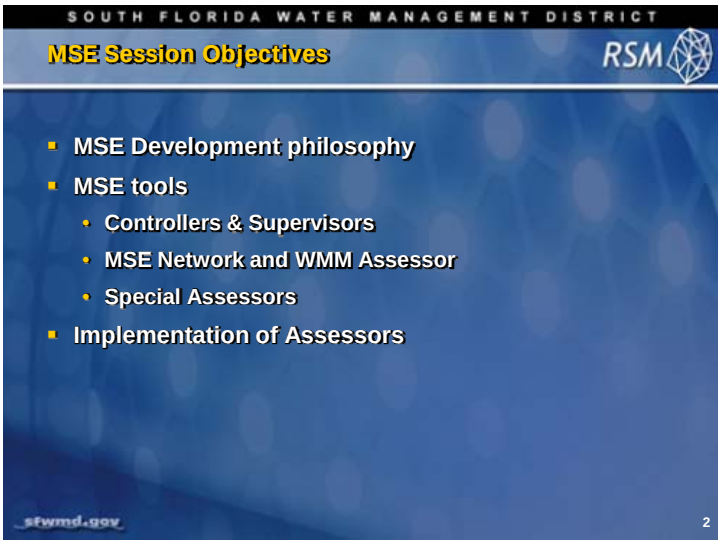
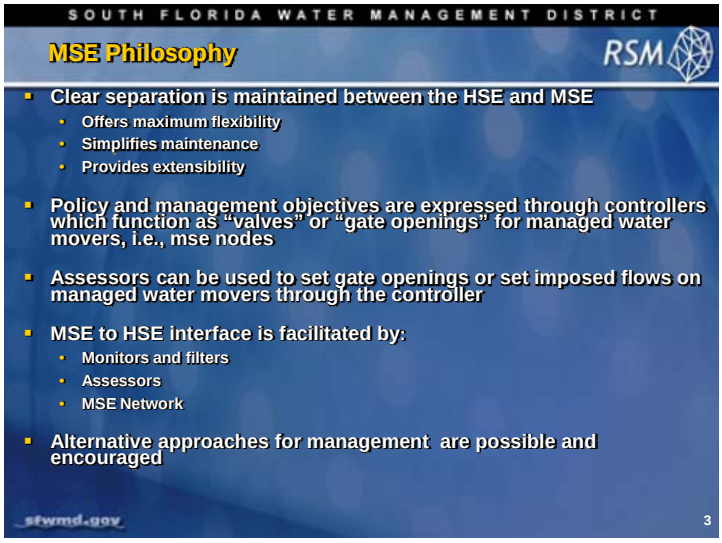
Lecture 15: Management Simulation Engine (MSE)

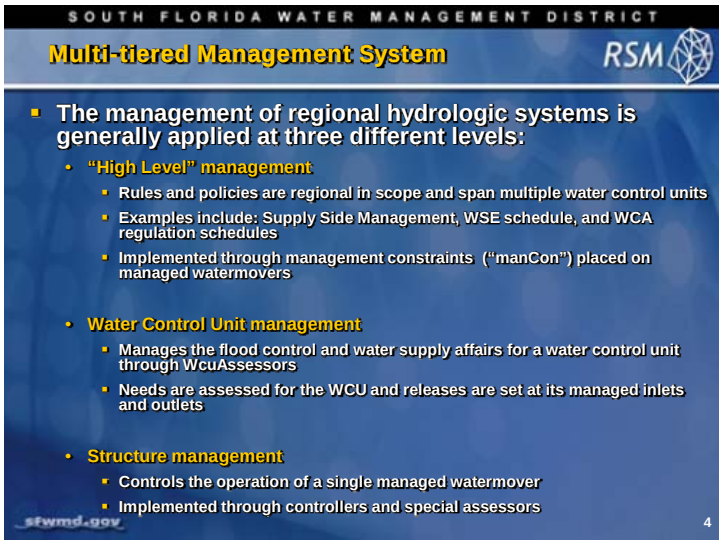
The lecture introduces the Management Simulate Engine (MSE), which is used to impose regional water management policies, local water level schedules and water construction rules on the network of water control structures.



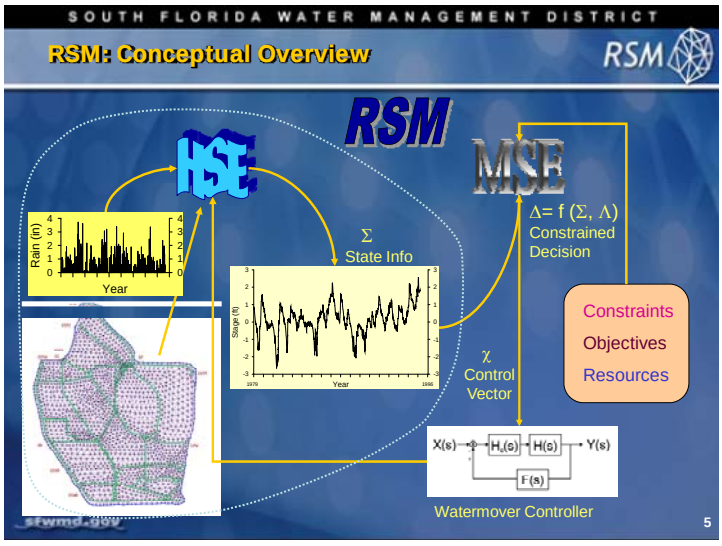
The session will be a discussion of these topics.



It was decided early in the development that the HSE and the MSE should be constructed separately. That would provide the greatest flexibility and simplify the maintenance. The HSE is a comprehensive hydrologic model by itself. The water management policies and management objectives are imposed on the HSE by setting the constraints implemented through controllers. There is a collection of assessors that determines how much flow should occur at each structure to meet water supply and flood control needs. There is a good connection between the HSE and the MSE through monitors and assessors at which MSE can obtain all HSE state values. The MSE provides for a high degree of flexibility in implementing alternative approaches for management.

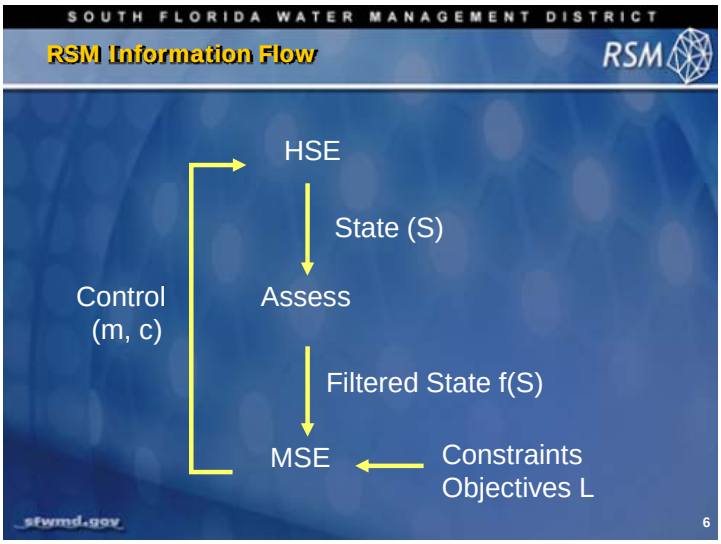


Management of the regional Central & South Florida Flood Control (CSFFC) project is a complex problem. It can be broken down into management at three different levels. At the highest level, the rules and policies are implemented at a high level broad-based regional approach through the application of constraints on how much water can pass through various structures. At the second level, flood control and water supply is managed at the water control unit (WCU) level as defined in the following slides. This is done by using assessors to determine the needs of the WCU and setting the flows at the inlets and outlets of the WCU. At the narrowest scope, controllers or special assessors are used to control specific structures. The future work will consist of determining what needs to be managed, at what level it should be implemented and creating the appropriate MSE functions.

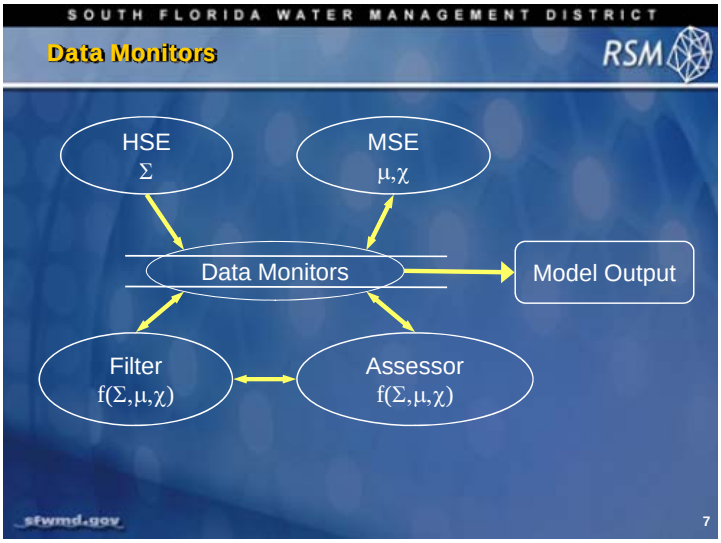


Conceptual overview of RSM includes two primary components: HSE and MSE. An early decision was made to completely separate the hydrologic computations from the management computations. This allows the user to make many flexible changes to the system management without affecting the hydrologic computations. HSE reads in boundary condition information and executes hydrologic computations, then estimates state information (heads in each waterbody and status of watermovers).

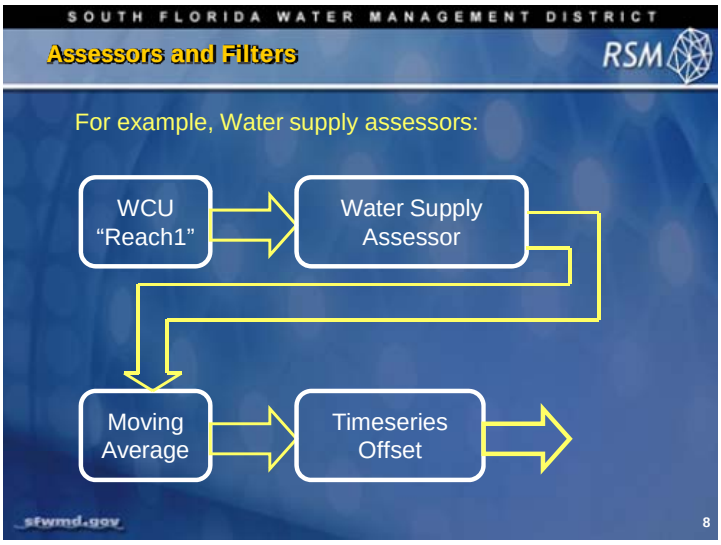
HSE controllers can react to state information, and enact watermover flow regulation within HSE. MSE reads in constraint and objective boundary condition information, and state information from HSE. MSE then produces a decision and control vector for HSE controllers.



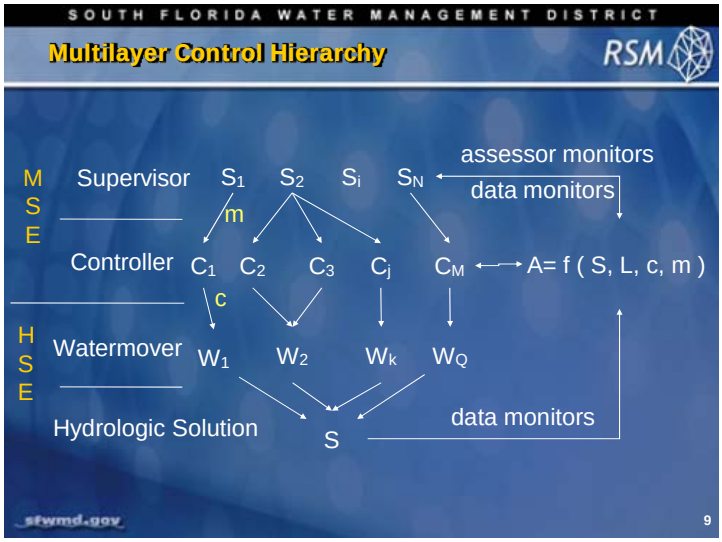
This is the information flow for the regional simulation model. The HSE provides the state information for the variables of interest including rainfall groundwater stage and canal stage. This information is passed to the MSE. The information may be the raw data or the data may be filtered by statistical analysis such as a moving average. The MSE takes the information from the state variables then applies certain rules, management constraints and controls the structures and watermovers in the HSE. The management constraints include the regional water management policies as well as local structure management rules.



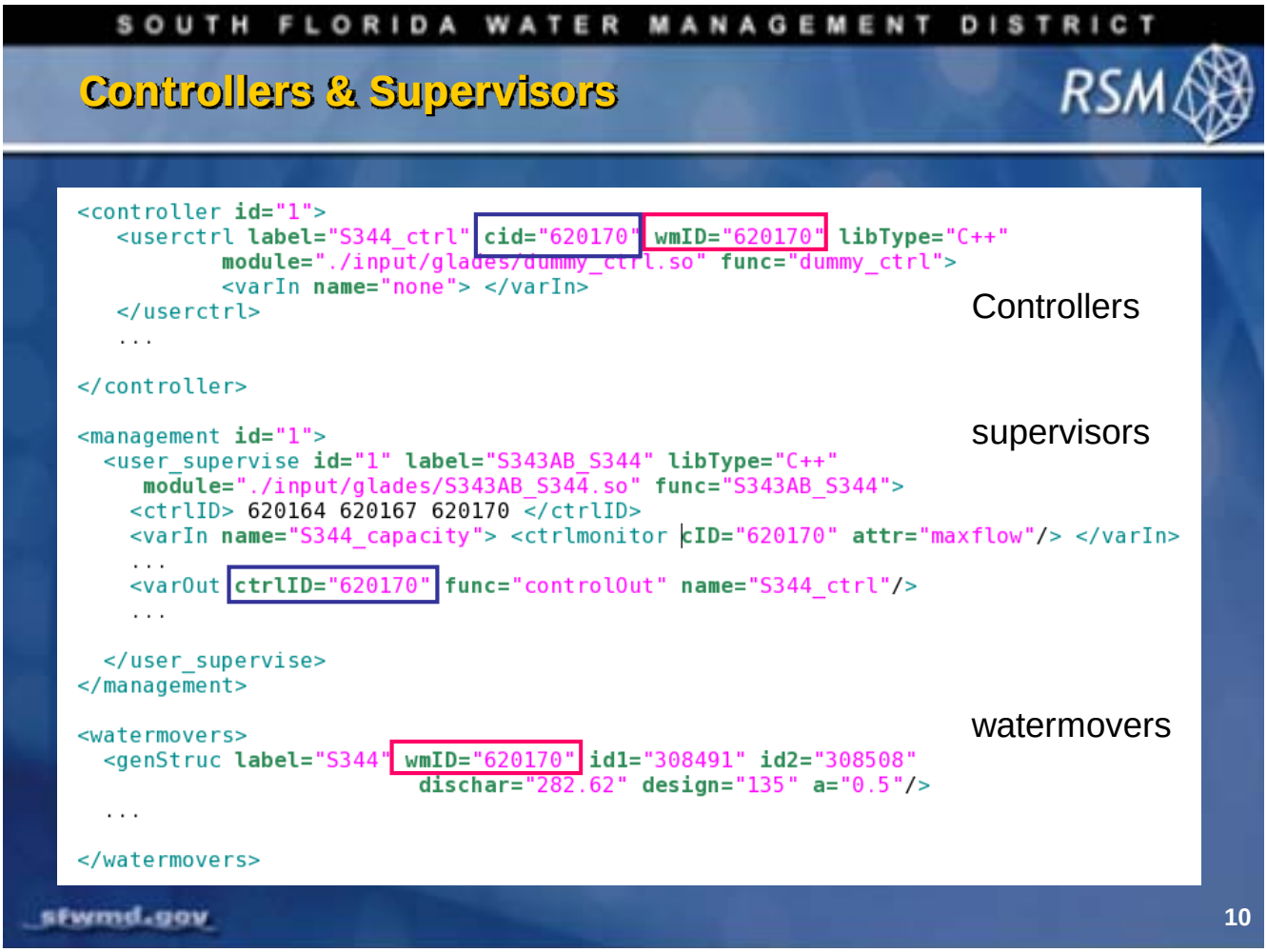
The data monitors are essential components of the interface between the HSE and the MSE. The monitors track the state information from the waterbodies, watermovers and MSE components as well as the output of filters and the assessors. The monitors produce the state information from waterbodies to the MSE as well as to the output. The MSE was created so that it uses the same data monitors as the HSE uses so no additional functionality had to be created for the MSE. The locations of the monitors are determined as part of the XML input.



The assessors are used to determine the volume of water in the selected WCUs for either flood control or water supply. Generally, assessors can be used to assess any state or dynamic variable in the model. A filter is an extension of a monitor that allows you to perform operations on the values from assessors. The common filters include moving averages, average of a group of monitors, and monitor comparisons. The typical output is a time series of values to be used by the MSE.

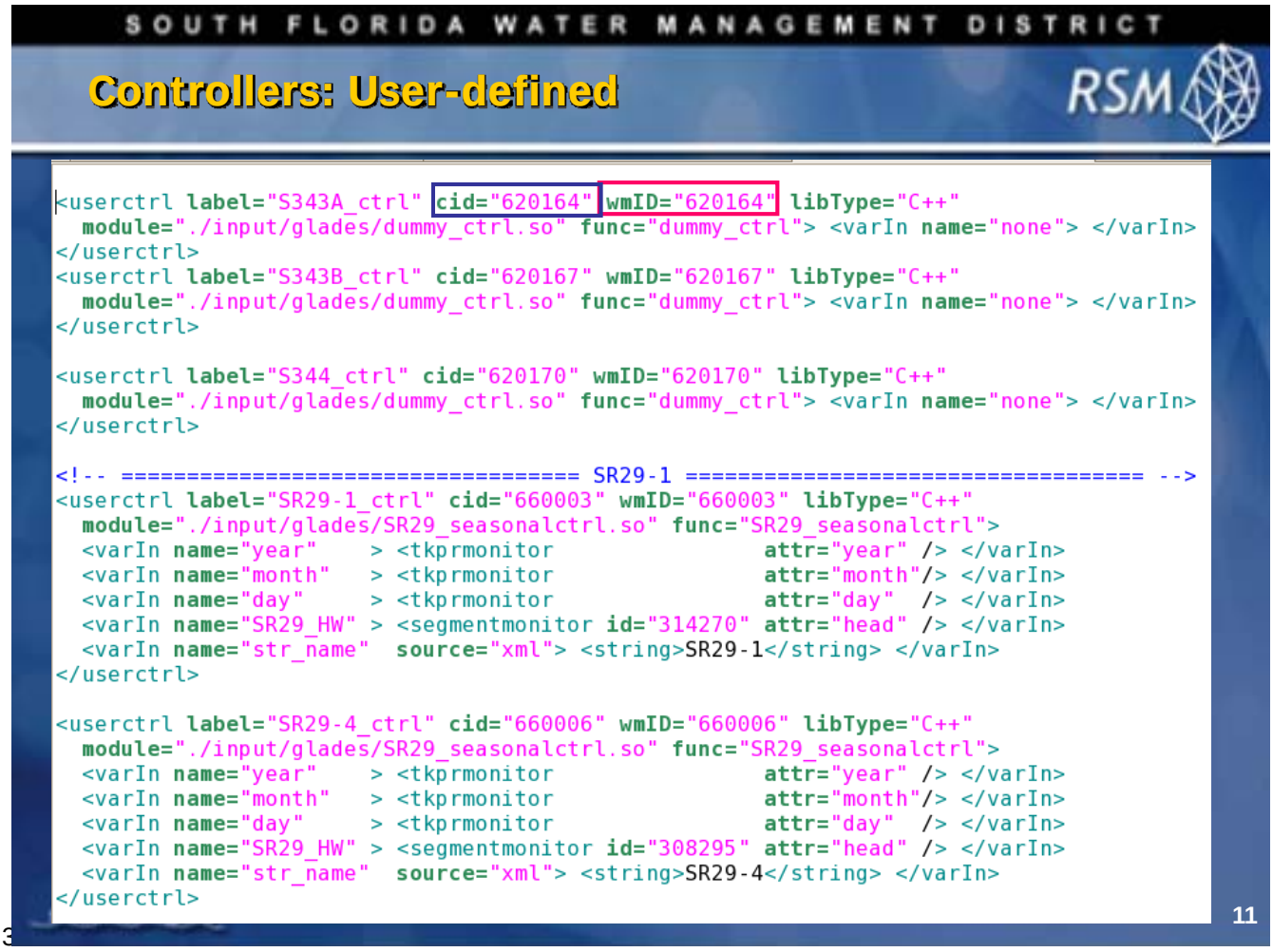


The MSE is a multilayer system. At the lowest level we have the HSE with the watermovers that control the water flow between the waterbodies. There are various data monitors that tell us the state information at various times. There are various assessors that provide information about the state of the system such as controller or gate operations in addition to hydrologic data. The MSE is a multilayer look at the system that allows supervisors to control multiple controllers and each controller to control multiple watermovers. The actions (A) taken by the MSE are a function of the state of the system (S), management constraints (L), the control settings (c), and management settings (m).

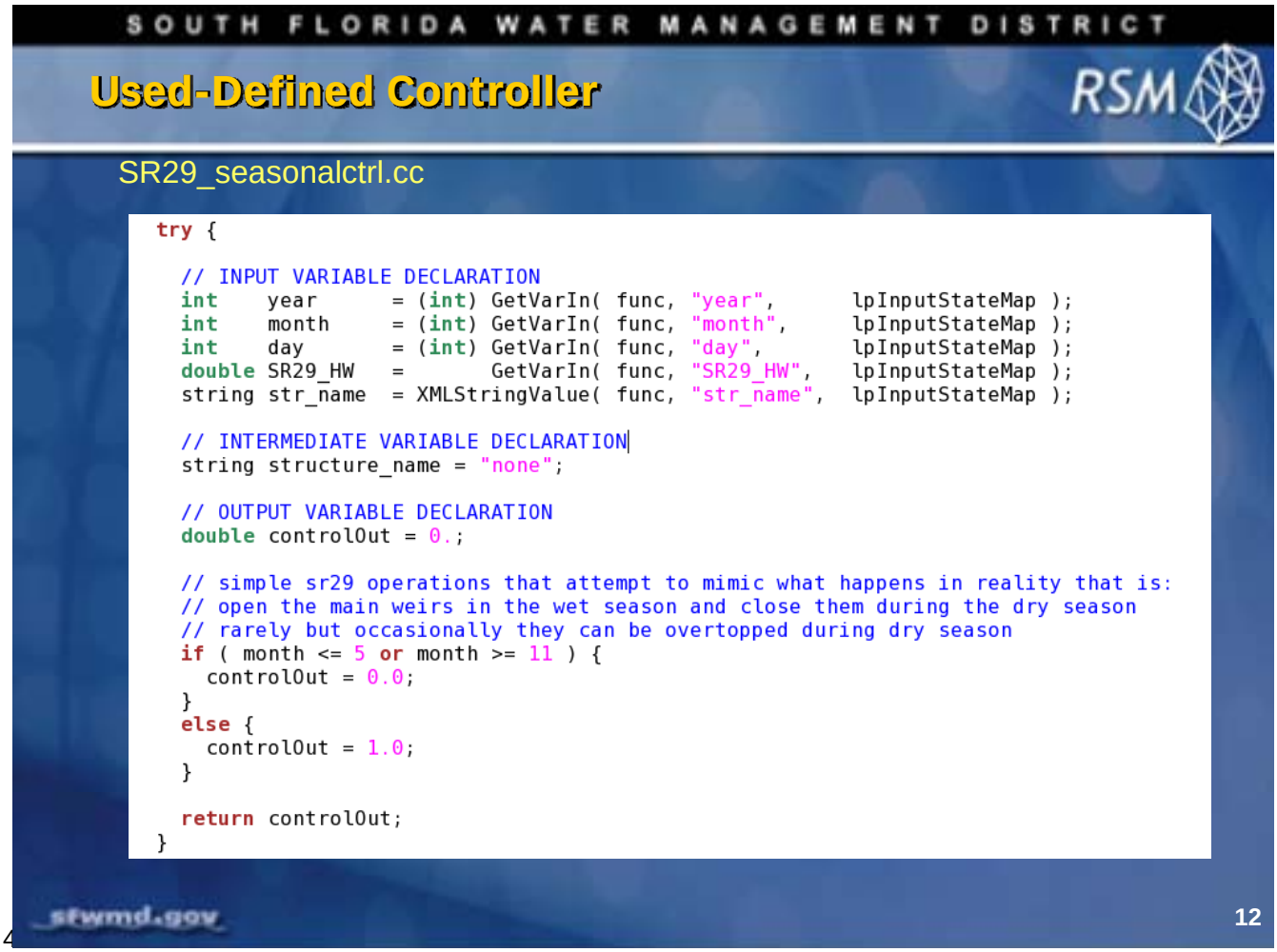


The controllers and supervisors are used to set the flow at specific structures based on local variables, seasonality or stage regulation schedules. Although controllers and supervisors cannot be used comprehensively due to computational expense, they are useful for selected structures.

The control functions are implemented using the user-defined controllers <**`userctrl`**>. These controllers are used where the control function is not a simple implementation of flow calculated elsewhere. The flow or control value is calculated in a C++ program and compiled into a library of control functions e.g. dummy.ctrl.so. The controllers are managed by a user_supervisor that controls one or more controllers <**`varOut`**> based on one or more inputs <**`varIn`**>. The controller in turn, controls a specific watermover. Each of these features occurs in separate XML blocks in the main run XML file.

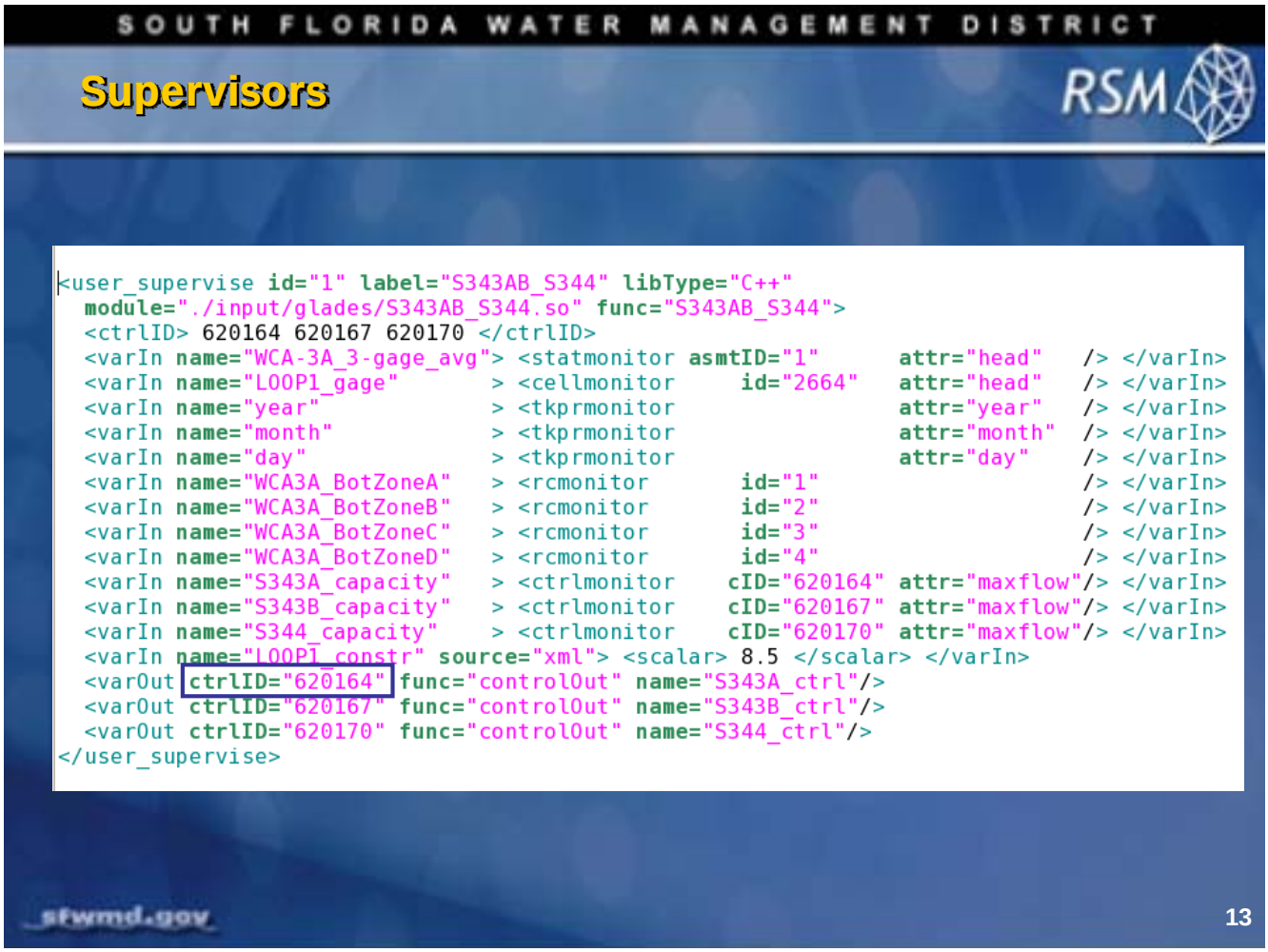


User defined controllers are specified in the controller block of the main run XML file. The specification can be a simple call to the controller C++ process program as with the controller for S343-344 structures or a more complex function that includes the necessary variables for the process program i.e., SR29 structures. The dummy_ctrl function simply passes the output from the supervisor to the controller.

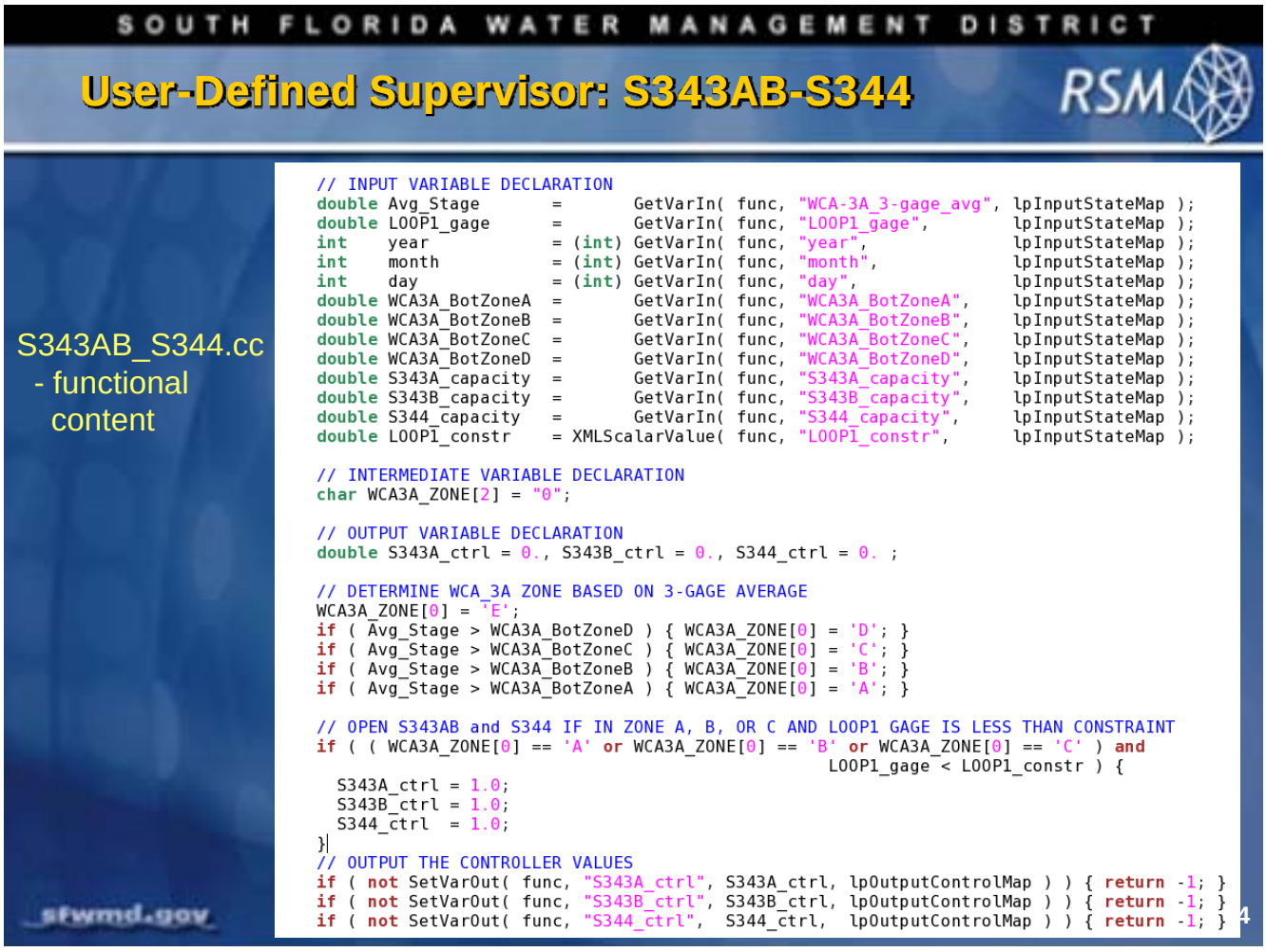


The user-controller process program for the S29 structures is simple, it only requires the data inputs from the model and sets the structure outflow to full flow during the wet season and closes the structure during the dry season. The source code is available in the **$RSM/data/glades_lecsa/input/glades/SR29_seasonalctrl.cc file**. The structure control can be adjusted by changing the function in this C++ source code file and recompiling the function into a shared library using the following command, all on one line:

```
gcc  SR29_seasonalctrl.cc
-BSymbolic
-fPIC
-shared
–o  SR29_seasonalctrl.so
-I../../../../trunk/src/
-I/opt/local/share2_64/include
```



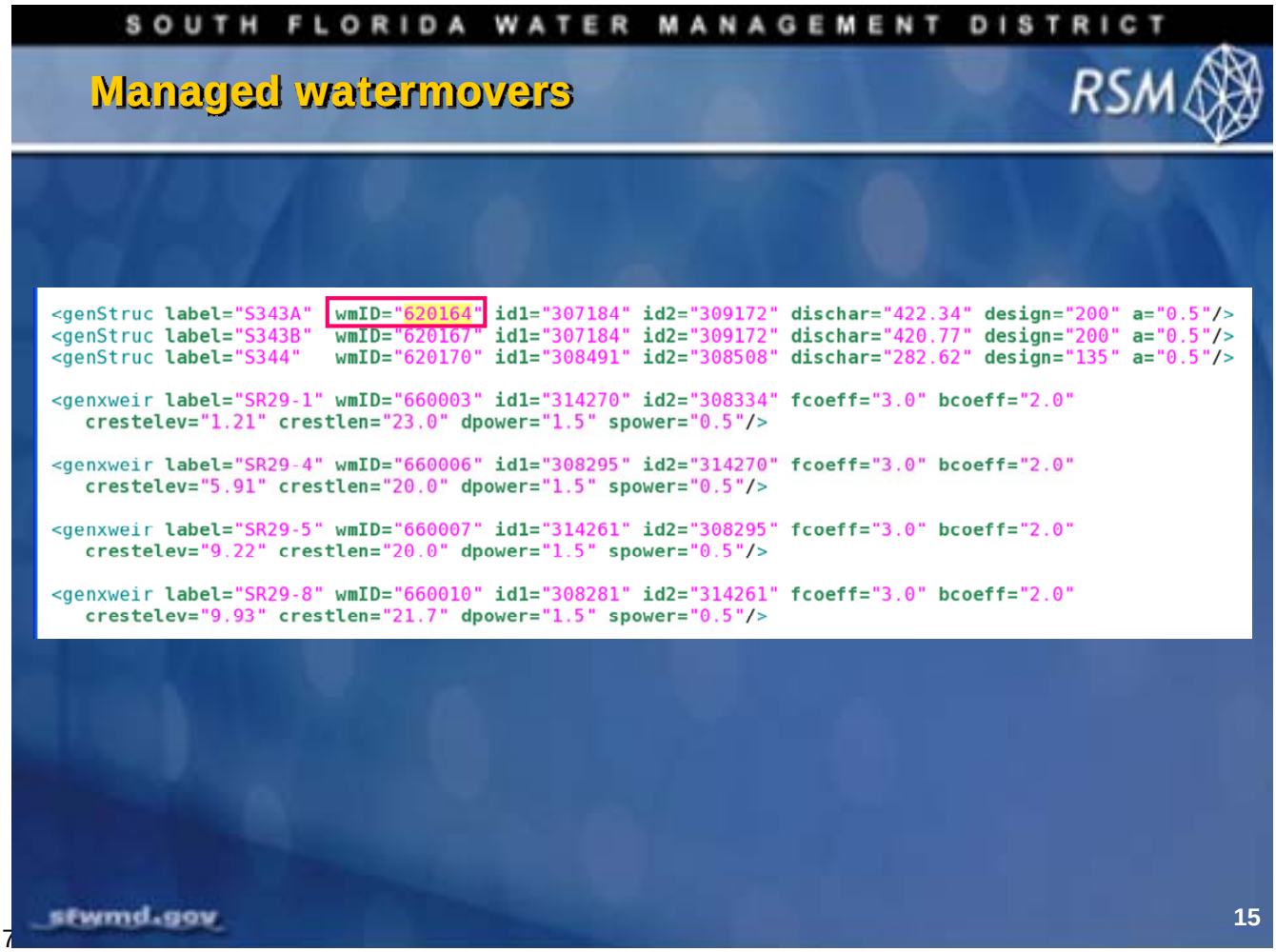
The user-supervisor uses the information processed by the controller function to set the "**`controlOut`**" variable for each of the structures. The inputs for the supervisor include several monitors including the control monitors processed the user-controller, rule-curve monitors and the statmonitor that uses an assessor to average the water levels in three cells to obtain the average water level in WCA-3A. The highlighted variable is the ID of the watermover that is controlled.



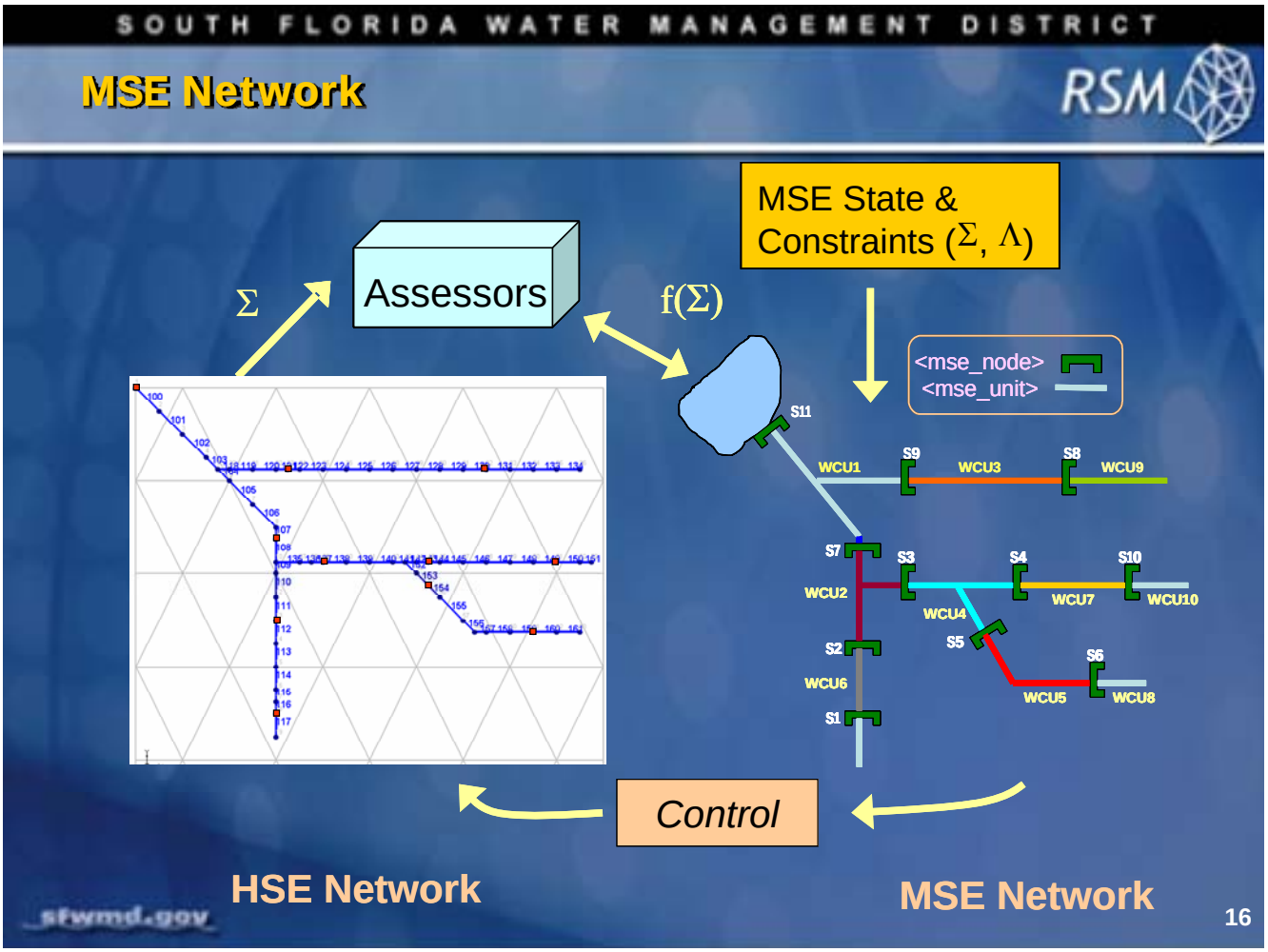

```
// INPUT VARIABLE DECLARATION
double Avg_Stage       =       GetVarIn( func, "WCA-3A_3-gage_avg", lpInputStateMap );
double LOOP1_gage      =       GetVarIn( func, "LOOP1_gage",        lpInputStateMap );
int    year            = (int) GetVarIn( func, "year",              lpInputStateMap );
int    month           = (int) GetVarIn( func, "month",             lpInputStateMap );
int    day             = (int) GetVarIn( func, "day",               lpInputStateMap );
double WCA3A_BotZoneA  =       GetVarIn( func, "WCA3A_BotZoneA",    lpInputStateMap );
double WCA3A_BotZoneB  =       GetVarIn( func, "WCA3A_BotZoneB",    lpInputStateMap );
double WCA3A_BotZoneC  =       GetVarIn( func, "WCA3A_BotZoneC",    lpInputStateMap );
double WCA3A_BotZoneD  =       GetVarIn( func, "WCA3A_BotZoneD",    lpInputStateMap );
double S343A_capacity  =       GetVarIn( func, "S343A_capacity",    lpInputStateMap );
double S343B_capacity  =       GetVarIn( func, "S343B_capacity",    lpInputStateMap );
double S344_capacity   =       GetVarIn( func, "S344_capacity",     lpInputStateMap );
double LOOP1_constr    = XMLScalarValue( func, "LOOP1_constr",      lpInputStateMap );

// INTERMEDIATE VARIABLE DECLARATION
char WCA3A_ZONE[2] = "0";

// OUTPUT VARIABLE DECLARATION
double S343A_ctrl = 0., S343B_ctrl = 0., S344_ctrl = 0. ;

// DETERMINE WCA_3A ZONE BASED ON 3-GAGE AVERAGE
WCA3A_ZONE[0] = 'E';
if ( Avg_Stage > WCA3A_BotZoneD ) { WCA3A_ZONE[0] = 'D'; }
if ( Avg_Stage > WCA3A_BotZoneC ) { WCA3A_ZONE[0] = 'C'; }
if ( Avg_Stage > WCA3A_BotZoneB ) { WCA3A_ZONE[0] = 'B'; }
if ( Avg_Stage > WCA3A_BotZoneA ) { WCA3A_ZONE[0] = 'A'; }

// OPEN S343AB and S344 IF IN ZONE A, B, OR C AND LOOP1 GAGE IS LESS THAN CONSTRAINT
if ( ( WCA3A_ZONE[0] == 'A' or WCA3A_ZONE[0] == 'B' or WCA3A_ZONE[0] == 'C' ) and
                                                    LOOP1_gage < LOOP1_constr ) {
  S343A_ctrl = 1.0;
  S343B_ctrl = 1.0;
  S344_ctrl  = 1.0;
}
// OUTPUT THE CONTROLLER VALUES
if ( not SetVarOut( func, "S343A_ctrl", S343A_ctrl, lpOutputControlMap ) ) { return -1; }
if ( not SetVarOut( func, "S343B_ctrl", S343B_ctrl, lpOutputControlMap ) ) { return -1; }
if ( not SetVarOut( func, "S344_ctrl",  S344_ctrl,  lpOutputControlMap ) ) { return -1; }
```

The user-supervisor for the S343-344 structures is more complicated than the one used for the S29 structures; it uses the WCA3A regulation stage schedule as well as the structure capacity to set the flow for the structures. Changes can be made to this controller source code file and recompiled using the following compile command (all on one line):

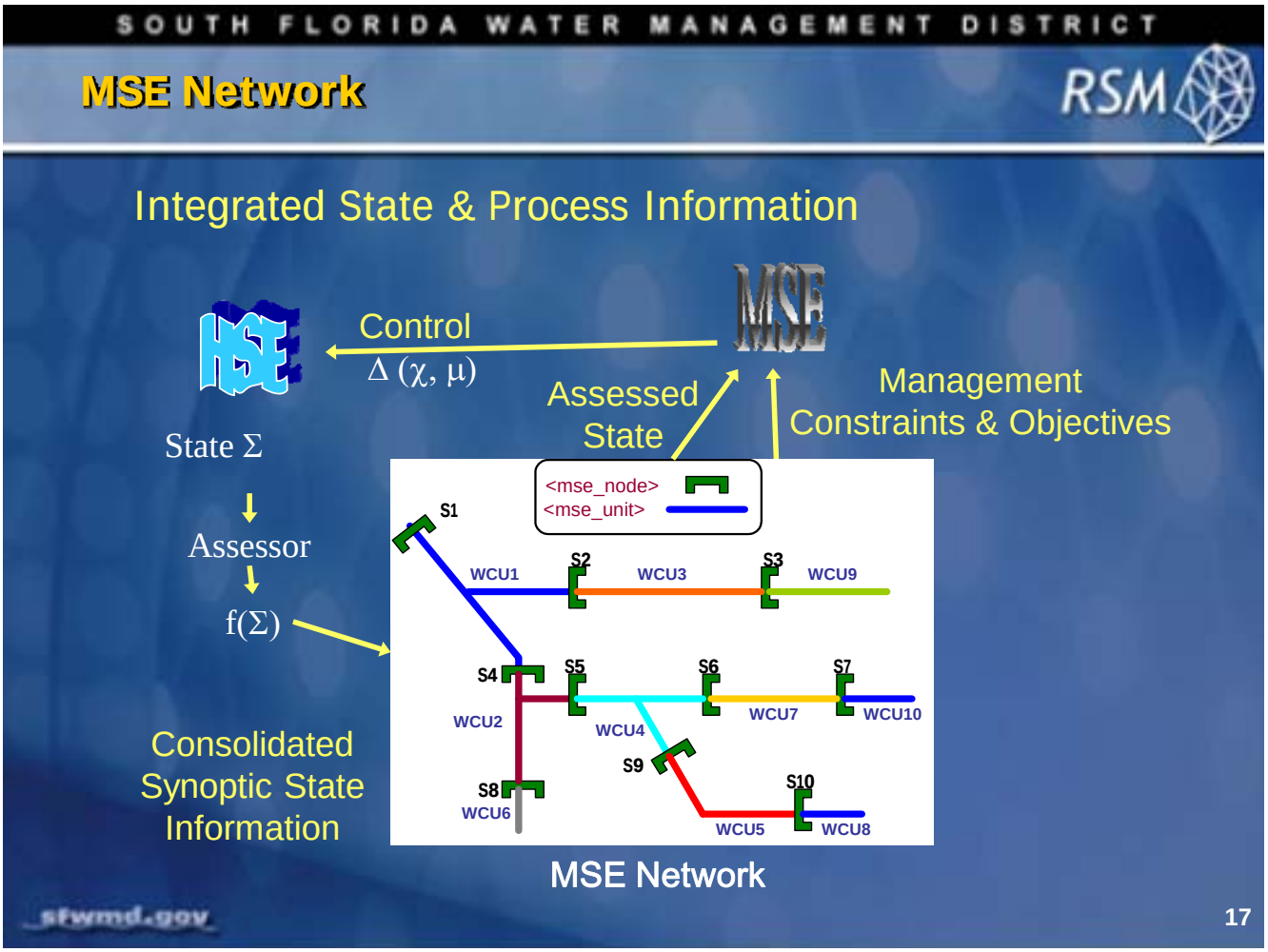
```
g++ S343AB_S344.cc -BSymbolic -fPIC -shared -o S343AB_S344.so
   -I../../../../trunk/src/ -I/opt/local/share2_64/include
```



The managed structures have watermovers where the flow is set by the controllers or the supervisors. The physical characteristics of the structure including the headwater and tailwater stages determine the flow capacity of the structure. The controller sets the flow within that capacity.

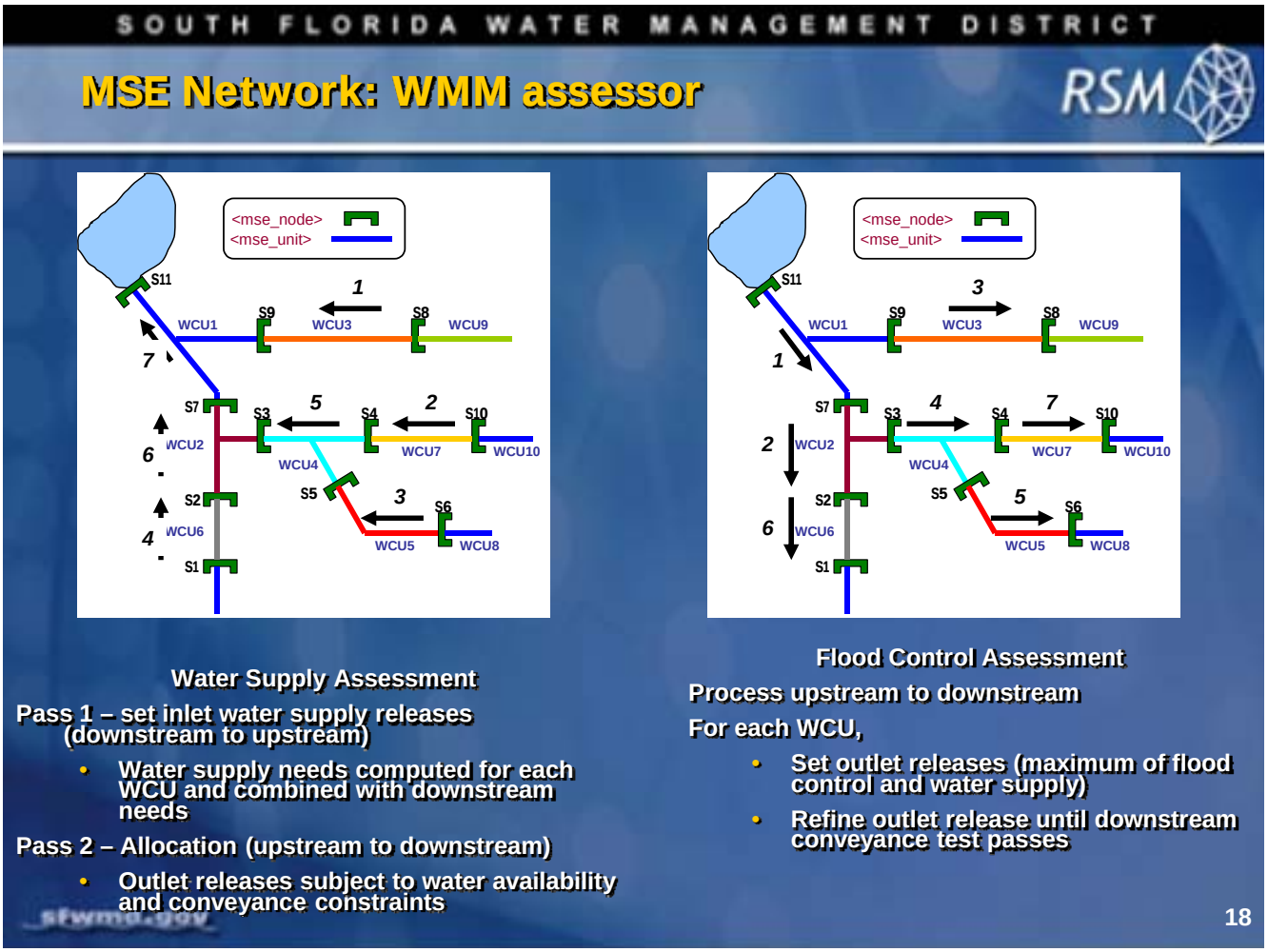


The MSE network is an abstraction of the HSE network to encapsulate the key components of the water control system without including the details of the HSE hydrology. The large number of segments within each WCU is unnecessary for managing the system. The MSE network provides a database on how you want the water management system to behave. This includes the flood control levels and/or water supply levels. Thus this information does not have to be carried by the HSE.

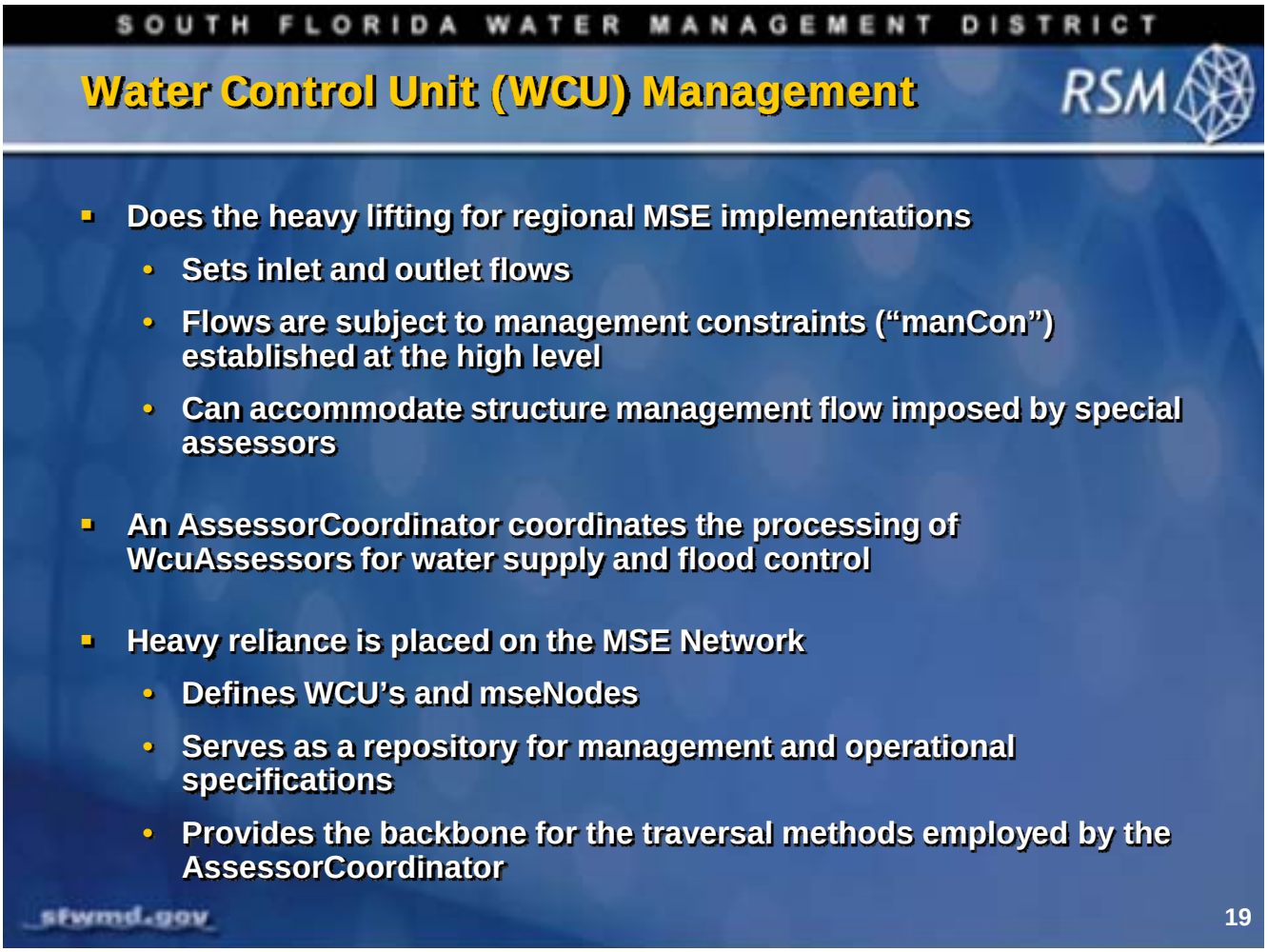


The MSE_network is an essential component of the MSE that links the components of the MSE together as well as linking the MSE to the HSE. The MSE network is an abstraction of the MSE information of the model and the hydrologic information in the model. To create an RSM-HSE you typically specify the canal network (segments) and mesh (cells) of the model. For the MSE to function it is not necessary to know the segments and cells. The MSE Network is an abstraction of the structures and the WCUs between the structures. The WCUs are composed of the segments that exist between the structures. The assessors take the state information from the HSE and provide consolidated state information for each WCU. This information, along with management objectives and constraints in the form of rules, is used to provide control information to the HSE.

The MSE network is specified in the xml. It is composed on nodes and WCUs. The nodes are tied to structures in the HSE. The nodes have various properties that can be abstracted from the HSE and include the rules that define water levels and flows at that node. The WCUs are the collection of waterbodies and control points and management constraints that describe how you want the system to behave.



This is an illustration of how the assessors are used to manage water supply (WS) and flood control (FC). This example, from Benchmark BM63, is a collection of fully defined WCUs that are canal reaches. The MSE_nodes are structures. This contains the details of the WCUs, the MSEnodes, the FC assessors and the WS assessors. The assessorCoordinator processes the FC assessors in the order in which they are presented in the xml file; order matters. They are processed in the order shown in the graphic on the right, from the upstream to downstream. After the FC assessors are applied, the WS assessors are processed from the downstream to the upstream inlet, determining the needs are to maintain water levels. This assessment determines the "needs". A second pass is made for the WS assessors from the upstream to the downstream allocating the available water by various means based on priorities defined by the users. Finally, another pass is made from downstream to upstream assigning the appropriate flood control needs for each structure. The final assigned flow is the larger of the flood control or water supply flows. The way these assessments are applied to the HSE is through controllers. The controller assesses the nodemonitor and than the controller sets the flow at the structure appropriately. The details are available in the documentation.



The WCU is where the MSE primarily determines the flow subject to the known management constraints. It should be remembered that the WCU is a waterbody; a lake, collection of cells etc… The inlets and outlets of the WCUs are nodes that can be controlled by special assessors. There is an AssessorCoordinator that controls how the WCU behaves. The MSE Network is the backbone for the implementation of the WCUs and the AssessorCoordinator. The MSE Network is the repository for the rules, policies and constraints as well as the structure characteristics.

SOUTH FLORIDA WATER MANAGEMENT DISTRICT

Water Control Unit (WCU)

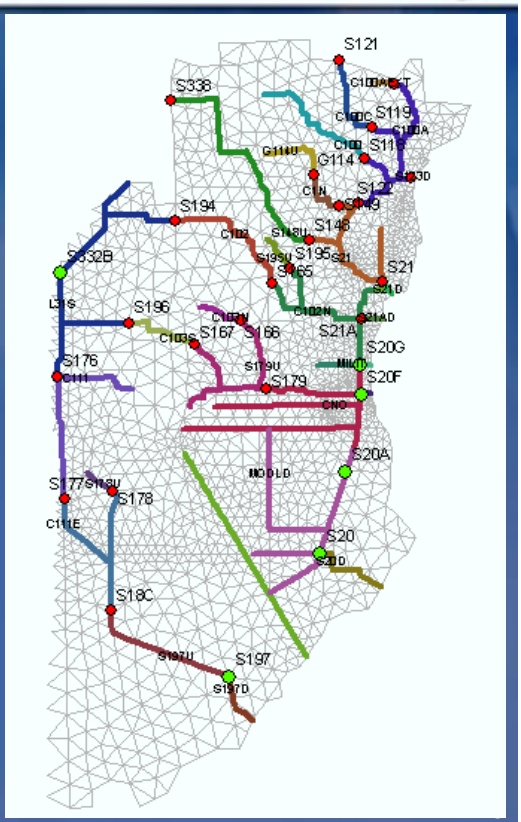
RSM

Typically, a collection of canal segments confined by MSE nodes that share the same operational constraints/ objectives.

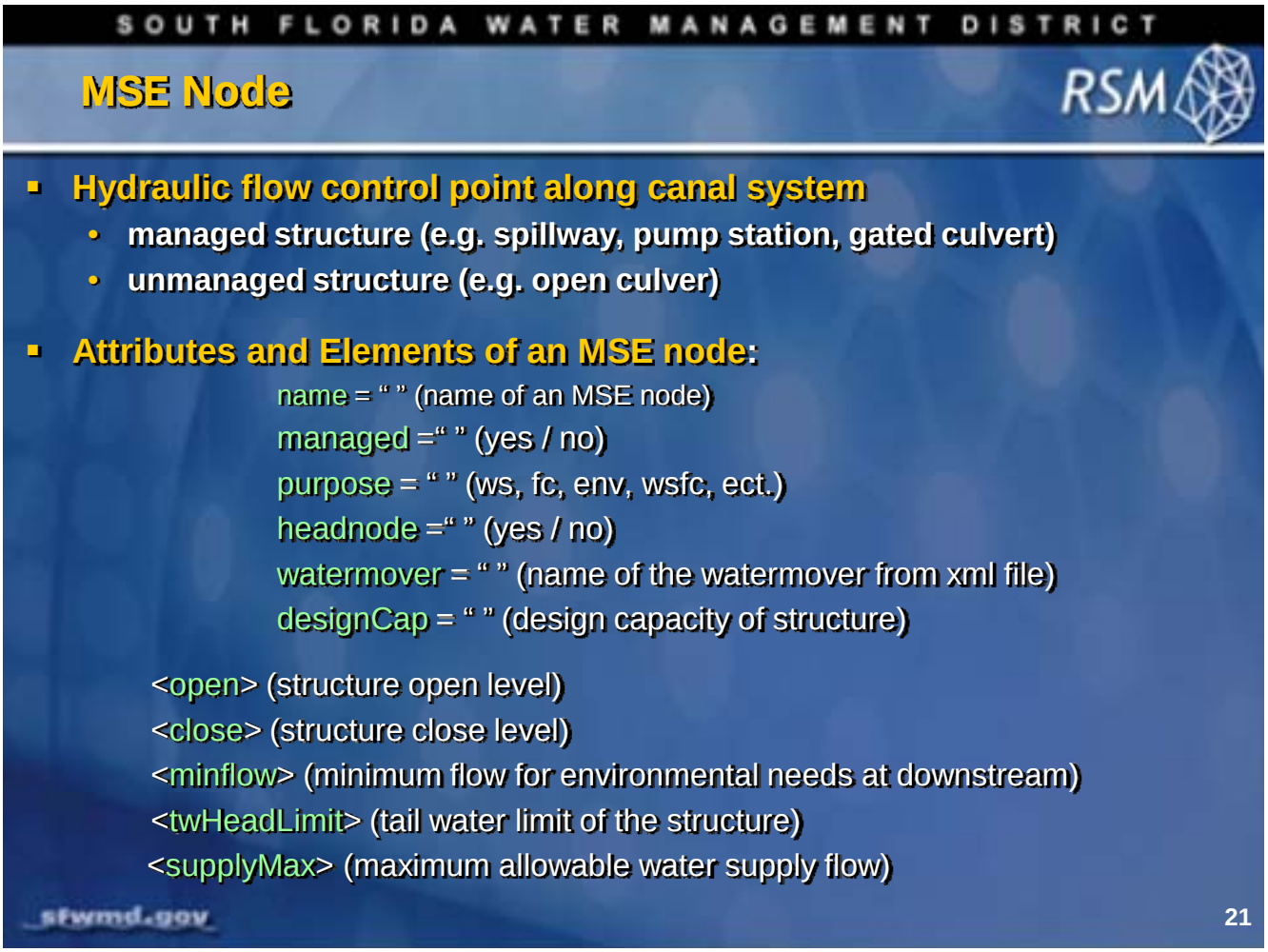
Elements of a WCU:

<hse_arcs>	list of segment ids
<hse_wbs>	list of waterbody ids
<wcds>	list of WCD ids
<lakes>	list of lakes ids
<basins>	list of basins ids
<impoundments>	list of impoundments ids
<inlet>	list of managed <mse_node>
<outlet>	list of managed <mse_node>
<maintLevel>	maintenance level for Water supply need
<fcLevel>	flood control level
<resLevel>	reserve level
<localLevel>	local level

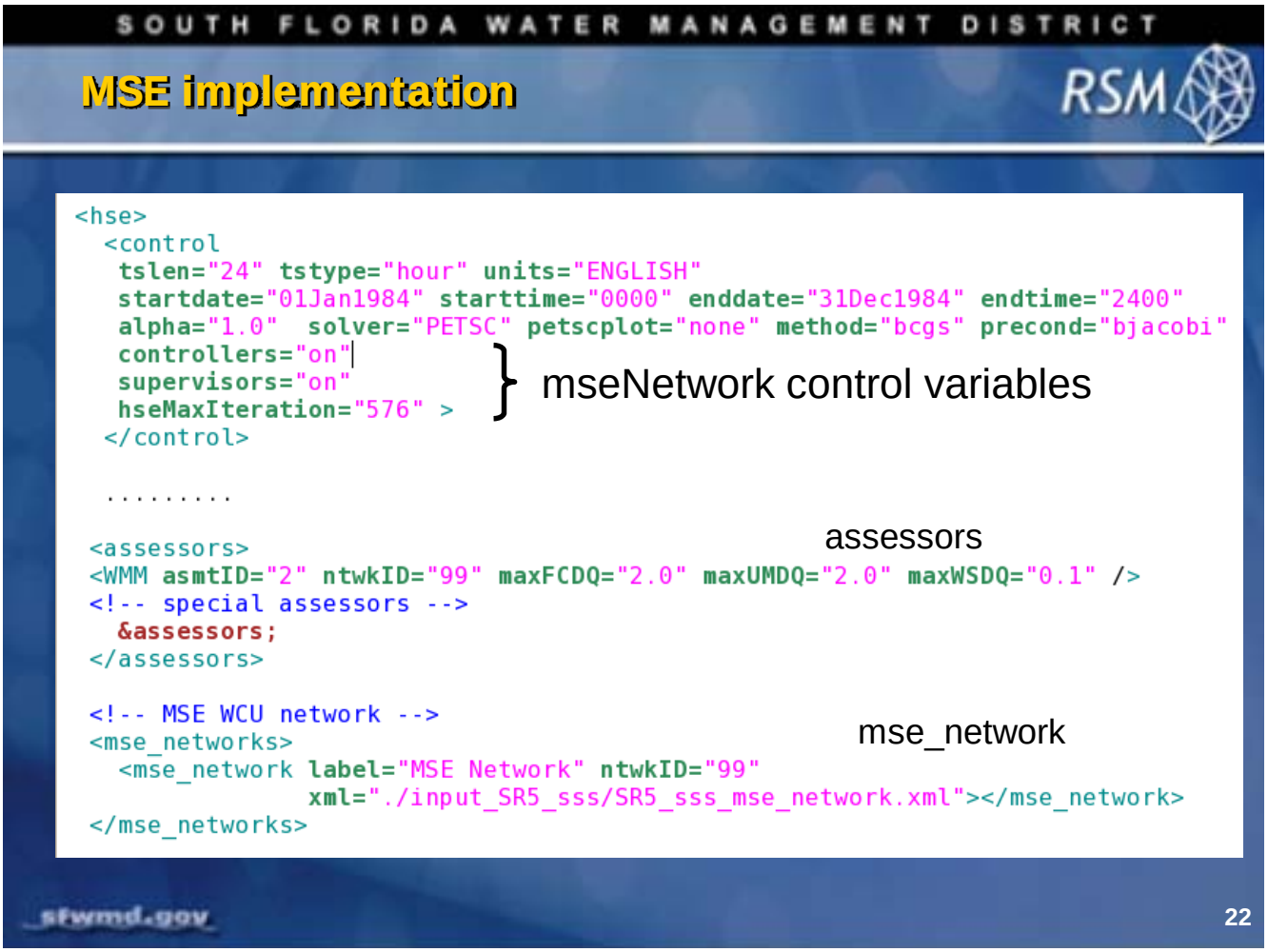
sfwmd.gov



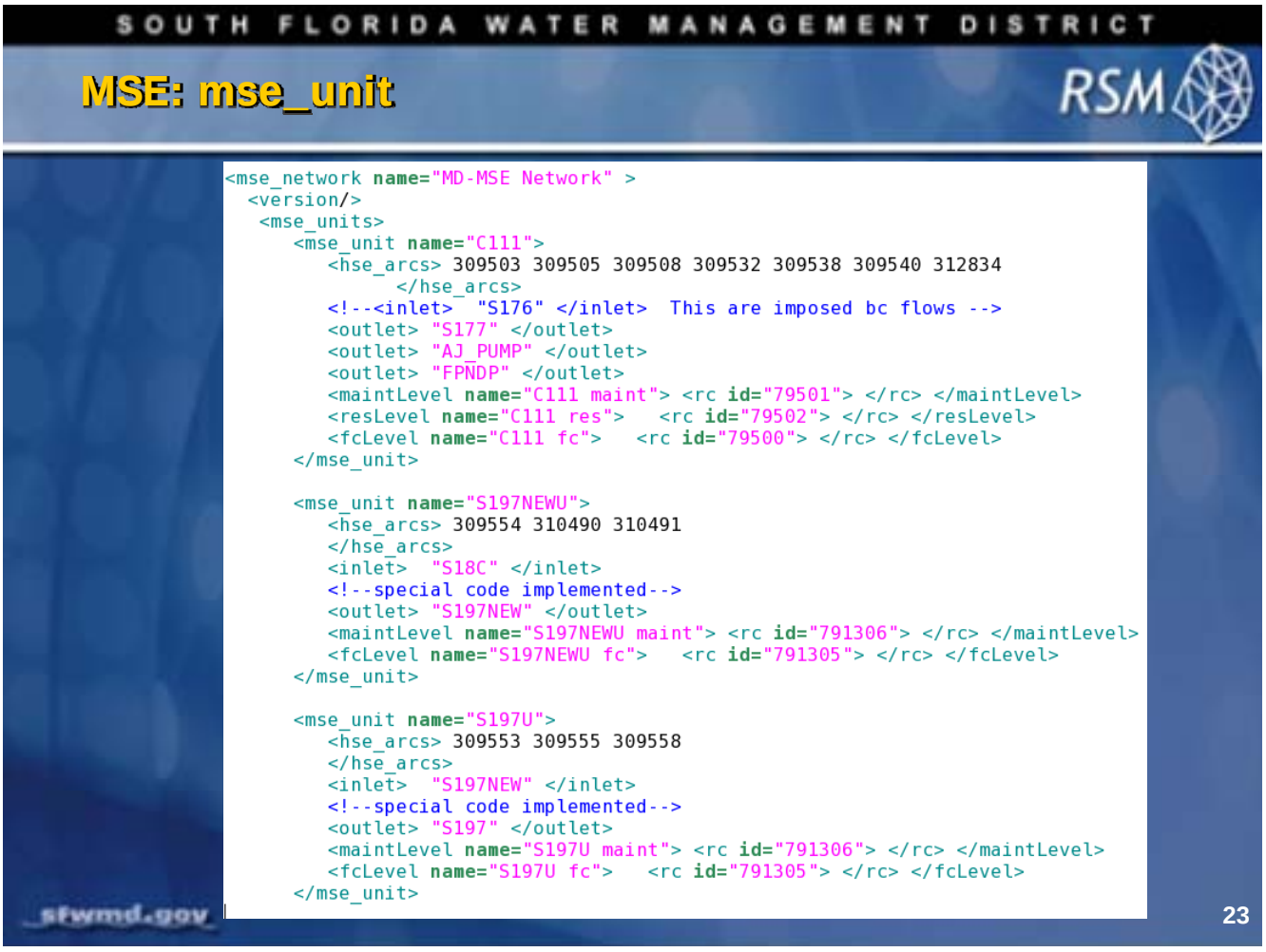
As an example, the Biscayne Bay coastal wetlands RSM is divided up into a number of WCUs. In this example, the WCUs are a collection of canal segments. Although the mesh cells drain to the WCUs, they are not defined as part of the WCUs. The WCUs can also be basins, cells, impoundments, WCDs or lakes. The elements include the canal segments and the inlet and outlet nodes. For each WCU it is necessary to define the flood control level, above which water is pumped out; maintenance level, the level that the WS assessors wish to maintain; the reserve level, below which water cannot be pumped out; and the local level, below which water cannot seep out or be used for evapotranspiration (ET).



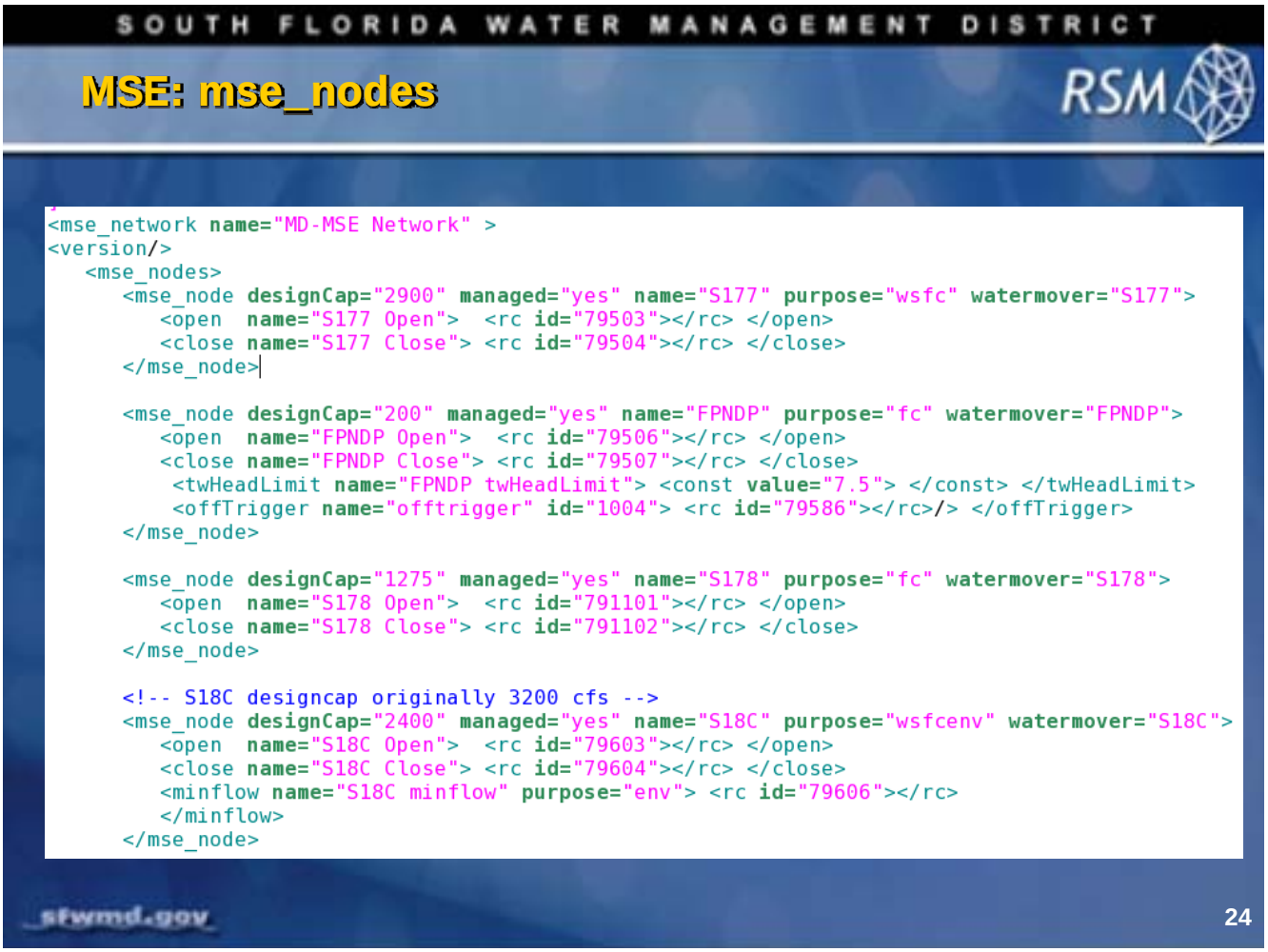
The second part of the MSE network is composed of the MSE nodes. The MSE nodes are the control points between the WCUs. An MSE node is typically a managed structure where the flow is controlled by setting gate openings, turning on pumps or an unmanaged structure such as a culvert of a fixed head weir. The node can be managed for flood control, water supply environmental or other purposes. Defining the purpose implements default assessors and controllers for operating the structure (watermover). The elements of the **<mse_node>** determine specific conditions when the structure can be open or closed and flow restrictions. The hydrologic data and operating rules are obtained from the SFWMD Structure Book.



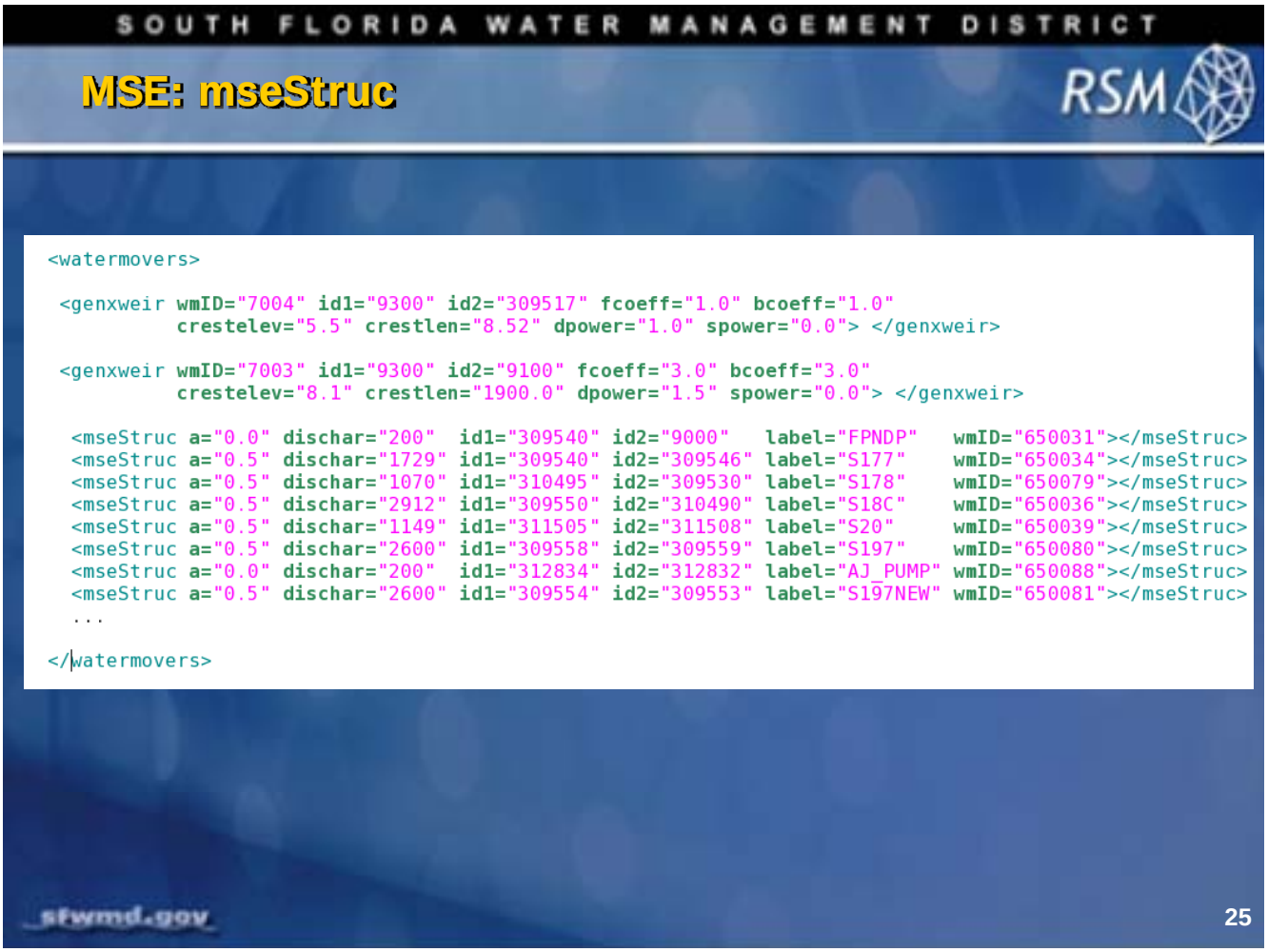
The implementation of the MSE requires several XML components. In the control block of the main XML file there are several attributes to control the MSE. The maximum iterations determine how long the MSE and the HSE will iterate until a solution is found. Remember, the MSE assessors are calculating estimated flood control and water supply requirements of the WCUs. The actual values will depend on the HSE solution for each structure for the head difference across the structure. The required flow determined by the MSE assessors is likely to be different from the flow calculated at the structure and as a result the RSM will iterate between the MSE and HSE solutions. The main XML file will contain the mse_network element that specified the name of the network and the location of the mse_network definition XML file. The main XML file will also contain the assessors or the file containing the assessors. Typically, the MSE will use the default assessors for water supply and flood control; only special assessors are specified in the XML file. For the water supply and flood control assessors the maximum difference for water supply flows, flood control flows and unmanaged flows between the HSE and the MSE solution are specified.



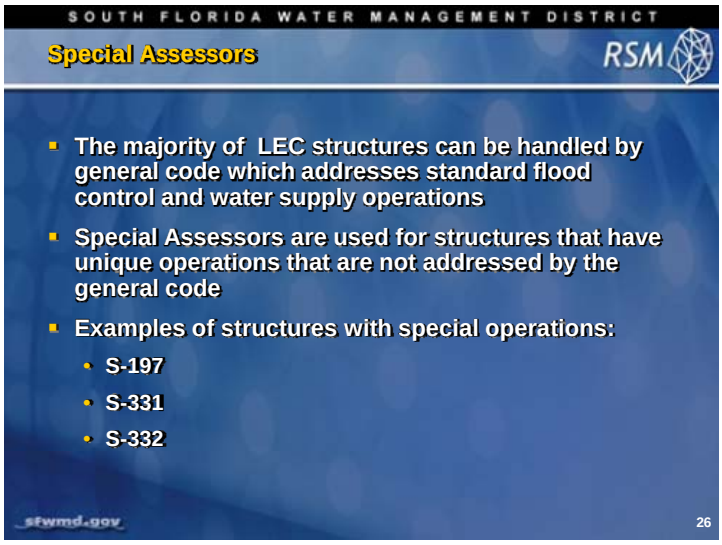
The typical MSE network will contain several WCU defined by the mse_units. Each mse_unit will have a list of waterbodies, typically a list of the canal segments from the HSE network that are to be included in the mse_unit. The inlet and outlet nodes are identified along with the stage targets within the WCU. The minimum stage targets include the flood control level **<fcLevel>** above which drainage is required and the water level maintenance level **<maintLevel>** below which water is requested from upstream. In these three WCUs, the target levels are set using rule curves that implement different water level targets from the dry season and wet season.



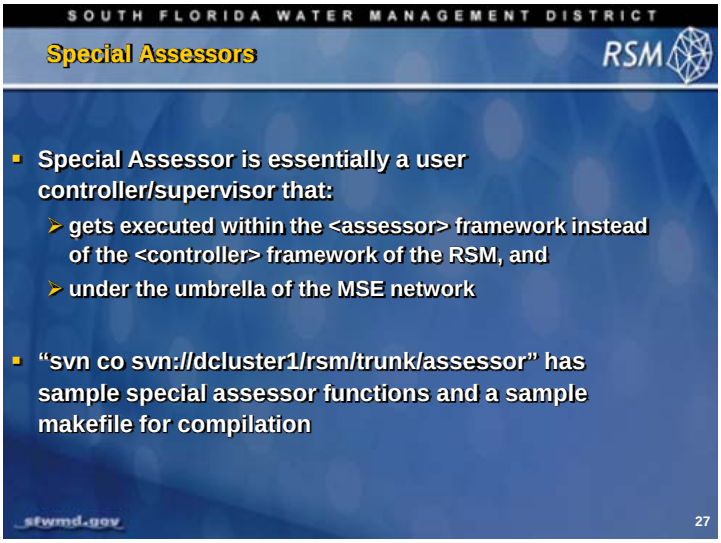
This is a typical implementation of the mse_nodes. Each node identifies the associated structure, purpose and design capacity. In the minimum, the water level are defined above which the structure is opened and below which the structure is closed. For these three mse_nodes the open and closed values are defined by rule curves. The values could be set as constants or time series. In these cases the rule curves define constant values. Typically, the rule curves are used to define seasonal or monthly targets. The mse_node for the FPNDP (Flood Protection pump) watermover has additional tailwater level restriction and a season restriction on pump operation. The S18C mse_node sets a monthly minimum flow for environmental protection. Notice that the purpose for each structure is different, they all have flood control but S177 also has water supply.



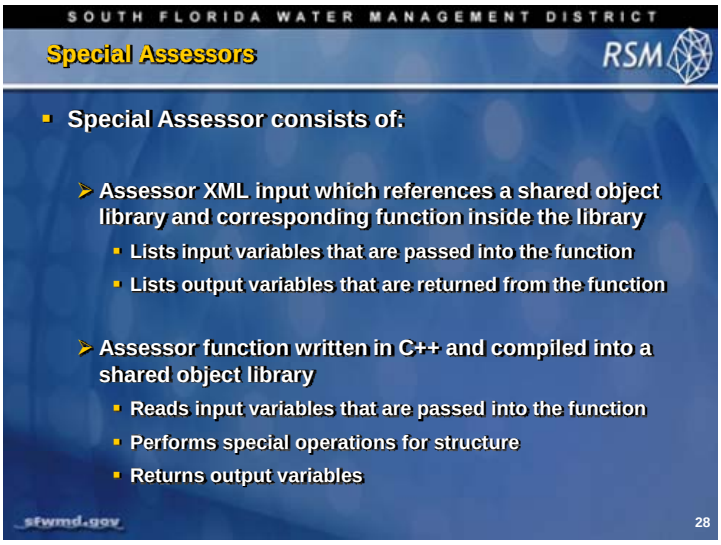
The flows determined by the MSE flood control, water supply and environmental assessors are implemented through the **<mseStruc>** watermovers. These watermovers define the discharge capacity of the structure and the mse_network determines the gain applied to the structure. As with all managed structures, these watermovers are controlled by controllers. The controllers are default controllers and visible to the user.



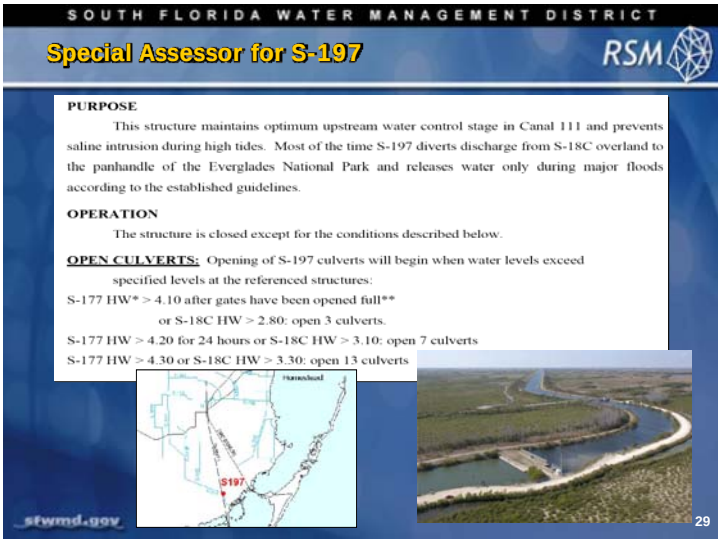
Occasionally there is a requirement for special assessors where the default assessors are not sufficient. Regional water management policies and practices can be implemented in the Lower East Coast Service Area (LECSA) using the standard RSM structures. This includes flood control that is managed using the upstream heads and water supply that is managed to maintain control levels in each WCU. The special assessors are necessary to control a few structures with complex water management. These include S-197, S-331 and S-332 in South Dade. It has not made sense to create generic controllers with these special requirements. There is no benchmark for the S-197 special assessor, but it can be seen in the C111 and other subregional models.



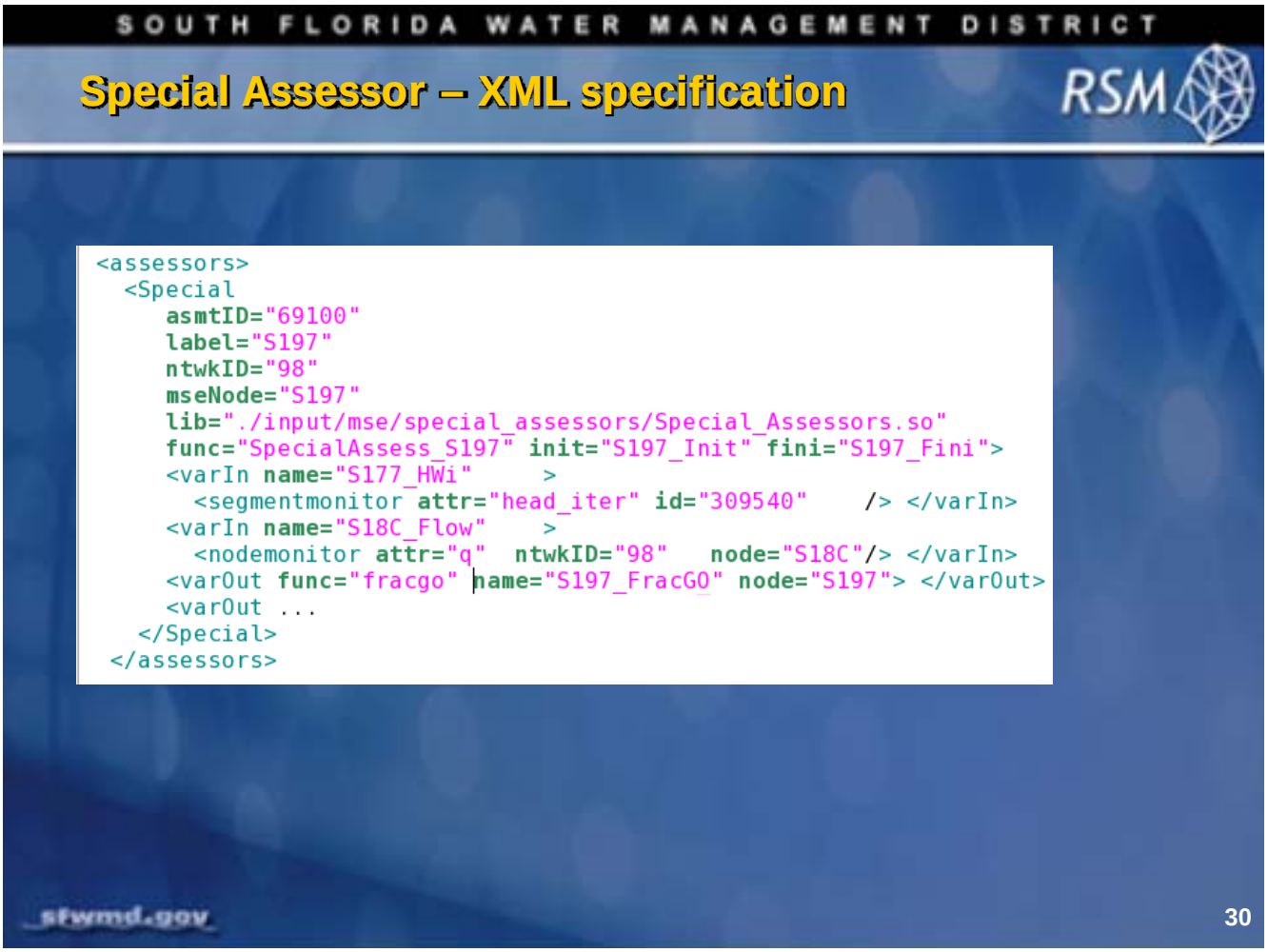
The special assessors are part of the MSE assessors so you need to have an MSE Network and you do not need standard controllers. It is necessary to check out the library of assessors. The source code is in that directory. There is a makefile for compiling the source code to create the shared library.



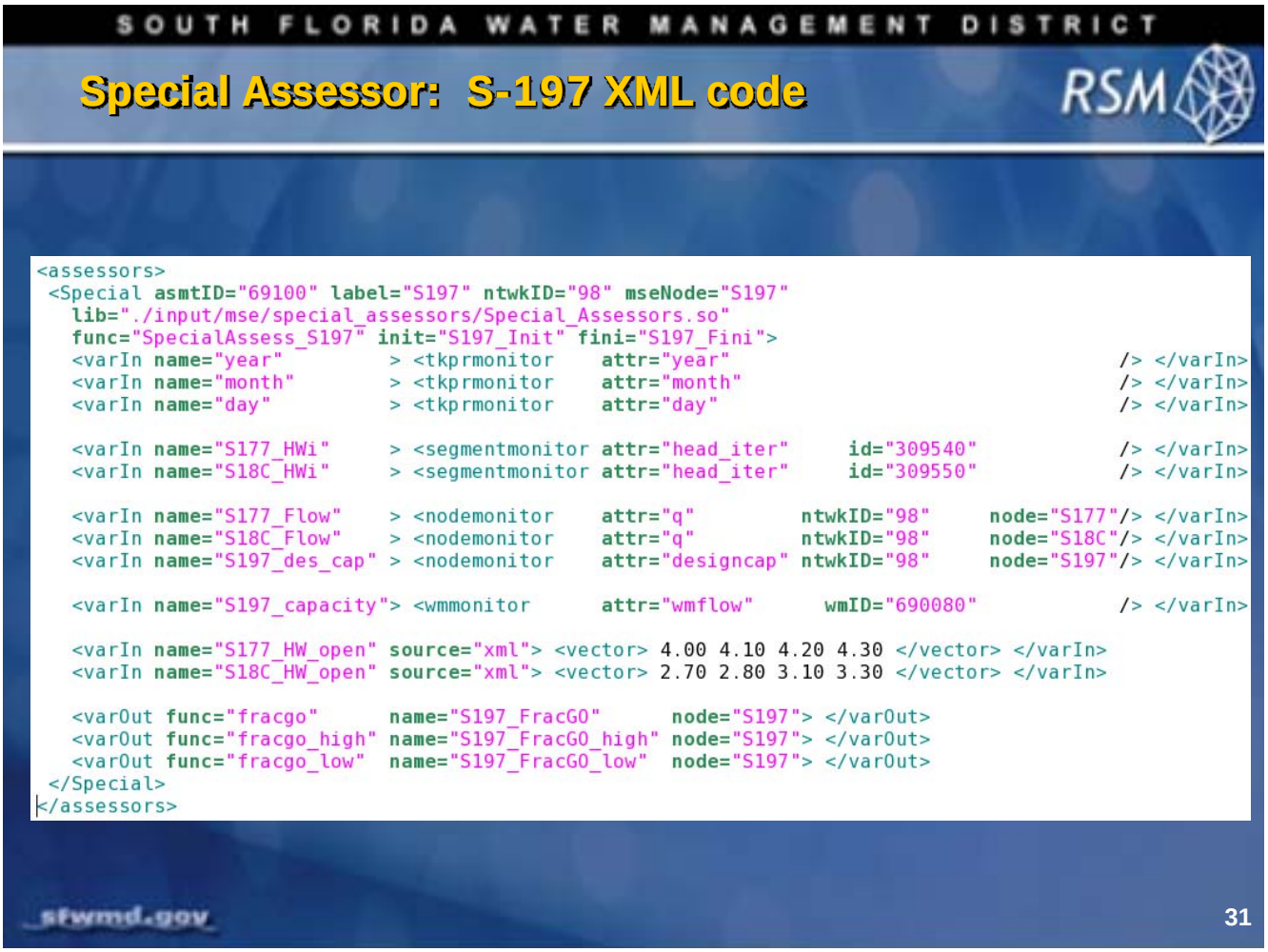
The Special Assessor has two components: the XML input and the compiled source code.



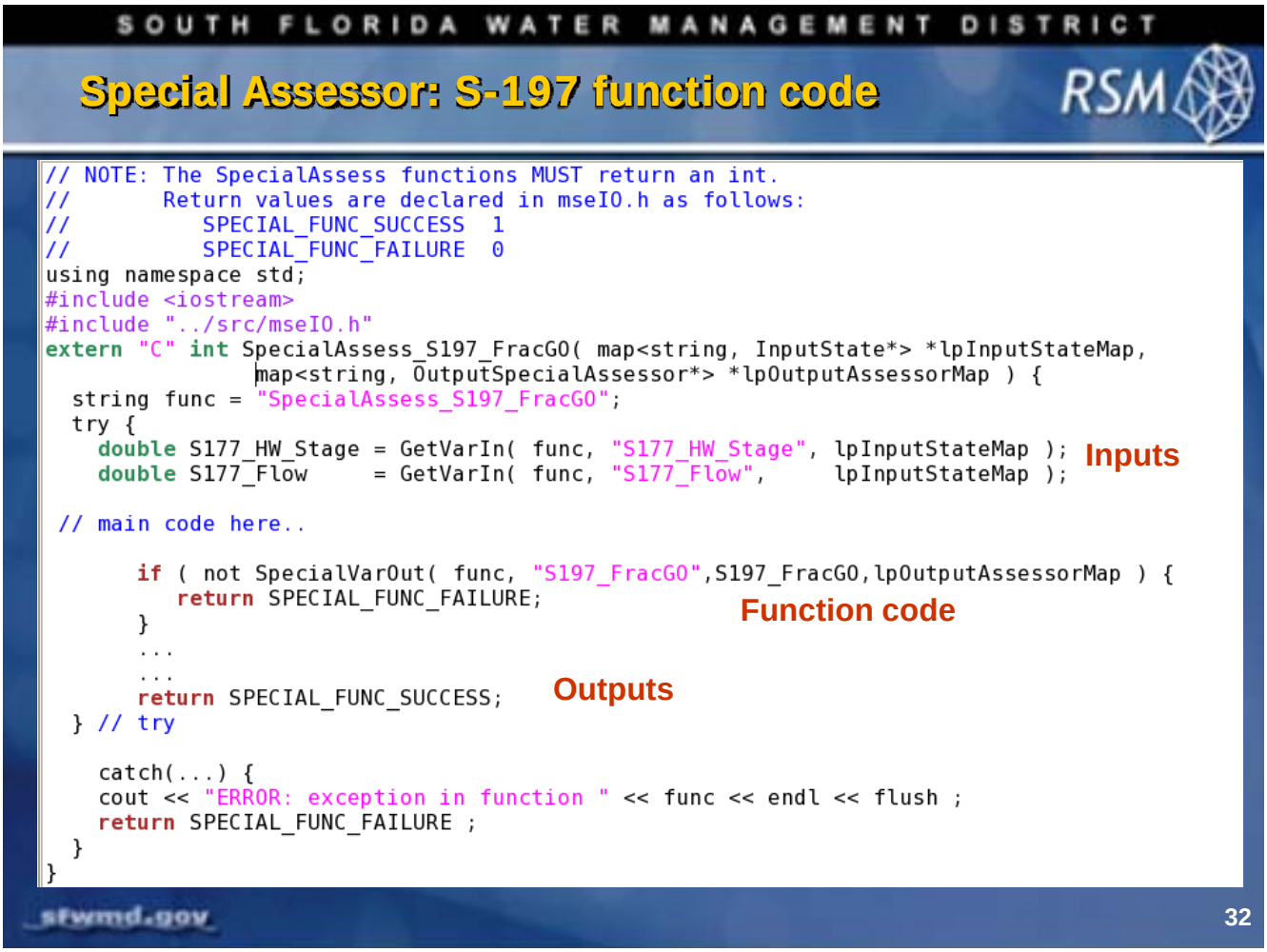
S-197 is a special structure located at the lower end of the canal system that is opened only to prevent serious flooding upstream. The structure has 13 culverts and the number of culverts that are opened depends on the head water levels at S-177 and S-18C structures located upstream. Since the structure operates more than one physical structure it was necessary to implement a special assessor to determine which culverts should be used.



The special assessor is defined in the run.xml file by the unique assessor identification number (asmtID), the network identification number (ntwkID), applicable MSE node (mseNode), the location of the shared function library (lib) and the function name in the shared library of compiled functions (func). The function may be written in Fortran, C or C++ language. The special assessor specification includes the list of input variables used by the special assessor and the output variables set by the assessor.

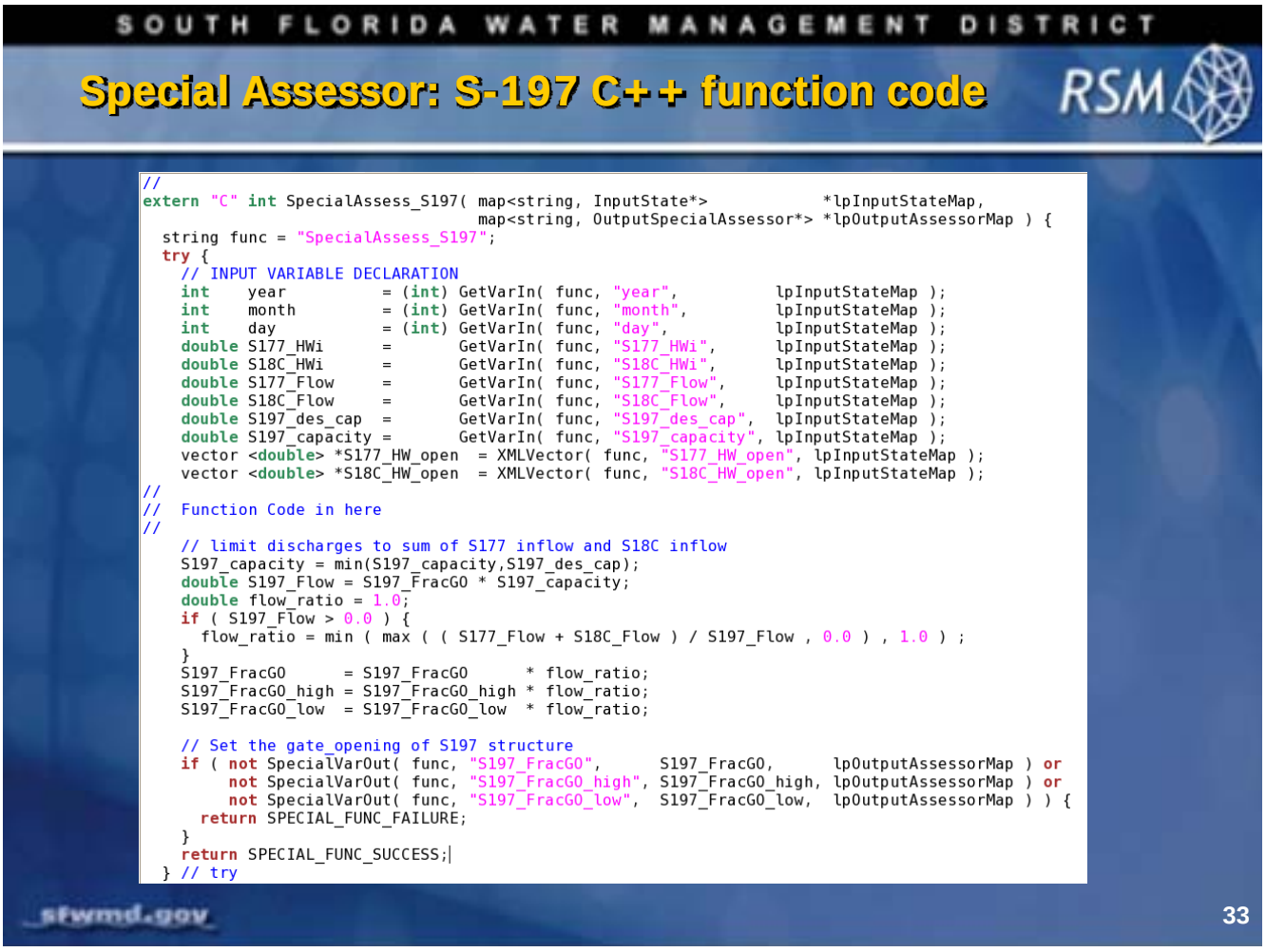


This is an example of the application of the management control applied to an individual structure. This is the management control of the S-197 structure. The management rules for the special assessor are written in C++ source code (presented below) and compiled in a shared library (SpecialAssessors.so). The various inputs for the assessor include the date (for seasonal stage requirements), head water stages, structure flows and rule curve vectors. The assessor can set the structure to zero flow (no operation), open three culverts, open seven culverts or open all 13 culverts.



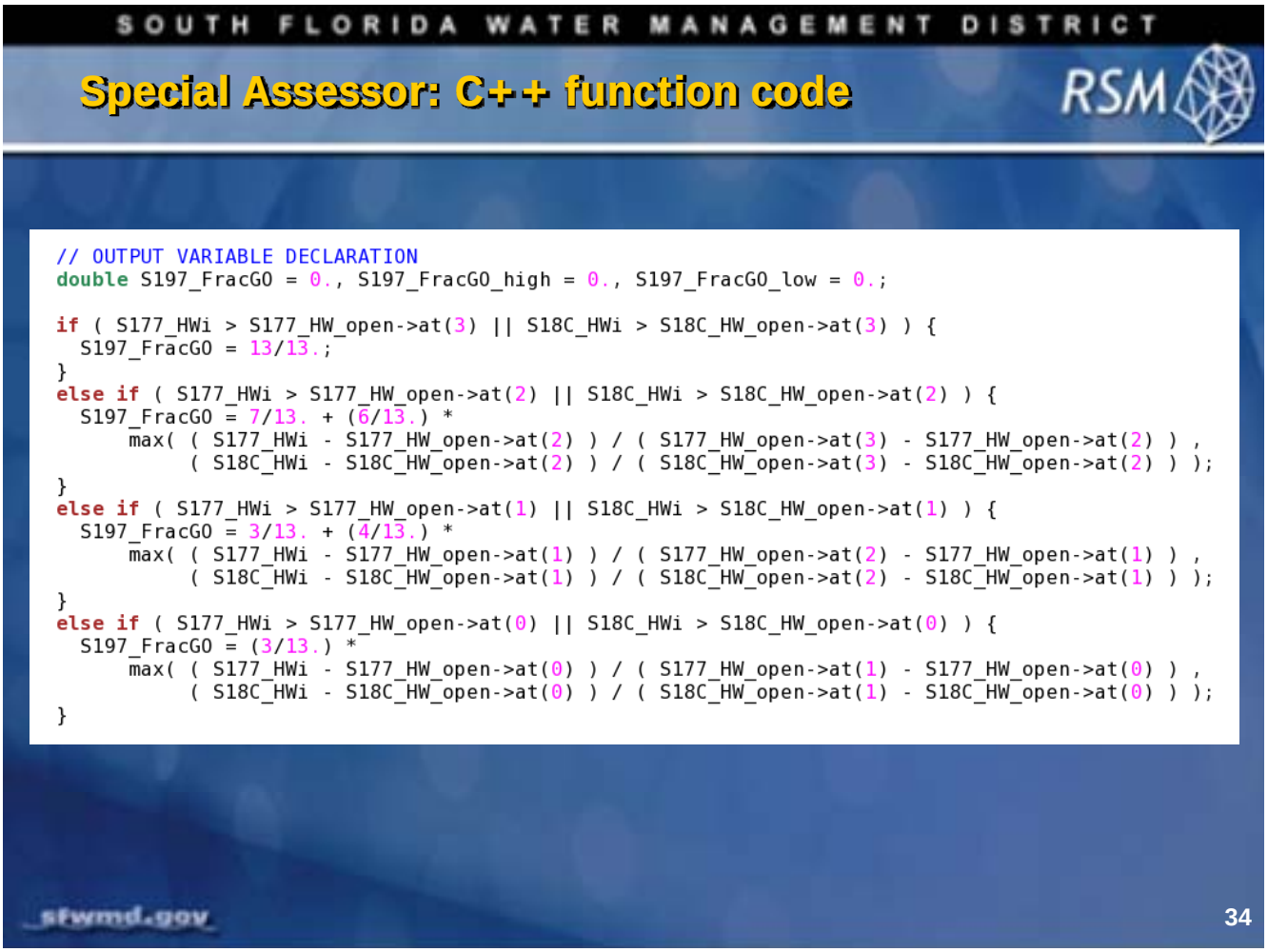
This is the standard source code template for a special assessor. There are a few statements that are changed:
1) extern function name: "**`SpecialAssess_S197_FracGO`**",
2) input variable names,
3) the main code that sets the output variables based on the input data, and
4) the return variable names.

As with all function codes, there is error trapping to avoid unreasonable results.




```
//
extern "C" int SpecialAssess_S197( map<string, InputState*>            *lpInputStateMap,
                                   map<string, OutputSpecialAssessor*> *lpOutputAssessorMap ) {
  string func = "SpecialAssess_S197";
  try {
    // INPUT VARIABLE DECLARATION
    int    year          = (int) GetVarIn( func, "year",          lpInputStateMap );
    int    month         = (int) GetVarIn( func, "month",         lpInputStateMap );
    int    day           = (int) GetVarIn( func, "day",           lpInputStateMap );
    double S177_HWi      =       GetVarIn( func, "S177_HWi",      lpInputStateMap );
    double S18C_HWi      =       GetVarIn( func, "S18C_HWi",      lpInputStateMap );
    double S177_Flow     =       GetVarIn( func, "S177_Flow",     lpInputStateMap );
    double S18C_Flow     =       GetVarIn( func, "S18C_Flow",     lpInputStateMap );
    double S197_des_cap  =       GetVarIn( func, "S197_des_cap",  lpInputStateMap );
    double S197_capacity =       GetVarIn( func, "S197_capacity", lpInputStateMap );
    vector <double> *S177_HW_open  = XMLVector( func, "S177_HW_open", lpInputStateMap );
    vector <double> *S18C_HW_open  = XMLVector( func, "S18C_HW_open", lpInputStateMap );
//
//  Function Code in here
//
    // limit discharges to sum of S177 inflow and S18C inflow
    S197_capacity = min(S197_capacity,S197_des_cap);
    double S197_Flow = S197_FracGO * S197_capacity;
    double flow_ratio = 1.0;
    if ( S197_Flow > 0.0 ) {
      flow_ratio = min ( max ( ( S177_Flow + S18C_Flow ) / S197_Flow , 0.0 ) , 1.0 ) ;
    }
    S197_FracGO      = S197_FracGO      * flow_ratio;
    S197_FracGO_high = S197_FracGO_high * flow_ratio;
    S197_FracGO_low  = S197_FracGO_low  * flow_ratio;

    // Set the gate_opening of S197 structure
    if ( not SpecialVarOut( func, "S197_FracGO",      S197_FracGO,      lpOutputAssessorMap ) or
         not SpecialVarOut( func, "S197_FracGO_high", S197_FracGO_high, lpOutputAssessorMap ) or
         not SpecialVarOut( func, "S197_FracGO_low",  S197_FracGO_low,  lpOutputAssessorMap ) ) {
      return SPECIAL_FUNC_FAILURE;
    }
    return SPECIAL_FUNC_SUCCESS;
  } // try
```

The special assessor is defined by the first five attributes. The varIn provides the input data that the S-197 special assessor requires. The requirements for this special assessor come directly from the SFWMD Structure Book that describes the operations of the CSFFCP. S-197 has thirteen culverts which can be set to zero, three, seven or 13 culverts open. The headwater levels are provided in the vector above. Note: the return variables must be continuous. The return variables are a range of gate openings for each group of culverts.



This is a fragment of the C++ function code that is included in the special assessor code. It is a series of if-then-else statements that determine the number of gates opened as well as the flow ratio for each group of gates based on the water levels at S177 and S18C structures.

This function code can be altered and recompiled using the g++ compiler and added to the function library to create a new special assessor code for the structure.

KNOWLEDGE ASSESSMENT

(pre- and post-lecture quiz to assess efficacy of training materials)

1. Why maintain a separation between MSE and HSE?
2. What are the three levels of MSE implementation?
3. How does MSE interact with HSE?
4. What is the key component of assessors?
5. What is the role of controllers in MSE?
6. How are user-defined controllers & supervisors implemented in MSE?
7. What is the role of supervisors in MSE?
8. How do managed watermovers behave?
9. What are the MSE nodes and MSE Water Control Units?
10. How are flood control and water supply needs processed?
11. What do special assessors do?
12. How are special assessors modified?

Answers

1. Creating a separate MSE allows for a greater flexibility in the development and implementation of the water resource management rules and policies for managing structure flows.

2. There are three levels of MSE implementation:
   - Execute high level regional rules and water management policies
   - Manage flood control and water supply demands for basins
   - Control operations of the individual structures.

3. The MSE gets information from the HSE through monitors and assessors, then the MSE implements management rules through controllers.

4. The key components of assessors are the data monitors. The data monitors can monitor any state or dynamic variable as well as perform operations on those values.

5. The controllers are necessary to set the flow for each managed structure. The MSE uses default controllers that are defined by the model and user-defined controllers.

6. The user-defined controllers and supervisors are implemented using standard C^{++} supervisor programs that contain the necessary logic to set the controllers. These programs can be edited and recompiled to change the management rules.

7. Supervisors control one or more controllers using the information from one or more monitors. Supervisors can be used to implement complex rules or regional rules that are implemented using multiple structures.

8. For managed watermovers, the MSE sets the flow for the structures up to the design flow, or the physically allowable flow for the current conditions.

9. The nodes define the structures controlling the flows to and from the WCU. The WCU defines the waterbodies included in the WCU, the inflows and outflows, and the water control elevations.

10. First the flood control assessments are made for each WCU starting at the upstream WCU and working downstream. Then the water supply requirements are evaluated for each WCU beginning with the lowest WCU and working upstream. Finally, the water supply allocations are made working downstream based on the remaining water supply and flood control needs.

11. The special assessors are used to provide complex assessments for the state of the system using various monitors and setting the flow for managed structures similar to the use of controllers.

12. Similar to user-defined controllers, special assessors are implemented through standardized, user edited C++ programs.

Lab 15: Water Management Model Assessors

Time Estimate: 4.0 hours

Training Objective: To investigate the implementation of watermover control in the Regional Simulation Model.

Watermover control is obtained directly through **Hydrologic Simulation Engine (HSE)** watermover controllers and indirectly through the use of assessors in the **Management Simulation Engine (MSE).** This lab explores the implementation of **user-defined controllers**, which have been implemented in the **HSE**, and the implementation of **WMM_assessors** to control watermovers.

NOTE:

For ease of navigation, you may wish to set an environment variable to the directory where you install the RSM code using the syntax

`setenv RSM <path>`

For SFWMD modelers, the path you should use for the NAS is:

`/nw/oomdata_ws/nw/oom/sfrsm/workdirs/<username>/trunk`

`setenv RSM /nw/oomdata_ws/nw/oom/sfrsm/workdirs/<username>/trunk`

Once you have set the RSM environment variable to your trunk path, you can use $RSM in any path statement, such as:

`cd $RSM/benchmarks`

Training files are currently located in the following directories:

```
INTERNAL_TRAINING
|
|____data
|       |__geographic
|       |__C111
|       |__rain+et
|       |__glades_lecsa
|       |__losa_eaa
|       |__BBCW
|
|____trunk
|       |___benchmarks
|       |___hpmbud
|
|____labs
```

Files for this lab are located in the **`labs/lab15_mse`** directory. Additional materials in the directory include:

`bm63b.pptx`

Activity 15.1 User-Specified Controllers

Overview

Activity 15.1 includes two exercises:

- **Exercise 15.1.1.** Modify <userctrl> in Benchmark 45.
- **Exercise 15.1.2.** Modify <userctrl> for SR29 in Glades-LECSA RSM

The **Regional Simulation Model (RSM)** has the capability of implementing user-defined controllers, which are described in **Benchmark 45.** There are four examples in **BM45 w**hich test the use of controllers and supervisors written in either the C++ or the C language. We will investigate the use of controllers in the C++ environment.

Exercise 15.1.1 Modify <userctrl> in Benchmark 45

1. In the trunk folder, copy the **`$RSM/trunk/benchmarks/BM45`** directory to new directory **`BM45a`**
   - **`cp -r BM45 BM45a`**
2. Run **test.script** to ensure **`BM45a`** is working correctly
3. Run the simple controller (**`run3x3.xml`**) from the **RSM Graphical User Interface (RSM GUI)**
4. Observe the results in **`t3x3out.dss`** using the **`HecDssVue`** utility
5. Select **canal S01 head**, **WM1 control** and **WM1 flow**. Keep the graph

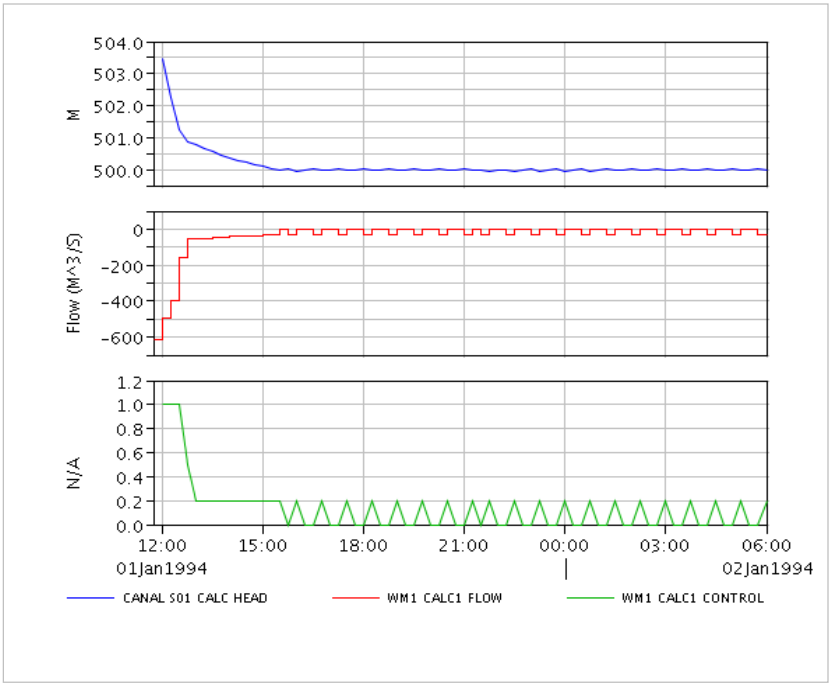


Figure 15.1 Time series of stage, flow and control-setting for the user_ctrl

6. Edit the **run file** (**`run3x3.xml`**) and turn off the **controller cid="`101`"**

HINT

Don't forget to remove or comment out <ctrlmonitors> for cid=101.

```
<!-- Watermovers to be controlled -->
<watermovers>
  <!-- discharge from canal segment 1 -->
  <hq_relation wmID="1" id="1" label="">
     <hq>
      0.0    0.0
      495.0  0.0
      499.0 -50.0
      500.0 -150.0
      501.0 -300.0
      510.0 -1000.0
     </hq>
  </hq_relation>
  <!-- inflow into canal segment 4 -->
  <hq_relation wmID="2" id="4" label="">
     <hq>
      0.0    0.0
      490.0  1000.0
      495.0  500.0
      499.0  250.0
      500.0  150.0
      501.0  50.0
      510.0  0.0
     </hq>
  </hq_relation>
</watermovers>

<controller id="1">
  <!-- Controller for discharge from segment 1 -->
  <userctrl cid="101" label="Segment 1 Ctrl " wmID="1" control="on"
   libType="C" module="./User_C.so" func="Segment1_Control" ctrlMin="0.0" ctrlMax="1.0" >
     <varIn name="Segment1"><segmentmonitor id="1" attr="head"/></varIn>
     <varIn name="Segment4"><segmentmonitor id="4" attr="head"/></varIn>
  </userctrl>
</controller>
<controller id="2">
  <!-- Controller for pumping into segment 4 -->
  <userctrl cid="102" label="Segment 4 Ctrl" wmID="2" control="on"
   libType="C" module="./User_C.so" func="Segment4_Control" ctrlMin="0.0" ctrlMax="1.0" >
     <varIn name="Segment1"><segmentmonitor id="1" attr="head"/></varIn>
     <varIn name="Segment4"><segmentmonitor id="4" attr="head"/></varIn>
  </userctrl>
</controller>
```

Figure 15.2 The watermovers and control functions used with user-specified controllers

7. Rerun model, display results and compare to the previous results
   - How does the **stage** behave?
8. Edit the **`userctrl.cc`** file and change the lower control value for segment 1 from 500 to 499.5

   The controller behavior is described in a C++ file that sets the control variable. The user-controller is implemented using the **`<controller>`** block in the xml file (**Fig 15.2**). The variables "**`Segment1_Control`**" and "**`Segment4_Control`**" are set in the **`userctrl.cc`** file (**Fig. 15.3**).

9. Recompile the **user-control library** by executing the **makefile script** using the "**make**" command
10. Rerun the model and compare current results to previous results
11. Change the output gain "**`controlOut`**" from 1.0 to 0.60 and run the model
    - Compare the results to previous results
12. Change the **`controlOut`** to ***6.0*** and observe the results

```
//------------------------------------------------------------------
// Function Segment1_Control for control of watermover into segment 1
//
extern "C" double Segment1_Control( map<string, InputState*> *lpInputStateMap ) {

  string func = "Segment1_Control";
  double controlOut = 0.;

  double segment1Head = GetVarIn( func, "Segment1", lpInputStateMap );
  // These are not used, but illustrate how to get values from XML input
  double xmlScalarVal = XMLScalarValue( func, "xmlScalar",         lpInputStateMap );
  double xmlVectorVal = XMLVectorValue( func, "xmlVector",  2,     lpInputStateMap );
  double xmlMatrixVal = XMLMatrixValue( func, "xmlMatrix",  2, 3, lpInputStateMap );

  // Provide control function based on input state variable
  if ( segment1Head > 502. )
    controlOut = 1.;
  else if ( segment1Head > 501. )
    controlOut = 0.5;
  else if ( segment1Head > 500. )
    controlOut = 0.2;
  else
    controlOut = 0.;

  return controlOut;
}
```

Figure 15.3 The userctrl.cc file used to set rules for controlling structure

Exercise15.1.2 Modify <userctrl> for SR29 in Glades-LECSA RSM

User-defined controllers can be used for specific structures but are rarely used in the current versions of the subregional models. Effective for controlling the behavior of individual structures, user-defined controllers have proven expensive to scale-up to the subregional or regional implementations.

In this exercise you will find the user-defined controllers and describe how they are used.

13. In the **`$RSM/data/glades_lecsa`** directory, find the **`run_81.xml`** file.

HINT

Find the `<controller>` block in the `run_81.xml` file. Input files are broken out into different directories by regions of the model.

14. Find the **controllers** for the **SR29** structures:
    - What are the **inputs** to the **controller**?
    - What is the file with the **control variables**?
    - What does the **control rule** in **SR29_seasonalctrl.cc** do?
    - What **watermovers** does this rule control?
    - What does the **control behavior** look like?
    - What are the **flow values** for the structures?
15. Run the model and graph the **flow data** for the **SR29** structures.

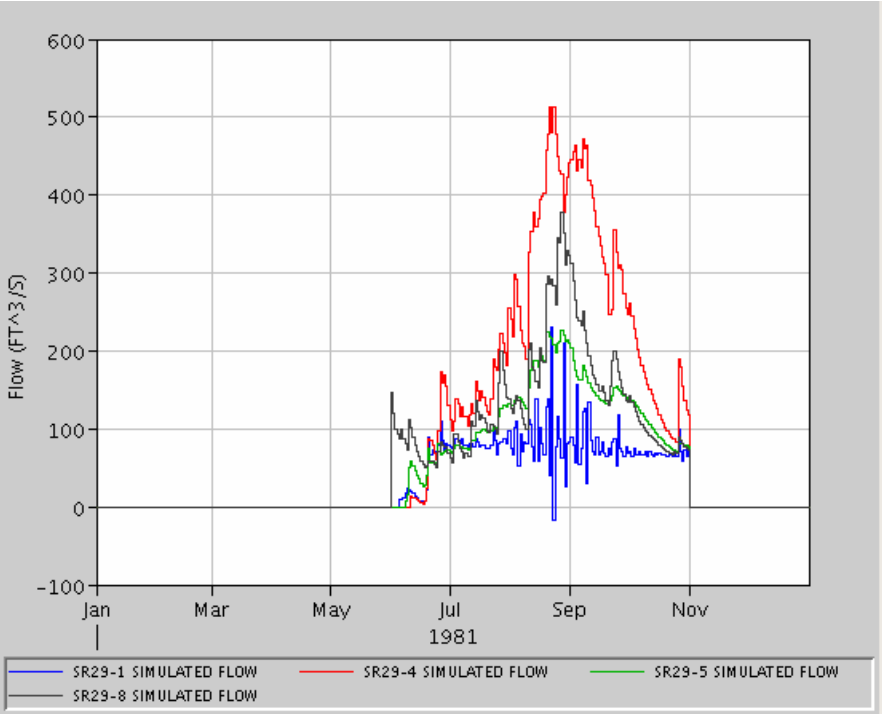


Figure 15.4 Simulated discharge at the SR29 weirs

16. Edit the **SR29_seasonalctrl.cc** and change the **dryseason control from 0.0** to ***0.2***.

17. Recompile the **control file** using the instructions in **SR29_seasonalctrl.cc.**

HINT

The compilation instructions are at the top of the file.

18. Run the model using **RSM GUI** and compare the results to the previous run.

Activity 15.2 WMM_Assessor (Benchmark 63b)

Overview

Activity 15.2 tests the behavior of the WMM_Assessor. This activity includes one exercise:

- **Exercise 15.2.1.** Run WMM_Assessors in BM 63b.

The **WMM_Assessor** uses **assessors** to determine the amount of water from the Hydrologic Simulation Engine (HSE) that needs to be moved to or from a water control unit to meet the **water supply (ws)** needs or **flood control (fc)** requirements for each basin. The user-entered **mse_units** and **mse_nodes** are used to construct the **<mse_network>**. The **wmm_assessor <WMM>** determines the required **management of the structures <mse_struc>** from "**off**" to full "**on**" at each managed structure, based on the requirements for each structure.

Exercise 15.2.1 Run WMM_Assessors in BM63b

In order to understand the structure of the **mse_network**, map out the network manually:

19. In the benchmark folder (**`$RSM/trunk/benchmarks`**) copy Benchmark 63b to the **`lab15_mse`** directory. All work for this activity will be completed in the **`lab15_mse/BM63b`** directory.
    - **`cp -r $RSM/trunk/benchmarks/BM63b $RSM/labs/lab15_mse`**
20. List the **`mse_network.xml`** file.
21. Create a diagram of the **mse_network** connecting the **mse_units** using the **mse_nodes**. Use **`bm63b.pptx`** as a starting point.
    - List the **water supply maintenance levels** next to each **`<mse_unit>`**
    - List the **maximum discharge capacity** next to each **`<mse_node>`** (structure)
    - List the **flood control level** for each reach (look at **`<mse_node>`** openings)
22. Construct a plot that shows the **discharge behavior** of **`<mseStruc>`** **S9** and **S8**.

    The flow is calculated using the standard weir equation as follows:

    $Q = \text{dischar}\ \Delta h^{a}$

    - The default for **a** is ***0.5*** (if not specified)
23. Add the **`<runDescriptor>`** element to the control section of **`wmm_assessor.xml`** file.
    - **`hseMaxIteration`** = "**`144`**"

- **runDescriptor** = "**forest**"

24. Run **Benchmark 63b** using the **RSM GUI**.
25. Observe the following data using HecDssVue from the RSM GUI:
    - Cell heads (**cellheads.dss**)
      Find the cell locations and WCU using the Groundwater Modeling System (GMS) graphic for mesh and network (**Fig. 15.1** in the Answer-lab15 file). If you don't have access to GMS software, you can use **bm63b.pptx** as a starting point.
      - What do these values indicate?
    - **Segment heads (segmentheads.dss)**
      - Which **WCUs** become flooded?
    - **Structure** flows (**structureflows.dss**)
    - **Nodeflows**
      - Plot the Q_{ws} and Q_{fc} for a couple of nodes. What do you see?
    - **Lake**
    - **Bcflows**
      (These are the flows into or out of the domain. This may be important where you are trying to produce net outflow, balance or net inflow.)
    - **Iterations**

    The **Management Simulation Engine (MSE)** sets the flows at the structures and these are imposed on the **Hydrologic Simulation Engine (HSE)** matrix solution. The **Regional Simulation Model** iterates the **HSE** solution until the **HSE** flows converge with the MSE flows.
26. Replace the **marsh Hydrologic Process Module (HPM)** with an **irrigated land HPM**. (See **Fig. 15.5**)
    - Change the **HPM** from **<layer1nsm>** (forest) to **<afsirs>** (turf) by substituting the code shown in **Fig. 15.5**.
    - Change the **runDescriptor** in the **<control>** block to "***turf***".
    - Save the model as **wmm_assessors_b.xml.**

```
<hpModules>
  <indexed file="hpm.index">
    <hpmEntry id="1">
      <afsirs>
        <afcrop label="turf" id="15" j1="01-01" jn="12-31" depth1="12"
depth2="24">
          <kctbl>
             1.00 1.00 1.00 1.00 1.00 1.00
             1.00 1.00 1.00 1.00 1.00 1.00
          </kctbl>
          <awdtbl>
             0.50 0.50 0.50 0.50 0.50 0.50
             0.50 0.50 0.50 0.50 0.50 0.50
          </awdtbl>
        </afcrop>
        <afirr label="MULTIPLE SPRINKLER" wtd="2.5">
          <irrmeth id ="4" eff="0.85" arzi="1.0" exir="0.7"></irrmeth>
          <irrmgmt label = "DROUGHT" trigcode="2" value="100"></irrmgmt>
        </afirr>
        <afsoil label="SANDY LOAM SOILS" depth="96" minwc="0.09"
maxwc="0.15" cond="1">
        </afsoil>
      </afsirs>
    </hpmEntry>
  </indexed>
</hpModules>
```

Figure 15.5 Example irrigated HPM (turf)

27. Run the model using the RSM GUI.
28. Observe the results compared to the previous run as "**forest**" by opening the following **DSS files**:
    - **cellheads.dss**
    - **segmentheads.dss**
    - **nodeflows.dss**
    - **iterations.dss**
29. Use **HecDssVue** and plot the "**forest**" **time series** and the "***turf***" **time series for heads**.
    - How have the **cell** and **segment heads** changed from the **forest HPMs**?
    - How have the **nodeflows** changed?
    - How has the number **of iterations** changed?
30. Change the **water management level** in each **WCU** by adding a "**reserve level**" to the **WCUs** at the upper end of the regional system. (This will allow water to be delivered to the lower reaches to allow those **WCUs** to maintain higher water tables.)

31. In the **mse_network.xml** file, add and replace the following lines to **mse_unit name= "Reach#" (Fig. 15.6).**

```
      <mse_unit name="Reach1">
         <hse_arcs> 100 101 102 103 104 105 106 107 118 119 120 121

         </hse_arcs>
         <inlet>  "S12" </inlet>
         <outlet> "S7"  </outlet>
         <outlet> "S9"  </outlet>
         <maintLevel name="reach1 maint"> <const value="4.5"/>
</maintLevel>
         <resLevel name="reach1 reserve"> <const value="3.5"/> </resLevel>
      </mse_unit>
```

Figure 15.6 Reach description for **mse_network.xml** (step 13).

- Add to **Reach1**:

```
         <resLevel name="reach1 reserve"> <const value="3.5"/> </resLevel>
```

- Add to **Reach2**:

```
         <resLevel name="reach2 reserve"> <const value="3.0"/> </resLevel>
```

- Add to **Reach3**:

```
         <resLevel name="reach3 reserve"><const value="2.75"/></resLevel>
```

32. Save the file as **mse_networkb.xml**.
33. Change **<mse_network>** in **wmm_assessorb.xml** to **xml="mse_networkb.xml".**
34. Change the **runDescriptor** in the **<control>** block to "**reserve**". Save the file.
35. Run **wmm_assessorb.xml** and observe the results (**Fig. 15.7**):
    - **cellheads.dss**
    - **segmentheads.dss**
    - **nodeflows.dss**
      - How have the **cell** and **segment heads** changed?
      - How have the **nodeflows** changed?

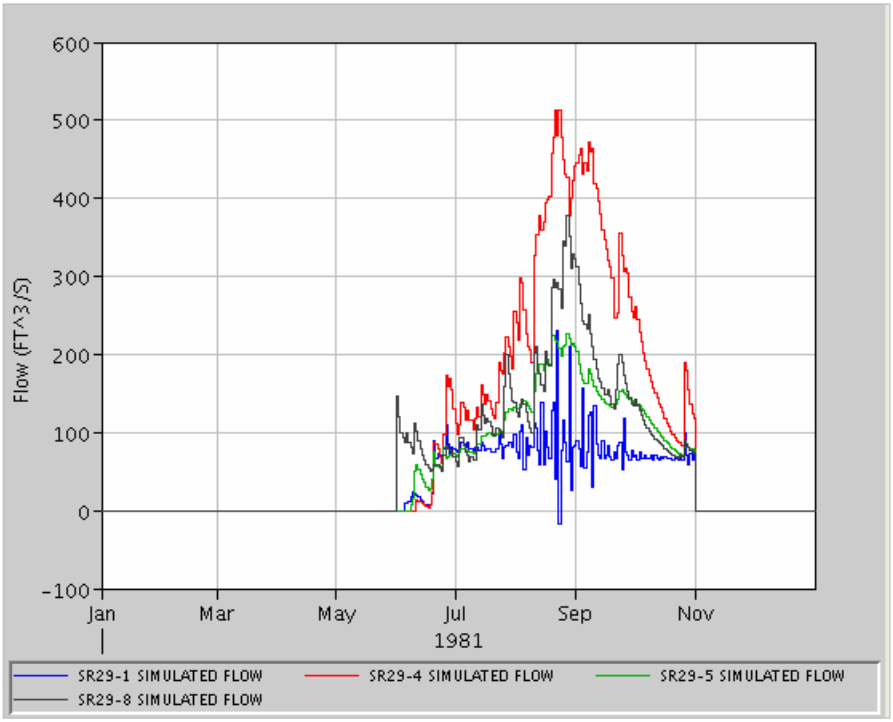


Figure 15.7 Resulting node flows for modified model

Answers for Lab 15

Exercise 15.1.1

7. Rerun model, display results and compare to the previous results

   How does the **stage** behave?
   With the very jagged controls removed the head and discharge curves became much smoother, but retained the same pattern as before.

10. Rerun model and compare to the previous results

    How does the **stage** behave?
    The stage decreases smoothly to 499.5 m, the flow steps down to 0 and the controller begins oscillating later.

11. Rerun model and compare to the previous results

    How does the **stage** behave?
    The stage and flow decrease more gradually.

12. Change the **`controlOut`** to **`6.0`** and compare to the previous results

    How does the **stage** behave?
    The stage and flow decrease more sharply.

Exercise 15.1.2

2. Find the **controllers** for the **SR29** structures:

   - What are the **inputs** to the **controller**?
     The inputs are year, month, day, SR29_HW which is a segment specific head, and str_name which is the structure name.
   - What is the file with the **control variables**?
     The file is **`input/glades/SR29_seasonalctrl.so`**
   - What does the **control rule** in **SR29_seasonalctrl.cc** do?
     It provides the operating rules for SR29 – If it is between November and May (inclusive), control out = 0 (closed), else control out = 1.0 (open). Basically, it determines whether the structure is open or closed.
   - What **watermovers** does this rule control?
     This controls wm-660003, wm-660006, wm-660007, and wm-660010.
   - What does the **control behavior** look like?
     The control behavior is binary, either open or closed. This decision is based on the structure parameters and system state.
   - What are the **flow values** for the structures?
     The flows are between 0 and 500cfs. Generally, they ramp up in June, peaking around September 1, and declining to zero around November 1.

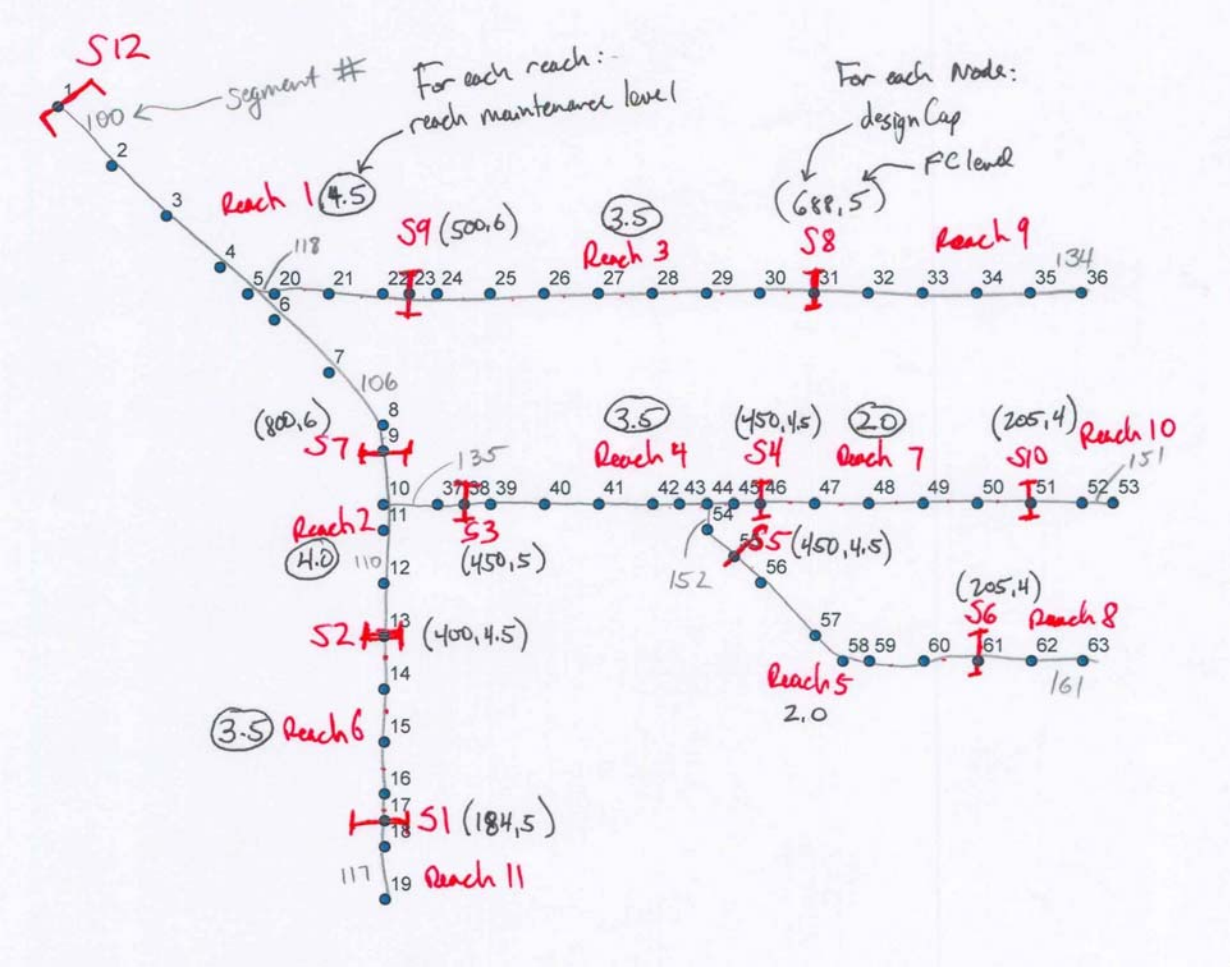
6. Rerun model and compare to the previous results

   How does the **stage** behave?
   This produces flow during the dry season.

Exercise 15.2.1

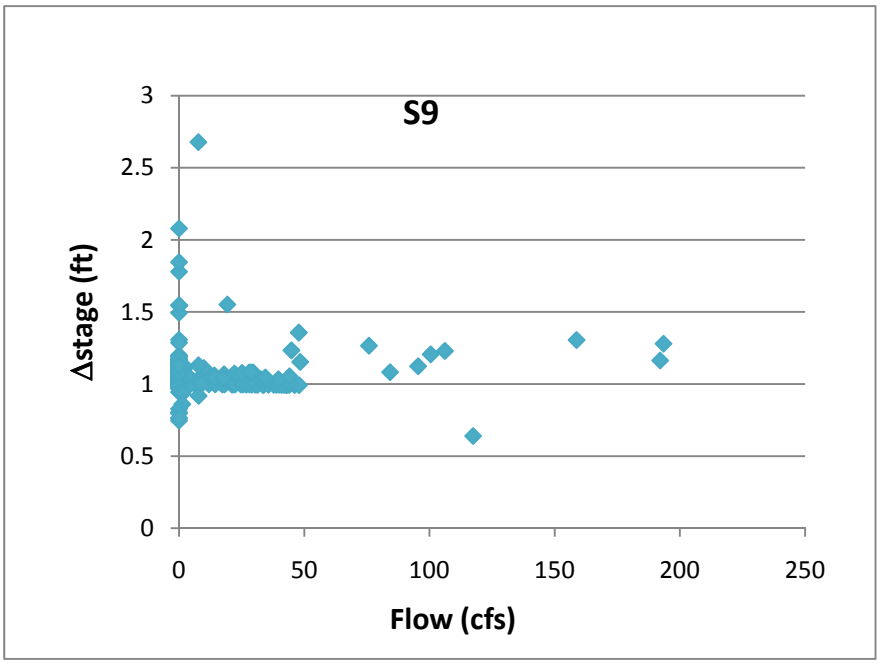
NOTE: A diagram of the MSE network and cell map can be found in the $RSM/labs/lab15_mse/answersBM63b/Figure15-1.jpg file.

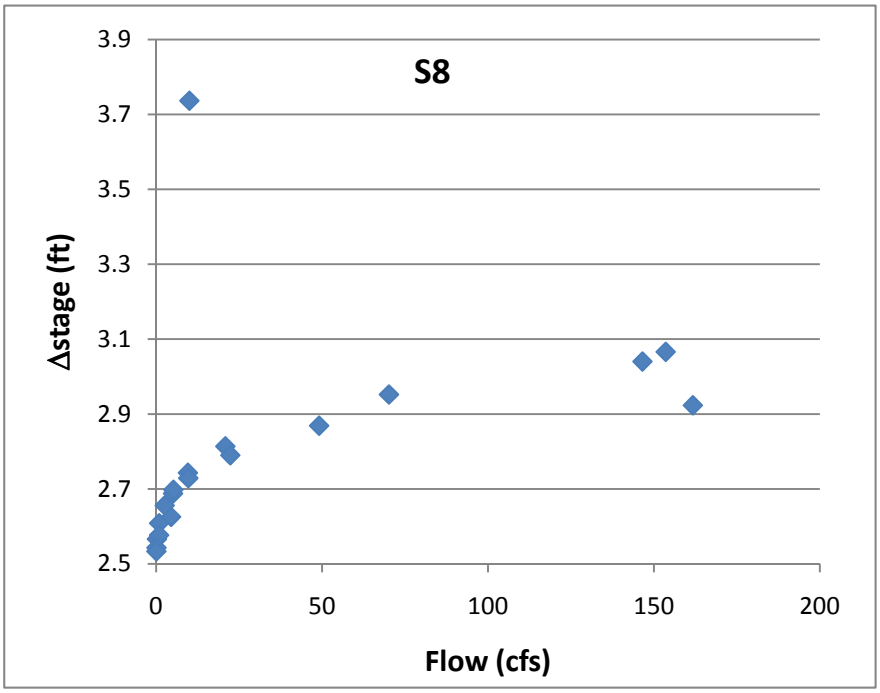
3. Create a diagram of the mse_network connecting the mse_units using the mse_nodes.



4. Construct a plot showing discharge behavior of **<mseStruc> S9** and **S8**.

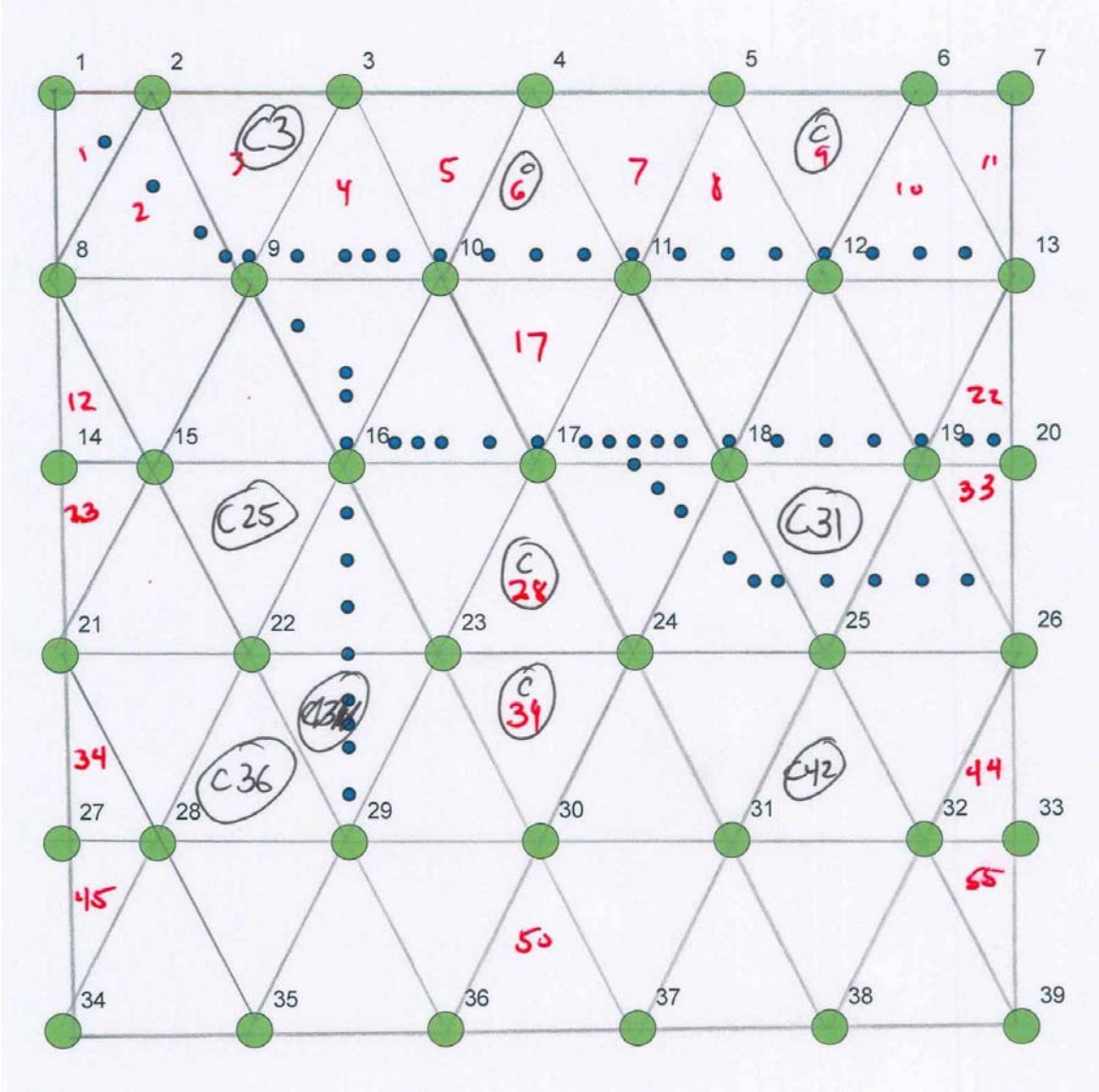
Plot ΔStage vs. flow for both structures using the simulated data from the segmenthead.dss file and nodeflow.dss file. S8 shows flow through a standardweir. S9 shows flow through a managed weir. See plots on next page.


S9
Δstage (ft)
Flow (cfs)
3
2.5
2
1.5
1
0.5
0
0
50
100
150
200
250


S8
Δstage (ft)
Flow (cfs)
3.9
3.7
3.5
3.3
3.1
2.9
2.7
2.5
0
50
100
150
200

7. Observe the following data using HecDssVue from the RSM GUI:

- Cell heads (cellheads.dss) Find the cell locations and WCU using the Groundwater Modeling System (GMS) graphic for mesh and network.



- What do these values indicate?
  These values indicate that the general flow (surface and groundwater) of the system is to the east and south. Cell 3 has the highest head and the lowest are cells 9, 36, and 39.
- Which **WCUs** become flooded?
  I was somewhat uncertain what the specific conditions for flooding were, so I assumed flooding occurs if the head in the segment was greater than the FC level in the downstream node. Reaches 1, 2, 3, and 4 (only barely, and briefly for reach 4) get flooded, provided this assumption.
- Plot the Q_{ws} and Q_{fc} for a couple of nodes. What do you see?
  The flood control flows only occur when the segments are flooded, corresponding to

the answer above. The water supply flows correspond to the upstream nodes very well. The highest amounts of water are provided to reaches 4, 5, and 7.

11. Use **HecDssVue** and plot the "**forest**" **time series** with the "***turf***" **time series**.

- How have the **cell** and **segment heads** changed from the **forest HPMs**?
  The turf head cells generally have a smaller and more stable value of head. The turf cell heads exhibit the same pattern as the forest cell heads but are generally lower, particularly in the summer months.

- How have the **nodeflows** changed?
  Increased water supply is necessary for the turf, but less flood control.

- How has the number **of iterations** changed?
  The maximum amount of iterations over the time period is nearly the same, but the turf consistently requires more iterations during the first half of the year.

17. Run **mse_network.xml** and observe the results:

- How have the **cell** and **segment heads** changed?
  There is virtually no change in the cell heads given the reserve level. Segment heads exhibit minimal change with heads typically being slightly lower during peak heads.

- How have the **nodeflows** changed?
  The node flows have minimal change due to the reserve level being set.

Index