APPENDIX C
ESTUARY INFLOW EVALUATION

Integrated Delaware River Basin Model: OASIS, DYNHYD5 and TOXI5

Documentation of Model Integration

January 2008

Introduction

Up to this point several standalone computer models have been used to simulate flow and solute transport in the lower reaches of the Delaware River and its estuary. Among these are the Delaware River Basin OASIS Flow Model, DYNHYD5, and TOXI5. It was realized that some of the regulatory applications of these models could be conducted more efficiently if these models were run in parallel, with a feedback loop to pass information back and forth between them. The models require feedback because the management of water (simulated in the OASIS model) affects the salt front in the Delaware River estuary (simulated by DYNHYD5/TOXI5). In turn, the management of water can be affected by the salt front.

Through this study we've integrated the existing DYNHYD5/TOXI5 model of the Delaware River estuary with the existing OASIS model of the Delaware River Basin. Both DYNHYD5 and TOXI5 were modified to run as *external modules* for OASIS. The external-module system allows other types of simulation models to exchange data with an OASIS model in each simulation time step.

The external modules are designed to be very flexible. The modeler should not have to change the source code to reconfigure the model. This document describes how the modules can be reconfigured. It also describes how the source code of the module can be changed if the current module design does not meet future modeling needs.

Models that were Integrated

OASIS

The Delaware River Basin OASIS Model (Delaware OASIS) is a reservoir operations and flow routing model of the Delaware River and its main tributaries. Users of Delaware OASIS can modify the physical configuration of the system and operating rules through data entered in input tables or in the Operations Control Language (OCL) input. Standalone simulations typically cover the whole period of record (currently about 70 years). The time-step size is one day. An important feature of any OASIS application, including Delaware OASIS, is that other computer programs in DLLs, known as *external modules*, can exchange data with the OASIS model.

DYNHYD5 and TOXI5

DYNHYD5 simulates the hydrodynamics and TOXI5 simulates the chloride transport in the tidal Delaware River and Bay, from Trenton to the mouth of the Delaware Bay. Standalone simulations typically cover one to three years. TOXI5 has a time-step size of 15 minutes; DYNHYD5 can accommodate time steps as small as 30 seconds. Prior to the model integration described by this document, DYNHYD5 and TOXI5 were run in series. First DYNHYD5 must be run. Then, TOXI5 uses the DYNHYD5 output for flow input. In this relationship, TOXI5 is completely dependent on DYNHYD5 output.

Overlap between OASIS and DYNHYD5/TOXI5

Each model represents a one-dimensional network of nodes, and they overlap geographically in their coverage of the tidal portion of the Delaware River. DYNHYD5/TOXI5 represents the estuary in much finer detail than OASIS. However, OASIS dynamically determines flow that results from the operating rules that are described in OASIS input. DYNHYD5 determines flow only from the input boundary conditions and the physics of flow. In the original version of DYNHYD5, management decisions would be implicit in the boundary conditions. Any change to management must be pre-processed into the time-series input that DYNHYD5 reads. In the OASIS-module

version of DYNHYD5, OASIS specifies the dynamically-determined upstream boundary conditions to DYNHYD5.

Information about DYNHYD5/TOXI5
The linked DYNHYD5/TOXI5 models were calibrated for water surface elevations, current velocities, and chloride concentrations throughout the Delaware estuary for the 19-month period from 2001 to 2003. Detailed calibrations results can be found from the DRBC's technical report, which can be obtained from http://www.state.nj.us/drbc/TMDL/HydroModelRptDec2003.pdf. Downstream boundary tidal conditions and constant inflows from point source discharges and minor tributaries have been modified for this linkage. More detailed description of this work can be obtained from DRBC staff at http://www.state.nj.us/drbc.

Design of the DYNHYD5/TOXI5 modules

Source-code changes
When creating the DYNHYD5 and TOXI5 modules, the goal was to interfere with the existing source code as little as possible. The source code of the external modules differs from the standalone versions in the following ways:

- **Flow of processing.** The protocol for an OASIS external module requires that the module be divided into three distinct phases of processing: 1) initialization, 2) time-step loop, and 3) shutdown. This is necessary because OASIS must call the module at least once for every OASIS time step. In general, it is necessary that the module respond by simulating a period of time equal to one OASIS time step. Thus, there are three subroutines in the module which OASIS can call:
    - ***MODULE_INIT:*** This routine is called only once, while OASIS is initializing.
    - ***MODULE_STEP:*** This routine is called every time OASIS evaluates a *Run_module* command. Thus, this routine is called at least once every OASIS time step.
    - ***MODULE_SHUT:*** This routine is called only once, when OASIS is shutting down.

  The existing code in DYNHYD5 and TOXI5 had to be divided so that it fit into these three phases. This was not extremely difficult, since the flow of both programs already generally followed the progression of the three phases. Some change was necessary, because in the original source code the three phases were not perfectly isolated from each other.

- **Routines for the interface.** New subroutines had to be added to the module which did not exist in the original source of the standalone programs. These routines include the callable routines *MODULE_INIT*, *MODULE_STEP*, and *MODULE_SHUTDOWN* described above. There are a few new subroutines that are only called from the three callable routines. Also, there are error-handling subroutines described below.

  The routines that are new to the module version (not found in the standalone version) are found in the following Fortran and C++ source-code files. None of the source code in these files is found in the original DYNHYD5 or TOXI5.

In DYNHYD5:	**In TOXI5:**
_DynHyd5_init.f90	_Toxi5_init.f90
_DynHyd5_shut.f90	_Toxi5_shut.f90
_DynHyd5_step.f90	_Toxi5_step.f90
_OASIS_mod_init.f90	_OASIS_mod_init.f90
_OASIS_module.f90	_OASIS_module.f90
mod_oasis.f90	mod_oasis.f90
mod_shutdown.f90	mod_shutdown.f90
_OASIS_mod_initC.cpp	_OASIS_mod_initC.cpp

The new routines have some effects on the way DYNHYD5 and TOXI5 run. The new routines manage the current working directory so that DYNHYD5 and TOXI5 execute in a folder specified by the OCL *:MODULE:* command. The new routines override input parameters in DYNHYD5/TOXI5 that specify the length of the run, so that the modules run as long as OASIS is running. The new routines also override variable-flow input from the DYNHYD5 input file, applying flow values from the OCL *Run_module* command instead. No entirely new subroutines were added to the original DYNHYD5/TOXI5 source-code files. However, some modifications were done to the original files for error handling and the use of scratch files as described below.

- **Error-handling.** In the standalone programs, error handling generally included use of the Fortran *STOP* command. This is not acceptable for an external module because OASIS should process the error and write final output files before terminating. Therefore, the use of the *STOP* command was replaced by a call to the new subroutine *FORTRAN_MOD_ERR_SHUTDOWN*. This subroutine flags an error and passes the error message back to OASIS. Furthermore, the error message is constructed by subroutine *ADD_LINE_TO_MESSAGE*. This subroutine is used to build a long error message that OASIS can display, which is otherwise very difficult in Fortran.

- **Reading and writing scratch files.** Whereas the original versions of DYNHYD5 and TOXI5 were run in series, the module versions run in parallel, and it is necessary for DYNHYD5 to pass data to TOXI5 once per 24-hour cycle (one OASIS time step). In order to make the least interference with the existing source code of DYNHYD5 and TOXI5, HydroLogics designed a scratch file that is based on the output file that DYNHYD5 was already writing. This file is named *scratch.HYD*, and contains all the information that DYNHYD5 calculated for a single simulated day. The nature of the scratch file is described in more detail below.

Any part of the original Fortran source-code files that was modified by HydroLogics for the creation of the external modules is marked with the comment:

```
! %% OASIS
```

DYNHYD5 and TOXI5 input and output

The DYNHYD5 module reads the same input files, does the same calculations, and writes the same output files a the standalone version. However, after the DYNHYD5 module reads the input file, *temp.INP*, all of the variable-inflow values are discarded. Instead the module assigns the variable-inflow values that are passed from OASIS each time step. The DYNHYD5 module still requires that the input file contain data for each of these inflows, and the formatting has to be consistent with the original DYNHYD5 input file requirements. However, the actual values entered do not matter. The current input file has 20 days worth of dummy values, all set to -.005.

DYNHYD5/TOXI5 modules create the same output files, in their entirety, as the original standalone programs. A modeler can thus do the same analysis of these files as with the standalone version.

However, if the optional parameter *Large_Output=0* is specified in the *:MODULE:* command, then DYNHYD5's original output files are not written by the module. This can save significant disk space, and the files are not needed by OASIS or the module version of TOXI5.

Additionally, the DYNHYD5 module creates a block of information stored in the file *varblock.TMP*, which can be pasted into the DYNHYD5 file *temp.INP* in the variable-inflow section. Thus, after a run with the integrated OASIS/DYNHYD5/TOXI5 models is finished, the modeler can run the standalone DYNHYD5 model with the same data that was used in the integrated model.

The original standalone DYNHYD5 writes data twice per simulated day to a file named *temp.HYD*. This file is rather large (approximately 100 MB per simulated day) due to the small time step that has been used in the Delaware estuary model. In the standalone versions of the estuary models, this becomes an input file for TOXI5.

HydroLogics added source code to the DYNHYD5 module that writes every single day's worth of data to a scratch file designated *scratch.HYD*. The file *scratch.HYD* contains all of the same data, in the exact same format, as in *temp.HYD*. However, whereas *temp.HYD* contains all the output data for the entire DYNHYD5 run, *scratch.HYD* contains only one simulated day's worth of data. The data in *scratch.HYD* is overwritten every simulated day. The module version of TOXI5 reads input data from *scratch.HYD* instead of from *temp.HYD*. The TOXI5 module still reads the original input file *tempwq.INP*. The module version of TOXI5 also uses an entirely new file, *segmile.tbl,* to correlate TOXI5 segments to DRBC river miles. *Segmile.tbl* is described in the implementation section below.

Since DYNHYD5 is not dependent on TOXI5, it is possible to execute the DYNHYD5 module without TOXI5. Note, however, that DYNHYD5 by itself does not pass any information to OASIS. It is not possible to execute the TOXI5 module without the DYHYD5 module running at the same time.

Interface between OASIS and DYNHYD5/TOXI5

General OCL syntax

The Operations Control Language (OCL), is a form of input to OASIS. An OASIS model can be designed with great flexibility due to the language-nature of OCL. In the integrated model, the interface between OASIS, DYNHYD5, and TOXI5 is controlled entirely by OCL commands.

There are two OCL commands that are critical to external modules: *:MODULE:* and *Run_module*.

:MODULE: Command

The *:MODULE:* command is used to *declare* the module. It identifies the file that contains the module, and specifies information about how the module is initialized. The *:MODULE:* command is an OCL *meta command*. This means that it only contains information about how OASIS is initialized. Unlike the *Run_module* command, it does not contain information that is re-evaluated every simulation time step.

Run_module Command

The *Run_module* command is an OCL *simulation command*. This means that it contains information that is re-evaluated every simulation time step. The *Run_module* command contains a list of values that are sent to the external module (the *Input* field). It also contains a list of OCL variables whose values are assigned by the module (the *Output* field).

For a complete description of the syntax of these commands, please refer to the OASIS user manual.

Specific OCL syntax for DYNHYD5 module

The DYNHYD5 external module is programmed to process certain information from the *:MODULE:* and *Run_module* commands.

```
:MODULE: DLL  DynHyd5 = [HomeDir]\DynHyd5_module.dll
    InitParam "Folder=DRBC-DH5 Large_Output=1"
```

The *:MODULE:* command tells OASIS to use a module named *DynHyd5*. The name *DynHyd5* is used to refer to this module in the *Run_module* command. This module is contained in the file *[HomeDir]\DynHyd5_module.dll* (*[HomeDir]* is the folder where *model.exe* is found). The *InitParam* field is used to pass two parameters named *Folder* and *Large_Output*. The module uses the value of the *Folder* parameter (*DRBC-DH5*) as the name of the folder that DYNHYD5 executes in. That is, DYNHYD5 will look for all input files and write all output files in this folder. The value of the parameter *Large_Output* tells the module whether to store the large output files that are created by standalone DYNHYD5. If *Large_Output=1*, the large output files are written. If *Large_Output=0*, the output files are not written. These files are not needed to run DYNHYD5 or TOXI5 with OASIS. However, they may be needed to analyze the output files in the same way as standalone DYNHYD5.

```
Run_Module : DynHyd5
{
   Input: { RunFlag, ParamType, DynHyd5JunctionNumber,
            OasisFlowValue }
   Output : {  }
}
Run_Module : DynHyd5
{
   Input : { RunFlag }
   Output : {  }
}
```

The *Run_module* command for DYNHYD5 is used for two purposes: passing flow values and executing. The different uses are determined by the first parameter, *RunFlag*. In neither case are there any output values that OASIS receives from the module. For the purpose of passing flow values, the module is programmed to receive one flow value per *Run_module* command. Thus, there are generally several *Run_module* commands with *RunFlag=0*, but there must be only one *Run_module* command with *RunFlag=1*. The input parameters of the *Run_module* command are:

RunFlag = [0 or 1].

- If *RunFlag*=0, DYNHYD5 is not executed, but the subsequent input parameters specify a value that OASIS passes to the module.
- If *RunFlag*=1, then the module does not read the parameters *ParamType*, *DynHyd5JunctionNumber*, and *OasisFlowValue* (they should be omitted). DYNHYD5 is executed for 24 simulated hours. *RunFlag* should equal 1 only in the last *Run_module* command. DHYHYD5 can not be run more than once per simulated OASIS day, so *RunFlag* must not equal 1 in more than one *Run_module* command. It would serve no purpose to follow the *Run_module* command with *RunFlag=1* with another *Run_module* command with *RunFlag=0*.

ParamType = [1]

- This parameter should be used only if *RunFlag=0*. *ParamType* specifies the type of data that is being passed in the *Run_module* command. The only value that is currently recognized is 1, which specifies that the value *OasisFlowValue* is a variable flow. If a future need arises, the module can be programmed to recognize other types of parameters.

DynHyd5JunctionNumber = [Integer]

- This parameter should be used only if *RunFlag=0*. The value should be the integer number of a junction defined for variable flow in the DYNHYD5 input file *temp.INP*.

OasisFlowValue = [Real number (CMS)]

- This parameter should be used only if *RunFlag=0*. This value should be a flow in CMS that OASIS determines for the DYNHYD5 junction number given by *DynHyd5JunctionNumber*. The module overrides the value from DYNHYD5 input file *temp.INP* with *OasisFlowValue*.

Specific OCL syntax for TOXI5 module

The TOXI5 external module is programmed to process certain information from the *:MODULE:* and *Run_module* commands.

```
:MODULE: DLL  Toxi5 = [HomeDir]\Toxi5_module.dll
    InitParam "Folder=DRBC-DH5 Large_Output=1"
```

This command tells OASIS to use a module named *Toxi5*. The name *Toxi5* is used to refer to this module in the *Run_module* command. This module is contained in the file *[HomeDir]*\ *Toxi5_module.dll* (*[HomeDir]* is the folder where *model.exe* is found). The *InitParam* field is used to pass two parameters named *Folder* and *Large_Output*. The module uses the value of the *Folder* parameter (*DRBC-DH5*) as the name of the folder that TOXI5 executes in. That is, TOXI5 will look for all input files and write all output files in this folder. Because the TOXI5 module is dependent on the scratch file generated by the DYNHYD5 module, it would not make sense to specify a TOXI5 *Folder* value different that what is used for DYNHYD5. The value of the parameter *Large_Output* tells the module whether to store the large output files that are created by standalone TOXI5. If *Large_Output=1*, the large output files are written. If *Large_Output=0*, the output files are not written. These files are not needed to run TOXI5 with OASIS. However, they may be needed to analyze the output files in the same way as standalone TOXI5.

```
Run_module : Toxi5
{
   Input: { RunFlag, ParamType, ParamInput }
   Output: { Return_Value }
}
```

The *Run_module* command for TOXI5 is used for two purposes: passing water-quality values and executing. Executing is done only when parameter *RunFlag* is 1. The passing of water quality values occurs whether or not *RunFlag* is 1. The module is programmed to send one water-quality value per *Run_module* command. There are four different types of water-quality information that can be sent by the TOXI5 module. In each *Run_module* command, the type of water quality information to be sent is specified by parameter *ParamInput*. There can be more than one *Run_module* command for TOXI5, but only one of these *Run_module* commands can have *RunFlag=1*, and that command should be the first one. The input and output parameters of the *Run_module* command are:

RunFlag = [0 or 1]

- If *RunFlag*=1, TOXI5 is executed for 24 simulated hours, and send one water-quality value to OASIS. *RunFlag* can equal 1 only in the first *Run_module* command.

- If *RunFlag*=0, TOXI5 is not executed, but one water-quality value from the previous 24-hour execution is sent to OASIS. It would not make sense to apply *RunFlag=0* in a *Run_module* command before the *Run_module* command where *RunFlag=1*.

ParamType = [1, 2, 3, or 4]

- If *ParamType*=1, then *Return_Value* is the river mile of the isochlor of *ParamInput* -- that is, the most upstream River Mile where chloride concentration exceeds *ParamInput*. The module does linear interpolation to find the precise river mile.
- If *ParamType*=2, then *Return_Value* is the TOXI5 segment number of the isochlor of *ParamInput* -- that is, the most upstream segment number where chloride concentration exceeds *ParamInput*.
- If *ParamType*=3, then *Return_Value* is the chloride concentration at river mile equal to *ParamInput*. The module does linear interpolation to find the precise concentration.
- If *ParamType*=4, then *Return_Value* is the chloride concentration at TOXI5 segment number *ParamInput*.

ParamInput = [chloride concentration (mg/L), River Mile, or TOXI5 segment number]

- If *ParamType*=1, *ParamInput* is a chloride concentration in mg/L.
- If *ParamType*=2, *ParamInput* is a chloride concentration in mg/L.
- If *ParamType*=3, *ParamInput* is a river mile
- If *ParamType*=4, *ParamInput* is a TOXI5 segment number.

ReturnValue = [River Mile, TOXI5 segment number, or chloride concentration (mg/L)]

- If *ParamType*=1, *Return_Value* is the river mile of the isochlor of *ParamInput*
- If *ParamType*=2, *Return_Value* is the TOXI5 segment number of the isochlor of *ParamInput*
- If *ParamType*=3, *Return_Value* is the chloride concentration (mg/L) at river mile *ParamInput*.
- If *ParamType*=4, *Return_Value* is the chloride concentration (mg/L) at TOXI5 segment number *ParamInput*

Modifying the source code of the DYNHYD5/TOXI5 modules

The DYNHYD5/TOXI5 modules were designed to be very flexible. However, if the module interface does not provide the necessary control, the source code can be modified.

Initialization Parameters
If new initialization parameters are needed, then the module must be programmed to process them. This is done in subroutine *MODULE_INITIALIZE* in file *_OASIS_mod_initC.cpp*. The comments indicate a section where processing loops through all arguments of the initialization parameters. Within this loop, each parameter should be handled by its own *IF* block. For example, the current parameter *FOLDER* is handled by this *IF* statement:

```
if( strnicmp(temp_string, "FOLDER=", 7)==0 )
```

Parameters passed in the *Run_module* command
If new values are to be passed by the *Run_module* command, then the module must be programmed to send or receive them. This is done in subroutine *MODULE_STEP* in file *_OASIS_module.f90*. The only argument of the subroutine is named *argument*, and it is an array of type *real*4*. That reflects the fact that all parameters of the *Run_module* command are floating-point numbers.

One of the first things that should occur in *MODULE_STEP* is that the elements of the *argument* array should be assigned to local variables. This is how the input values from the *Run_module* command are assigned to variables in *MODULE_STEP*.

One of the last things that should occur in *MODULE_STEP* is that the values of local variables are assigned to elements of the *argument* array. This is how variables in *MODULE_STEP* are assigned to output variables (OCL variables) in the *Run_module* command.

As described above, the DYNHYD5 and TOXI5 modules have been designed so that *Run_module* command can be used as a sort of inquiry command. The second input parameter is a code for which type of data to retrieve, and the succeeding parameters identify for which segment or junction data should be retrieved or assigned. It is suggested that any changes to the module conform to this protocol. That is, a new code can be defined. The module can be programmed to recognize this code and respond appropriately. Additional *Run_module* commands that apply this code can then be added to the OCL input.

Limitations of the DYNHYD5/TOXI5 modules

The DYNHYD5 and TOXI5 modules have limitations that are inherited from the original standalone programs.

1. DYNHYD5 was originally coded to be limited to 700 variable flow time steps. This practically translates to 700 daily variable inflow steps. This was increased to 55,000. Tests indicate that the module runs properly for runs greater than 5 simulated years in length. With 55,000 variable inflow values, the longest possible run should be 150 simulated years.
2. DYNHYD5/TOXI5 output files are very large – combined about 100 Megabytes per month when using the supplied time steps. On Windows computers with hard drives formatted with the FAT32 file system, this will artificially limit the length of runs when a single output files reaches 4 Gigabytes. Computers with hard drives formatted with the NTFS file system

will not have this limitation. However, to run these files with OASIS, the only output that is strictly necessary are the small scratch files. The module versions have been programmed so that the large output files can be turned off using the *Large_output=0* parameter in the *:MODULE:* command.

3. The DYNHYD5/TOXI5 modules have been tested with OASIS for position analysis. The tests indicate that the module will work properly for position analysis. However, it should be noted that not all DYNHYD5/TOXI5 output is saved when doing position analysis. Position analysis is accomplished by running the OASIS model multiple times – once for each position analysis trace. With each trace run, the DYNHYD5/TOXI5 output files are overwritten. This is probably not a problem for the modeler, since the most important results can be stored by OASIS. Anyway, it would be difficult to analyze the large number of DYNHYD5/TOXI5 output files from a position analysis.
4. Examination of the original DYNHYD5 source code indicates that the Restart Function (using file *temp.RST*) contains errors, and should not be used.

Modifications to input files for the integrated model.

DYNHYD5 and TOXI5 input and output

TOXI5 and DYNHYD5 files are placed in the folder specified by the *Folder* parameter in the *:MODULE:* commands. The folder name *DRBC-DH5* was chosen for this folder, although the folder name is easily changed. The folder is assumed to be a subfolder of the OASIS run folder.

segmile.tbl – This is a new file not used by standalone TOXI5. It contains a table of values listing the river mile that corresponds to each TOXI5 segment number. The format is strictly column-based. Each row contains a 3-digit TOXI5 segment number followed by an 8-digit river mile value with up to 2 decimal places. The file starts at the highest river mile (133.3) and continues to the lowest river mile (8.0).

For Example:

```
 76   133.3
 75   132.0
 74   130.6
...
 84    18.6
 85     8.0
```

temp.inp– This file is used by the standalone TOXI5 and DYNHYD5, but the input data was modified slightly for use with the module versions. The variable inflow data was abbreviated, since the module was configured to override variable inflow data from this file (values passed through the OASIS *Run_module* commands are used instead). Each inflow is assigned 20 dummy data points (the number 20 is arbitrary). The value of each data point is unimportant. With each variable inflow, a note was added to identify the corresponding OASIS node number.

For Example:

```
*VARIABLE INFLOW DATA:  negative=inflow; positive=withdrawal ****
   12
        90        20          Del. Riv. at Trenton          [OASIS:365]
   1.  0 0     -.005   2.  0 0     -.005   3.  0 0     -.005   4.  0 0     -.005
   5.  0 0     -.005   6.  0 0     -.005   7.  0 0     -.005   8.  0 0     -.005
   9.  0 0     -.005  10.  0 0     -.005  11.  0 0     -.005  12.  0 0     -.005
  13.  0 0     -.005  14.  0 0     -.005  15.  0 0     -.005  16.  0 0     -.005
  17.  0 0     -.005  18.  0 0     -.005  19.  0 0     -.005  20.  0 0     -.005
```

```
…
        24        20          Inflows                      [OASIS:540]
   1.  0 0     -.005   2.  0 0     -.005   3.  0 0     -.005   4.  0 0     -.005
   5.  0 0     -.005   6.  0 0     -.005   7.  0 0     -.005   8.  0 0     -.005
   9.  0 0     -.005  10.  0 0     -.005  11.  0 0     -.005  12.  0 0     -.005
  13.  0 0     -.005  14.  0 0     -.005  15.  0 0     -.005  16.  0 0     -.005
  17.  0 0     -.005  18.  0 0     -.005  19.  0 0     -.005  20.  0 0     -.005
```

tempwq.inp – This file is used by standalone TOXI5. The module version of TOXI5 uses the file in exactly the same way. This file was not modified in any way from the version that DRBC provided.

Modifications to OASIS model input:
Of the OASIS input files, only OCL files were modified, as described here.

_module_declare.ocl – This is a new file. It contains the *:MODULE:* commands for DYNHYD5 and TOXI5, as described in the syntax section

main.ocl – This file existed in the previous model, but contains some modifications for the integration with DYNHYD5 and TOXI5. A new *:SUBSTITUTE:* command sets a flag (*[UseToxi5]*) with which you can specify whether OASIS should use original regression formula or the TOXI5 module to model the salt front. New *:INCLUDE:* commands are also used to apply the new OCL files. The file also contains a new *SOLVE* command. In the previous version of the model, the *SOLVE* command was implicit only. In the integrated model, the *SOLVE* command must be explicitly applied because DYNHYD5 and TOXI5 are called after flow information has been solved in each OASIS time step.

udef_list.ocl – This file existed in the previous model, but contains some modifications for the integration with DYNHYD5 and TOXI5. *Udef* commands were added for variables that were used to store information from TOXI5.

For example:

```
...
Udef : _SaltFrontMile_T5 init{[InitSalt] , [InitSalt] , [InitSalt] , [InitSalt] ,
       [InitSalt] , [InitSalt] , [InitSalt] }

Udef : _SaltFrontMovAv_T5 init{[InitSalt] , [InitSalt] , [InitSalt] , [InitSalt] ,
       [InitSalt] , [InitSalt] , [InitSalt] }

/* For DynHyd5 / Toxi5 Module testing and reference */

Udef : _SegNum_T5
Udef : _ConcMile_T5
Udef : _ConcSeg_T5
...
```

dynhyd5.ocl – This is a new file. It contains all *Run_module* commands for DYNHYD5. A *Run_Module* command is required for every DYNHYD5 variable flow that is set by the OASIS model. DYNHYD5 is configured to process flow in CMS, so a conversion from MG (the units used by OASIS) is necessary. The final *Run_module* command tells the DYNHYD5 module to simulate one day.

For example:

```
Run_Module : DynHyd5
{
   Input : { 0, 1, 90, Convert_Units{ Flow365.994, MG, CMS} }
   Output : { }
```

```
}
…
/* After all inputs are sent to the DynHyd5 Module, run it.  All output goes
   to Toxi5 through scratch files, so no data transfer is necessary between Oasis
*/

Run_Module : DynHyd5
{
   Input : {1}
   Output : { }
}
```

toxi5.ocl – This is a new file. It contains all *Run_module* commands for TOXI5. The first *Run_module* command tells TOXI5 to simulate one day. Then, a *Run_module* command is required for every value that OASIS retrieves from TOXI5.

For example:

```
Run_Module : Toxi5
{
   Input : { 1, 1, 250 }
   Output : { _SaltFrontMile_T5 }
}
Run_Module : Toxi5
{
   Input : { 0, 2, 250 }
   Output : { _SegNum_T5 }
}
Run_Module : Toxi5
{
   Input : { 0, 3, _SaltFrontMile_T5 }
   Output : { _ConcMile_T5 }
}
Run_Module : Toxi5
{
   Input : { 0, 4, _SegNum_T5 }
   Output : { _ConcSeg_T5 }
}
```

salt_front.ocl – This file existed in the previous model, but contains some modifications for the integration with DYNHYD5 and TOXI5. The *[UseToxi5]* flag, declared in *Main.ocl*, is applied here. If it *[UseToxi5]* equals 1, the model uses the TOXI5 output to create a moving average of the river mile where the salt front is located. If *[UseToxi5]* is not equal to 1, then the model uses the previously existing regression relationship to create a moving average of the river mile where the salt front is located.

For example:

```
/*  If the [UseToxi5] flag is used, it uses the value returned from the Toxi5 Module  */

:IF: { [UseToxi5] = 1 }

   Set : _SaltFrontMile_T5  { Value: _SaltFrontMile(-1) }

   Set : _SaltFrontMovAv_T5
   {
       Condition: Abs_Period <= 7
       Value : _SaltFrontMile_T5

       Condition: default
       Value : accumulate { _SaltFrontMile_T5 , -7 , -1 } / 7
   }

/* Otherwise Use the current approximation method */
//:ELSE:

...
```

Modifications to OASIS GUI input:

***GUI.ini* –** This file (which is always used by the OASIS GUI) contains one important modification for the integrated model. The new parameter *AddnlCopyFiles* is applied such that the GUI will copy the DYNHYD5 and TOXI5 input files whenever the GUI is used to copy a run folder. The new parameter appears as such:

```
AddnlCopyFiles=DRBC-DH5\*.inp;DRBC-DH5\*.tbl
```

Where *DRBC-DH5* is the name chosen for the folder that contains DYNHYD5/TOXI5 input files.

Modifications to OASIS database:

***Statdata.mdb* –** Each run needed to updated to work with the latest version of OASIS. This was accomplished by using Hydrologics' OASISConv.exe program. A double-entry in the lookup table, *Nevers_Rel*, had to be manually deleted in OASIS to get the run to work.