Boundary flows are not indexed according to the node or channel numbers. It indexed in the qext table according to the order (total 789) and connected with a particular node (in qext table attach\_obj\_name be external node# and attach\_obj\_no be internal node#)

Reservoirs are indexed in a reservoir\_node\_connect table with reservoir names and connecting node#

When and where read\_fixed.f, process\_fix.f get called? Never been called. The code is not used anymore.

**The code needs to be cleaned up:**

1. fixedData.f line 1716 both get\_behavior\_filename(array) and get\_behavior\_inflile\_name(array) return same value.

Former is used by Aaron’s code and later is used by mine.

1. The io\_files are filled in the subroutine process\_io\_file(model, filetype, io, interval, filename), which is called by buffer\_input\_common line 112, which is called by ptmlocal.f line 208.

**Don’t compile using DLL mode. There is a run time lib issue. Using debug or release modes instead. They will produce a dll, which is in D:\dsm2\_development\dsm2\build\_vs2008sp1\_if11\all\Debug).**

**Copy the ptm.dll to dsm2\_development\historical\_run\bin**

**Compile DSM2:**

**If only ptm part is changed only need step 4.**

**Has to be python 2.x**

1. Run dsm2\_development/dsm2/src/input\_storage/generate.py
2. Open visual studio: dsm2\_development/input\_storage/build\_vs2008/all/input\_storage\_all.sln

**In Solution Explorer**: Solution ‘input\_storage\_all’(7projects) right click **Rebuild Solution**

**The compiled lib file is in dsm2\_development/lib/input\_storage\_vs9\_\*.lib**

1. Open visual studio: dsm2\_development/oprule/build\_vs2008/all/OpRuleAll.sln

**In Solution Explorer**: Solution ‘OpRuleAll’(5 projects) right click **Rebuild Solution**

**compiled lib file is in dsm2\_development/lib/oprule\_\*.lib**

1. Open visual studio: dsm2\_development/dsm2/build\_vs2008sp1\_if11 /all/ all\_intel.sln

**In Solution Explorer**: Solution ‘all\_intel’(17 projects) right click **Rebuild Solution**

Depending on the little window checked on the tool bar, the executables will be in

dsm2\_development/dsm2/build\_vs2008sp1\_if11 /all/Debug if Debug is checked

dsm2\_development/dsm2/build\_vs2008sp1\_if11 /all/Release if Release is checked

dsm2\_development/dsm2/build\_vs2008sp1\_if11 /all/dll if dll is checked

**Search for entire solution:**

A little window next to the Mixed platforms in the tool bar or shift+ctrl+f

**Test run dsm2**

D:\delta\dsm2\_v8\studies\playground> hydro historical\_hydro.inp

Note: The new compiled executable needs to be copied to D:\delta\dsm2\_v8\bin

**DSM2 PTM Fortran code:**

PTMFixedData.java line 61 initialize(filename)

PTMFixedData.java Native void initialize (…)

fixedData.f subroutine init\_fixed\_data (…)

fixedData.f line 35 **call** read\_ptm(filename)

**in ptmlocal.f – this is main entrance from PTM java to PTM Fortran**

**line 151 subroutine read\_ptm(init\_input\_file)**

1. ***prepare and initialize buffers, etc.***

**ine 178 call dsm2\_init():** initialize all arrays and logical variables. Nothing to do anything input files

**line 180 call h5open\_f(istat)** prepare to read H5 nothing to do with read in hydro tide file

1. ***read in evertything in .inp files to buffers w/o processing them for later Fortran programs use***

**line 191 call input\_text(input file name):**

in input\_stroge.f ::line 21 soubrouteine input\_text(filename)

in input\_stroge.f ::line 41 read\_buffer\_from\_text(….): only read in all text w/o processing text substitutioon (separate text from numbers and set up proper buffers w/o numbers)

in input\_stroge.f ::line 56 read\_buffer\_from\_text(….): put numbers to the buffers

read\_buffer\_from\_text(….) is automatically generated by generate.py in input\_stroage\_fortran.f90 line 154

1. ***Process the buffer data for later Fortran programs use***
2. **line 192 call process\_initial\_text(): to read in env var and scalar data only**

in input\_stroge.f ::line 118 soubrouteine process\_initial\_text

process envvar:

input\_stroage.f::line 133 call envvar\_query\_from\_buffer: get envname, envval, defined in envar\_input\_storage.fi line 60 (auto generated by generate.py)

input\_stroage.f::line 134 call add\_envvar: add envname, envval to buffer, defined in envvar.f to add env var name and value to the array of envvar\_t to be used later

process scalar:

input\_stroage.f::line 140 call scalar\_query\_from\_buffer: get scalar name, value, defined in scalar\_input\_storage.fi line 50 (auto generated by generate.py)

input\_stroage.f::line 141 call process\_scalar: put scalar data into suitable storage for later use, defined in process\_scalar.f line 21.

1. **Line 208 call buffer\_input\_common(): process common items**
2. **line 209 call buffer\_input\_ptm(): to read in ptm related input data, such as injection data**

in buffer\_input\_ptm.f ::line 21 soubrouteine pubber\_input\_ptm():

buffer\_input\_ptm.f ::line 57 particle\_insertion\_buffer\_size():

buffer\_input\_ptm.f ::line 59 **call** particle\_insertion\_query\_from\_buffer (…):

this two functions are generated by generate.py automatically and called from input\_storage\_fortran.f90, which include particle\_insertion\_input\_storage.fi. Thses functions help to get data out of the particular buffer (if you have new block needs to be added, you need to create those new functions by adding blocks to generate.py) and also This is where those generated functions get used.

buffer\_input\_ptm.f ::line 65 **call** process\_particle\_injection(…): defined in process\_ptm.f line 218 to add the buffered data to the fortran array to be used later.

1. **Lines 193 – 201: read in tide data???**

**call** initialize\_runtimes

**call** buffer\_input\_tidefile() !

**call** read\_grid\_from\_tidefile() !

**call** buffer\_input\_grid() ! processes grid

**call** read\_tide\_head: read in hydro tide file

1. **Line 211 write\_input\_buffers(): Write all buffers to text in the order they were defined???**
2. ***DSM2 PTM input structure:***

**PTMlocal.f: read\_ptm**

**Input\_storage.f: read in text**

**Input\_storage.f buffer\_input\_ptm.f**

**Env vars Scalars PTM specific data**

**generate.py envvar.f generate.py process\_scalar.f generate.py process\_ptm.f**

1. ***Java calls: All native function is called***

All fixed data (such as particle insertion info) are in fixedData.f: e.g.,

fixedData.f line 1259 – 1317: where particle insert info gets called for Java.

All dynamic data (such as flow, stage info) are in dynamic.f

**How does JNIT work:**

1. Write native methods in java classes
2. Use javah to create head files: all DWR\_DMS\_PTM\_\*.h in the native folder
3. Manually Implement .h files using dynamicDate.h and fixedData.h in the same directory to create .cpp files. DynamicDate.h and fixedData.h call the Fortran functions

All DSM2 solution is in:

D:\dsm2\_development\dsm2\build\_vs2008sp1\_if11\all\all\_intel.sln

Common:

Module common\_ptm

Fixed:

Subroutine buffer\_input\_ptm()

Subroutine check\_fixed\_ptm

Process\_ptm.f

Input\_storage:

Particle\_insertion\_input\_storage ……

Ptm\_dll

Ptm\_fortran

Ctrl+Shift+F: search