4. Object Level Accounting Methods

In simulation, local inflows and gains/losses are typically input by the user, set by rules, or calculated by methods on the objects. For example, local inflows to a reservoir may be input on the Hydrologic Inflow slot and daily Evaporation may be calculated from a monthly table. These components represent a change in the physical amount of water in the object and must also be represented by the accounting system.

A structure, called Object Level Accounting Methods, are used to allocate physical gains and/or losses to the accounting system. They can also be used to reconcile the physical and accounting systems. They are called "object level" because they allocate water from a given simulation object to one or more accounts on that object.

OBJECT LEVEL ACCOUNTING METHODS

Object Level Accounting methods are used to allocate physical gains, losses, and local inflows on an object to accounting slots on that object. In addition, they can be used to reconcile the physical and accounting system.

There are two types of Object Level Accounting Methods, *Compiled* and *User Defined*.

4.1 Compiled Accounting Methods

COMPILED ACCOUNTING METHODS

Compiled accounting methods are hard-coded Object Level Accounting Methods. These methods were created by CADSWES and are available in a library. They represent either commonly used functionality or methods that are too complex to represent as user defined methods

Examples of compiled accounting methods include commonly used methods like the Zero Slot Inflow method in the Reservoir Account Slot Inflow category which assigns zero to each account's Slot Inflow. In addition, there are complex methods that were too difficult to implement in RPL. For example, there is the Heron Inflow Calculation; a method specific to basins that have Rio Grande water types. The

following table briefly describes each of the compiled methods (the default method for each category is shown in **orange**; the general methods are also described in more detail after the table):

(Obj)Category	Method	Description
Agg Diversion Account Reconciliation	**No Method**	No action - never executed
Bifurcation Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Confluence Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
	Rio Grande Inflow 2[a]	Confluence^RioGrande.Slot Inflow is set equal to the Inflow 2. All other accounts' Slot Inflow set to 0.0
	Sidewater Inflow 2 [a]	Distribute Confluence.Inflow 2 to the Floriston Rate and Undes account Slot Inflow.
Control Point Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
	Copy Slot to Slot Inflows	On the specified account, set the Slot Inflow equal to the Local Inflow. Set all other Slot Inflows to 0.0. Click **HERE (Section 4.1.3)**
Distribution Canal Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Diversion Object Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Inline Pump Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Pipe Junction Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Pipeline Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
Reach Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
	Reconcile Rio Grande Outflow [a]	Reach^RioGrande.Slot Inflow is equal to the Reach.Outflow minus sum of WT(SanJuan).Outflow minus RioGrande.Inflow. Reach^WT(SanJuan).Slot Inflow = 0[b]
	Rio Grande Local Inflow [a]	Reach^RioGrande.Slot Inflow is set equal to the Reach.Local Inflow. All other accounts' Slot Inflows are set to 0.0
	Provo River Local Inflow [a]	Reach^ProvoRiver.Slot Inflow is set equal to the Local Inflow.
	NIC Local Inflow [a]	Reach^NIC.Slot Inflow is set equal to the Local Inflow.
	Copy Slot to Slot Inflows	On the specified account, set the Slot Inflow equal to the Local Inflow. Set all other Slot Inflows to 0.0. Click **HERE (Section 4.1.3)**

(Obj)Category	Method	Description
Reach Account Gain Loss	**No Method**	No action - never executed
	San Juan Gain Loss [a]	Sets Gain Loss on accounts with water types Rio Grande and San Juan. Many accounting slots are registered as dependencies.
Reservoir Account Slot Inflow Reservoirs: • Storage • Level Power • Sloped Power • Pumped Storage	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
	Heron Inflow Calculation	Sets Slot Inflow on accounts with water types Rio Grande and San Juan. Many accounting slots are registered as dependencies.
	Pooled Account Slot Inflow [a]	Pooled.SlotInflow is set equal to the object's Hydrologic Inflow Net
	Donner Inflow [a]	Basin specific
	Prosser Uncomm [a]	Basin specific
	Copy Slot to Slot Inflows	On the specified account, set the Slot Inflow equal to the Hydrologic Inflow. Set all other Slot Inflows to 0.0. Click **HERE (Section 4.1.3)**
Reservoir Account Gain Loss Reservoirs: • Storage • Level Power • Sloped Power • Pumped Storage	**No Method**	No action - never executed
	Heron Gain Loss Calculation [a]	Sets Gain Loss on accounts with water types Rio Grande and San Juan. Many accounting slots are registered as dependencies.
	El Vado Loss Calculation [a]	
	Nambe Falls Loss Calculation [a]	
	Elephant Butte Loss Calculation [a]	
	Elephant Butte Loss with RG Compact [a]	Sets Gain Loss on accounts with water types Rio Grande and San Juan Includes logic for the RG Compact. Many accounting slots are registered as dependencies.
	Abiquiu Loss Calculation [a]	Sets Gain Loss on accounts with water types Rio Grande and San Juan. Many accounting slots are registered as dependencies.
	Jemez Loss Calculation [a]	
	Cochiti Loss Calculation [a]	
Reservoir Account Reconciliation	**No Method**	No action - never executed
Water User Account Reconciliation	**No Method**	No action
Stream Gage Account Slot Inflow	**No Method**	No action - never executed
	Zero Slot Inflows	Sets all Slot Inflows to 0.0
	Reconcile Rio Grande Outflow [a]	StreamGage^RioGrande.Slot Inflow is equal to the StreamGage.Outflow minus Sum of WT(SanJuan).Outflow minus RioGrande.Inflow. StreamGage^WT(SanJuan).Slot Inflow = 0 [b] Many accounting slots are registered as dependencies.
	All Rio Grande	Sets Slot Inflow on accounts with water types Rio Grande
	All San Juan Chama [a]	Sets Slot Inflow on accounts with water types San Juan

a.Contact riverware-support@colorado.edu for more information on these basin specific methods.
b.WT(<name>) indicates all accounts with Water Type <name>)

Following is a description of the generally and commonly used compiled methods. Each method is executed according to its selected execution time as described **HERE (Section 4.5)**. Listed are the default execution times for each method.

4.1.1 No Method

The default No Method is a no-action method and does nothing. In fact, the method has a static execution time of "Never" meaning it will never execute.

4.1.2 Zero Slot Inflows

The Zero Slot Inflows method on many of the object's account slot inflow category sets the Slot Inflow to 0.0 for each account. By default, it is executed once at the beginning of the run, but the user can change this if desired.

4.1.3 Copy Slot to Slot Inflows

The **Copy Slot to Slot Inflows** method copies the object's local inflow to the target account's Slot Inflow and sets the other accounts' Slot Inflow to zero.

Selecting the **Copy Slot to Slot Inflows** method instantiates the following slot:

☞ **TARGET ACCOUNT**

Type: List Slot
Units: NO UNITS
Description: This slot contains one account on the object. This account's Slot Inflow will be set equal to the local inflow.
Information: Only one account is allowed in this list and the account must be on the same object as this slot. Also, the account must be a storage or passthrough account. This slot can be set (by account name, water type, or water owner) for many objects at once using the Multiple Object Method Selector **HERE (ObjectUserInterface.pdf, Section 2.11.2)**.
I/O: Required Input
Supply Links: No

The method is only available in the <Object> Account Slot Inflow category on the following objects with one of the specified local inflow methods selected:

Object	Local Inflow Category	Possible methods	local inflow slot
Control Point	Local Inflow Calculation	Input Local Inflow	Local Inflow
Reach	Local Inflow Solution Direction	Input Local Inflow Calc Local Inflow Contingent Local Inflow Local Inflow Downstream Only	Local Inflow

Object	Local Inflow Category	Possible methods	local inflow slot
Reservoirs: Storage, Level Power	hydrologicInflow CalculationCategory	solveHydrologicInflow inputHydrologicInflow Hydrologic Inflow and Loss	Hydrologic Inflow
Reservoirs: Sloped Power, Pumped Storage	hydrologicInflow CalculationCategory	inputHydrologicInflow	Hydrologic Inflow

The **Copy Slot to Slot Inflows** method will first assign zero to the current timestep value of the Slot Inflow slot of ALL accounts on the object. It will then copy the current timestep value in the object's "local inflow slot" (as shown in the above table) to the specified Target Account's Slot Inflow slot. By default this method will be given an execution time of **Beg of Timestep Once** meaning it will be executed once at each timestep before simulation starts. This execution time can be changed by the user.

4.2 User Defined Accounting Methods

Alternatively, User Defined Accounting Methods are created by the user in the RiverWare Policy Language (RPL).

USER-DEFINED ACCOUNTING METHODS

User-Defined Accounting Methods are Object Level Accounting Methods created by the user in the RiverWare Policy Language,.

For example, User Defined Accounting Methods can be used to specify how Hydrologic Inflow on a reservoir is allocated to the accounts on that reservoir or how the seepage in a reach is charged to the passthrough accounts on the reach. Often the methods depend on policy decision and are not physically based. For example, seepage in a reach may be charged only to the allocatable flow for that reach, not to any of the other water released from upstream reservoirs for downstream diversions. As a result, the methods are basin specific and must be created for each basin.

Because the methods are basin specific, the user must define and configure these methods for each basin. On storage and passthrough accounts, there are slots called Slot Inflow and Gain Loss. These are the slots in the accounting system that are set by the User Defined Accounting Methods to allocate local inflows and apportion gains and losses, respectively. This is done in the RiverWare Policy Language (RPL) using the Accounting Method Set Editor. In this editor, there are pre-configured policy groups for the allowed Objects/Accounts and the intended actions. For example, there is a policy group for the Reservoir Account Slot Inflow and Reservoir Account Gain Loss. These policy groups correspond to categories on the appropriate object.

Example

In DeepLake, the Hydrologic Inflow is shared equally amongst three accounts in a Deep Lake: A, B, and C. The Storage Account Slot Inflow method would be the following:

DeepLake ^ "A.Slot Inflow"[] = DeepLake."Hydrologic Inflow"[] / 3

DeepLake ^ “B.Slot Inflow”[] = DeepLake.“Hydrologic Inflow”[] / 3

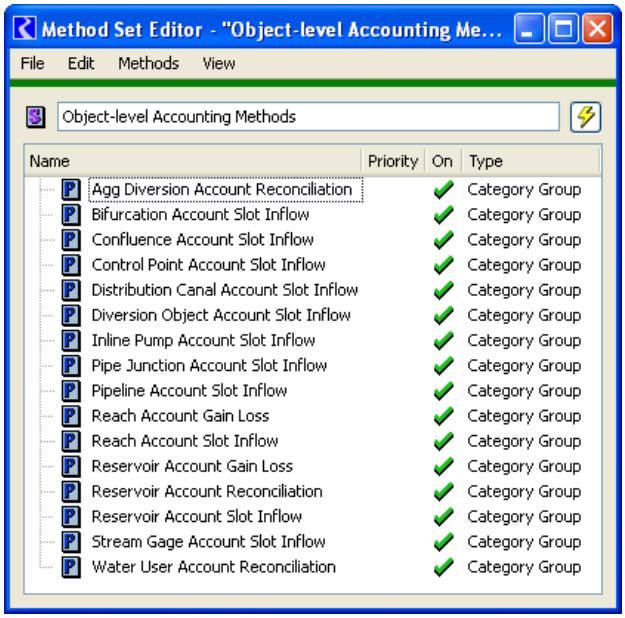
DeepLake ^ “C.Slot Inflow”[] = DeepLake.“Hydrologic Inflow”[] / 3

When the user creates a new Method in one of these policy groups, the new method appears as a method in the appropriate category on the object’s Accounting Methods tab and can be selected by the user. By selecting this method on the object, the user is telling the object that the selected User Defined Accounting Method should apply to the accounts on that object. As a result, the method created for each object applies to all of the accounts on the object. Multiple assignment statements may be necessary to set all necessary slots.

Following is a list of the policy groups / categories in the Accounting Method Set Editor and the associated objects and the accounts to which they apply:

Policy Group / Category	Object	Account(s)
Agg Diversion Account Reconciliation	Aggregate Diversion	Diversion
Bifurcation Account Slot Inflow	Bifurcation	Passthrough
Confluence Account Slot Inflow	Confluence	Passthrough
Control Point Account Slot Inflow	Control Point	Passthrough
Distribution Canal Account Slot Inflow	Distribution Canal	Passthrough
Diversion Object Account Slot Inflow	Diversion Object	Passthrough
Inline Pump Account Slot Inflow	Inline Pump	Passthrough
Pipe Junction Account Slot Inflow	Pipe Junction	Passthrough
Pipeline Account Slot Inflow	Pipeline	Passthrough
Reach Account Gain Loss	Reach	Passthrough
Reach Account Slot Inflow	Reach	Passthrough
Reservoir Account Gain Loss	Storage Reservoir Level Power Reservoir Pumped Storage Reservoir Sloped Power Reservoir	Storage and/or Passthrough
Reservoir Account Reconciliation	Storage Reservoir Level Power Reservoir Pumped Storage Reservoir Sloped Power Reservoir	Storage and/or Passthrough
Reservoir Account Slot Inflow	Storage Reservoir Level Power Reservoir Pumped Storage Reservoir Sloped Power Reservoir	Storage and/or Passthrough
Stream Gage Account Slot Inflow	Gage	Passthrough
Water User Account Reconciliation	Water User	Diversion

User Defined Accounting Methods, although written in RPL, do not behave the same as rules. Although the methods are prioritized in the method set editor, the priority is not used. Instead, there can only be one method selected for each object. The methods execute according to their specified execution time **HERE (Section 4.5)**.



Although the methods are specific to a basin, there are a few features that can be used to generalize the methods. The keyword *ThisObject* (note, there are no space) can be used in place of a specific object name. When called, it will replace this with the name of the object from which the method is called.

Example:

Following is a reach Pass Through Gain Loss method that makes use of the "ThisObject" syntax. It would be useful on this basin because each reach has two accounts, Fish and Farmers. The Fish account gets charged with all loss, Farmers get none. Each reach in the model could then use this method.

ThisObject ^ "Fish.Gain Loss"[] = ThisObject. "Total GainLoss"[]

ThisObject ^ "Farmers . Gain Loss"[] = 0 ["acre-feet"]

Note: Gain Loss in the accounting system is a volume.

Example:

Following is an example setting all of the account's Slot Inflow to zero using a ForEach loop (note, this is just a sample method, this functionality can be accomplished using the compiled **Zero Slot Inflows** method):

```
FOREACH (STRING account IN AccountNamesByAccountType(ThisObject, "ALL"))
      ThisObject^(account CONCAT ".Slot Inflow")[] = 0 ["cfs"]
ENDFOREACH
```

More information on RPL and its use can be found **HERE (RPLUserInterface.pdf, Section 1)**.

4.3 Execution Time

You are able to control when each Object Level Accounting Method is executed. Because Object Level Accounting Methods set the Slot Inflow and Gain Loss values on the accounts and these are required knowns for the account solution, this execution time allows you to control when the accounts solve. Depending on the application, you should execute the method as soon as possible once the information

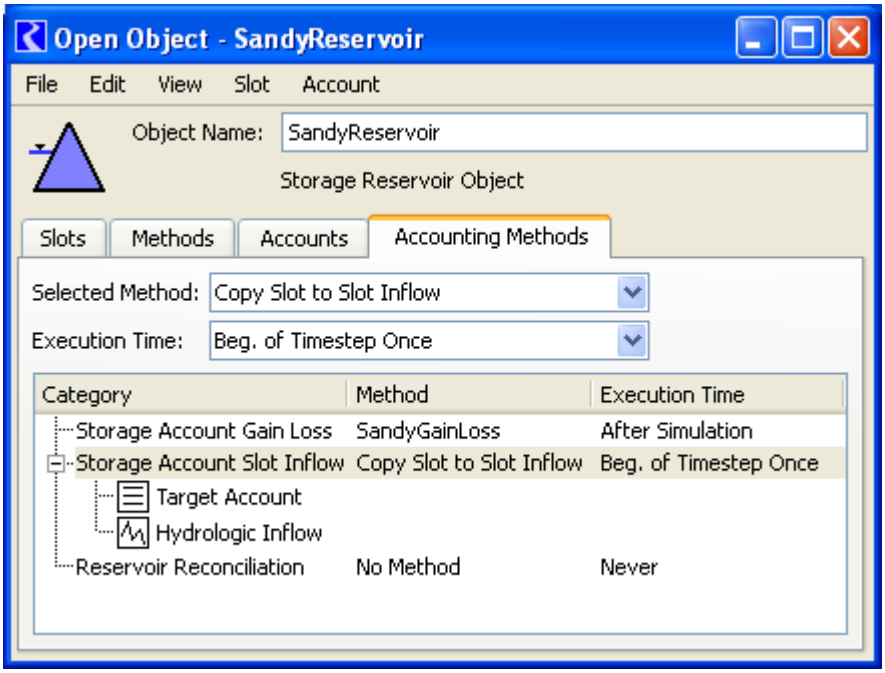
is known. For example, if you input Local Inflow, then you can configure your Slot Inflow method to execute at the Beginning of the run. Following are the possible execution times:

Execution Time	Description	Dependencies
Never	The method is never executed. This is only available for the default no-action method for each category.	None
Beg. of Run	At the beginning of the run, the method is executed once per timestep.	None
Beg. of Timestep Once	The method is executed once before each timestep's simulation.	None
Beg. of Timestep	The method is executed before each timestep's simulation and as dependencies change. For this execution time, the method registers accounting slot dependencies. Therefore, if a value in a dependent accounting slot changes, then the method re-executes. Note, this could be have performance implications if slot dependencies cause methods to re-fire.	Accounting Slots
After Simulation	Execute the method after each timestep's simulation is complete and as dependencies change. Note this is the default for compiled methods and the only execution time in releases prior to RiverWare 5.1. Technically, the method is executed as part of the “accounting beginning of timestep” which occurs after all rules have executed and all dispatching is complete. Each method registers accounting slot dependencies when executed. Therefore, if a value in a dependent accounting slot changes, then the method will re-execute.	Accounting Slots

The Object Level Accounting Methods can only access information that exists at the time they are called or the method will terminate early. For example, the Object Level Accounting Methods set to execute “After Simulation” can only access accounting values at the previous timestep as accounts may not have solved yet. Usually, these methods can access physical values at the current timestep as the simulation has likely solved when “After Simulation” Object Level Accounting Methods are called. Input values can be accessed by any OLAM.

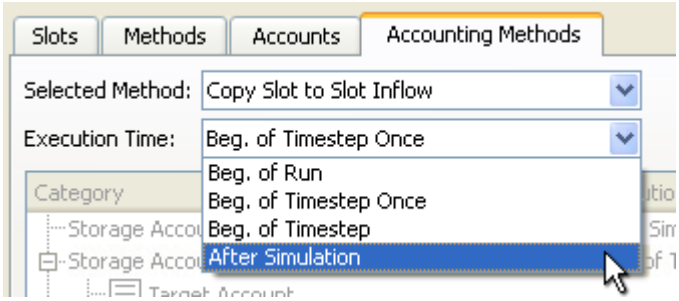
4.4 Selecting OLAMs and execution time

OLAMs and their Execution Time are selected from the Open Object **Accounting Methods** tab:



The **Accounting Methods** tab works similar to the Methods tab. You select a category, then choose a method from the **Selected Method:** box at the top of the tab. An adjacent combo box labeled **Execution Time:** allows you to choose the Execution Time. The category / method list on this tab also contains the "Execution Time" column showing the currently chosen execution time for each selected method.

Additionally, you can select methods on multiple objects using the Multiple Object Method Selector described generally **HERE (ObjectUserInterface.pdf, Section 2)** and in terms of OLAMs **HERE (ObjectUserInterface.pdf, Section 2.11)**.



4.5 Reconciliation

User defined accounting methods (and rules discussed later) can be used to ensure that the accounting system is reconciled with the physical system, i.e. the sum of the accounting releases is the same as the physical release and the sum of account storages is the same as the total object's storage. RiverWare has no automatic checks to ensure that physical and paper water are reconciled. Each basin is unique and operates differently. As a result, it is the responsibility of the modeler to reconcile the two systems according to the legislation and operations of the basin. There are river basin that have legally decided that the accounting system can vary from the physical system on a daily basis and the total reconciliation happens on a monthly (or longer) timescale.

There are reconciliation categories on the aggregate diversion, reservoir, and water user objects. These categories can hold methods that set the Gain Loss slot, Slot Inflow slot, and/or accounting supplies. Reconciliation can also be accomplished using rules in a rulebased model. Further information on rules and accounting is given in **HERE (Section 6)**.

5. How the Accounting System Solves

The accounting system does not dispatch in the same way that the physical water network simulation is dispatched, but there are some similarities. Account solution is triggered by the assignment of a value to a slot on the account. Assignment to account slots occurs in one of six ways: by user input, by a rule, by propagation through a supply, account solution, account level method, and by an object-level accounting method on the object that references physical water quantities to introduce water into the accounting system (paper water). More information on each type of assignment is presented in the table **HERE (pg. 5)**.

Accounts solve whenever sufficient known slot values are present to run a solution method. This is analogous to dispatching in the simulation network; however, in the case of the accounting system, this means that accounts may solve when a user edits the account values through the user interface, a rule sets a value, or a method sets a value. Therefore, a run is not required to make a single account solve.

Each time an account slot is given a value (during resetting of user input, execution of object level methods, rule execution, or during account solution), the account holding the slot is notified, and performs its own checks for over- and under-determination, and possibly solves at one or more timesteps. Some accounts solve "into the future" under some circumstances, even though the controller might be at a different timestep.

The accounts solve for as many timesteps as there is the required known data. This means that the account solution behaves similarly to a spreadsheet solution. If the user inputs an Outflow for each timestep in the accounting period for a storage object and all of the inflows, slot inflow and Gain Loss are known, then the account will solve for Storage at each timestep. This action is performed when the user hits the enter button. If the user then changes an Outflow value at the start of the accounting period, then the storage for the entire period will change because the solution equation for the storage account depends on the previous timestep's Storage. By changing one Outflow, the Storage for the whole period is affected. Editing values in the accounting system causes the accounting system to solve in the same manner described above.

There are three controllers that allow accounting: Post-Simulation Accounting, Inline Simulation and Accounting, and Inline Rulebased Simulation and Accounting. These three controllers and their solution steps during a run are described in the following subsections.

5.1 Post-Simulation Accounting

The simulation is run, then the post-simulation accounting process is run. This process clears output values on accounting slots and invokes the object-level accounting methods (in that order). Accounts solve when they have the required knowns and resolve when a value in the solution equation changes.

Simulation Steps: The Post-simulation controller follows the procedure outlined below in the process of a run:

- Initialization and Beginning of Run for simulation
  - Clear all output and values from previous runs for all timesteps in all series slots.
  - Set user inputs and propagate user inputs across links for all timesteps.
- Simulation Beginning of Run
  - Evaluate Beginning of run expression slots for all timesteps.
  - Execute Beginning of Run methods for all objects.
- For each timestep:
  - Set the controller clock to the timestep time.
  - Execute Beginning of Timestep methods for all objects.
  - Evaluate Beginning of timestep, current timestep only expression slots
  - Dispatch objects until the queue is empty, simulating the effects of the user inputs and default values.
  - Execute End of Timestep methods on all objects.
  - Evaluate end of timestep, current timestep only Expression slots
- Execute End of Run Simulation methods on all objects.
- Evaluate End of Run expression slots.

- Initialize Accounting Run
  - Clear Iteration count
  - Clear Account States, i.e. Output values and Output flags
  - Set Accounting User Inputs
- Accounting Beginning of Run
  - Execute Beginning of Run accounting methods for all objects.
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Run**
- For each timestep,
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Timestep** and **Beg. of Timestep Once.**
  - Execute the Object Level Accounting Methods that are set to execute **After Simulation.**
  - If dependent accounting slot have changed, re-execute as necessary Object Level Accounting Methods that are set to execute at **Beg. of Timestep** or **After Simulation**.

5.2 Inline Simulation and Accounting

For each timestep, the physical simulation and accounting method execution are intermixed. Accounts solve when they have the required knowns and resolve when a value in the solution equation changes.

Simulation Steps: The simulation controller follows the procedure outlined below in the course of a run:

- Initialization Simulation Run

  - Clear all output and values from previous runs for all timesteps in all series slots.
  - Set user inputs and propagate user inputs across links for all timesteps.
- Initialize Accounting Run
  - Clear Iteration count
  - Clear Account States, i.e. Outputs and Output flags
  - Set Accounting User Inputs
- Simulation Beginning of Run
  - Evaluate Beginning of run expression slots for all timesteps.
  - Execute Beginning of Run methods for all objects.
- Accounting Beginning of Run
  - Execute Beginning of Run accounting method for all objects
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Run**
- For each timestep:
  - Set the controller clock to the timestep time.
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Timestep** and **Beg. of Timestep Once.**
  - Execute Beginning of Timestep methods for all objects.
  - Evaluate Beginning of timestep, current timestep only expression slots
  - Dispatch objects until the queue is empty, simulating the effects of the user inputs and default values.
  - Execute End of Timestep methods on all objects.
  - Execute the Object Level Accounting Methods that are set to execute **After Simulation.**
  - If dependent accounting slot have changed, re-execute as necessary Object Level Accounting Methods that are set to execute at **Beg. of Timestep** or **After Simulation**.
  - Evaluate end of timestep, current timestep only Expression slots
- Execute End of Run Simulation methods on all objects.
- Evaluate End of Run expression slots.

5.3 Inline Rulebased Simulation and Accounting

For each timestep, the rulebased simulation is run where slots are set by rules, the simulation objects dispatch, and user defined accounting methods execute. Remember that accounts solve when they have the required knowns and re-solve when a value in the solution equation changes. This can happen at any time throughout the course of the run or even outside of the run. This section describes the inline rulebased simulation and accounting steps in more detail. First, the run steps are presented in paragraph format, then they are presented in bullet format.

Rulebased simulation is described **HERE (RulebasedSimulation.pdf, Section 1)**. As a reminder, execution of rules and dispatching (simulation) of objects are inter-mixed and may be mutually-dependent. At each timestep, the controller alternates between two steps:

1. Processing the object dispatch queue. Objects check their dispatch conditions and may try to dispatch (or re-dispatch) after a "dispatch slot" on the object changes value. An object maintains a list of its dispatch slots and dispatching only occurs if it has sufficient known values.

2. Executing rules from the agenda one at a time. When a rule is successful, all its slot assignments are made. If it is not successful, none of its assignments are made, i.e. never are some, but not all of the assignments made. When the rule succeeds, the slot cell whose value was changed is given the priority of the rule and the "R" flag. These values show up in the rules analysis dialogues. Each rule maintains a list of the slots on which the outcome of the rule is "dependent", namely those slots that were read during the rule execution. A rule will re-execute after one or more of its dependency slots changes value. Thus, objects that are dispatching can cause rules to re-execute by changing the values of slots upon which the rules depend, while rules that change dispatch slot values can cause simulation objects to dispatch. In the accounting system, a rule can change an account slot's value, which may cause the account to solve and may propagate changes throughout the accounting system. This can happen during a rule execution, but not during an object dispatch.

At the start of the timestep (after beginning of timestep behavior has occurred where object level accounting methods may be executed), the dispatch queue (step 1) is fully processed until empty. This simulates the effects of user inputs. In this processing of the dispatch queue, objects may dispatch one or more times or may not dispatch at all depending on the dispatch slots that are set or changed throughout the course of the dispatching.

Once the queue is empty, the rules controller moves to the second step and starts with the full set of enabled rules on its "agenda" in priority order. This ensures that each enabled rule gets at least one chance to execute. The controller tries to execute the first rule on the agenda, in user specified order, at which time the controller removes the rule from its agenda. A successful rule may put objects on the simulation dispatch queue. After a rule succeeds and sets a value, the controller returns to step 1 and dispatches all objects on the queue.

When the dispatch queue is again empty, the controller returns to step 2 and executes the next rule on its agenda. Once a rule succeeds, it returns to step one and processes the dispatch queue. It continues alternating until both the dispatch queue and the rules agenda are empty.

When all simulation dispatching is complete and the rules agenda is empty, Object Level Accounting Methods may execute and the simulation moves to the next timestep.

Rulebased Simulation Steps: The rulebased simulation controller follows the procedure outlined below:

- Initialization Rulebased Simulation Beginning of Run
  - Clear all output and values set by rules from previous runs for all timesteps in all series slots.
  - Set the controller priority to 0.
  - Set user inputs and propagate user inputs across links for all timesteps.
- Initialize Accounting Run
  - Clear Iteration count
  - Clear Account States, i.e. Outputs and Output flags

  - Set Accounting User Inputs
- Rulebased Simulation Beginning of Run
  - Evaluate Beginning of run expression slots for all timesteps.
  - Execute Beginning of Run methods for all objects.
- Accounting Beginning of Run
  - Execute Beginning of Run accounting method for all objects
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Run**
- For each timestep:
  - Set the controller clock to the timestep time.
  - Execute the Object Level Accounting Methods that are set to execute at **Beg. of Timestep** and **Beg. of Timestep Once.**
  - Set controller priority to 0.
  - Execute Beginning of Timestep methods for all objects.
  - Evaluate expression slots that have evaluate-at-beginning-of-timestep selected
  - Put all rules on the agenda (in priority order - whether 3,2,1 or 1,2,3 is user-selectable)
  - Do the following two processes until both the dispatch queue and the rules agenda are empty
    - Process 1 - Process the Dispatch Queue: Dispatch objects (as in basic Simulation) until the queue is empty, simulating the effects of any user inputs and default values and any recent changes to slots by rules. For each slot changed by this dispatch:
      - Put all rules that depend on the changed slot on the agenda, if it's not already there.
      - If the slot is a dispatch slot, check dispatch conditions and if necessary put the object containing the slot on the simulation dispatch queue
      - If the slot is an accounting slot, notify the account that it may need to solve.
    - Once the Queue is empty, move on to process 2
    - Process 2 - Execute a Rule on the agenda: Set the controller priority to the priority of the next rule on the agenda (either in order 3,2,1 or 1,2,3 based on user-selection), and fire this rule. If not successful, continue firing one rule at a time until a rule is successful and at least one slot is set in the model. Each rule is removed from the agenda after it fires. Once a rule is successful:
      - Apply the slot changes, giving the changed slot cell the priority of the controller (i.e. the last rule that fired)
      - Add to this rule's dependency's list each slot that was read by this rule.
      - For each slot set by this rule, put all rules that depend on the changed slot on the agenda if the rule is not already there.
      - For each slot changed by this rule, if the slot is a dispatch slot, check dispatch conditions and if necessary, place the object containing the slot on the simulation dispatch queue; and if the slot is an accounting slot, notify the account that it may need to solve.
    - Return to Process 1 and repeat until the Agenda and the Queue are empty.
  - Set controller priority to zero, and execute End of Timestep methods on all objects.
  - Execute the Object Level Accounting Methods that are set to execute **After Simulation.**
  - If dependent accounting slot have changed, re-execute as necessary Object Level Accounting

Methods that are set to execute at **Beg. of Timestep** or **After Simulation**.
  - Evaluate end of timestep, current timestep only Expression slots
- Execute End of Run Simulation methods on all objects.
- Evaluate End of run expression slots.

6. Accounting with Rules

In an Inline Rulebased Simulation and Accounting run, rules can be used for two purposes: setting slots on the physical system and setting supplies in the accounting system. In the accounting system, rules are used to move water between two accounts by setting Supplies. To clarify the two types of assignments, we will define simulation rules/assignments as follows:

SIMULATION OR PHYSICAL RULES/ASSIGNMENTS

Rules and/or assignments that set values in the physical simulation system. For example, these rules set reservoir Outflow, Storage, or Pool Elevation.

We will also define accounting rules/assignments as follows:

ACCOUNTING RULES/ASSIGNMENTS

Rules and/or assignments that set slots or supplies in the accounting system

Both simulation rules, i.e. rules that set outflows and storages, and accounting rules, i.e. rules that set supplies, can be part of the same ruleset and even assignments within the same rule. Accounting rules may be interspersed with physical rules that set simulation slots. Or, all of the Simulation rules can execute, then all of the accounting rules can execute. The structure of the ruleset depends on the application. In addition, accounting assignments and simulation assignments can be within the same rule. For example, when a physical release is made, the release can be allocated to the appropriate supplies in the same rule. Following is a description of the possible ways an accounting model can interact with the physical simulation:

After-the-Fact Accounting: The physical simulation is run completely without any rule effects, i.e. all objects dispatch because of input data. Rules are used to determine how the water already released in the physical simulation should be classified in the accounting system. This type of application has only Accounting Rules, all of the physical simulation is the result of input data. Typically this type of system is used in an operations mode to determine how yesterday's final releases should be classified. For