition, stored in the procdef file (currently a Nefis file). This allows re-using the same code for different substances undergoing the same processes. An example would be the volatilisation of organic compounds that is modelled with the same formula, using different parameters for different substances. This will be further explained below.

4.2.1 Process definition

The box below shows an example (simplified) process definition for the reaeration of oxygen.

```

RearOXY                      Reaeration of oxygen
REAR      ; module name
123       ; TRswitch
          4; # input items for segments
OXY       10.0000    x Dissolved Oxygen                      (g/m3)
SaturOXY  -999.000   x saturation concentration              (gO2/m3)
KLRear    1.00000    x reaeration transfer coefficient        (m/d)
Depth     -999.000   x depth of segment                      (m)
          0; # input items for exchanges
          1; # output items for segments
SatPercOXY          x Actual saturation percentage O2         (%)
          0; # output items for exchanges
          1; # fluxes
dREAROXY           x reaeration flux of dissolved oxygen      (gO2/m3/d)
          1; # stoichiometry lines
OXY       dREAROXY   1.00000
          0; # stoichiometry lines dispersion arrays

```

```
0# stoichiometry lines velocity arrays
END
```

Note:

- 4 input items are defined, each with an id, a name, a unit and an optional default value;
- 1 output item is defined;
- 1 flux is defined “dREAROXY”,
- that acts on the substance “OXY”.

4.2.2 FORTRAN code

The corresponding FORTRAN code is listed below:

```

subroutine Rear      ( pmsa  , fl    , ipoint , increm, noseq , &
                     noflux , iexpnt , iknmrk , noq1  , noq2  , &
                     noq3   , noq4   )
! *****
!
!   IMPLICIT NONE
!
!   Type      Name      I/O Description
!
real(4) pmsa(*)      ! I/O Process Manager System Array
real(4) fl(*)         ! O  Array of fluxes made by this process in mass/volume/time
integer ipoint( 5)   ! I  Array of pointers in pmsa to get and store the data
integer increm( 5)   ! I  Increments in ipoint for segment loop
integer noseq        ! I  Number of computational elements
integer noflux        ! I  Number of fluxes, increment in the fl array
integer iexpnt(4,*)   ! I  From, To, From-1 and To+1 exchange pointers
integer iknmrk(*)     ! I  Active-Inactive, Surface-water-bottom, see manual for use
integer noq1          ! I  Nr of exchanges in 1st direction
integer noq2          ! I  Nr of exchanges in 2nd direction
integer noq3          ! I  Nr of exchanges in 3rd direction
integer noq4          ! I  Nr of exchanges in the bottom
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) O2          !   concentration of dissolved oxygen [g/m3]
real(4) OXSAT        !   saturation concentration of dissolved oxygen [g/m3]
real(4) REARKL        !   reaeration transfer coefficient [m/d]
real(4) DEPTH         !   actual depth of the water column [m]
real(4) SATPERC       !   SATURATION PERCENTAGE (%)
real(4) FL1          !   reaeration flux [g/m3/d]
!
integer iflx         !   pointer to flux variable
integer ikmrk1        !   first segment attribute
integer ikmrk2        !   second segment attribute
!
! *****
!
      ipnt      = ipoint
      iflx      = 1
!
      do 9000 iseg = 1 , noseq
        CALL DHKMRK(1,IKNMRK(ISEG),IKMRK1)
        IF (IKMRK1,EQ.1) THEN
!
!           Compute saturation percentage for all layers
!
          O2      = PMSA(ipnt( 1))
          OXSAT   = PMSA(ipnt( 2))
          SATPERC = O2 / OXSAT * 100.

```



```

CALL DHKMRK(2, IKNMRK(ISEG), IKMRK2)
IF ((IKMRK2.EQ.0).OR.(IKMRK2.EQ.1)) THEN

!       Compute flux only for a segment with an air-water-interface
REARKL  = PMSA(ipnt( 3))
DEPTH   = PMSA(ipnt( 4))
FL1 = REARKL * ( OXSAT - O2 ) / DEPTH
ELSE
    FL1 = 0.0
ENDIF
ELSE
    SATPERC = 0.0
ENDIF

fl(iflx) = FL1
pmsa( ipnt( 5) ) = SATPERC

iflx = iflx + noflux
ipnt = ipnt + increm

!
9000 continue
!
return
end subroutine

```

Note:

- the uniform call for all processes;
- the input and output items array “pmsa” has a variable length;
- there is a start location (“ipoint”) and an increment for every next segment (“increm”) in “pmsa” for every input and output item;
- the array “fl” stores the computed fluxes (in $M.L^{-3}.T^{-1}$);
- information about the exchanges (grid) is available (but not used);
- the dhkmrk subroutine is used to evaluate the attributes of the segments:
 - the first attribute (“ikmrk1”) is used to calculate for active water segments only (ikmrk1 = 1);
 - the second attribute (“ikmrk2”) is used to find segments with an air-water interface (ikmrk2 = 0 indicates a segment with both an air-water and a sediment-water interface; ikmrk2 = 1 indicates a segment with an air-water interface only).

The process definition together with the actual input file determine how the process is instructed to access the “pmsa” array. The instructions are passed via the “ipoint” and “increm” arrays. The input can be derived from different sources. In order of preference:

- a state variable;
- model input:
 - function of space and time;
 - function of time;
 - function of space;
 - constant;
- output from another process;
- default value.

Input items can also refer to predefined input elements of DELWAQ, like the time step, the volume, the flow etc.

4.3 The Open Processes Library (OPL)

The Open Processes Library was created some years ago to allow users to add their own processes to the library. A Graphical User Interface allows the user to create the process definitions and convert them into a <proces_def> file, which is the format used by DELWAQ and the GUI. It also creates a framework FORTRAN file for every defined process. These FORTRAN files need to be compiled and linked into a dynamic link library, called by DELWAQ2 at run time.

We note that the OPL can be used to create new stand-alone processes, but also to add processes to the standard PL, using the existing variables.

4.4 Managing the processes library

Managing the processes library consists of two tasks. First, all required FORTRAN files need to be available either directly in DELWAQ2.exe, or in the OPL dynamic link library (which is specified in a command line argument to DELWAQ2.exe).

Next, all processes definitions should be properly compiled in the <proces_def> file (which is specified in a command line argument to DELWAQ1.exe).

The processes definitions are stored as a relational database with 4 primary tables, and 6 tables fixing relations between these primary tables (see Appendix A). The primary tables are:

- Substances Groups (clusters of associated state variables, used by the GUI only).
- Items (all quantities playing a role in any process).
- Fortran subroutines.
- Processes.

There are two ways to manage this processes definitions database: (1) use the OPL GUI, and (2) use the waqpb_import and waqpb_export tools.

4.4.1 Use the OPL GUI

Managing the processes definitions with the OPL GUI is recommended for inexperienced users, but will only be feasible for small interventions and simple processes. The way of processing is shown in Figure 4.3. All required actions are supported by the GUI and saving the processes library will create the required <proces_def> file.

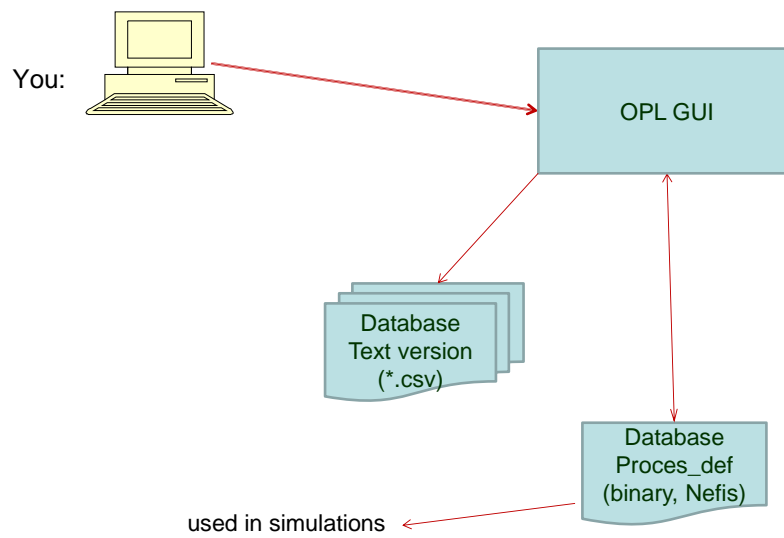


Figure 4.3 Managing the processes definitions database by using the OPL GUI

4.4.2 Use the waqpb tools

Managing the processes definitions with the waqpb_import and waqpb_export tools is recommended for experienced users only, but will allow larger interventions and handling complex processes, because the user has a better overview. The way of processing is shown in Figure 4.4. The management process involves the following steps:

- creating a first version of the processes definition database in a text file version (<*.csv>);
- using the waqpb_export tool to create an ascii definitions file (<procesm.asc>), where the definitions are now organised per process (as in Section 4.2.1);
- creating a modified ascii definitions file (e.g. <proces.asc>) by manually editing, which contains the modifications the user wants to make;
- using the waqpb_import tool to create an updated processes definition database in a text file version (<*.csv>);
- using the waqpb_export tool again to create the updated processes definition database in a binary format (<proces_def>) for DELWAQ simulations;
- optionally, the newly generated ascii definitions file (<procesm.asc>) can be compared to the originally edited file (<proces.asc>) to check the correct processing of the modifications.

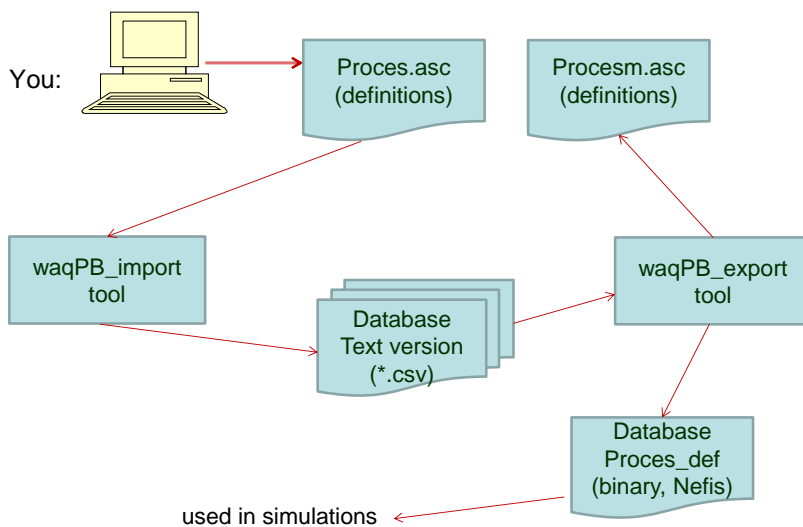


Figure 4.4 Managing the processes definitions database by using the waqpb tools

4.5 The waqpb_import and waqpb_export tools

4.5.1 How to use waqPB_import

The application is controlled by command line arguments. The following arguments are recognised:

`-pdf<file>:` specifies the name of the ascii processes definition file (default name: <proces.asc>);

- duprol: makes the programme read an ascii processes definition file from the DUPROL parser, (forces the -newtab and -newfrm arguments);
- newtab: makes the programme create a new Processes Library based on the specified ascii processes definition file only, neglecting the additional information in the already existing Processes Library (default is that the specified ascii processes definition file is supplemented with information already present in the existing Processes Library);
- newfrm: makes the programme use the new format of the ascii processes definition file (default is the old format, which uses a common string for the item description and the item unit).

Note

- one or more spaces between “-pdf” and the filename are not allowed;
- the filename may not include embedded blanks;
- the latest version is backward compatible: using no arguments will reproduce the functionality of the previous version;
- no matter how you use the import tool, only the processes in the ascii processes definition file will be included in the Processes Library; the existing processes will be removed; if you want to merge existing processes and new processes, this has to be done manually on the basis of the ascii file.

The use of the –newtab option will remove all existing information from the Processes Library. This concerns information not included in the ascii file, in particular:

- all existing substance groups will be deleted, and replaced by a single group called “DummyGroup”;
- all existing items not present in any process will be deleted;
- items occurring more than once in the ascii file will be attributed the name and unit first occurring in the file.

4.5.2 How to use waqPB_export

The application is controlled by command line arguments. The following arguments are recognised:

- versionxxxx: specifies the Processes Library version number xxxx (a real);
- serialyyyy: specifies the Processes Library serial number yyyy (an integer);
- newfrm: makes the programme use the new format of the ascii processes definition file (default is the old format, which uses a common string for the item description and the item unit).

Note

- one or more spaces between “-version” and the version number are not allowed;
- one or more spaces between “-serial” and the serial number are not allowed;
- the previous version prompts the user for input, this is no longer supported;
- the programme exports an ascii processes definition file called “procesm.asc”, which contains all processes; the previously supported selection options are no longer available;
- the programme exports Processes Library tables in Latex format only.

4.5.3 Definition of the ASCII proces definition file (new format)

The file is ASCII. To allow using embedded blanks without the need to use quotes, we use a fixed format. The file does not recognise comment lines. It is possible however to add comments at the end of a line, outside the range that is read (starting from column 106).

First line: the number of processes (integer, free format).

Next follow the indicated number of processes descriptions. Per process:

- Id (col 1-10) and name (col 11-60)
- Name (col 1-6) of FORTAN subroutine
- So-called TRswitch (integer, col 1-3)
- Nr of input items (segment) (integer, col 1-10), per item:
 - Id (col 1-10), default (col 11-28), PLCT switch (col 29), name (col 31-80) and unit (col 86-105)¹;
- Nr of input items (exchange) (integer, col 1-10), per item:
 - Id (col 1-10), default (col 11-28), PLCT switch (col 29), name (col 31-80) and unit (col 86-105)²;
- Nr of output items (segment) (integer, col 1-10), per item:
 - Id (col 1-10), PLCT switch (col 29), name (col 31-80) and unit (col 86-105)³;
- Nr of output items (exchange) (integer, col 1-10), per item:
 - Id (col 1-10), PLCT switch (col 29), name (col 31-80) and unit (col 86-105)⁴;
- Nr of calculated fluxes (integer, col 1-10), and per flux:
 - Id (col 1-10), PLCT switch (col 29), name (col 31-80) and unit (col 86-105)⁵;
- Nr of fluxes added to the model equations (integer, col 1-10), and per term:
 - Substance (col 1-10), flux (col 13-22) and scale factor (col 25-34);
- Nr of extra dispersion terms added to the model equations based on output items (integer, col 1-10), and per term:
 - Substance (col 1-10), item (col 13-22) and scale factor (col 25-34);
- Nr of extra advection terms added to the model equations based on output items (integer, col 1-10), and per term:
 - Substance (col 1-10), item (col 13-22) and scale factor (col 25-34);
- A closing line "END" (col 1-3).

Next follow optional blocks to define items that can act as state variables:⁶

- Nr of active substances (integer, col 1-10), and per substance:
 - Id (col 1-10), PLCT switch (col 29), name (col 31-80) and unit (col 86-105);
- Nr of inactive substances (integer, col 1-10), and per substance:
 - Id (col 1-10), PLCT switch (col 29), name (col 31-80) and unit (col 86-105).

The box below provides an example.

¹ In the old format, the unit field is lacking and the unit is merged into the name

² In the old format, the unit field is lacking and the unit is merged into the name

³ In the old format, the unit field is lacking and the unit is merged into the name

⁴ In the old format, the unit field is lacking and the unit is merged into the name

⁵ In the old format, the unit field is lacking and the unit is merged into the name

⁶ Missing in old format.

```

2
HydDuflow                      HydDuflow
HYDDFL      ; naam module
123         ; waarde van TRswitch
3; aantal invoer grootheden op segment niveau
Volume      -999.000    x volume of computational cell            m3
Surf        -999.000    x horizontal surface area of a DELWAQ segment m2
DELT        -999.000    x timestep for processes                  d
0; aantal invoer items op exchange niveau
2; aantal uitvoer grootheden op segment niveau
Z           x water depth                                         m
dt          x timestep of Duflow-quality calculation in seconds   sec
0; aantal uitvoer items op exchange niveau
0; aantal fluxen
0; aantal basis stochiometrie termen
0 ; aantal basis stochiometrie termen dispersie-array
0 ; aantal basis stochiometrie termen velocity-array
END
pclk401                      pclk401
PCLAKE      ; naam module
123         ; waarde van TRswitch
338; aantal invoer grootheden op segment niveau
sDDiatW     0.5    x DW water diatoms                            mgDW/l
sDGreenW    0.5    x DW water greens                             mgDW/l
sDBlueW     3      x DW water blue-greens                        mgDW/l
sDZoo       0.05   x DW zooplankton                              mgDW/l
sDIMW       2      x DW water inorganic matter                   mgDW/l
sDDetW      10     x DW water detritus                           mgDW/l
sDFish      4      x DW whitefish                                gDW/m2
Etc

fNH4NLoadD  0.75   x ammonium fraction of external dissolved N-load -
Z           2      x water depth                                  m
dt          86400   x timestep of Duflow-quality calculation in seconds sec
0; aantal invoer items op exchange niveau
1028; aantal uitvoer grootheden op segment niveau
chkDPhyt    x /* dry weight alga states */
chkDWeb     x /* dry weight foodweb/vegetation states */
chkDAbio    x /* dry weight abiotic states */
chkD        x /* dry weight states */
chkPPhyt    x /* phosphorus alga states */
chkPWeb     x /* phosphorus foodweb/vegetation states */
Etc

tSiTotSys   x /* total change of silica in the system [gSi/m2/da
tSiTotIn    x /* total flux of silica to the system [gSi/m2/day]
tSiErr      x /* error in silica mass-balance [gSi/m2/day] */
dYearNumbe  x

```

```

0; aantal uitvoer items op exchange niveau
57; aantal fluxen
D0sDDiatW          x
D0sDGreenW         x
D0sDBlueW          x
D0sDIMW            x
D0sDDetW           x
D0sDZoo            x

Etc

D0PAdsInit         x
D0Time             x
D0YearNumb         x

57; aantal basis stochiometrie termen
sDDiatW   D0sDDiatW   1.0
sDGreenW  D0sDGreenW  1.0
sDBlueW   D0sDBlueW   1.0
sDIMW     D0sDIMW     1.0
sDDetW    D0sDDetW    1.0
sDZoo     D0sDZoo     1.0
sDFish    D0sDFish    1.0

Etc

Time      D0Time      1.0
YearNumb  D0YearNumb  1.0

0 ; aantal basis stochiometrie termen dispersie-array
0 ; aantal basis stochiometrie termen velocity-array

END

23; aantal getransporteerde stoffen
sDDiatW          x DW water diatoms          mgDW/l
sDGreenW         x DW water greens           mgDW/l

Etc

sSiDetW          x Si water detritus          mgSi/l

34; aantal stoffen op de bodem
sDFish           x DW whitefish               gDW/m2
sDPisc           x DW predatory fish          gDW/m2

Etc

YearNumb          x number of years past (= zero during first year)  -
END

```

4.5.4 Specific functionality for DUPROL

If used with the option DUPROL, waqpb_import has the following additional functionality:

- the HydDufLOW process is inserted;
- all “FLOW” items get the default value -999., so that the PL derives these items from the HydDufLOW process;
- a so called “predefined set” is written, to import the model directly in the Sobek user interface (files *.0, *.sub, *.des).

A Processes definition database structure

No	Table	Columns	Field
p1	substances groups SGRP grpsub.csv	ID NM: name	C30 C50
p2	"items" ITEM items.csv	ID SE: segment level: 'x' = yes, ' ' = no EX: exchange level: 'x' = yes, ' ' = no DE: default value UN: unit NM: name AG: reference to item for averaging <i>input</i> in multi-grid models DA: reference to item for averaging <i>output</i> in multi-grid models WK: watercolumn: 'x' = yes, ' ' = no GR: reference to substance group id	C10 C1 C1 R4 C20 C50 C10 C10 C1 C30
p3	FORTTRAN subroutines FORT fortran.csv	ID (refers to program code and to documentation files)	C10
p4	processes (or PDFs) PROC proces.csv	ID CO: transport code ("123") FO: reference to FORTTRAN-subroutine id. NM: name	C10 I4 C10 C50
p5	configurations CONF config.csv	ID (used to refer to license files) NM: name	C10 C50

No	Table	Columns	Field
r1	relations between configurations and processes con_pro.csv	Matrix ICNPRO, defined as follows: inner loop over configurations outer loop over processes value: 1 = included, 0 = not included	I4
r2	relations between configurations and substances con_sub.csv	CID: reference to configuration SID: referene to item (substance)	C10 C10

r3	input items (relations between items and processes) INPU inputs.csv	PR: reference to proces IT: reference to item NM: serial number DE: default/noddefault (Y=yes,N=no, G=-888, B=-101, M=-11, O=-1) DO: documented? 'x' = yes, ' ' = no SX: 1 = segment, 0 = exchange	C10 C10 I4 C1 C1 I4
r4	output items (relations between items and processes) OUTP outputs.csv	PR: reference to proces IT: reference to item NM: serial number DO: documented? 'x' = yes, ' ' = no SX: 1 = segment, 0 = exchange	C10 C10 I4 C1 I4
r5	output fluxes (relations between processes and fluxes) OUTF outpflx.csv	PR: reference to proces FL: reference to item (flux) NM: serial number DO: documented? 'x' = yes, ' ' = no	C10 C10 I4 C1
r6	stochi lines (relations between fluxes and substances) STOC stochi.csv	FL: reference to item (flux, segment level) SU: reference to item (substance, segment level) SC: scale factor	C10 C10 R4
r7	velocity stochi lines (relations between items on exchange level and substances) VELO velocs.csv	IT: reference to item (velocity, exchange level) SU: reference to item (substance, segment level) SC: scale factor	C10 C10 R4
r8	dispersion stochi lines (relations between items on exchange level and substances) DISP disps.csv	IT: reference to item (dispersion, exchange level) SU: reference to item (substance, segment level) SC: scale factor	C10 C10 R4
r9	modelled variables (tab5.prn)	CI: reference to configuration IT: referene to item	C10 C10