UPPER RIO GRANDE WATER OPERATIONS MODEL
RULES DOCUMENTATION
Summary

Table of Contents

DISCLAIMER 1
Introduction 1
Rulebased Simulation 2
  Functions – General Discussion 3
Reservoir Operations 4
  Heron Reservoir 4
  El Vado Reservoir 4
    Operation for Rio Grande Water 4
    Restrictions of the Rio Grande Compact 5
    Water Right Constraints on Operation of Rio Grande Water 5
    Operation for San Juan-Chama Water 5
  Abiquiu Reservoir 5
    Operation for Rio Grande Water 5
    Flood Carryover Storage 6
    Conservation Storage 6
  Cochiti Reservoir 6
    Operation for Rio Grande Water 6
    Operation for San Juan-Chama Project Water 6
  Jemez Canyon Reservoir 7
  Elephant Butte Reservoir 7
    Operation for Rio Grande Water: 7
    Operation for San Juan-Chama Project Water: 7
  Caballo Reservoir 7
General Rules Description 8
  General 8
  Exchanges, credits and debits 8
  San Juan Diversion Rules 8
  Heron Reservoir Rules 9
  El Vado Reservoir Rules 9
  Abiquiu Reservoir Rules 9
  Cochiti Lake Rules 10
  Jemez Canyon Reservoir Rules 10
  Elephant Butte RESERVOIR Rules 10
  Caballo Reservoir Rules 10
San Juan-Chama release priorities 10
  Priority Solution Example 11
    Assumptions: 11
    Solution 12
APPENDIX A – Simulation and Rulebased Simulation Overview 13
Introduction 13

Simulation 13
Example of Simulation..... 14
Rulebased Simulation 16
APPENDIX B – SPECIFIC RULES DESCRIPTION 21
FOREWORD 21
RULES CODE AND DESCRIPTIONS DOCUMENTATION (filename; "waterops.ruleset.10.16.02.mss"): 21

Lists of Tables & Figures

Table 1 - Release type assumed priorities for Abiquiu Reservoir 11
Table 2 - Account priorities for Abiquiu Reservoir 12
Table 3 - Steps Of Rulebased Simulation 19

Figure 1 - Example Model For Demonstrating The Simulation Controller In RiverWare 15
Figure 2 - Rulebased Simulation Example Model 18

UPPER RIO GRANDE WATER OPERATIONS MODEL
RULES DOCUMENTATION
Summary

DISCLAIMER

This (draft) document is a product of Upper Rio Grande Water Operations Model (URGWOM) and is intended to assist technical reviewers understand how the Water Operations Model logic works. It is primarily to be used in the technical review of the Water Operations Model at this unfinished stage of development, and eventually will evolve into the final Rules documentation for URGWOM. It reflects the URGWOM Technical Team's interpretation of operational policy, regulations, preferences, and other decision-making logic. URGWOM is not the designer of these policies and regulations, but simply an interpreter and user. For the actual policies, laws, regulations, and implementations, the user is referred to the responsible agencies. This document, which is substantially composed of code and descriptions taken verbatim from a recent Water Operations Model Ruleset (filename; "waterops.ruleset.10.16.02.mss"), may contain errors in interpretations, statements, and applications of such laws, regulations, and policies. URGWOM and the URGWOM Team members are not responsible for errors that might result from the use of the information provided herein.

INTRODUCTION

URGWOM has a number of components of substantial development. Two of them are: The physical structure, and the operational logic. This document is an evolving explanation of the operational logic ("Rules"), which are also evolving. The physical structure has undergone substantial development and documentation as presented in the (annual) Technical Reviews the past few years. This is the second official draft of the Rules document published (The first dealt only with the "Rio Chama Test-Case" and came out in September of 1998.) As a "draft document", this document is not complete, but begins to communicate understanding of the "Rules" in terms other than just the code itself. The approach taken is to provide an actual printout of a current Water Operations Model Ruleset, with descriptions of each Rule, and Function alongside each. (Ideally, another method of presenting logic such as flowcharting would be used, but the resources do not allow for that type of effort.) Note that the best way to track Rules is from within RiverWare (an option not available to most reviewers) because you can track through the Rule/Function paths by clicking the next Function name. (Of course, a comprehensive review would have the reviewer track through every possible logic path, which would be a tremendous task!) Since this is a living document describing an evolving Ruleset, it may not include descriptions of every item, and it may be outdated fairly quickly. It will be updated as resources allow.

In basins where multi-objective river and reservoir system modeling is needed, such as the Rio Grande Basin extending slightly beyond New Mexico's northern and southern borders, modeling uses a set of independent policy objectives to drive a solution of releases and/or other control actions that satisfy the objectives as well as possible. The main discretionary control executed by "Rules" on this river/reservoir system is in setting the outflow amounts from the reservoirs. This makes sense, because in the real world of operation, outflow is what we directly control, not reservoir elevation, inflow, evaporation, or precipitation! Typical objectives reflect such policies as meeting downstream demands, preventing flooding, providing flows for aquatic and riparian habitat, providing flows and lake levels for recreation, protecting senior water rights, and other regulation policies. Conflicting objectives compete by a system of prioritization. For example, flood control for public safety is always at the top of priorities, and recreational requirements are often very low. Generally in RiverWare, "Groups" of Rules, as the Rules themselves, are placed in order of priority, highest to least. This is not always true, because through the use of logical statements, you can circumvent the priority system. (We accept this inconsistency because in the real world, priorities like these are not linear or 2-dimensional, in other words, priorities vary based upon

conditions and so you cannot strictly adhere to one hard priority arrangement for all cases.) However, higher-to-lower is the general ordering of prioritized Groups and Rules. The reviewer should be made aware that some lower priority objectives are placed before higher priority objectives, but in the actual process, do not govern over them. This might be made to happen by making the lower priority objective (placed at a higher priority position) check to see if certain higher-objective values are set before it sets its own value.

In RiverWare, the logic of the operating policies is included in modeling by running what is referred to as "Rulebased Simulation." In the absence of control of intakes, penstocks, slide gates, and other such managed facilities, RiverWare's "Simulation" run-mode would sufficiently model how the natural river system transports water through a system of unmanned/uncontrolled reservoirs. Conversely, with the above facilities in operation, understanding of how decisions are made must be provided for, in the form of a prioritized set of Rules. These Rules are developed using RiverWare's Rule language, "RiverWare Policy Language" (RPL). The written Rules attempt to embody and implement all of the consistent, repeatable decision-making done by water operators in actual operations. Of course, special cases and deviations are a more difficult issue, possibly not suitable for coding as Rules. The Rules themselves provide for absolute consistency in operations (although actual operations are seldom absolutely consistent), and the documentation ideally would explain all of the logic.

Some basic definitions are needed to meaningfully describe Rules (the reviewer should refer to RiverWare training and manuals to fully understand concepts): A RiverWare "object" is a graphical representation of a physical thing like a reservoir, river reach, or streamgage, that contains data, customized "methods" for doing things, and contractor "accounts". An example of an object would be El Vado Reservoir, which includes information on its elevation/storage relationship, outflow features, etc. A "slot" is the placeholder for a value, a table of values, or a set of timeseries values residing within an object, that can be filled ("set") by the user directly, by importing data, or by Rules. An example of a slot would be "Pool Elevation", which is a timeseries slot including values for the entire run period, and is generally solved for internally based upon total reservoir mass balance. User input has the highest priority in setting slots – Rules cannot overwrite what has been input by a user. In the absence of data, a slot contains a "NaN" or a series of them. This stands for "Not-a-Number", and is essentially an identifiable no-value placeholder for an empty space in a slot. Rules can recognize and overwrite NaN's. Each Rule can set one or more slots.

The need for Rules results from too many unknowns to solve the physical system. In other words, we don't know for example, the outflows from reservoirs because they are controlled by operators, and are not just dependent upon inflows and reservoir elevations. Rules step in to make decisions that physical system cannot solve.

Note that the Rules are developed as much as possible to avoid "hard-coding" numbers into them. That is, wherever a number like channel capacity flow is needed, a data object/table slot is created to hold that number, instead of typing the number into the Rule or Function code. Therefore, these numbers can be changed as studies determine necessary, to study "what-if" scenarios, and to primarily give the normal user the ability to make the change, not a Rules-Coding person. This practice is a standard good programming practice we attempt to employ, to the best usability and proper development of the Water Operations Model.

RULEBASED SIMULATION

"Rulebased Simulation" is a RiverWare term for computerized reproduction in which values undeterminable by pure "Simulation", are set by prioritized Rules. Rules typically have an IF-THEN-ELSE structure that examines the state of the system in the antecedent, and then sets values in slots accordingly. The "state of the system" is usually the result of input from forecasting. Higher priority Rules can overwrite values set by lower priority Rules, but not vice-versa. This summary provides just a general overview on how the Rulebased simulation controller works. (A more detailed description is provided in Appendix A.)

As discussed above, the Rules are listed in order of priority. Each Rule is assigned a numerical integer value, with the lowest (1) being the highest priority Rule. The Rules are subdivided into "policy Groups," typically by reservoir and by special functional needs. The model and Rules execute upstream to downstream, therefore, the upstream reservoir Rules have higher priority than downstream reservoir Rules. The implications of this are for example: Abiquiu's top priority Rule is actually a lower priority than El Vado's lowest priority Rule. This is not a problem though, because reservoirs operate somewhat independently and do not set each other's slots. Also, another convention is used to negate priority problems: Assign responsibility to the upstream reservoir to check the downstream reservoirs' conditions prior to doing something detrimental to downstream operations.

Rules "fire", or in other words "execute" in an iterative fashion. The priority system gives higher Rules the ability to overwrite slots set by lower Rules, or conversely, to have the slots remain as they set them if and when lower priority Rules attempt changes. But normally, higher Rules have insufficient information to solve on the first iteration, and must re-fire after lower Rules have set some slot values. However, if a higher Rule can solve on the first pass, the lower Rules must recognize this prior to attempting to set the same slots, if they wish to set other slots the higher Rule did not set. For any Rule to be successful setting any of the slots, it must not attempt to set any slots that have been set by higher Rules. If it does, it will fail not only on that slot, but on all slots it addresses due to its junior priority. The way around this is; for each slot setting assignment in the Rule, an "IsNaN" ("Is it a NaN") check is used to see if a number has been assigned to that slot. If it "IsNaN", then an assignment will be attempted, but if it is a number, no attempt will be made and the Rule will move on to the remaining slots (instead of completely failing and kicking completely out of the Rule). For both the checked former slot and the subsequent slots it will be fully successful. Set slots remain set until the next timestep, unless overwritten by higher priority Rules, and assume the priority of the Rules that set them. Rules are best named descriptively; for readability of the code, for example; "AbiquiuFloodControl".

FUNCTIONS – GENERAL DISCUSSION

Functions are named logical and mathematical expressions of code called by Rules or other Functions, which return one value each time they are called. The values they return reside only temporarily in memory, and disappear as soon as they provide their values to the calling Functions or Rules. Functions cannot set slots, but are used by Rules to do so. For example, if the Function "ElVadoRGRelease ()" is called, its logic goes to work using current values in slots it refers to and using the results of other Functions it calls to return for instance; "500 CFS". This number is invisible to the user, unlike if written to a slot, and is just provided to the caller, and purged. A Function is also named (by the user) descriptively, so the user can understand what the number returned is, (in the example case; the amount of Rio Grande water to be released from El Vado).

There are two types of "Internal Functions" used by RiverWare: "Pre-Defined Functions" provided by CADSWES for general RiverWare use, such as; "FlowToVolume ()", which converts a flow in CFS to a volume per timestep in AF/timestep; and "User-Defined Functions" such as "HeronOutflow ()". The Functions we are documenting here are User-Defined Functions. User-Defined Functions were written by the URGWOM development team (including consultants). (For documentation on RiverWare Pre-Defined Functions, refer to RiverWare help, which is launched from RiverWare.)

Using Functions is often optional, as the Rules can usually include all of the lines of code. (In fact, the code syntax in Functions is almost identical to that used in Rules.) However, this would result in long Rules that are difficult to read and decipher. Functions allow the subdivision of Rules into shorter, more readable groupings of code. More importantly, for code that is used repeatedly, having one Function called by name numerous times is much more efficient than rewriting all its code in every spot it is needed. Use of Functions also makes debugging of Rulesets easier, as you can trace the calling of Functions easier than through lengthy Rule statements. Functions and Rules are the fundamental building blocks of a Ruleset.

RESERVOIR OPERATIONS

This section is dedicated to explaining the workings of the system from the operators' perspective. Following are the current policies and typical operations for each reservoir project that are implemented in the model.

HERON RESERVOIR

Heron Reservoir is operated in accordance with P.L. 87-483 and in compliance with the Rio Grande Compact. Two basic principles control the water release schedule from Heron Reservoir. The first is the authorized development of San Juan-Chama Project supplemental irrigation and municipal and industrial water demands that result in increased depletion of the Rio Grande. These depletions are offset by releases of San Juan-Chama water from Heron Reservoir sufficient to assure that no residual effect occurs to natural waters of the Rio Grande due to project operations. In addition, downstream contractors such as the City of Albuquerque and the Middle Rio Grande Conservancy District (MRGCD) convey other project waters past Otowi for use. No Rio Grande water can be stored in Heron, therefore it is released as soon as it is known of per the accounting procedures, and it is feasible to release it. (This is usually after the end of the month, because the accounting-determined amount is not known until the month ends.)

Secondly, carry-over storage is not permitted in Heron Reservoir (unless by "Waivers", as discussed later) from one year to the next. Contracted water not called to be released by December 31, would remain in Heron Reservoir as part of the project supply and would no longer belong to the individual contractor. In the past, Reclamation negotiated temporary waivers with contractors that allowed carry-over until April 30, in order to provide release rates on the Rio Chama that would enhance the fishery between El Vado and Abiquiu Reservoirs during the winter and provide flexibility in managing river flows. The no carry-over stipulation results in the various contractors seeking storage in reservoirs downstream of Heron for their unused water. El Vado, Abiquiu, Jemez Canyon, and Elephant Butte reservoirs have been used for storage of San Juan-Chama waters. Another factor that influences Heron releases is ice cover on the reservoir and the resulting safety issues. If Heron is drawn down too quickly when iced over or nearly completely iced over, hazardous conditions develop. Releases would be terminated until conditions are safe. During late March or April, any San Juan-Chama Project water not released because of unsafe winter operation conditions, will be released at a time when it is assured to meet the same purposes as if it had been released during the winter months, provided the necessary waivers have been granted.

The Natural Resources Conservation Service/National Weather Service (NRCS/NWS) coordinated runoff forecast is used to estimate the period of time during the spring runoff that the flow of the Rio Chama is expected to exceed channel capacity below Abiquiu Reservoir, thus preventing a release of San Juan-Chama Project water from Abiquiu Dam. Between the times of ice melt on the reservoir and the times when spring runoff reaches full channel capacity, San Juan-Chama replacement water requirements downstream are estimated and are released from Heron Reservoir.

EL VADO RESERVOIR

El Vado Dam was originally constructed to provide conservation storage for a supplemental irrigation supply for the MRGCD lands along the Rio Grande from Cochiti Dam to below Socorro, New Mexico. Because El Vado Dam was constructed after 1929 (completed in 1935), the operation of the reservoir for storage and release of Rio Grande water is subject to Prior and Paramount (Indian Water Rights) storage and releases, and the Rio Grande Compact. Water imported into the Rio Grande basin via the San Juan-Chama Project and stored in El Vado Reservoir is not subject to storage and release restrictions of the Rio Grande Compact.

Operation for Rio Grande Water

The basic concept in operating El Vado Reservoir involves the storage of natural inflow that is in excess of current MRGCD and other needs below El Vado Dam. The major storage season is during the spring runoff, and then storage can be released during the irrigation season to users in the Middle Rio Grande

Valley as needed. In accordance with Prior And Paramount requirements, a monthly-varying minimum pool of Rio Grande water is held in storage from March through October, with higher priority than any other requirements. This amount is adjusted monthly based upon the NRCS/NWS March, April, and May Otowi forecasts, amounts of Rio Grande captured in El Vado, and calls for the water by the Pueblos.

Restrictions of the Rio Grande Compact

Article VII of the Rio Grande Compact provides that no storage of Rio Grande water in El Vado Reservoir can take place when usable water in Project Storage (non-credit-water and non-SJ-C Contractor-water storage in Elephant Butte and Caballo Reservoirs) is less than 400,000 acre-feet (except for Prior and Paramount waters). Article VI provides that any Rio Grande water stored in El Vado Reservoir must be held in storage to the extent of New Mexico's accrued debit under the Compact. Nearly all of the post-compact storage capacity in New Mexico is located in El Vado Reservoir.

Water Right Constraints on Operation of Rio Grande Water

El Vado is operated to store native water (Prior and Paramount) for the Six Middle Rio Grande Pueblos of; Cochiti, Santo Domingo, San Felipe, Santa Ana, Sandia and Isleta. The Bureau of Indian Affairs and the Bureau of Reclamation compute the amount of storage required and release of Indian storage is made only when the natural flow of the Rio Grande is insufficient to adequately supply irrigation to 8,847 acres of Indian lands.

Additionally, no storage of native water except for Prior and Paramount can be made at El Vado Reservoir when to do so would deprive acequias along the Rio Chama downstream of El Vado of water to which they are entitled. In 1971, the State Engineer required that El Vado Reservoir be operated during the irrigation season to pass all the natural flow of the Rio Chama up to 100 cfs, as determined at Abiquiu Dam, during the irrigation season.

Operation for San Juan-Chama Water

El Vado Reservoir operation is affected by the San Juan-Chama Project in two ways. The first is that San Juan-Chama Project water released from Heron Dam for use downstream of El Vado Reservoir is simply passed through. The second is that storage of large volumes of San Juan-Chama Project water in El Vado Reservoir may take place for extended periods of time. The MRGCD has contracted for 20,900 acre-feet per year of San Juan-Chama Project water and maintains as much of this water in El Vado Reservoir as conditions permit. In addition, the MRGCD has contracted with various contractors of San Juan-Chama Project water to allow for those contractors to store in El Vado Reservoir.

ABIQUIU RESERVOIR

Operation for Rio Grande Water

Abiquiu Dam and Reservoir is operated for flood and sediment control in accordance with conditions and limitations stipulated in the Flood Control Act of 1960 (PL 86-645). Reservoir regulation for flood control is also coordinated with the operation of Jemez Canyon Reservoir, Cochiti Lake, and Galisteo Reservoir (an ungated structure). Abiquiu Reservoir is operated to limit flow in the Rio Chama, insofar as possible, to the downstream channel capacities of 1,800 cfs for the reach below Abiquiu Dam, 3,000 cfs for the reach below the mouth of the Rio Ojo Caliente, and on the Rio Grande main stem, 10,000 cfs below the mouth of the Rio Chama. Irrigation releases from El Vado Reservoir are passed through the reservoir. Typically, if Rio Grande inflows exceed downstream channel capacities during April and May, Abiquiu captures this peak of snowmelt runoff, and releases it during June and early July. However, any Rio Grande storage remaining after the natural flow at Otowi drops below 1,500 cfs (July 1st or later) is carried over (see Flood Carryover Storage section below) and not released until November 1st or later.

Operation for San Juan-Chama Water

In 1981, PL 97-140 authorized the storage of 200,000 acre-feet of San Juan-Chama water in Abiquiu Reservoir. The City of Albuquerque has obtained a storage easement to elevation 6,220 feet. Real estate interests have not been obtained above elevation 6,220 feet to accommodate the full 200,000 acre-feet as authorized. The San Juan–Chama capacity is annually reduced due to the estimated sediment

deposition. San Juan-Chama storage is held below elevation 6,220 feet and released as requested by the storing entities, except when releasing Rio Grande water at channel capacity (due to its flood-control priority). The San Juan-Chama pool also serves to increase the sediment trap efficiency, and enhance the recreation, fish and wildlife opportunities in the reservoir.

Flood Carryover Storage

Abiquiu Reservoir is required to retain any floodwater remaining in storage after July 1st of each year, when the natural inflow to Cochiti Lake is less than 1,500 cfs and so long as there is at least 212,000 acre-feet of capacity available at Cochiti Lake to control runoff from summer thunderstorms. This floodwater storage is the result of high inflows coming into the reservoir during spring runoff. As previously stated, the Corps has downstream channel capacity restrictions that prohibit passing all of the natural inflow. The Rio Chama has a channel capacity of 1,800 cfs below Abiquiu Reservoir, flows must not exceed 3,000 cfs at Chamita gage and there must be no more than 10,000 cfs at the Otowi gage. These restrictions result in temporary storage of natural water as well as releasing as much as downstream restrictions allow. Depending on the volume of water from spring runoff, Abiquiu Reservoir has either been able to safely pass inflow without any carryover or has locked-in as little as 3,500 (1994) acre-feet to as much as 212,000 acre-feet (1987).

The natural water that is locked-in must remain in storage until the end of irrigation season (November 1). Any natural inflow to the reservoir during the lock-in period is passed through the reservoir. After October 31st, the Corps can release the carryover, and release rates are coordinated with state and other federal agencies, and are generally used to maintain minimum flows during the winter.

Conservation Storage

In 2000, an agreement was formed to allow for up to 3-years of storage of natural flow in Abiquiu (and Jemez Canyon) to help maintain middle valley flows during the latter part of the summer and beyond, for silvery minnow habitat. This temporary agreement was implemented in the model by creating another Rio Grande Account in Abiquiu (and in Jemez), and by creating Rules understanding how to store and release this water. Flow diverted to the conservation water pool can only be stored when all downstream demands are met. Releases are made when there is not enough natural flow and San Juan-Chama water combined to meet downstream demands.

COCHITI RESERVOIR

Operation for Rio Grande Water

Congress authorized Cochiti Dam in 1960 for flood and sediment control. Operating Rules specified in P.L. 86-485 provide that the Dam is to be operated to bypass the maximum possible rate of flow that can be carried in the channel through the middle valley without causing flooding. Cochiti Lake flood control operations are primarily focused on the Middle Rio Grande Valley, controlling flows between Cochiti Lake and Elephant Butte Reservoir to current maximum channel capacities of 7,000 cfs in Albuquerque and about 4,500 cfs at the San Marcial Railroad Bridge.

As with Abiquiu, when inflow exceeds the capacity of the downstream channel, storage is retained in the reservoir and held until downstream channel conditions allow for its release, provided that, after July 1, there is 1500 cfs or more of native inflow and that a minimum of 212,000 acre-feet of storage is available in Cochiti Reservoir to control summer flood flows. Flood Carryover storage that is “locked-in” is released beginning November first. (See discussion under Flood Carryover storage at Abiquiu Reservoir).

An algorithm for balanced flood releases between Cochiti and Jemez, when there is flood storage in both reservoirs is given in their respective Water Control Manuals (WCM). Use of the balanced operation tends to cause fluctuations in each reservoirs release to maintain flood control pools.

Operation for San Juan-Chama Project Water

Public Law 88-293 authorized the release of 50,000 acre-feet of San Juan-Chama Project water for the initial filling of a permanent pool of 1200 acres in Cochiti Reservoir, and thereafter, sufficient water

annually to offset the evaporation from such area. A portion of the release of San Juan-Chama Project water is used to offset evaporation loss from the water surface of a small wetland on the Santa Fe River above Cochiti Dam. San Juan-Chama Project water destined for Elephant Butte or as supplemental irrigation water for MRGCD is passed through Cochiti.

JEMEZ CANYON RESERVOIR

Jemez Canyon Dam and Reservoir was authorized by the Flood Control Act of 1948, and is operated in tandem with Cochiti Lake to control flows through the middle valley. In 1979, a sediment control pool was established and maintained within that portion of the reservoir capacity allocated for sediment deposition. The water stored in the sediment control was completely evacuated by 2001. Flood storage, if any, is accumulated atop the sediment control pool and released as soon as possible thereafter. Jemez Canyon Reservoir is operated to prevent carryover storage of floodwater, and in conjunction with Cochiti to prevent flows exceeding downstream channel capacities. Currently there is no continuing authorization to operate Jemez for sediment retention, so it is essentially a dry reservoir.

As stated in the Cochiti operations section, balanced flood operations are attempted between Jemez and Cochiti. Also, refer to the above write-up on Abiquiu for the temporary “Rio Grande Conservation Storage” management implemented in the year 2000.

ELEPHANT BUTTE RESERVOIR

Operation for Rio Grande Water:

Elephant Butte Reservoir is the principal storage facility for the Rio Grande Project, delivering stored water for downstream use under contract between the Bureau of Reclamation and the Elephant Butte Irrigation District in New Mexico and the El Paso County Water Improvement District No. 1 in Texas. Elephant Butte Reservoir is also operated to ensure that the U. S. 1906 Treaty obligation with Mexico to deliver 60,000 acre-feet per annum at the Acequia Madre headgate in Mexico can be met. The river channel below Elephant Butte has a design capacity of 5,000 cfs.

Operation for San Juan-Chama Project Water:

In 1981, Congress authorized the Secretary of the Interior to enter into contracts for storage of San Juan–Chama Project water in Elephant Butte Reservoir. This Act (P. L. 97-140) provided that the amount of evaporation loss and spill chargeable to San Juan-Chama Project water shall be accounted under procedures established by the Rio Grande Compact Commission.

San Juan-Chama Project Water may also be stored in Elephant Butte Reservoir for recreation purposes. Originally established at 50,000 acre-feet, water in the recreation pool has been substantially diminished in size because of spill of this water from Elephant Butte Reservoir. This spill resulted from the Rio Grande Compact Commission accounting requirements that SJ-C water is to spill before native water.

CABALLO RESERVOIR

Caballo Reservoir is a re-regulating reservoir that works in conjunction with Elephant Butte Reservoir. Water released from the Elephant Butte power plant during winter generation is impounded for irrigation use during the following summer. The International Boundary and Water Commission (IBWC) manages flood control operations in and below Caballo Reservoir. Caballo is authorized for 100,000 acre-feet of flood control storage to limit flow in the Rio Grande to no more than 11,000 cfs at or below American Dam in El Paso, Texas.

GENERAL RULES DESCRIPTION

GENERAL

Following are descriptions of the general logic of the Rules for each project. More detailed descriptions (comments within the Ruleset) of each Rule and Function are given in the Specific Rules Descriptions nestled within the code.

The basic solution methodology of the water operations model is top down. In other words, the model solves in a downstream direction from the San Juan-Chama Project diversions above Heron Reservoir (Rio Chama) and the Lobatos gage (Rio Grande) near the Colorado/New Mexico border down to and including Caballo Reservoir and on to El Paso.

As a general Rule, each reservoir makes a “total release” decision based upon the physical system. In this case “total release” is defined as both San Juan-Chama (SJ-C) and Rio Grande. This total release is then reconciled with the accounting system (which keeps track of the different types of water and contractors). Reconciliation of the accounting system to the physical system is made by first releasing any Rio Grande water that was computed that needed to be released. Any remaining water is assumed to be SJ-C. In case that the physical “total release” is greater than the sum of the Rio Grande and SJ-C release, the excess release is determined to be Rio Grande. If the total release is less than the computed Rio Grande release, all release is Rio Grande. Rules then allocate and distribute each of the different types of water (SJ-C and Rio Grande) to individual contractors based upon a priority system involving contractor, release type, and in the case of Heron, destination. A discussion on how the priority system works to distribute contractor SJ-C water is presented below.

EXCHANGES, CREDITS AND DEBITS

Rules are also used to track exchanges of water for Rio Grande depletions or between accounts. These exchanges also keep track of credits and debits between various accounts. For example, Nambe Falls involves an exchange of Rio Grande water for SJ-C water. Nambe Falls is not currently modeled in URGWOM, but there is a mechanism (an account) in the model and Ruleset to release exchange water for the Nambe Falls project. Release of SJ-C water to offset depletions of Rio Grande water caused by the operation of the Nambe Falls Project must be made as soon as practical from the Nambe Falls SJ-C account in Heron to replace the depleted Rio Grande water. This water must be delivered to the Rio Grande at Otowi. This is modeled in the URGWOM model by inputting historical patterned depletions. When such an input depletion is encountered by the Rules, a series of exchanges are set up. The first exchange is set up between Otowi and the Albuquerque Abiquiu account. This exchange is seen as a demand for water by the Albuquerque Abiquiu account and if there is enough storage available in Abiquiu and the Nambe Falls Heron account has enough water to pay Albuquerque Abiquiu account back, Albuquerque Abiquiu account releases enough water to offset the depletion, by exchange. At that point, another exchange is set up between Nambe Falls account at Heron and Albuquerque Account in Abiquiu that will entirely repay the Albuquerque Abiquiu account. This second exchange is examined by the Rules computing Heron Reservoir’s release and eventually water is released from Nambe Falls Heron account in order to repay its debt to Albuquerque Abiquiu. This same process is used to propagate demands for every account with the exception of MRGCD, Reclamation’s Supplemental Water Program (SWP), and Cochiti Recreation Pool. Demands for these accounts are directly input or computed by the model.

SAN JUAN DIVERSION RULES

The SJ-C diversions Rules are the first to execute because they are at the top of the system. These Rules determine how much water will be diverted from the San Juan River Basin into the Rio Grande Basin based on statutory conditions and operating practices and priorities. Statutory conditions providing the bases for Rules set are, water availability based on bypass requirements, annual diversion volume limitation, and decade diversion volume limitation. Operating practices and priorities that are included as

Rules are tunnel and feeder capacities and operating priorities based on diversion levels, and available storage capacity available in Heron. After the SJ-C diversion Rules execute, both the SJ-C and Rio Grande or "total inflow" into Heron may then be determined.

HERON RESERVOIR RULES

The Rules for Heron then determine what the "total outflow" (the physical outflow) from Heron will be. Initially this is computed as the sum of the Rio Grande Release plus any SJ-C releases. The Rio Grande release is determined by looking at the Rio Grande inflow into the reservoir as well as any incidental storage of Rio Grande water in Heron. The goal for the operation of Heron is to avoid impairment of rights to native water. In other words, release any Rio Grande water as soon after it comes into the reservoir as possible. In the absence of knowing the actual daily Rio Grande storage amounts, the Rules implement a system of releases during the course of the month under a variety of considerations to ensure Rio Grande storage is released or maintained at very low levels.

The total SJ-C release is determined by several factors. The SJ-C release factors are; maintaining the minimum release below El Vado, rafting releases, whether a waiver has been granted to allow carry-over storage and delivery of contractor allocations. After the initial outflow is determined, the higher priority Rules for Heron check other physical factors such as the need to spill, minimum and maximum storage, maximum allowable draw-down conditions, reservoir ice cover and maximum release. This checked physical release is then reconciled against the accounting system. These reconciled Rio Grande and SJ-C releases are then distributed to specific accounts based upon the priority system described in the San Juan–Chama Release Priorities section below.

EL VADO RESERVOIR RULES

After Heron Rules have executed, the El Vado total inflow and the breakdown of the total inflow between native and SJ-C water is known. Once again, an initial total outflow is computed by the sum of the Rio Grande and SJ-C releases. The Rio Grande release is computed as the maximum of the MRGCD demand (not met by downstream Rio Grande inflow) and a target storage algorithm. Target storages are input by the user. To meet a storage target, reservoir inflow is forecasted and the Rules release water in excess of storage targets. The Rio Grande release is also constrained by Pueblo Indian water storage and release requirements and time dependent (seasonal or monthly) minimum releases. The SJ-C release is computed by the summation of any flow through accounts, exchange debts to Otowi, exchange debts to accounts in Abiquiu, rafting releases, and minimum flows.

This initial total outflow is then checked by higher priority Rules to ensure compliance with flood control criteria based on maximum pool elevations, outlet works capacity, downstream channel capacity, available capacity of Abiquiu Reservoir and the Rio Grande Compact requirements of Article VI, (retain debit water in storage in post compact reservoirs), Article VII (no increase in storage when "Usable Water in Project Storage" is less than 400,000 acre-feet) and Article VIII (Texas call for release of debit storage) to determine a final total release. The total outflow is then reconciled to the accounting system and the native and SJ-C releases for individual contractors are determined by a priority system.

ABIQUIU RESERVOIR RULES

After El Vado Rules have been executed and it's outflow computed, Abiquiu total inflow might then be determined. The initial total outflow is determined by summing the Rio Grande release and the SJ-C release. As Abiquiu cannot (in normal circumstances) store any Rio Grande water, the Rio Grande outflow is computed by looking at the amount of Rio Grande inflow and releasing the inflow and any incidental Rio Grande storage. In instances where releases would exceed downstream channel capacity, Rio Grande storage is induced. This storage is "locked-in" when the flood control criteria of Public Law 86-645 are invoked. These are, that during the July 1st through October 31st period, flood waters are retained in storage when there is a minimum of 212,000 acre-feet of vacant capacity in Cochiti Reservoir, and the natural inflow to Cochiti Reservoir (sum of Rio Chama near La Puente and Rio Grande at

Embudo) is less than 1,500 cfs. This flood carryover storage is released after October 31, and must be completely evacuated by March 31 of the following year. The default in the Rules is to compute a constant release for the period of November 1 through March 31 of the flood carryover storage available on October 31. The user can override the default method by directly inputting the flood carryover release per preferred schedule.

The SJ-C release for Abiquiu is computed by summing any flow through accounts, meeting any exchange demands, and by meeting computed MRGCD, Albuquerque, and Reclamation SWP demands below Cochiti. The SJ-C release is also constrained by a user input maximum SJ-C release. After the initial total outflow is computed, it is checked to see that it conforms to flood control criteria spillway, (outlet works and downstream channel capacity), outflow restrictions (stepped release etc.), and minimum flow requirements for tailwater fishery. This checked total outflow is then reconciled to the accounting system and distributed to individual accounts by the priority system.

COCHITI LAKE RULES

Cochiti's operation is very similar to Abiquiu's. Its main criterion is to release any Rio Grande water and incidental storage and meet user input demands for MRGCD, Albuquerque, and Reclamation's SWP. It also uses the priority system to distribute water to individual accounts, such as a pass-through account for delivering water to downstream users such as MRGCD.

JEMEZ CANYON RESERVOIR RULES

Jemez's operation is very similar to Abiquiu and Cochiti's. Its main criterion is to release any Rio Grande water and incidental storage. It is also set up to operate to capture water for a sediment pool in the allocated sediment space. Water for the earlier sediment pool and evaporation replacement were exchanged from RG to SJ-C, along with a release from Heron that's exchanged from SJ-C to RG at the RioGrandeJemez confluence. As of 2000, the authorization for the sediment pool has expired, and it is not currently in operation. The Rules are still set up allow a sediment pool, if so desired.

ELEPHANT BUTTE RESERVOIR RULES

Since only flood control operations are allowed to be incorporated in the Rules at and below Elephant Butte at this time, irrigation releases are directly input (in a data object). The flood control Rules will override these releases when the content of Elephant Butte Reservoir exceeds the prudent flood space criteria (50,000 AF vacant from April through September and 25,000 AF from October through March).

CABALLO RESERVOIR RULES

Caballo's operations are set up similarly to Elephant Butte's; irrigation releases are directly input (in a data object), and flood control Rules only override releases when flood conditions exist. Caballo's Flood Operations Criteria is a release diagram based on 2-hour instantaneous inflow rates, compromising its application in a daily model. Therefore, the current Ruleset releases one-half of the inflow when in flood operations. (It would be preferred to use a release diagram that approximates the 2-hour release diagram releases on a daily basis, but currently only one timestep period can be used per instance of RiverWare).

SAN JUAN-CHAMA RELEASE PRIORITIES

As mentioned earlier, reconciled total accounting releases are distributed by a priority system. This priority system uses 2 to 3 different sets of priorities (depending upon the reservoir) to determine which account(s) will release water. These sets of priorities include release type, destination, and account.

The highest priority of these sets of priorities is the release type priority. For example, the release type priorities for Abiquiu are FlowThrough, OtowiPaybacks, AlbuquerqueLoan (Albuquerque will let other contractors borrow from them), CochitiRecPoolPayback, MRGCD, and Reclamation. Each of these release types has a priority that can change through the year and has from one to fifteen different accounts associated with it. For example, the OtowiPaybacks release type has seven different accounts

associated with it (Albuquerque, Bernalillo, Espanola, LosAlamos, SantaFe, Taos, and Twining). Each of these associated accounts has a storage pool in Abiquiu and therefore, they can be used to pay any debt to Otowi from Abiquiu directly.

The next highest priority set (if used) is the destination priority set, of which only Heron Reservoir uses. The destination priority set is used on Heron to determine whether it is a higher priority to send water to accounts in El Vado or in Abiquiu. Heron releases are somewhat unique, in that there are no direct demands for releases from Heron (all direct demands are made from El Vado or Abiquiu and releases from Heron are made to repay El Vado or Abiquiu. It can be set up to, in effect, cause the demands to be met directly from Heron though). Releases from Heron are generally timed to help El Vado to maintain its minimum release, help produce rafting releases, or to make specified delivery of a contractors allocation, if maintaining minimum releases and rafting releases do not require enough water to force contractors water out of Heron. This priority set, like the others, can change up to 12 times through the year.

The lowest priority set is the account priority set. Each storage account on a reservoir has a priority that can change up to 12 times through the year. This priority set is used to determine which particular account will receive water if SJ-C releases are not sufficient to meet all demands for a release priority. To illustrate how the priorities work, several assumptions about the state of the model are made, as described as follows.

PRIORITY SOLUTION EXAMPLE

Assumptions:

Assume on a given day that the reconciled SJ-C release from Abiquiu is 500 cfs, of which 300 cfs is flow through water (water released from either El Vado or Heron going to a destination downstream of Abiquiu) passing through Abiquiu, MRGCD owes 150 cfs to the CochitiRecPool, Albuquerque owes 10 cfs to the CochitiRecPool, Bernalillo has a debt at Otowi of 150 cfs, and Espanola has a debt of 25 cfs at Otowi. Additionally, there is no borrowing from the Albuquerque pool. The priorities of each release type and account for Abiquiu are shown in Tables 1 and 2, respectively.

Table 1 - Release type assumed priorities for Abiquiu Reservoir

	Effective through Julian Day				
	74	90	212	304	365
RELEASE TYPE	Priority	Priority	Priority	Priority	Priority
Reclamation S.W.P.	6	6	6	6	6
OtowiPaybacks	4	4	4	4	4
Albuquerque Borrow	2	2	2	2	2
FlowThrough	1	1	1	1	1
NMISC	5	5	5	5	5
CochitiRecPoolPayback	3	3	3	3	3
MRGCD	7	7	7	7	7

Table 2 - Account priorities for Abiquiu Reservoir

	Effective through Julian Day				
ACCOUNT	74 Priority	90 Priority	212 Priority	304 Priority	365 Priority
Albuquerque	2	2	2	2	2
Bernalillo	5	5	5	5	5
Espanola	3	3	3	3	3
LosAlamos	4	4	4	4	4
MRGCD	1	1	1	1	1
Reclamation S.W.P.	6	6	6	6	6
SantaFe	7	7	7	7	7
Taos	8	8	8	8	8
Twining	9	9	9	9	9
Belen	10	10	10	10	10
LosLunas	11	11	11	11	11
NambeFalls	12	12	12	12	12
RedRiver	13	13	13	13	13
Uncontracted	14	14	14	14	14

Solution

The first thing to happen is that all debts or demands associated with a particular release type are met in order of priority. From Table 1, it can be seen that the first priority for release type in Abiquiu is FlowThrough. From the assumptions, there is 300 cfs of flow through water and therefore, all 300 cfs of flow through releases would occur. This leaves 200 cfs to distribute to the other priorities (the total of 500 cfs less the 300 cfs of flow through). The second release type priority is Albuquerque Borrow, which has been set to zero acre-feet, so 200 cfs still remains to be distributed. The third release type priority is CochitiRecPoolPayback, which has two accounts associated with it (MRGCD and Albuquerque). MRGCD has a debt of 150 cfs to the CochitiRecPool, and from Table 2 it can be seen that MRGCD has the highest account priority (between MRGCD and Albuquerque), so the entire 150 cfs debt would be met leaving 50 cfs to be distributed. Albuquerque also owes the CochtitiRecPool 10 cfs, therefore, it would be the next priority and its debt would be met in its entirety. This would leave 40 cfs to be distributed to other priorities. From Table 1, the fourth priority release type is OtowiPaybacks. For Abiquiu this priority has seven accounts associated with it (Albuquerque, Bernalillo, Espanola, LosAlamos, SantaFe, Taos, Twining). Espanola has a debt of 25 cfs to Otowi, and from Table 2 is can be seen that Espanola has a priority of 3, which is higher than the priority of 5 for Bernalillo. Therefore, Espanola would be met first and completely, leaving just 15 cfs to use toward meeting the 150 cfs debt that Bernalillo has at Otowi. So on the next time step, the Bernalillo debt at Otowi would be 150 cfs less 15 cfs (or 135 cfs) plus any additional debt that it may accumulate on the next time step.

APPENDIX A – SIMULATION AND RULEBASED SIMULATION OVERVIEW

INTRODUCTION

Rulebased Simulation can only be understood once the fundamentals of basic Simulation have been mastered. It would be impossible to describe how the URGWOM model will be implemented within the RiverWare framework without an understanding how both simulation and Rulebased simulation work within RiverWare. This section is a brief introduction to how these two *controllers* are *currently* implemented.

A RiverWare model of a river system consists of objects such as reservoirs, river reaches, diversions, confluences and water users, which are linked together to form a network. Each object "contains" physical process models and data to support those models. The links between the objects represent flow continuity, and in some cases, water surface elevation dependencies between the objects.

SIMULATION

In simulation, the physical process models range from simple linear mass balance equations to complex nonlinear processes such as dynamic routing, flow-varying losses, hydropower generation, release and spill Functions, consumptive use, and others. The user tailors the process models to meet the needs of the model and to be consistent with the size of the computational timestep. The tailoring is accomplished by selecting methods from a menu of possible model methods. For example, on a river reach object, a method of routing is selected. The simulation solution is driven by input data that provides enough known values to solve for the unknowns.

There are various solution methods (Functions) on an object corresponding to the various combinations of inputs and outputs that can solve the basic equation of the object. For a reservoir, the simplified basic equation of conservation of mass is:

$$Storage_t = Storage_{t-1} + Inflow_t \Delta t - Outflow_t \Delta t$$

Assuming that the previous storage ($Storage_{t-1}$) is known, the equation has three unknowns: Inflow, Outflow and Storage at the current timestep. Three solution methods are needed, each with solution conditions consisting of the known and unknown slots as follows:

Solve for Storage given Inflow and Outflow
Solve for Inflow given Storage and Outflow
Solve for Outflow given Storage and Inflow

Each object "knows" about its own data and solution methods and recognizes when it receives a new value in any one of its slots. A slot can get a new value from three sources: user input, a value propagated across a link from another object, and output set by the physical process methods. Whenever an object gets a new slot value, it compares its current set of known and unknown slots with the solution conditions for all the solution methods. If a set of solution conditions are met, that solution method is executed. For example, if a reservoir's storage slot is input and it receives inflow from a link propagation from an upstream object, then it will execute a method which solves for outflow. When the outflow slot is set, the value will propagate downstream if that slot is linked to the inflow of a downstream object.

Too much (conflicting) information results in an error which terminates the run; not enough information results in parts of the model left unsolved. Diagnostic and run analysis utilities alert the user to these situations and help to identify the exact location of the data problems. In the cases where multiple slot linkages make the objects mutually dependent on a solution, the objects iterate until a solution meets convergence criteria.

The objects solve at whatever timestep they receive new information, allowing some flexibility in specifying models in which the solution is not strictly propagating from upstream to downstream and forward in time. River reaches with time lags may solve for inflow given outflow, setting the inflow value at a previous timestep and propagating that value upstream. Target operations on reservoirs are solved to meet a future target storage by adjusting the reservoir's outflow over a specified range of timesteps.

Simulation allows the user to control the modeled operations by inputting reservoir releases, storage levels, or other data, directly resulting in a solution. This scenario-based modeling is especially useful for fine-tuning daily operations or predicting the response of the system to particular inputs for special studies.

EXAMPLE OF SIMULATION

Simulation in RiverWare is best explained by the use of a simple example model. Consider the model illustrated in Fig 1. The model consists of four objects, 2 reservoirs (StorageReservoir0 and StorageReservoir1), Reach0, and Confluence0. The outflow of StorageReservoir0 is linked to the inflow of Reach0, the outflow of Reach0 is linked to inflow1 of Confluence0 and the outflow of StorageReservoir1 is linked to inflow2 of Confluence0. Linking in RiverWare means that if a value is set either by user input or by the solution of the objects mass balance, that value is propagated along the link to whatever object that it is linked to. An example would be that if the outflow of StorageReservoir0 gets set, that same the value would then be propagated to the inflow of Reach0 or visa versa. To successfully run simulation within RiverWare, the model must be completely determined, meaning that the user must supply enough information so that all objects in the workspace have enough information or will obtain enough information to compute its mass balance.

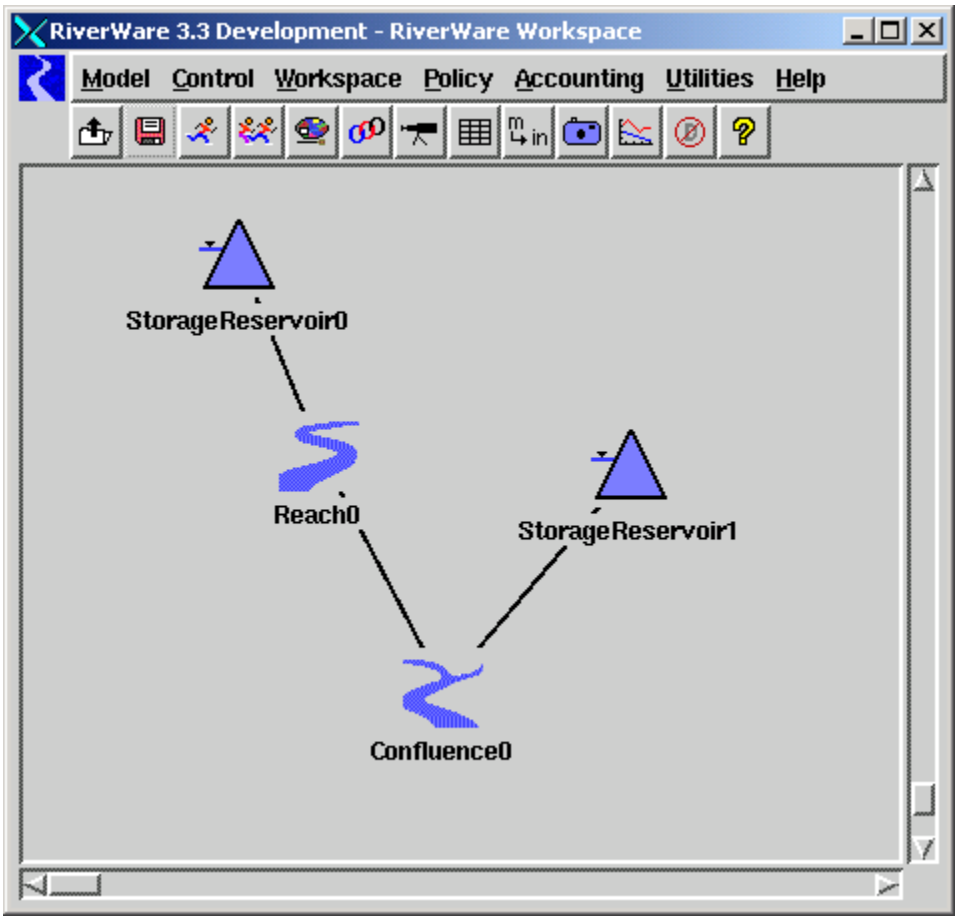
To illustrate how the simulation controller in RiverWare works, two different situations relating to the model in **Fig. 1** will be presented:

1. Solving Downstream
2. Solving Upstream.

Solving Downstream - To illustrate solving downstream, suppose that a user wants to use the model in **Fig. 1** to compute what the flow at Confluence0 would be, given that the user knows the inflow into each reservoir and there are storage targets on each reservoir. To solve this problem the user would have to supply both reservoirs, their inflows, initial storages, and their current storages (or storage targets). The model would then be completely determined and would solve in a downstream direction. The model would solve in a fashion similar to the following:

> Since, both StorageReservoir0 and StorageReservoir1 reservoirs now know what their inflow, and previous and current storages are, both reservoirs would have enough information to mass balance and they would both solve for their outflows. The outflow of StorageReservoir0 would then propagate to the inflow of Reach0 and the outflow of StorageReservoir1 would propagate to inflow2 on Confluence0. Since Reach0 now knows its inflow, it has enough information to mass balance and compute its outflow, which is then propagated to inflow1 on Confluence0. Now Confluence0 knows inflow1 and inflow2 so it can compute its outflow and the model is completely solved.

Figure 1 - Example Model For Demonstrating The Simulation Controller In RiverWare



.

Solving Upstream - To illustrate solving upstream, suppose that a user wants to use the model shown in Fig 1 to compute what the inflow into each reservoir would need to be in order to meet a particular outflow and inflow1 requirements at Confluence0. Additionally, the storage at each reservoir is fixed due to storage restrictions. From these assumptions, it should be apparent that the only free variables are the inflows to each reservoir. To solve this problem the model needs solve upstream. The model would solve in a fashion similar to the following:

> Since, Confluence0 knows its outflow and inflow1 it has enough information to mass balance and solve for inflow2. Inflow2 would get propagated to the outflow of StorageReservoir1, now StorageReservoir1 knows both its outflow and storage, so it would solve for its inflow. Inflow1 from Confluence0 would get propagated up to the outflow of Reach0 giving Reach0 enough information to solve for its inflow which would then be propagated up to the outflow of StorageReservoir0 which now knows its outflow and storage and so could solve for its inflow and the model is completely solved.

NOTE: Depending on which routing method is selected for Reach0, the model may not be able to solve upstream. Some routing methods (such as the variable time lag) cannot solve upstream.

From the above two examples of how the simulation controller within the RiverWare works, it should be readily apparent RiverWare can easily solve in any direction depending upon what inputs that the user provides to the model. It should also be apparent that this type of simulation has limited applications, as the user, via inputs to the model, determines all of the operational policy and a workable solution is

usually found by many iterative runs with the user changing inputs until the desired behavior of the river basin system is achieved. The use of straight simulation makes it very difficult, if not impossible, to compare how two different operational criteria would change the operation of a river basin over a long period of time because the user essentially has to supply every decision at every timestep through the entire simulation.

RULEBASED SIMULATION

Rulebased simulation solves in a similar fashion to simulation except that before the run, the system is under-determined, i.e., there are not enough known variables to solve the system. Whenever no objects can solve, control is passed to the Rule processor, which executes the prioritized Rules one at a time. Each Rule has the opportunity to examine the state of the system and set values of decision variables according to the logic of the operating policies. As a result, the objects have additional information, which can be used to further the simulation. Even after the objects have solved one or more times, higher priority Rules can reset the values of variables that have already been set by lower priority Rules or by simulating the effects of other Rules.

Rulebased simulation is the controller in RiverWare that directs the simulation of a river basin network based on user inputs and specified operating policies or Rules. With Rule based simulation, the simulation behavior of the river basin network can modified via the Rules without recompiling the RiverWare framework code.

There are three main components that make up Rulebased simulation within RiverWare; a Rule language, a Rule processor and a Rulebased simulation controller. The first component is used directly by RiverWare to create, modify and debug Rules. The remaining two components are used internally by RiverWare to execute Rules and control the simulation. The following is a brief description of these three main components.

Rule Language

The Rules are logical statements formulated by the modeler and written within RiverWare, in a special language, RiverWare Policy Language (RPL), which is interpreted at runtime. Thus, the Rules are data which can be saved and modified without having to recompile a program.

One of the major features of Rulebased simulation is the use of a structured editor that allows the user to quickly develop customized Rules and Functions to emulate the desired behavior in the model. Each Rule has a unique priority and can set one or many slots in the model workspace. Rules are basically formatted as "if-then-else" statements where the “if” often refers to the current state of the system (values of slots), and the "then else” sets values (slots) on the system.

Rule Processor

The Rule processor is responsible for the management and execution of the Rules and, as such, is the “traffic cop” between the user-specified Rules and the RiverWare simulator. It loads, unloads, stores and updates the Rules. It retrieves data from RiverWare that the Rules need, including the values of any variables in the RiverWare workspace, and can also set any value in the RiverWare workspace based on what the user specifies in the Rules.

Execution of Rules consists of keeping a list (referred to as the agenda), ordered by priority, of all Rules that need to execute and then executing or “firing” the highest priority Rule from the agenda. The agenda is updated during Rulebased simulation as Rules are executed and as variables on the RiverWare workspace change. A Rule is added to the agenda at the start of every timestep and whenever the value

of any of its dependencies change during the timestep. A Rule is removed from the agenda after it is executed.

Rulebased Simulator Controller

The final component is the Rulebased simulation controller itself. The Rulebased simulation controller orchestrates the alternation between execution of Rules and solving of the simulation during a Rulebased simulation run. It manages the dispatching of objects on the dispatch queue, just as the simulation controller does. When no objects can dispatch, the controller invokes the Rule processor. The Rule processor fires Rules on the agenda, beginning with the highest priority, until a Rule is successful. Then, the controller again processes the dispatch queue.

There are more steps involved, but generally the sequence of the Rulebased simulation controller is as follows:

1. Dispatch objects without using Rules until the system is completely solved or the dispatch queue is empty (no objects have enough information to dispatch).

2. Execute the highest priority Rule on the agenda and return to step 1.

3. If no Rules exist on the agenda, simulation for the timestep is complete.

In Rulebased simulation, objects will solve, just as in simulation if they are exactly specified. However, at the point that no more objects can dispatch (i.e., some objects are under determined), the Rule system is consulted for additional information. If a candidate Rule applies, it will fire and one or more slots in RiverWare may be changed. If the object (or others due to links) can dispatch, they will. This connection between Rules and the simulator continues until either the Rule agenda is empty or all objects have dispatched. In the former case, some objects are still under determined and warnings will be issued and the simulation will continue. In the latter case, the Rule agenda is consulted one last time to ensure that the priorities of the Rules have been satisfied. If not, the highest priority Rule will fire and the process will attempt to continue.

It should be noted that the Rule processor is **NOT** allowed to overwrite any user-supplied inputs; i.e., user-inputs are implicitly the highest priority. This premise forces the simulator to rely on some Rules to provide the necessary information to initially dispatch certain objects. Higher priority Rules may eventually override the effects of initial Rules, but under all circumstances, user-inputs are preserved.

Example of Rulebased Simulation

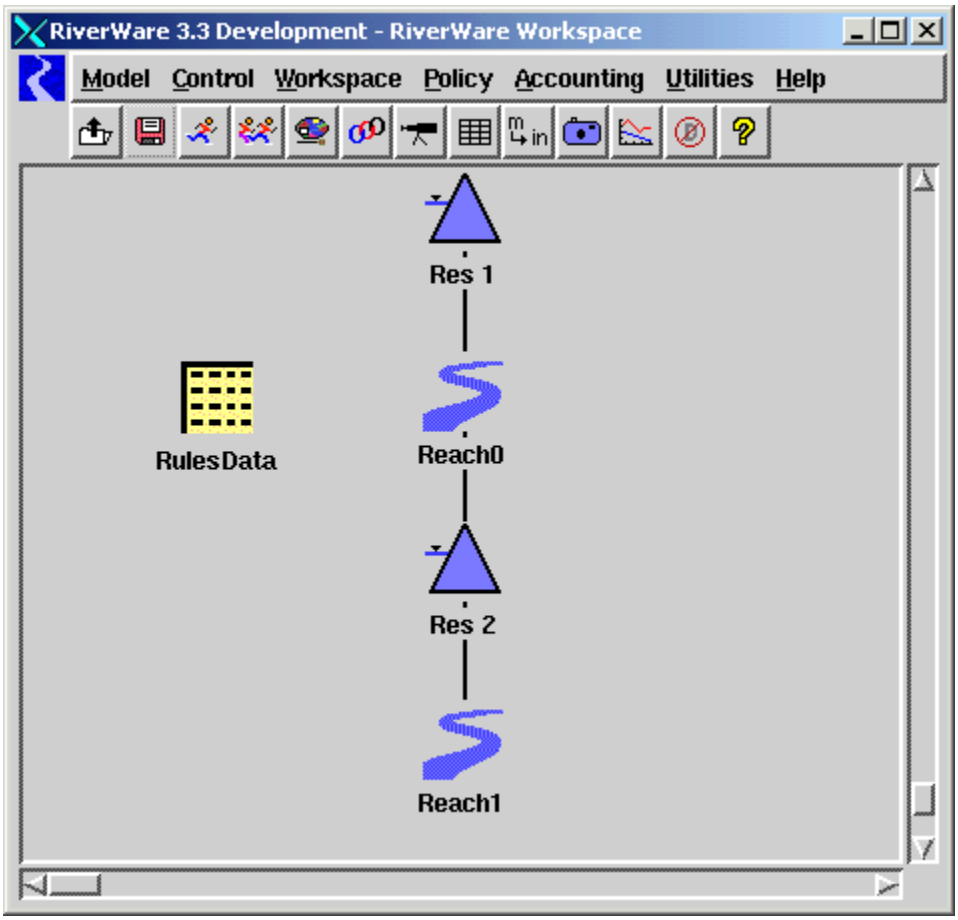
To further illustrate how Rulebased simulation works within RiverWare, consider the model in Fig. 2. This simple model consists of two reservoirs (Res1 and Res2), two reaches (Reach0 and Reach1) and one data object that contains information used by the Rules.

There are three Rules that control how this model will solve; these three Rules along with their dependencies are listed below in order of priority:

1. Check the outflow of Res1 against a minimum flow, if the outflow is less than the minimum flow, set the outflow of Res1 to the minimum flow. The dependency for this Rule is Reach1 outflow.

2. Set a target storage for Res1. The dependency for this Rule is Res1 storage.
3. Set a target outflow for Res2. The dependency for this Rule is Res2 outflow.

Figure 2 - Rulebased Simulation Example Model



The four inputs into the model are shown below:

1. Inflow into Res1 (this is input on the inflow slot on Res1)
2. Minimum flow for Res1 (this is input in the RulesData data object)
3. Target storage for Res1 (this is input in the RulesData data object)
4. Target outflow for Res2 (this is input in the RulesData data object)

Given the model in **Fig. 2** the three operation Rules and the inputs into the model, the Rulebased simulation would precede as shown in **Table 3**. **Table 3** shows the steps that would take place during one timestep when solving this model. First the action is listed, then the Rules that are on the agenda (candidates for firing), any slots that may get set by the Rule, and finally a list of slots and their current priority.

Table 3 - Steps Of Rulebased Simulation

Action	Rules on Agenda	Values Set with Rules	Priority of Values
At the first of the timestep all Rules get added to the agenda, since only the inflow into Res1 is known, no object has enough information to dispatch, so the Rule controller will execute the highest priority Rule on the agenda which is Rule 1.	Rule 1 Rule 2 Rule 3	nothing set	inflow Res1 = 0 outflow Res1 = storage Res1 = inflow Reach0 = outflow Reach0 = inflow Res2 = outflow Res2 = storage Res2 =
Execute Rule 1, which is to see if the outflow of Res1 is greater than the minimum release. Because, the outflow has not been set the Rule will fail at this time.	Rule 1 Rule 2 Rule 3	nothing set	inflow Res1 = 0 outflow Res1 = storage Res1 = inflow Reach0 = outflow Reach0 = inflow Res2 = outflow Res2 = storage Res2 =
Execute highest priority Rule on the agenda (Rule 2) Rule 2 sets the storage of Res1 and now Res1 has enough information to dispatch. Res1 will now solve for its outflow. Note Res1 outflow and storage are now set with a priority of 2, corresponding to the priority of the Rule which set them. Also note that the storage of Res2 is priority 2R, indicating it was set by a Rule. Rule 2 is now removed from the agenda, but is added on again because its dependency Res1 storage has changed.	Rule 2 Rule 3	storage Res1 outflow Res1	inflow Res1 = 0 outflow Res1 = 2 storage Res1 = 2R inflow Reach0 = outflow Reach0 = inflow Res2 = outflow Res2 = storage Res2 =
The outflow of Res1 is now propagated to the inflow of Reach0. Reach0 now has enough information to dispatch and solves for its outflow. The outflow of Reach0 is then propagated to the inflow of Res2. Note that the priority of the Rule that set the outflow of Res1 is propagated (priority 2) along with the value. Also note that the dependency for Rule 1 (Reach0 outflow) is changed and so Rule 1 gets added back on the agenda.	Rule 2 Rule 3	nothing set with Rules, all values are set with propagation	inflow Res1 = 0 outflow Res1 = 2 storage Res1 = 2R inflow Reach0 = 2 outflow Reach0 = 2 inflow Res2 = 2 outflow Res2 = storage Res2 =
Res2 only knows its inflow and so it still does not have enough information to dispatch. Therefore the highest priority Rule on the agenda is fired (Rule 1). Lets assume that the outflow set with Rule 2 was less than the minimum release and so Rule 1 (since it is a higher priority than Rule 2) sets the outflow of Res1 to the minimum release with a priority of 1. Now Res1 is overdetermined. However this is where the unique priority of the Rules come into play. Notice that the inflow of Res1 has a priority of 0, the storage has a	Rule 1 Rule 2 Rule 3	outflow Res1	inflow Res1 = 0 outflow Res1 = 1R storage Res1 = 2R inflow Reach0 = 2 outflow Reach0 = 2 inflow Res2 = 2 outflow Res2 = storage Res2 =

priority of 2R and the outflow a priority of 1R, therefore Res1 will solve for its storage since it is the lowest priority. Rule 1 is now removed from the agenda.			
The outflow of Res1 is now propagated to the inflow of Reach0. It overwrites the previous value of inflow to Reach0 because it has a higher priority. Reach0 now resolves for its outflow. The outflow of Reach0 is then propagated to the inflow of Res2. Note that the priority of the Rule that set the outflow of Res1 is once again propagated (priority 1) along with the value. Also note that the dependency for Rule 1 (Reach0 outflow) is changed and so Rule 1 gets added back on the agenda.	Rule 2 Rule 3	nothing set with Rules, all values are set with propagation	inflow Res1 = 0 outflow Res1 = 1R storage Res1 = 1 inflow Reach0 = 1 outflow Reach0 = 1 inflow Res2 = 1 outflow Res2 = storage Res2 =
Res2 still only knows its inflow and so it still does not have enough information to dispatch. Therefore the highest priority Rule on the agenda is fired (Rule 1). This time the outflow of Res1 is greater than or equal to the minimum release so it fails. Rule 2 then fires, and set the storage of Res1, however note the priorities, inflow is 0 and can't be changed and the outflow is priority 1 that was set with a Rule therefore Res1 will redispatch solving for storage with priority 1. This gives the same results as before and so Rule 2 will fail. Finally Rule 3 fires and sets the outflow of Res2 with a priority of 3. Note that the dependency for Rule 3 is the outflow of Res2 therefore Rule 3 is added back on the agenda.	Rule 1 Rule 2 Rule 3	storage Res2	inflow Res1 = 0 outflow Res1 = 1R storage Res1 = 1 inflow Reach0 = 1 outflow Reach0 = 1 inflow Res2 = 1 outflow Res2 = 3 storage Res2 =
Now Res2 has enough information to dispatch (it knows its inflow and outflow) and so it will solve for its storage and set it with a priority of 3. Rule 3 will fire one more time since it is on the agenda, however, it will fail because the value of the outflow of Res2 is already set to the same value. Rule 3 is now removed from the agenda.	Rule 3	nothing set with Rules	inflow Res1 = 0 outflow Res1 = 1R storage Res1 = 1 inflow Reach0 = 1 outflow Reach0 = 1 inflow Res2 = 1 outflow Res2 = 3R storage Res2 = 3
All objects have now dispatched and there are no Rules left on the agenda, so this will conclude the timestep. The controller would then execute the next timestep.	no Rules left on agenda	end of timestep	inflow Res1 = 0 outflow Res1 = 1R storage Res1 = 1 inflow Reach0 = 1 outflow Reach0 = 1 inflow Res2 = 1 outflow Res2 = 3 storage Res2 = 3

APPENDIX B – SPECIFIC RULES DESCRIPTION

FOREWORD

This portion of the document is taken from an actual recent Water Operations Model Ruleset (filename; "waterops.ruleset.10.16.02.mss"). It is both the code and descriptions, very close to how they appear in the Ruleset text file. For many reviewers, the greatest benefits will come from reading the individual Rule and Function descriptions, which are plain English interpretations of what they are doing. To completely understand the Rules, of course one would need to follow the logic trails for each possible action the Ruleset can take, and there are very many! Such a level of simplifying documentation is not available now, because no RiverWare tools currently exist to automatically create it, although development is slated for the future. Current resource limitations prevent flowcharting.

RULES CODE AND DESCRIPTIONS DOCUMENTATION (FILENAME; "waterops.ruleset.10.16.02.mss"):

```
# RiverWare_Ruleset 4.1 Release
# Created 15:07 October 18, 2002
#
RULESET
AGENDA_ORDER DESCENDING;
DESCRIPTION "";
PRECISION   8;
BEGIN

  POLICY_GROUP   "Exchanges";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "SetAlbuquerqueJemezEXs";
    DESCRIPTION          "This Rule sets the exchanges in place for Albuquerque to payback any
debts o Otowi that they may incur. Additionally for any exchange that may be in effect for Jemez.
Setting the exchange is the first part of the process of how debts to Otowi are made. ";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      "RGOtowiAlbuquerqueAbiquiuEX.Borrow" [] := "VolumeToFlow"( $ "DeliveryRequests.Albuquerque"
[], @"Current Timestep" );

      "NMISCJemezNMISCAbiquiuEX.Borrow" [] := 0.00000000 ["cfs"];

    END;

    RULE                 "SetAlbuquerqueLoanEXs";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "AlbuquerqueWillLoan"(  );
    BEGIN

      FOREACH (LIST slotValueList IN "AlbuquerqueLoanEXs"( "AlbuquerqueAbiquiuEXAccounts"(  ) ))
DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueList ) CONCAT ".Borrow" [] :=
"VolumeToFlow"( GET NUMERIC @INDEX 1.00000000 FROM slotValueList, @"Current Timestep" );

      ENDFOREACH;

    END;

    RULE                 "SetHeronElVadoEXs";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      FOREACH (LIST slotValue IN "HeronElVadoEXs"(  )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := GET NUMERIC @INDEX 1.00000000
FROM slotValue;
```

```
      ENDFOREACH;

    END;

    RULE                 "SetNoAlbuquerqueLoanEXs";
    DESCRIPTION          "This Rule sets the basic exchanges of San Juan Chama water for Rio
Grande water as requested by
the New Mexico's state Engineer's office to offset depletions of Rio Grande water by San Juan
Chama
contractors.

This particular Rule only sets these exchanges if Albuquerque has no water to loan in its pool in
Abiquiu.  Therefore each indivdual contractor must make the payback from their own pool.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT "AlbuquerqueWillLoan"(  );
    BEGIN

      FOREACH (LIST exchanges IN "MakeOtowiExchangeList"( "AbiquiuEXStorageAccounts"(  ),
"Abiquiu" )) DO
            ( GET STRING @INDEX 1.00000000 FROM exchanges ) CONCAT ".Borrow" [] :=
"VolumeToFlow"( "DeliveryRequestFor"( GET STRING @INDEX 0.00000000 FROM exchanges ), @"Current
Timestep" );

      ENDFOREACH;

      FOREACH (LIST exchanges IN "MakeOtowiExchangeList"( "StorageInElVadoOnlyAccounts"(  ),
"ElVado" )) DO
            ( GET STRING @INDEX 1.00000000 FROM exchanges ) CONCAT ".Borrow" [] :=
"VolumeToFlow"( "DeliveryRequestFor"( GET STRING @INDEX 0.00000000 FROM exchanges ), @"Current
Timestep" );

      ENDFOREACH;

      FOREACH (LIST exchanges IN "MakeOtowiExchangeList"( "StorageInHeronOnlyAccounts"(  ),
"Heron" )) DO
            ( GET STRING @INDEX 1.00000000 FROM exchanges ) CONCAT ".Borrow" [] :=
"VolumeToFlow"( "DeliveryRequestFor"( GET STRING @INDEX 0.00000000 FROM exchanges ), @"Current
Timestep" );

      ENDFOREACH;

    END;

  END;

  POLICY_GROUP   "Water Leases";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "Heron Lease Amounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "AlbuquerqueHeronToReclamationHeron.Supply" [];
    BEGIN

      FOREACH (LIST slotValue IN "HeronReclamationLeaseList"(  )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := "VolumeToFlow"( GET NUMERIC
@INDEX 1.00000000 FROM slotValue, @"Current Timestep" );

      ENDFOREACH;

    END;

    RULE                 "ElVado Lease Amounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      FOREACH (LIST slotValue IN "ElVadoReclamationLeaseList"(  )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := "VolumeToFlow"( GET NUMERIC
@INDEX 1.00000000 FROM slotValue, @"Current Timestep" );

      ENDFOREACH;

    END;
```

```
    RULE                 "Abiquiu Lease Amounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      FOREACH (LIST slotValue IN "AbiquiuReclamationLeaseList"(  )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := "VolumeToFlow"( GET NUMERIC
@INDEX 1.00000000 FROM slotValue, @"Current Timestep" );

      ENDFOREACH;

    END;

    RULE                 "Set MRGCD Loans";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      FOREACH (LIST slotValue IN "GetSortedReclamationMRGCDLoans"( $
"HeronData.ReclamationFromMRGCDLoan" [] )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := GET NUMERIC @INDEX 1.00000000
FROM slotValue;

      ENDFOREACH;

      FOREACH (LIST slotValue IN "GetSortedAlbuquerqueMRGCDLoans"( $
"HeronData.AlbuquerqueFromMRGCDLoan" [] )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValue ) [] := GET NUMERIC @INDEX 1.00000000
FROM slotValue;

      ENDFOREACH;

    END;

    RULE                 "Set Reclamation ElVado Abiquiu Payback";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "ReclamationElVadoToMRGCDElVado.Supply" [];
    BEGIN

      "ReclamationElVadoToMRGCDElVado.Supply" [] := IF ( "ElVadoIsPriority"(  ) )
   THEN
      "VolumeToFlow"( "Min"( "GetAccountDebt"( "ReclamationElVadoToMRGCDElVado.Supply" ),
"PreviousAccountStorage"( "Reclamation", % "ElVado" ) ), @"Current Timestep" )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

      "ReclamationAbiquiuToMRGCDAbiquiu.Supply" [] := "VolumeToFlow"( "Min"( "GetAccountDebt"(
"ReclamationAbiquiuToMRGCDAbiquiu.Supply" ), "Max"( "PreviousAccountStorage"( "Reclamation", %
"Abiquiu" ), 0.00000000 ["acre-feet"] ) ), @"Current Timestep" );

      "AlbuquerqueAbiquiuToAlbuquerqueAbiquiuMRGCDAbiquiuAbiquiu.Supply" [] := "VolumeToFlow"(
"Min"( "GetAccountDebt"( "AlbuquerqueAbiquiuToAlbuquerqueAbiquiuMRGCDAbiquiuAbiquiu.Supply" ),
"Max"( "PreviousAccountStorage"( "Albuquerque", % "Abiquiu" ), 0.00000000 ["acre-feet"] ) ),
@"Current Timestep" );

    END;

  END;

  POLICY_GROUP   "Forecast Errors";
  DESCRIPTION    "";
  ACTIVE         FALSE;
  BEGIN

    RULE                 "ForecastErrorPercent";
    DESCRIPTION          "Computes percent forecast error given if forecast error is available.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "IsFirstDayOfMonth"(  ) AND IsNaN $ "ForecastData.PercentForecastError"
["EndOfMonth"(  )];
    BEGIN

      $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] := IF ( NOT IsNaN $
"ForecastData.ForecastError" ["EndOfMonth"(  )] )
   THEN
      IF ( "GetMonth"( @"Current Timestep" ) <= 7.00000000 )
```

```
      THEN
         IF ( "EstimateElVadoInflow"( @"24:00:00 March 1, Current Year", @"24:00:00 July Max
DayOfMonth, Current Year" ) != 0.00000000 ["acre-feet"] )
         THEN
            $ "ForecastData.ForecastError" ["EndOfMonth"(  )] / "EstimateElVadoInflow"(
@"24:00:00 March 1, Current Year", @"24:00:00 July Max DayOfMonth, Current Year" )
         ELSE
            0.10000000
         ENDIF
      ELSE
         IF ( "GetYear"( @"Current Timestep" ) == "GetYear"( @"Start Timestep" ) )
         THEN
            0.04000000
         ELSE
            "Min"( 0.10000000, $ "ForecastData.PercentForecastError" [@"24:00:00 July Max
DayOfMonth, Current Year"] )
         ENDIF
      ENDIF
   ENDIF;

    END;

    RULE                 "RewindRandomFile";
    DESCRIPTION          "Dummy Rule to rewind the random number file.
";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      PRINT IF ( @"Current Timestep" == @"Start Timestep" )
   THEN
      "ResetRanDev"( TRUE, @"24:00:00 October Max DayOfMonth, 1983" )
   ENDIF;

    END;

    RULE                 "CalculatedForecastError";
    DESCRIPTION          "Computes a forecast error using a random number adjusted for historic
forecast error of given reservoir.  Seed of random number generator
is a constant so that runs can be replicated.  Random numbers are
currently from a file generated by an old CRSS function.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "IsFirstDayOfMonth"(  ) AND IsNaN $ "ForecastData.ForecastError"
["EndOfMonth"(  )];
    BEGIN

      $ "ForecastData.ForecastError" ["EndOfMonth"(  )] := IF ( "Abs"( "ComputeForecastError"(  )
) > $ "ForecastData.MaximumForecastError" ["ElVado", "GetMonthAsString"( @"Current Timestep" )] )
   THEN
      IF ( "ComputeForecastError"(  ) < 0.00000000 ["acre-feet"] )
      THEN
         $ "ForecastData.MaximumForecastError" ["ElVado", "GetMonthAsString"( @"Current Timestep"
)] * - 1.00000000
      ELSE
         $ "ForecastData.MaximumForecastError" ["ElVado", "GetMonthAsString"( @"Current Timestep"
)]
      ENDIF
   ELSE
      "ComputeForecastError"(  )
   ENDIF;

    END;

  END;

  POLICY_GROUP   "Diversions And Demands";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "SetRioChamaDiversionAndDepletionRequested";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"Start Timestep" AND IsNaN $
"AbvAbiquiuDiversions:Monastery.Diversion Requested" [];
    BEGIN

      FOREACH (DATETIME date IN @"Start Timestep" TO @"Finish Timestep") DO
            FOREACH (OBJECT acequia IN "ListSubbasin"( "BelowAbiquiuDiversions" )) DO
```

```
            acequia & "Diversion Requested" [date] := IF ( IsNaN acequia & "Diversion Requested"
[date] )
   THEN
      "GetDiversionRequested"( "StringifyAggObjectElement"( acequia ), % "RioChamaDiversionData",
date )
   ENDIF;

            acequia & "Depletion Requested" [date] := IF ( IsNaN acequia & "Depletion Requested"
[date] )
   THEN
      "GetDepletionRequested"( "StringifyAggObjectElement"( acequia ), % "RioChamaDiversionData",
date )
   ENDIF;

      ENDFOREACH;

      ENDFOREACH;

    END;

    RULE                 "SetMiddleRioGrandeDiversionRequested";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"Start Timestep" AND IsNaN $
"BlwCochitiDiversions:EastSideMain.Diversion Requested" [];
    BEGIN

      FOREACH (DATETIME date IN @"Start Timestep" TO @"Finish Timestep") DO
            FOREACH (OBJECT diversion IN "ListSubbasin"( "MiddleRioGrandeDiversions" )) DO
            diversion & "Diversion Requested" [date] := IF ( IsNaN diversion & "Diversion
Requested" [date] )
   THEN
      IF ( "StringifyAggObjectElement"( diversion ) == "AlgodonesDrain" )
      THEN
         - 1.00000000 * "GetDiversionRequested"( "StringifyAggObjectElement"( diversion ), %
"MiddleRioGrandeDiversionData", date )
      ELSE
         "GetDiversionRequested"( "StringifyAggObjectElement"( diversion ), %
"MiddleRioGrandeDiversionData", date )
      ENDIF
   ENDIF;

      ENDFOREACH;

      ENDFOREACH;

      FOREACH (DATETIME date IN @"Start Timestep" TO @"Finish Timestep") DO
            FOREACH (OBJECT diversion IN "ListSubbasin"( "LowerMiddleRioGrandeDiversions" )) DO
            diversion & "Diversion Request" [date] := IF ( IsNaN diversion & "Diversion Request"
[date] )
   THEN
      "GetDiversionRequested"( "StringifyObject"( diversion ), % "MiddleRioGrandeDiversionData",
date )
   ENDIF;

      ENDFOREACH;

      ENDFOREACH;

    END;

    RULE                 "ComputeReleaseToMeetMinimumCentralFlow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "MiddleValleyDemands.MinReleaseForCentral" [] AND NOT IsNaN
"MinTargetFlow"( "Central", 1.00000000 ["day"] );
    BEGIN

      $ "MiddleValleyDemands.MinReleaseForCentral" [] := "Max"( "ReleaseToMeetMinFlowAtCentral"(
1.00000000 ["day"] ), "ReleaseToMeetMinFlowAtCentral"( 2.00000000 ["day"] ) );

    END;

    RULE                 "ComputeReleaseToMeetMinimumIsletaFlow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "MiddleValleyDemands.MinReleaseForIsleta" [] AND NOT IsNaN
"MinTargetFlow"( "Isleta", 1.00000000 ["day"] );
    BEGIN
```

```
      $ "MiddleValleyDemands.MinReleaseForIsleta" [] := "MaxItem"( {
"ReleaseToMeetMinFlowAtIsleta"( 1.00000000 ["day"] ) , "ReleaseToMeetMinFlowAtIsleta"( 2.00000000
["day"] ) , "ReleaseToMeetMinFlowAtIsleta"( 3.00000000 ["day"] ) } );

    END;

    RULE                "ComputeReleaseToMeetMinimumSanAcaciaFlow";
    DESCRIPTION         "";
    ACTIVE              TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "MiddleValleyDemands.MinReleaseForSanAcacia" [] AND NOT IsNaN
"MinTargetFlow"( "SanAcacia", 2.00000000 ["day"] );
    BEGIN

      $ "MiddleValleyDemands.MinReleaseForSanAcacia" [] := "MaxItem"( {
"ReleaseToMeetMinFlowAtSanAcacia"( 2.00000000 ["day"] ) , "ReleaseToMeetMinFlowAtSanAcacia"(
3.00000000 ["day"] ) , "ReleaseToMeetMinFlowAtSanAcacia"( 4.00000000 ["day"] ) } );

    END;

    RULE                "ComputeReleaseToMeetMinimumSanMarcialFlow";
    DESCRIPTION         "";
    ACTIVE              TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "MiddleValleyDemands.MinReleaseForSanMarcial" [] AND NOT IsNaN
"MinTargetFlow"( "SanMarcial", 3.00000000 ["day"] );
    BEGIN

      $ "MiddleValleyDemands.MinReleaseForSanMarcial" [] := "MaxItem"( {
"ReleaseToMeetMinFlowAtSanMarcial"( 3.00000000 ["day"] ) , "ReleaseToMeetMinFlowAtSanMarcial"(
4.00000000 ["day"] ) , "ReleaseToMeetMinFlowAtSanMarcial"( 5.00000000 ["day"] ) } );

    END;

    RULE                "SetCochitiMinimumFlow";
    DESCRIPTION         "";
    ACTIVE              TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "MiddleValleyDemands.MinimumFlow" ["DatePlusXTimesteps"(
@"Current Timestep", 1.00000000 ["day"] )];
    BEGIN

      FOREACH (DATETIME date IN @"Current Timestep" TO "DatePlusXTimesteps"( @"Current Timestep",
1.00000000 ["day"] )) DO
            $ "MiddleValleyDemands.MinimumFlow" [date] := IF ( IsNaN $
"MiddleValleyDemands.MinimumFlow" [date] )
   THEN
      "ComputeCochitiMinimumFlow"(  )
   ENDIF;

      ENDFOREACH;

    END;

    RULE                "ComputeAbiquiuMRGCDDemand";
    DESCRIPTION         "";
    ACTIVE              TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.ComputedMRGCDDemand" [] AND IsNaN $
"AbiquiuData.MRGCDDemand" ["DatePlusXTimesteps"( @"Current Timestep", 1.00000000 ["day"] )];
    BEGIN

      $ "AbiquiuData.ComputedMRGCDDemand" [] := ( "AbiquiuComputedMRGCDDemand"( 1.00000000
["day"] ) + "AbiquiuComputedMRGCDDemand"( 0.00000000 ["day"] ) ) / 2.00000000;

    END;

    RULE                "SetAbiquiuMRGCDDemand";
    DESCRIPTION         "";
    ACTIVE              TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.MRGCDDemand" ["DatePlusXTimesteps"( @"Current
Timestep", 1.00000000 ["day"] )];
    BEGIN

      FOREACH (DATETIME date IN @"Current Timestep" TO "DatePlusXTimesteps"( @"Current Timestep",
1.00000000 ["day"] )) DO
            $ "AbiquiuData.MRGCDDemand" [date] := $ "AbiquiuData.ComputedMRGCDDemand" [];

      ENDFOREACH;

    END;
```

```
    RULE                 "SetElVadoMRGCDDemand";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.MRGCDDemand" [];
    BEGIN

      $ "ElVadoData.MRGCDDemand" [] := "ComputeElVadoMRGCDDemand"( 1.00000000 ["day"] ) -
"LetterWaterAdjustment"( % "ElVado", @"Current Timestep" );

    END;

    RULE                 "SetMinElVadoMRGCDRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.ComputedMinMRGCDRelease" [];
    BEGIN

      $ "ElVadoData.ComputedMinMRGCDRelease" [] := "ComputeMinElVadoRGDemand"(  );

    END;

    RULE                 "ComputedMinAbiquiuRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.ComputedMinRGDemandRelease" [] AND IsNaN $
"AbiquiuData.MinRGDemandRelease" ["DatePlusXTimesteps"( @"Current Timestep", 1.00000000 ["day"]
)];
    BEGIN

      $ "AbiquiuData.ComputedMinRGDemandRelease" [@"Current Timestep"] := "Max"(
"AbiquiuMinRGDemand"( @"Current Timestep" ), "MinRGOutflow"( % "Abiquiu" ) );

    END;

    RULE                 "SetMinAbiquiuRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.MinRGDemandRelease" ["DatePlusXTimesteps"(
@"Current Timestep", 1.00000000 ["day"] )];
    BEGIN

      FOREACH (DATETIME writeDate IN @"Current Timestep" TO "DatePlusXTimesteps"( @"Current
Timestep", 1.00000000 ["day"] )) DO
            $ "AbiquiuData.MinRGDemandRelease" [writeDate] := $
"AbiquiuData.ComputedMinRGDemandRelease" [];

      ENDFOREACH;

    END;

    RULE                 "SetEstimatedAbquiuToCochitiLosses";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.EstimatedAbiquiuToCochitiLosses" [];
    BEGIN

      $ "AbiquiuData.EstimatedAbiquiuToCochitiLosses" [] := $ "AbiquiuData.MRGCDDemand" [] - $
"MiddleValleyDemands.MRGCD" [@"Current Timestep + 1 Timesteps"];

    END;

    RULE                 "AbiquiuTotalFlowToMeetTarget";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.TotalFlowNeededToMeetTarget" [];
    BEGIN

      $ "AbiquiuData.TotalFlowNeededToMeetTarget" [] := ( "AbiquiuMinFlowsDemand"( 1.00000000
["day"] ) + "AbiquiuMinFlowsDemand"( 0.00000000 ["day"] ) ) / 2.00000000;

    END;

  END;

  POLICY_GROUP   "Preparatory Rules";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN
```

```
    RULE                 "SetCummulativeAccountFillRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" > @"Start Timestep" AND IsNaN $
"HeronData.CumulativeAccountFillRelease" [@"Previous Timestep"];
    BEGIN

      $ "HeronData.CumulativeAccountFillRelease" [@"Previous Timestep"] := IF ( @"Previous
Timestep" == "AccountFillDateGivenDate"( % "Heron", @"Previous Timestep" ) OR @"Previous
Timestep" == @"24:00:00 January 1, Current Year" )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      $ "HeronData.CumulativeAccountFillRelease" [@"Previous Timestep - 1 Timesteps"] + WITH LIST
supplies = "MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "ElVadoStorageAccounts"(  ) ),
"PrioritizedReleaseTypeAccounts"( "ElVadoStorageAccounts"(  ), "AccountFill", % "Heron", %
"ElVado" ), % "Heron", % "Heron" ) CONCAT "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"AbiquiuRootStorageAccounts"(  ) ), "PrioritizedReleaseTypeAccounts"(
"AbiquiuRootStorageAccounts"(  ), "AccountFill", % "Heron", % "Abiquiu" ), % "Heron", % "Heron" )
DO
         FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
            result + "FlowToVolume"( "ZeroNaNs"( "SlotifyString"( supply ), @"Previous Timestep"
), @"Current Timestep" )
         ENDFOR
      ENDWITH
   ENDIF;

    END;

    RULE                 "SetSanJuanContractorAllocations";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"24:00:00 January 1, Current Year" AND IsNaN
"Heron^" CONCAT "Albuquerque" CONCAT ".Begin Year Allocation" [];
    BEGIN

      FOREACH (STRING account IN "RootStorageAccounts"( % "Heron" )) DO
            "Heron^" CONCAT account CONCAT ".Begin Year Allocation" [] := IF ( IsNaN "Heron^"
CONCAT account CONCAT ".Begin Year Allocation" [] )
   THEN
      $ "HeronData.SanJuanContractorAllocations" [account, "Allocation"]
   ENDIF;

      ENDFOREACH;

    END;

    RULE                 "CompactVIIUsableStorage";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      $ "RioGrandeCompactData.CompactVIIUsableWater" [] := "Max"( "PreviousAccountStorage"(
"RioGrande", % "ElephantButte" ) - "NMDebit"(  ) - "CODebit"(  ), 0.00000000 ["acre-feet"] ) +
"PreviousStorage"( % "Caballo" );

    END;

    RULE                 "SetPercentRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.PercentRGRelease" [] AND @"Current Timestep" <=
"TargetFillDate"( % "ElVado" );
    BEGIN

      FOREACH (DATETIME date IN @"Current Timestep" TO @"Current Timestep + 13 Timesteps") DO
            $ "ElVadoData.PercentRGRelease" [date] := "ComputePercentElVadoRGRelease"(  );

      ENDFOREACH;

    END;

    RULE                 "SetElVadoIndianStorageReqAprilAndMarch";
    DESCRIPTION          "This Rule sets the El Vado Indian storage requirement for each month on
the first day of the month.
The computation looks from the current month through October to determine how much storage is
required and sets that value on the IndianStorageReq slot on the ElVadoData Data Object.";
    ACTIVE               TRUE;
```

```
    RULE_EXEC_CONSTRAINT IsNaN "ElVadoData.IndianStorageReq" ["EndOfMonth"(  )] AND
"IsFirstDayOfMonth"(  ) OR ( ( @"Current Timestep" == @"Start Timestep" AND NOT
"IsFirstDayOfMonth"(  ) ) AND IsNaN "ElVadoData.IndianStorageReq" ["EndOfMonth"(  )] );
    BEGIN

      "ElVadoData.IndianStorageReq" [@"24:00:00 Current Month Max DayOfMonth, Current Year"] :=
IF ( "IsIrrigationSeason"(  ) AND @"Current Timestep" <= @"24:00:00 April 1, Current Year" )
   THEN
      "SumList"( "ComputeIndianStorageReq"( @"Current Timestep" ) )
   ENDIF;

    END;

    RULE                 "SetElVadoIndianStorageReqAfterApril";
    DESCRIPTION          "This Rule sets the El Vado Indian storage requirement for each month on
the first day of the month.
The computation looks from the current month through October to determine how much storage is
required and sets that value on the IndianStorageReq slot on the ElVadoData Data Object.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "IrrigationSeason"(  ) AND @"Current Timestep" >= @"24:00:00 April 30,
Current Year" AND ( IsNaN "ElVadoData.IndianStorageReq" ["EndOfMonth"(  )] AND
"IsFirstDayOfMonth"(  ) OR ( ( @"Current Timestep" == @"Start Timestep" AND NOT
"IsFirstDayOfMonth"(  ) ) AND IsNaN "ElVadoData.IndianStorageReq" ["EndOfMonth"(  )] ) );
    BEGIN

      FOREACH (NUMERIC index IN "CreateNumericList"( 0.00000000, ( LENGTH "MonthlyDATETIMEList"(
@"Current Timestep", @"24:00:00 October 1, Current Year" ) ) - 1.00000000, 1.00000000 )) DO
            "ElVadoData.IndianStorageReq" [GET DATETIME @INDEX index FROM "MonthlyDATETIMEList"(
@"Current Timestep", @"24:00:00 October 1, Current Year" )] := WITH LIST storageReqs =
"ComputeIndianStorageReq"( @"Current Timestep" ) DO
      FOR ( NUMERIC index2 IN "CreateNumericList"( index, ( LENGTH storageReqs ) - 1.00000000,
1.00000000 ) ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
         result + GET NUMERIC @INDEX index2 FROM storageReqs
      ENDFOR
   ENDWITH;

      ENDFOREACH;

    END;

    RULE                 "SetPreviousMonthIndianStorageRequirement";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.PreviousMonthIndianStorageReq" ["MaxDate"(
@"24:00:00 Previous Month Max DayOfMonth, Current Year", @"Start Timestep" )] AND
"IsFirstDayOfMonth"(  ) OR ( ( @"Current Timestep" == @"Start Timestep" AND NOT
"IsFirstDayOfMonth"(  ) ) AND IsNaN $ "ElVadoData.PreviousMonthIndianStorageReq" ["MaxDate"(
@"24:00:00 Previous Month Max DayOfMonth, Current Year", @"Start Timestep" )] );
    BEGIN

      $ "ElVadoData.PreviousMonthIndianStorageReq" ["MaxDate"( @"24:00:00 Previous Month Max
DayOfMonth, Current Year", @"Start Timestep" )] := IF ( "GetMonthAsString"( @"Current Timestep" )
== "March" OR NOT "IrrigationSeason"(  ) )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      "ComputeMonthlyIndianStorageReq"( @"Current Timestep" )
   ENDIF;

    END;

    RULE                 "ZeroElVadoIndianStorageReq";
    DESCRIPTION          "This Rule sets the El Vado Indian storage requirement for each month on
the first day of the month.
The computation looks from the current month through October to determine how much storage is
required and sets that value on the IndianStorageReq slot on the ElVadoData Data Object.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "ElVadoData.IndianStorageReq" [@"24:00:00 Current Month Max
DayOfMonth, Current Year"];
    BEGIN

      "ElVadoData.IndianStorageReq" [@"24:00:00 Current Month Max DayOfMonth, Current Year"] :=
0.00000000 ["acre-feet"];

    END;

    RULE                 "ComputeHeronRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
```

```
    RULE_EXEC_CONSTRAINT IsNaN $ "HeronData.RGOutflow" [];
    BEGIN

      $ "HeronData.RGOutflow" [] := "HeronRGOutflow"(  );

    END;

    RULE                 "ComputeElVadoRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.RGOutflow" [];
    BEGIN

      $ "ElVadoData.RGOutflow" [] := IF ( "CompactVIIIsInEffect"(  ) )
   THEN
      IF ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) < "IndianStorageRequirement"(  )
OR "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
"IndianStorageRequirement"(  ), @"Current Timestep" ) < "ElVadoRGRelease"(  ) )
      THEN
         "IndianStorageRequirementRelease"(  )
      ELSE
         "ElVadoRGRelease"(  )
      ENDIF
   ELSE
      "Min"( "ElVadoRGRelease"(  ), "IndianStorageRequirementRelease"(  ) )
   ENDIF;

    END;

    RULE                 "ComputeHeronSJRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "HeronData.SJOutflow" [];
    BEGIN

      $ "HeronData.SJOutflow" [] := IF ( "HeronSJOutflow"(  ) < 1.00000000 ["cfs"] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "HeronSJOutflow"(  )
   ENDIF;

    END;

  END;

  POLICY_GROUP   "San Juan Diversions";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "ComputeAnnualSJDiversion";
    DESCRIPTION          "This Rule sums up the total annual San Juan Chama Diversion for each
calendar year.  Note this Rule only executes on
January 1st of each year.  This information is used with the EODTotal function which computes the
total ten year
San Juan Chama diversion.";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"24:00:00 January 1, Current Year" AND NOT
@"Current Timestep" == @"Start Timestep" AND IsNaN $ "SanJuanChamaDiversions.AnnualDiversion"
[@"24:00:00 December 31, Previous Year"];
    BEGIN

      $ "SanJuanChamaDiversions.AnnualDiversion" [@"Previous Timestep"] := "FlowToVolume"(
"SumSlotOverTime"( "SanJuanChamaDiversions.Total", "DateMax"( @"Start Timestep", @"24:00:00
January 1, Previous Year" ), @"Previous Timestep" ), @"Previous Timestep" );

    END;

    RULE                 "San Juan Diversions";
    DESCRIPTION          "Author: Brad Vickers, Wave Engineering Inc.
Date:    August, 1998

Comments:

This Rule sets the diversion slot for each of the San Juan Chama Diversions (Blanco Diversion, ";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN
```

```
      FOREACH (OBJECT diversion IN "ListSubbasin"( "San Juan Diversions" )) DO
            diversion & "Diversion Request" [] := IF ( "IsMaintenanceSwitchOn"( diversion ) )
   THEN
      "MaintenanceFlow"( diversion )
   ELSE
      IF ( "DiversionIsLimited"(  ) )
      THEN
         IF ( "TargetTotalDiversion"(  ) < $ "SanJuanChamaRules.SmallDivThreshold" [0.00000000,
0.00000000] )
         THEN
            "SmallDivCalc"( diversion, "TargetTotalDiversion"(  ) )
         ELSE
            "LargeDivCalc"( diversion, "TargetTotalDiversion"(  ) )
         ENDIF
      ELSE
         "Capacity"( diversion )
      ENDIF
   ENDIF;

      ENDFOREACH;

      $ "OsoTunnelInlet.Diversion Request" [] := "Capacity"( % "OsoTunnelInlet" );

      $ "AzoteaTunnelInlet.Diversion Request" [] := "Capacity"( % "AzoteaTunnelInlet" );

    END;

  END;

  POLICY_GROUP   "Heron ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "SetCarryover";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"24:00:00 January 1, Current Year";
    BEGIN

      FOREACH (STRING account IN "HeronRootDownstreamStorageAccounts"(  )) DO
            "Heron^" CONCAT account CONCAT ".Carry Over" [] := IF ( "PreviousYearIsWaiverYear"(
) )
   THEN
      "PreviousAccountStorage"( account, % "Heron" )
   ELSE
      0.00000000 ["acre-feet"]
   ENDIF;

      ENDFOREACH;

      FOREACH (STRING account IN "HeronNonDownstreamStorageAccounts"(  )) DO
            "Heron^" CONCAT account CONCAT ".Carry Over" [] := 0.00000000 ["acre-feet"];

      ENDFOREACH;

    END;

    RULE                 "HeronCheckSpill";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      "Heron.Outflow" [] := IF ( "HeronMustSpill"(  ) )
   THEN
      "Min"( "ComputeMaxOutflow"( % "Heron" ), "HeronMaxRGStorageToRel"(  ) )
   ENDIF;

    END;

    RULE                 "HeronCheckMinMaxStorage";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN "Heron.Storage" [];
    BEGIN

      "Heron.Outflow" [] := IF ( "Heron.Storage" [] < "MinStorage"( % "Heron" ) )
```

```
   THEN
      "ComputeOutflowAtGivenStorage"( % "Heron", "MinStorage"( % "Heron" ) )
   ELSE
      IF ( "Heron.Storage" [] > "MaxStorage"( % "Heron" ) )
      THEN
         "Min"( "HeronMaxRGStorageToRel"(  ) + "SJOutflow"( % "Heron" ), "GetMaxOutflowGivenHW"(
% "Heron", "MaxElevation"( % "Heron" ), @"Current Timestep" ) )
      ENDIF
   ENDIF;

    END;

    RULE                 "HeronCheckDeltaStorage";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Heron.Pool Elevation" [];
    BEGIN

      "Heron.Outflow" [] := IF ( "ComputeDeltaPoolElev"( % "Heron" ) > $
"HeronData.maxDeltaPoolElev" [0.00000000, 0.00000000] )
   THEN
      "HeronLimitDeltaStorage"(  )
   ENDIF;

      PRINT IF ( "ComputeDeltaPoolElev"( % "Heron" ) > $ "HeronData.maxDeltaPoolElev"
[0.00000000, 0.00000000] )
   THEN
      "Heron Change in Storage limited to" CONCAT STRINGIFY $ "HeronData.maxDeltaPoolElev"
[0.00000000, 0.00000000] CONCAT " feet/day"
   ENDIF;

    END;

    RULE                 "HeronOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "Heron.Outflow" [];
    BEGIN

      $ "Heron.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "Heron" ) )
   THEN
      "Min"( "RGOutflow"( % "Heron" ) + "SJOutflow"( % "Heron" ), "ComputeMaxOutflow"( % "Heron"
) )
   ELSE
      "TotalOutflow"( % "Heron" )
   ENDIF;

    END;

    RULE                 "SetHeronRGAccount";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Heron.Inflow" [] AND NOT IsNaN $ "Heron.Outflow" [] AND
IsNaN "RioGrandeHeronToRioGrandeHeronSeepage.Supply" [];
    BEGIN

      "RioGrandeHeronToRioGrandeHeronSeepage.Supply" [] := "ReconcileRGOutflow"( % "Heron" );

      $ "HeronData.ReconciledRGOutflow" [] := "ReconcileRGOutflow"( % "Heron" );

    END;

    RULE                 "SetHeronSJAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Heron.Inflow" [] AND NOT IsNaN $ "Heron.Outflow" [] AND
IsNaN "MRGCDHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" [];
    BEGIN

      FOREACH (LIST slotValueRemaining IN "PrioritizedAccountReleases"( % "Heron",
"ReconcileSJOutflow"( % "Heron" ) )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueRemaining ) [] := GET NUMERIC @INDEX
1.00000000 FROM slotValueRemaining;

      ENDFOREACH;

      $ "HeronData.ReconciledSJOutflow" [] := "ReconcileSJOutflow"( % "Heron" );

    END;
```

```
    RULE                 "ComputeElVadoSJRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVadoData.SJOutflow" [];
    BEGIN

      $ "ElVadoData.SJOutflow" [] := WITH NUMERIC ElVadoSJRelease = "TotalElVadoSJRelease"(  ) DO
      IF ( NOT "ElVadoRaftingSeason"(  ) AND "IrrigationSeason"(  ) )
      THEN
         "Min"( "MaximumSJOutflow"( % "ElVado" ), "Max"( $ "ElVadoData.ComputedMinMRGCDRelease"
[] - "RGOutflow"( % "ElVado" ), 0.00000000 ["cfs"] ) + "TotalElVadoFlowThruAccounts"(  ) )
      ELSE
         IF ( ElVadoSJRelease < "Max"( "MinOutflow"( % "ElVado" ) - "RGOutflow"( % "ElVado" ),
0.00000000 ["cfs"] ) AND "VolumeToFlow"( "TotalAccountStorageAvailable"(
"ElVadoAbiquiuCommonStorageAccounts"(  ), % "ElVado", % "Abiquiu" ), @"Current Timestep" ) +
"ElVadoOtowiDebt"(  ) > "Max"( "MinOutflow"( % "ElVado" ) - "RGOutflow"( % "ElVado" ), 0.00000000
["cfs"] ) )
         THEN
            "Max"( "MinOutflow"( % "ElVado" ) - "RGOutflow"( % "ElVado" ), 0.00000000 ["cfs"] )
         ELSE
            IF ( ElVadoSJRelease < 10.00000000 ["cfs"] )
            THEN
               0.00000000 ["cfs"]
            ELSE
               ElVadoSJRelease
            ENDIF
         ENDIF
      ENDIF
   ENDWITH;

    END;

    RULE                 "ZeroWaivers";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "WaiverDate"(  ) + 1.00000000 ["day"] == @"Current Timestep" AND
"PreviousYearIsWaiverYear"(  );
    BEGIN

      FOREACH (STRING account IN "HeronDownstreamWaiverStorageAccounts"(  )) DO
            account CONCAT "HeronToFederalSanJuanHeron.Supply" [] := "VolumeToFlow"(
"PreviousWaiverBalance"( account ), @"Current Timestep" );

      ENDFOREACH;

      FOREACH (STRING account IN "HeronDownstreamWaiverStorageAccounts"(  )) DO
            "HeronData." CONCAT account CONCAT "WaiverBalance" [] := 0.00000000 ["acre-feet"];

      ENDFOREACH;

    END;

    RULE                 "SetWaiverBalance";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "HeronData.AlbuquerqueWaiverBalance" [];
    BEGIN

      FOREACH (STRING account IN "HeronDownstreamWaiverStorageAccounts"(  )) DO
            "HeronData." CONCAT account CONCAT "WaiverBalance" [] := IF ( "WaiversAreInEffect"(
) AND @"Current Timestep" == @"24:00:00 January 1, Current Year" )
   THEN
      "Max"( "PreviousAccountStorage"( account, % "Heron" ) - "FlowToVolume"( "WaiverSupply"(
account, % "ElVado" ), @"Current Timestep" ) - "FlowToVolume"( "WaiverSupply"( account, %
"Abiquiu" ), @"Current Timestep" ), 0.00000000 ["acre-feet"] )
   ELSE
      IF ( "WaiversAreInEffect"(  ) AND account != "Albuquerque" )
      THEN
         "Max"( "PreviousWaiverBalance"( account ) - "FlowToVolume"( "WaiverSupply"( account, %
"ElVado" ), @"Current Timestep" ) - "FlowToVolume"( "WaiverSupply"( account, % "Abiquiu" ),
@"Current Timestep" ), 0.00000000 ["acre-feet"] )
      ELSE
         IF ( "WaiversAreInEffect"(  ) AND account == "Albuquerque" )
         THEN
            "Max"( "PreviousWaiverBalance"( account ) - "FlowToVolume"( "WaiverSupply"( account,
% "ElVado" ), @"Current Timestep" ) - "FlowToVolume"( "WaiverSupply"( account, % "Abiquiu" ),
@"Current Timestep" ) - "FlowToVolume"( "SupplyFlow"(
```

```
"AlbuquerqueHeronToAlbuquerqueHeronMRGCDElVadoHeronSeepage.Supply" ), @"Current Timestep" ),
0.00000000 ["acre-feet"] )
         ELSE
            0.00000000 ["acre-feet"]
         ENDIF
      ENDIF
   ENDIF;

      ENDFOREACH;

    END;

  END;

  POLICY_GROUP   "ElVado ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "ElVadoChannelCapacity";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElVado.Outflow" [] AND $ "ElVado.Outflow" [] >
"MaxOutflow"( % "ElVado" );
    BEGIN

      $ "Indian.Indian Met" [] := "ElVadoIndianMet"(  );

      $ "RioGrandeCompactData.Texas Met" [] := "ElVadoTexasMet"(  );

      $ "ElVado.Outflow" [] := "MaxOutflow"( % "ElVado" );

    END;

    RULE                 "ElVadoFloodControl";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElVado.Pool Elevation" [] AND $ "ElVado.Pool Elevation" []
> "MaxAllowableElevation"( % "ElVado" );
    BEGIN

      $ "ElVado.Outflow" [] := "Max"( "Min"( "ComputeMaxOutflow"( % "ElVado" ), "SolveOutflow"( %
"ElVado", $ "ElVado.Inflow" [], "ElevationToStorage"( % "ElVado", "MaxAllowableElevation"( %
"ElVado" ) ), "PreviousStorage"( % "ElVado" ), @"Current Timestep" ) ), "MinOutflow"( % "ElVado"
) );

    END;

    RULE                 "ElVadoIndianPuebloWaterRight";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      % "Indian" & "Indian Met" [] := IF ( "ElVadoSpringRunoff "(  ) OR
"ElVadoSummerIrrigationSeason"(  ) )
   THEN
      IF ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) <= 0.00000000 ["acre-feet"] )
      THEN
         0.00000000 ["cfs"]
      ELSE
         IF ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) <= "ElVadoIndianStorage"(  ) )
         THEN
            IF ( "IndianCall"(  ) > 0.00000000 ["cfs"] )
            THEN
               "Min"( "Min"( "IndianCall"(  ), "VolumeToFlow"( "IndianStorageRequirement"(  ),
@"Current Timestep" ) ), "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ),
@"Current Timestep" ) )
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    RULE                 "ElVadoRioGrandeCompactVI ";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN % "RioGrandeCompactData" & "Texas Met" [];
```

```
    BEGIN

      % "Indian" & "Indian Met" [] := "ElVadoRGCompactVIIndianMet"(  );

      % "RioGrandeCompactData" & "Texas Met" [] := "ElVadoRGCompactVITexasMet"(  );

    END;

    RULE                 "ElVadoMinimumRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElVado.Outflow" [] AND $ "ElVado.Outflow" [] <
"MinOutflow"( % "ElVado" );
    BEGIN

      $ "ElVado.Outflow" [] := "MinOutflow"( % "ElVado" );

    END;

    RULE                 "ElVadoOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElVado.Outflow" [];
    BEGIN

      $ "ElVado.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "ElVado" ) )
   THEN
      "Max"( "Min"( "RGOutflow"( % "ElVado" ) + "SJOutflow"( % "ElVado" ), "ComputeMaxOutflow"( %
"ElVado" ) ), 0.10000000 ["cfs"] )
   ELSE
      "TotalOutflow"( % "ElVado" )
   ENDIF;

    END;

    RULE                 "SetElVadoRGAccount";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElVado.Inflow" [] AND NOT IsNaN $ "ElVado.Outflow" [] AND
IsNaN $ "RioGrandeElVadoToRioGrandeBlwElVado.Supply" [];
    BEGIN

      $ "RioGrandeElVadoToRioGrandeBlwElVado.Supply" [] := "ReconcileRGOutflow"( % "ElVado" );

      $ "ElVadoData.ReconciledRGOutflow" [] := "ReconcileRGOutflow"( % "ElVado" );

    END;

    RULE                 "SetElVadoSJAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElVado.Inflow" [] AND NOT IsNaN $ "ElVado.Outflow" [] AND
IsNaN
"MRGCDHeronMRGCDMiddleValleyDiversionsElVadoToMRGCDHeronMRGCDMiddleValleyDiversionsBlwElVado.Supp
ly" [];
    BEGIN

      FOREACH (LIST slotValueRemaining IN "PrioritizedAccountReleases"( % "ElVado",
"ReconcileSJOutflow"( % "ElVado" ) )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueRemaining ) [] := GET NUMERIC @INDEX
1.00000000 FROM slotValueRemaining;

      ENDFOREACH;

      $ "ElVadoData.ReconciledSJOutflow" [] := "ReconcileSJOutflow"( % "ElVado" );

    END;

  END;

  POLICY_GROUP   "Abiquiu ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "AbiquiuFloodControl";
    DESCRIPTION          "";
    ACTIVE               TRUE;
```

```
    RULE_EXEC_CONSTRAINT NOT IsNaN "Abiquiu.Outflow" [] AND "ReservoirIsSpillingUnreg"( %
"Abiquiu" );
    BEGIN

      "Abiquiu.Outflow" [] := "AbiquiuFCOutflow"(  );

    END;

    RULE                 "AbiquiuTemporaryFlowsForMaintenance";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT "IsMaintenanceSwitchOn"( % "Abiquiu" ) AND IsNaN $ "Abiquiu.Outflow" [];
    BEGIN

      $ "Abiquiu.Outflow" [] := "MaintenanceFlow"( % "Abiquiu" );

    END;

    RULE                 "AbiquiuSteppedRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN "Abiquiu.Outflow" [] AND NOT IsNaN $
"AbiquiuData.ChannelCapacityRuleHasFired" [] AND "SteppedReleaseIsNeeded"( % "Abiquiu" );
    BEGIN

      "Abiquiu.Outflow" [] := "DetermineSteppedRelease"( % "Abiquiu" );

    END;

    RULE                 "AbiquiuChannelCapacityRestrictions";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN "Abiquiu.Outflow" [];
    BEGIN

      "Abiquiu.Outflow" [] := IF ( "Abiquiu.Outflow" [] - "AbiquiuReleaseForChannelCapacity"(  )
> 1.00000000 ["cfs"] )
   THEN
      "AbiquiuReleaseForChannelCapacity"(  )
   ENDIF;

      $ "AbiquiuData.ChannelCapacityRuleHasFired" [] := 1.00000000;

    END;

    RULE                 "Abiquiu PreEvacuation Rule";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Outflow" [] AND "IsPreEvacuationRequired"(  );
    BEGIN

      $ "Abiquiu.Outflow" [] := "DeterminePreEvacFlow"(  );

    END;

    RULE                 "AbiquiuRGCarryOver";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.RGCarryOverRelease" [] AND
"FloodCarryOverReleaseSeason"(  );
    BEGIN

      $ "AbiquiuData.RGCarryOverRelease" [] := "RGCarryOverRelease"( % "Abiquiu" );

      $ "AbiquiuData.RGCarryOverLeft" [] := "RGCarryOverLeft"( % "Abiquiu" );

    END;

    RULE                 "ComputeAbiquiuRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.RGOutflow" [];
    BEGIN

      $ "AbiquiuData.RGOutflow" [] := IF ( "ComputeAbiquiuRGConsInflow"(  ) > 0.00000000 ["cfs"]
)
   THEN
      "AbiquiuMinRGOutflow"(  ) - "ComputeAbiquiuRGConsInflow"(  )
   ELSE
```

```
      "AbiquiuMinRGOutflow"(  )
   ENDIF;

    END;

    RULE                 "ComputeAbiquiuMinFlowsDemand";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.ComputedMinFlowsDemand" [] AND IsNaN $
"AbiquiuData.MinFlowsDemand" ["DatePlusXTimesteps"( @"Current Timestep", 1.00000000 ["day"] )];
    BEGIN

      $ "AbiquiuData.ComputedMinFlowsDemand" [] := "Max"(
"TotalFlowAtAbiquiuNeededToMeetMinTarget"(  ) - ( ( "AbiquiuMRGCDDemandMetWithSJ"(  ) +
"RGOutflow"( % "Abiquiu" ) ) + "LetterWaterAdjustment"( % "Abiquiu", @"Current Timestep" ) ),
0.00000000 ["cfs"] );

    END;

    RULE                 "SetAbiquiuMinFlowsDemand";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.MinFlowsDemand" ["DatePlusXTimesteps"( @"Current
Timestep", 1.00000000 ["day"] )];
    BEGIN

      FOREACH (DATETIME date IN @"Current Timestep" TO "DatePlusXTimesteps"( @"Current Timestep",
1.00000000 ["day"] )) DO
            $ "AbiquiuData.MinFlowsDemand" [date] := $ "AbiquiuData.ComputedMinFlowsDemand" [];

      ENDFOREACH;

    END;

    RULE                 "AbiquiuMinimumFlows";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Outflow" [] AND $ "Abiquiu.Outflow" [] <= "Max"(
"MinOutflow"( % "Abiquiu" ), "MinRGOutflow"( % "Abiquiu" ) );
    BEGIN

      $ "Abiquiu.Outflow" [] := "Max"( "MinOutflow"( % "Abiquiu" ), "MinRGOutflow"( % "Abiquiu" )
);

      PRINT "Warning, outflow below Abiquiu Dam is less than " CONCAT STRINGIFY "Max"(
"MinOutflow"( % "Abiquiu" ), "MinRGOutflow"( % "Abiquiu" ) ) CONCAT "!";

    END;

    RULE                 "SetAlbuquerqueLoanAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "AlbuquerqueAbiquiuToReclamationAbiquiu.Supply" [] OR IsNaN
"AlbuquerqueAbiquiuToNMISCAbiquiu.Supply" [] OR IsNaN "AlbuquerqueAbiquiuToMRGCDAbiquiu.Supply"
[];
    BEGIN

      FOREACH (LIST slotValueList IN "GetSortedLoanAccounts"( "MakeSupplyList"(
"AlbuquerqueTwelveTimesList"(  ), "PrioritizedAccounts"( % "Abiquiu",
"AlbuquerqueAbiquiuEXAccounts"(  ) ), % "Abiquiu", % "Abiquiu" ), "MakeSupplyList"(
"PrioritizedAccounts"( % "Abiquiu", "AlbuquerqueAbiquiuEXAccounts"(  ) ),
"RGOtowiAlbuquerqueAbiquiuEXAccounts"( "PrioritizedAccounts"( % "Abiquiu",
"AlbuquerqueAbiquiuEXAccounts"(  ) ) ), % "Abiquiu", % "BlwAbiquiu" ), "PrioritizedAccounts"( %
"Abiquiu", "AlbuquerqueAbiquiuEXAccounts"(  ) ), % "Abiquiu" )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueList ) [] := GET NUMERIC @INDEX
1.00000000 FROM slotValueList;

      ENDFOREACH;

      "AlbuquerqueAbiquiuToReclamationAbiquiu.Supply" [] := IF ( "AlbuquerqueWillLoanToAccount"(
"Reclamation" ) )
   THEN
      "VolumeToFlow"( "Min"( "MaximumAlbuquerqueLoan"( "Reclamation" ), "Max"( "FlowToVolume"( $
"AbiquiuData.MinFlowsDemand" [], @"Current Timestep" ) - "PreviousAccountStorage"( "Reclamation",
% "Abiquiu" ), 0.00000000 ["acre-feet"] ) ), @"Current Timestep" )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;
```

```
      "AlbuquerqueAbiquiuToNMISCAbiquiu.Supply" [] := IF ( "AlbuquerqueWillLoanToAccount"(
"NMISC" ) )
   THEN
      "VolumeToFlow"( "GetAccountDebt"( "NMISCAbiquiuToNMISCAbiquiuNMISCJemezBlwAbiquiu.Supply"
), @"Current Timestep" )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

      "AlbuquerqueAbiquiuToMRGCDAbiquiu.Supply" [] := IF ( "AlbuquerqueWillLoanToAccount"(
"MRGCD" ) )
   THEN
      "VolumeToFlow"( "Max"( "FlowToVolume"( "AbiquiuMRGCDDemand"( @"Current Timestep" ),
@"Current Timestep" ) - "PreviousAccountStorage"( "MRGCD", % "Abiquiu" ), 0.00000000 ["acre-
feet"] ), @"Current Timestep" ) + "MRGCDAbiquiuSJDemandLoan"(  )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    RULE                 "AbiquiuLockedIn";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      $ "Abiquiu.Locked In" ["LookAhead"(  )] := IF ( IsNaN $ "Abiquiu.Locked In" ["LookAhead"(
)] )
   THEN
      "LockinCarryOverLookAhead"( % "Abiquiu" )
   ENDIF;

      $ "Abiquiu.Locked In" [] := IF ( IsNaN $ "Abiquiu.Locked In" [] )
   THEN
      "LockinCarryOver"( % "Abiquiu" )
   ENDIF;

    END;

    RULE                 "ComputeAbiquiuSJRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.SJOutflow" [];
    BEGIN

      $ "AbiquiuData.SJOutflow" [] := "AbiquiuSJOutflow"(  );

    END;

    RULE                 "RestrictSJOutflow";
    DESCRIPTION          "";
    ACTIVE               FALSE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Outflow" [];
    BEGIN

      $ "Abiquiu.Outflow" [] := IF ( "RGOutflow"( % "Abiquiu" ) < $
"AbiquiuData.EstimatedAbiquiuToCochitiLosses" [] AND $ "Abiquiu.Outflow" [] > 500.00000000
["cfs"] )
   THEN
      "Min"( "RGOutflow"( % "Abiquiu" ) / 0.17000000, $ "Abiquiu.Outflow" [] )
   ENDIF;

    END;

    RULE                 "AbiquiuOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "Abiquiu.Outflow" [];
    BEGIN

      $ "Abiquiu.Outflow" [] := "InitialAbiquiuOutflow"(  );

    END;

    RULE                 "SetAbiquiuRGConservationAccount";
    DESCRIPTION          "";
    ACTIVE               TRUE;
```

```
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Inflow" [] AND NOT IsNaN $ "Abiquiu.Outflow" [] AND
IsNaN "RioGrandeAbiquiuToRioGrandeConservationAbiquiu.Supply" [];
    BEGIN

      "RioGrandeAbiquiuToRioGrandeConservationAbiquiu.Supply" [] := "ComputeAbiquiuRGConsInflow"(
);

    END;

    RULE                 "SetAbiquiuRGAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Inflow" [] AND NOT IsNaN $ "Abiquiu.Outflow" [] AND
IsNaN "RioGrandeAbiquiuToRioGrandeBlwAbiquiu.Supply" [];
    BEGIN

      "RioGrandeAbiquiuToRioGrandeBlwAbiquiu.Supply" [] := "ReconcileRGOutflow"( % "Abiquiu" );

    END;

    RULE                 "SetAbiquiuSJAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Abiquiu.Inflow" [] AND NOT IsNaN $ "Abiquiu.Outflow" [] AND
IsNaN
"MRGCDElVadoMRGCDMiddleValleyDiversionsAbiquiuToMRGCDElVadoMRGCDMiddleValleyDiversionsBlwAbiquiu.
Supply" [];
    BEGIN

      FOREACH (LIST slotValueRemaining IN "PrioritizedAccountReleases"( % "Abiquiu",
"ReconcileSJOutflow"( % "Abiquiu" ) )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueRemaining ) [] := GET NUMERIC @INDEX
1.00000000 FROM slotValueRemaining;

      ENDFOREACH;

    END;

    RULE                 "ReconciledRGandSJReleases";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "AbiquiuData.ReconciledRGOutflow" [] AND IsNaN $
"AbiquiuData.ReconciledSJOutflow" [];
    BEGIN

      $ "AbiquiuData.ReconciledRGOutflow" [] := "ReconcileRGOutflow"( % "Abiquiu" );

      $ "AbiquiuData.ReconciledSJOutflow" [] := "ReconcileSJOutflow"( % "Abiquiu" );

    END;

  END;

  POLICY_GROUP   "Cochiti ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "CochitiLockedIn";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      $ "Cochiti.Locked In" ["LookAhead"(  )] := IF ( IsNaN $ "Cochiti.Locked In" ["LookAhead"(
)] )
   THEN
      "LockinCarryOverLookAhead"( % "Cochiti" )
   ENDIF;

      $ "Cochiti.Locked In" [] := IF ( IsNaN $ "Cochiti.Locked In" [] )
   THEN
      "LockinCarryOver"( % "Cochiti" )
   ENDIF;

    END;

    RULE                 "CochitiRGCarryOver";
    DESCRIPTION          "";
```

```
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "CochitiData.RGCarryOverRelease" [];
    BEGIN

      $ "CochitiData.RGCarryOverRelease" [] := "RGCarryOverRelease"( % "Cochiti" );

      $ "CochitiData.RGCarryOverLeft" [] := "RGCarryOverLeft"( % "Cochiti" );

    END;

    RULE                  "ComputeCochitiRGRelease";
    DESCRIPTION           "";
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "CochitiData.RGOutflow" [];
    BEGIN

      $ "CochitiData.RGOutflow" [] := "Max"( "CurrentRGInflow"( % "Cochiti" ) + "VolumeToFlow"(
"PreviousAccountGainLoss"( "RioGrande", % "Cochiti" ), @"Current Timestep" ) +
"CochitiRGStorageAdjustment"(  ) + "RGCarryOverRelease"( % "Cochiti" ), "MinRGOutflow"( %
"Cochiti" ) );

    END;

    RULE                  "ComputeCochitiSJRelease";
    DESCRIPTION           "";
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "CochitiData.SJOutflow" [] AND NOT IsNaN $
"CochitiData.RGOutflow" [];
    BEGIN

      $ "CochitiData.SJOutflow" [] := "CochitiSJOutflow"(  );

    END;

    RULE                  "CochitiFloodControl";
    DESCRIPTION           "";
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND "ReservoirIsSpillingUnreg"( %
"Cochiti" );
    BEGIN

      "Cochiti.Outflow" [] := "CochitiFCOutflow"(  );

    END;

    RULE                  "CochitiWCMBalancedRelease";
    DESCRIPTION           "";
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [];
    BEGIN

      $ "Cochiti.Outflow" [] := IF ( ( $ "Cochiti.Outflow" [] - "CochitiBalancedOperation"(  ) >
10.00000000 ["cfs"] OR $ "Jemez.Outflow" [] - "JemezBalancedOperation"(  ) > 10.00000000 ["cfs"]
) AND "IfCochitiJemezBalancedOperation"(  ) )
   THEN
      "CochitiBalancedOperation"(  )
   ENDIF;

      $ "Jemez.Outflow" [] := IF ( ( $ "Cochiti.Outflow" [] - "CochitiBalancedOperation"(  ) >
10.00000000 ["cfs"] OR $ "Jemez.Outflow" [] - "JemezBalancedOperation"(  ) > 10.00000000 ["cfs"]
) AND "IfCochitiJemezBalancedOperation"(  ) )
   THEN
      "JemezBalancedOperation"(  )
   ENDIF;

    END;

    RULE                  "SanMarcialChannelCapacityRule";
    DESCRIPTION           "";
    ACTIVE                TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [] AND
IsNaN $ "CochitiData.MaxReleaseForSanMarcialChannelCap" [];
    BEGIN

      $ "CochitiData.MaxReleaseForSanMarcialChannelCap" [] :=
"CochitiReleaseForSanMarcialChannelCapacity"(  );

    END;
```

```
    RULE                 "CentralChannelCapacityRule";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [] AND
IsNaN $ "CochitiData.MaxReleaseForCentralChannelCap" [];
    BEGIN

      $ "CochitiData.MaxReleaseForCentralChannelCap" [] :=
"CochitiReleaseForCentralChannelCapacity"(  );

    END;

    RULE                 "JemezSanMarcialChannelCapacity";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [] AND
IsNaN $ "JemezData.MaxReleaseForSanMarcialChannelCap" [];
    BEGIN

      $ "JemezData.MaxReleaseForSanMarcialChannelCap" [] := $
"CochitiData.MaxReleaseForSanMarcialChannelCap" [];

    END;

    RULE                 "CochitiSteppedRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [] AND
NOT IsNaN $ "CochitiData.ChannelCapacityRuleHasFired" [] AND "SteppedReleaseIsNeeded"( %
"Cochiti" );
    BEGIN

      $ "Cochiti.Outflow" [] := "DetermineSteppedRelease"( % "Cochiti" );

      $ "Jemez.Outflow" [] := "DetermineSteppedRelease"( % "Jemez" );

    END;

    RULE                 "CochitiChannelCapacityRestrictions";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Jemez.Outflow" [];
    BEGIN

      $ "Cochiti.Outflow" [] := IF ( $ "Cochiti.Outflow" [] + $ "Jemez.Outflow" [] -
"CochitiReleaseForChannelCapacity"(  ) > "ChannelCapacityTolerance"( % "Cochiti",
"SanMarcialFloodway" ) )
   THEN
      "CochitiReleaseForChannelCapacity"(  ) - "JemezFloodRelease"(  )
   ENDIF;

      $ "Jemez.Outflow" [] := IF ( $ "Cochiti.Outflow" [] + $ "Jemez.Outflow" [] -
"CochitiReleaseForChannelCapacity"(  ) > "ChannelCapacityTolerance"( % "Cochiti",
"SanMarcialFloodway" ) )
   THEN
      "JemezFloodRelease"(  )
   ENDIF;

      $ "CochitiData.ChannelCapacityRuleHasFired" [] := 1.00000000;

    END;

    RULE                 "CochitiMinimumFlows";
    DESCRIPTION          "";
    ACTIVE               FALSE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Outflow" [];
    BEGIN

      PRINT IF ( $ "Central.Gage Inflow" [] < "MinimumMonthlyFlows"( % "Cochiti", "Central" ) )
   THEN
      "Warning, outflow below Cochiti Dam is less than " CONCAT STRINGIFY "MinOutflow"( %
"Cochiti" ) CONCAT "!"
   ENDIF;

    END;

    RULE                 "CochitiOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
```

```
    RULE_EXEC_CONSTRAINT IsNaN $ "Cochiti.Outflow" [];
    BEGIN

      $ "Cochiti.Outflow" [] := "InitialCochitiOutflow"(  );

    END;

    RULE                 "SetCochitiRGAccount";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "RioGrandeCochitiToRioGrandeBlwCochitiDiversionsReach.Supply" []
AND NOT IsNaN $ "Cochiti.Outflow" [] AND NOT IsNaN $ "Cochiti.Inflow" [];
    BEGIN

      "RioGrandeCochitiToRioGrandeBlwCochitiDiversionsReach.Supply" [] := "ReconcileRGOutflow"( %
"Cochiti" );

    END;

    RULE                 "SetCochitiSJAccounts";
    DESCRIPTION          "This Rule was changed to make it possible to loan water from the
Cochiti Rec Pool in order to keep the supply for MRGCD whole.
This loaning of water will only occur if MRGCD has enough water in storage to pay the loan back.
It now has the same behavior
as the Rules for the rest of the multicontractor pools.

Brad Vickers, WAVE Engineering Inc.  1/15/01
";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Cochiti.Inflow" [] AND NOT IsNaN $ "Cochiti.Outflow" [] AND
IsNaN
"MRGCDAbiquiuMRGCDMiddleValleyDiversionsCochitiToMRGCDAbiquiuMRGCDMiddleValleyDiversionsBlwCochit
iDiversionsReach.Supply" [];
    BEGIN

      FOREACH (LIST slotValueList IN "PrioritizedAccountReleases"( % "Cochiti",
"ReconcileSJOutflow"( % "Cochiti" ) )) DO
            ( GET STRING @INDEX 0.00000000 FROM slotValueList ) [] := GET NUMERIC @INDEX
1.00000000 FROM slotValueList;

      ENDFOREACH;

    END;

  END;

  POLICY_GROUP   "Jemez ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "SetJemezRGRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Jemez.Inflow" [] AND IsNaN $ "JemezData.RGOutflow" [];
    BEGIN

      $ "JemezData.RGOutflow" [] := "CorpsProjectsRGOutflow"( % "Jemez" );

    END;

    RULE                 "SetJemezSJRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "JemezData.SJOutflow" [];
    BEGIN

      $ "JemezData.SJOutflow" [] := 0.00000000 ["cfs"];

    END;

    RULE                 "JemezOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      $ "Jemez.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "Jemez" ) )
   THEN
```

```
      IF ( "IsNOTFirstTimestep"(  ) AND ( "JemezAccumulatedDeliveryRequestInACRE-FT"(  ) -
"JemezAccumulatedSJCaptureInACRE-FT"(  ) ) > 1.00000000 ["acre-feet"] )
      THEN
         "EstimateJemezOutflowForExchange"(  )
      ELSE
         "RGOutflow"( % "Jemez" ) + "SJOutflow"( % "Jemez" )
      ENDIF
   ELSE
      "TotalOutflow"( % "Jemez" )
   ENDIF;

    END;

    RULE                 "SetJemezAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      "RioGrandeJemezToRioGrandeBlwJemez.Supply" [] := "ReconcileRGOutflow"( % "Jemez" );

      "JemezSedimentPoolJemezToJemezSedimentPoolJemezMRGCDMiddleValleyDiversionsBlwJemez.Supply"
[] := "ReconcileSJOutflow"( % "Jemez" );

    END;

    RULE                 "RGTransfer";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      "RioGrandeJemezToRioGrandeJemezJemezSedimentPoolJemez.Supply" [] := "JemezExchange"(  );

    END;

  END;

  POLICY_GROUP   "Elephant Butte ";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "ElephantButteMaxDesignCapacityExceeded";
    DESCRIPTION          "";
    ACTIVE               FALSE;
    RULE_EXEC_CONSTRAINT TRUE;
    BEGIN

      $ "ElephantButte.Outflow" [] := IF ( "IsMaxCapacityExceeded"( % "ElephantButte" ) )
   THEN
      "GetMaxOutflowGivenInflow"( % "ElephantButte", % "ElephantButte" & "Inflow" [], @"Current
Timestep" )
   ENDIF;

    END;

    RULE                 "ElephantButteOutflowRestrictions";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "ElephantButte.Outflow" [];
    BEGIN

      $ "ElephantButte.Outflow" [] := IF ( $ "ElephantButte.Outflow" [] -
"ElephantButteComputedMaxOutflow"(  ) > 10.00000000 ["cfs"] )
   THEN
      "ElephantButteComputedMaxOutflow"(  )
   ENDIF;

    END;

    RULE                 "ElephantButteOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "ElephantButte.Outflow" [];
    BEGIN

      $ "ElephantButte.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "ElephantButte" ) )
   THEN
```

```
      "DownstreamDemands"( % "ElephantButte" )
   ELSE
      "TotalOutflow"( % "ElephantButte" )
   ENDIF;

    END;

    RULE                 "SetElephantButteAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN "RioGrandeElephantButteToRioGrandeBlwElephantButte.Supply" [] AND
NOT IsNaN $ "ElephantButte.Outflow" [];
    BEGIN

      "RioGrandeElephantButteToRioGrandeBlwElephantButte.Supply" [] := $ "ElephantButte.Outflow"
[];

      "AlbuquerqueElephantButteToAlbuquerqueBlwElephantButte.Supply" [] := 0.00000000 ["cfs"];

    END;

  END;

  POLICY_GROUP   "Caballo Flood Control Rules";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "CaballoFloodControlRelease";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "CaballoData.FloodRelease" [];
    BEGIN

      $ "Caballo.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "Caballo" ) )
   THEN
      IF ( NOT IsNaN $ "Caballo.Outflow" [] AND "IsFloodReleaseRequired"( % "Caballo", $
"Caballo.Inflow" [], $ "Caballo.Outflow" [] ) )
      THEN
         "ComputeCaballoFloodRelease"( % "Caballo", $ "Caballo.Inflow" [], $ "Caballo.Outflow" []
)
      ENDIF
   ELSE
      "TotalOutflow"( % "Caballo" )
   ENDIF;

      $ "CaballoData.FloodRelease" [] := IF ( IsNaN "TotalOutflow"( % "Caballo" ) )
   THEN
      IF ( NOT IsNaN $ "Caballo.Outflow" [] AND "IsFloodReleaseRequired"( % "Caballo", $
"Caballo.Inflow" [], $ "Caballo.Outflow" [] ) AND IsNaN $ "CaballoData.FloodRelease" [] )
      THEN
         "ComputeCaballoFloodRelease"( % "Caballo", $ "Caballo.Inflow" [], $ "Caballo.Outflow" []
)
      ENDIF
   ELSE
      "TotalOutflow"( % "Caballo" )
   ENDIF;

    END;

    RULE                 "CaballoOutflowRestrictions";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT NOT IsNaN $ "Caballo.Outflow" [];
    BEGIN

      $ "Caballo.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "Caballo" ) )
   THEN
      IF ( $ "Caballo.Outflow" [] - "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $
"BlwCaballo.Gage Inflow", $ "CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] ) > $
"CaballoData.ChannelCapacities" ["Tolerance", "ElPaso"] )
      THEN
         "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $ "BlwCaballo.Gage Inflow", $
"CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] )
      ENDIF
   ELSE
      "TotalOutflow"( % "Caballo" )
   ENDIF;
```

```
    END;

    RULE                 "CaballoOutflow";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT IsNaN $ "Caballo.Outflow" [];
    BEGIN

      $ "Caballo.Outflow" [] := IF ( IsNaN "TotalOutflow"( % "Caballo" ) )
   THEN
      "DownstreamDemands"( % "Caballo" )
   ELSE
      "TotalOutflow"( % "Caballo" )
   ENDIF;

    END;

  END;

  POLICY_GROUP   "Accounting Checks";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "OutputTotalAccounts";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" > @"Start Timestep + 2 Timesteps" AND IsNaN $
"AccountingCheck.HeronRGOutflow" [@"Previous Timestep - 1 Timesteps"];
    BEGIN

      $ "AccountingCheck.HeronRGOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "HeronSeepage", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.HeronSJOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "HeronSeepage", "SanJuan", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.HeronTotalOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "HeronSeepage", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" ) + "SumAccountSlotsByWaterType"( % "HeronSeepage", "SanJuan", "Outflow", @"Previous
Timestep - 1 Timesteps" );

      $ "AccountingCheck.ElVadoRGOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwElVado", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.ElVadoSJOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwElVado", "SanJuan", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.ElVadoTotalOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwElVado", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" ) + "SumAccountSlotsByWaterType"( % "BlwElVado", "SanJuan", "Outflow", @"Previous
Timestep - 1 Timesteps" );

      $ "AccountingCheck.AbiquiuRGOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwAbiquiu", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.AbiquiuSJOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwAbiquiu", "SanJuan", "Outflow", @"Previous Timestep - 1
Timesteps" );

      $ "AccountingCheck.AbiquiuTotalOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwAbiquiu", "RioGrande", "Outflow", @"Previous Timestep - 1
Timesteps" ) + "SumAccountSlotsByWaterType"( % "BlwAbiquiu", "SanJuan", "Outflow", @"Previous
Timestep - 1 Timesteps" );

      $ "AccountingCheck.CochitiRGOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwCochitiDiversionsReach", "RioGrande", "Outflow", @"Previous
Timestep - 1 Timesteps" );

      $ "AccountingCheck.CochitiSJOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwCochitiDiversionsReach", "SanJuan", "Outflow", @"Previous
Timestep - 1 Timesteps" );
```

```
      $ "AccountingCheck.CochitiTotalOutflow" [@"Previous Timestep - 1 Timesteps"] :=
"SumAccountSlotsByWaterType"( % "BlwCochitiDiversionsReach", "RioGrande", "Outflow", @"Previous
Timestep - 1 Timesteps" ) + "SumAccountSlotsByWaterType"( % "BlwCochitiDiversionsReach",
"SanJuan", "Outflow", @"Previous Timestep - 1 Timesteps" );

    END;

    RULE                 "ComputeDifference";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" > @"Start Timestep + 2 Timesteps" AND IsNaN $
"AccountingCheck.HeronDifference" [@"Previous Timestep - 1 Timesteps"];
    BEGIN

      $ "AccountingCheck.HeronDifference" [@"Previous Timestep - 1 Timesteps"] := $
"Heron.Outflow" [@"Previous Timestep - 1 Timesteps"] - $ "AccountingCheck.HeronTotalOutflow"
[@"Previous Timestep - 1 Timesteps"] + $ "HeronSeepage.Inflow2" [@"Previous Timestep - 1
Timesteps"];

      $ "AccountingCheck.ElVadoDifference" [@"Previous Timestep - 1 Timesteps"] := $
"ElVado.Outflow" [@"Previous Timestep - 1 Timesteps"] - $ "AccountingCheck.ElVadoTotalOutflow"
[@"Previous Timestep - 1 Timesteps"];

      $ "AccountingCheck.AbiquiuDifference" [@"Previous Timestep - 1 Timesteps"] := $
"Abiquiu.Outflow" [@"Previous Timestep - 1 Timesteps"] - $ "AccountingCheck.AbiquiuTotalOutflow"
[@"Previous Timestep - 1 Timesteps"];

      $ "AccountingCheck.CochitiDifference" [@"Previous Timestep - 1 Timesteps"] := $
"Cochiti.Outflow" [@"Previous Timestep - 1 Timesteps"] - $ "AccountingCheck.CochitiTotalOutflow"
[@"Previous Timestep - 1 Timesteps"] - $ "BlwCochitiDiversionsReach.Diversion" [@"Previous
Timestep - 1 Timesteps"];

    END;

  END;

  POLICY_GROUP   "SummingRules";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  BEGIN

    RULE                 "SumFlows";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"24:00:00 August 1, Current Year";
    BEGIN

      $ "SumFlows.ElVadoLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF (
@"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.ElVadoLocalInflow", @"24:00:00 March 1, Current
Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow",
@"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" )
   ENDIF + "SumFlowsToVolume"( $ "AzoteaWillow.Inflow2", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" );

      $ "SumFlows.Otowi" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF ( @"24:00:00
March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.Otowi", @"24:00:00 March 1, Current Year",
@"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "Otowi.Gage Inflow", @"Start Timestep",
@"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "Otowi.Gage Inflow", @"24:00:00 March 1, Current Year", @"24:00:00
July 31, Current Year" )
   ENDIF;

      $ "SumFlows.Lobatos" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF ( @"24:00:00
March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.Lobatos", @"24:00:00 March 1, Current Year",
@"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "Lobatos.Gage Inflow", @"Start Timestep",
@"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "Lobatos.Gage Inflow", @"24:00:00 March 1, Current Year", @"24:00:00
July 31, Current Year" )
```

```
   ENDIF;

      $ "SumFlows.LobatosToCerroLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"]
:= IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.LobatosToCerroLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $
"LobatosToCerroLocalInflow.Local Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "LobatosToCerroLocalInflow.Local Inflow", @"24:00:00 March 1, Current
Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.RedRiverBlwFishHatchery" [@"24:00:00 August 1, Current Year - 1 Timesteps"] :=
IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.RedRiverBlwFishHatchery", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "RedRiverBlwFishHatchery.Gage
Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "RedRiverBlwFishHatchery.Gage Inflow", @"24:00:00 March 1, Current
Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.CerroToTaosLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"] :=
IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.CerroToTaosLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "CerroToTaosLocalInflow.Local
Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "CerroToTaosLocalInflow.Local Inflow", @"24:00:00 March 1, Current
Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.RioPuebloDeTaosAtLosCordovas" [@"24:00:00 August 1, Current Year - 1
Timesteps"] := IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.RioPuebloDeTaosAtLosCordovas", @"24:00:00 March
1, Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $
"RioPuebloDeTaosAtLosCordovas.Gage Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year"
)
   ELSE
      "SumFlowsToVolume"( $ "RioPuebloDeTaosAtLosCordovas.Gage Inflow", @"24:00:00 March 1,
Current Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.TaosToEmbudoLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"] :=
IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.TaosToEmbudoLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "TaosToEmbudoLocalInflow.Local
Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "TaosToEmbudoLocalInflow.Local Inflow", @"24:00:00 March 1, Current
Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.ElVadoToAbiquiuLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"]
:= IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.ElVadoToAbiquiuLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $
"ElVadoToAbiquiuLocalInflow.Local Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "ElVadoToAbiquiuLocalInflow.Local Inflow", @"24:00:00 March 1,
Current Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.AbiquiuToChamitaLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"]
:= IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.AbiquiuToChamitaLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $
"AbiquiuToChamitaLocalInflow.Local Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year"
)
   ELSE
      "SumFlowsToVolume"( $ "AbiquiuToChamitaLocalInflow.Local Inflow", @"24:00:00 March 1,
Current Year", @"24:00:00 July 31, Current Year" )
```

```
   ENDIF;

      $ "SumFlows.EmbudoToOtowiLocalInflow" [@"24:00:00 August 1, Current Year - 1 Timesteps"] :=
IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.EmbudoToOtowiLocalInflow", @"24:00:00 March 1,
Current Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $
"EmbudoToOtowiLocalInflow.Local Inflow", @"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "EmbudoToOtowiLocalInflow.Local Inflow", @"24:00:00 March 1, Current
Year", @"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.EmbudoCreekAtDixon" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF (
@"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.EmbudoCreekAtDixon", @"24:00:00 March 1, Current
Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "EmbudoCreekAtDixon.Gage Inflow",
@"Start Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "EmbudoCreekAtDixon.Gage Inflow", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.Embudo" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF ( @"24:00:00
March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.Embudo", @"24:00:00 March 1, Current Year",
@"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "Embudo.Gage Inflow", @"Start Timestep",
@"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "Embudo.Gage Inflow", @"24:00:00 March 1, Current Year", @"24:00:00
July 31, Current Year" )
   ENDIF;

      $ "SumFlows.OtowiRG" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF ( @"24:00:00
March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.OtowiRG", @"24:00:00 March 1, Current Year",
@"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "Otowi^RioGrande.Inflow", @"Start Timestep",
@"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "Otowi^RioGrande.Inflow", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.ChangeInElVadoRGStorage" [@"24:00:00 August 1, Current Year - 1 Timesteps"] :=
IF ( @"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      $ "ElVado^RioGrande.Storage" [@"24:00:00 August 1, Current Year - 1 Timesteps"] - $
"SumFlowsHistorical.ChangeInElVadoRGStorage" [@"24:00:00 March 1, Current Year"]
   ELSE
      $ "ElVado^RioGrande.Storage" [@"24:00:00 August 1, Current Year - 1 Timesteps"] - $
"ElVado^RioGrande.Storage" [@"24:00:00 March 1, Current Year"]
   ENDIF;

      $ "SumFlows.SanMarcial" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF (
@"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.SanMarcial", @"24:00:00 March 1, Current Year",
@"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "SanMarcialFloodway.Gage Inflow", @"Start
Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "SanMarcialFloodway.Gage Inflow", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" )
   ENDIF;

      $ "SumFlows.SanMarcialLFCC" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := IF (
@"24:00:00 March 1, Current Year" < @"Start Timestep" )
   THEN
      "SumFlowsToVolume"( $ "SumFlowsHistorical.SanMarcialLFCC", @"24:00:00 March 1, Current
Year", @"Start Timestep - 1 Day" ) + "SumFlowsToVolume"( $ "SanMarcialLFCC.Gage Inflow", @"Start
Timestep", @"24:00:00 July 31, Current Year" )
   ELSE
      "SumFlowsToVolume"( $ "SanMarcialLFCC.Gage Inflow", @"24:00:00 March 1, Current Year",
@"24:00:00 July 31, Current Year" )
   ENDIF;

    END;
```

```
    RULE                 "SumOtowi";
    DESCRIPTION          "";
    ACTIVE               TRUE;
    RULE_EXEC_CONSTRAINT @"Current Timestep" == @"24:00:00 August 1, Current Year";
    BEGIN

      $ "SumFlows.OtowiRGTotal" [@"24:00:00 August 1, Current Year - 1 Timesteps"] := $
"SumFlows.OtowiRG" [@"24:00:00 August 1, Current Year - 1 Timesteps"] + $
"SumFlows.ChangeInElVadoRGStorage" [@"24:00:00 August 1, Current Year - 1 Timesteps"];

    END;

  END;

  UTILITY_GROUP "Abiquiu Lockin Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuConstantRGCarryOverRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( NOT IsNaN $ "AbiquiuData.RGCarryOverRelease" [@"Previous Timestep"] AND $
"AbiquiuData.RGCarryOverRelease" [@"Previous Timestep"] > 0.00000000 ["cfs"] )
   THEN
      "Min"( $ "AbiquiuData.RGCarryOverRelease" [@"Previous Timestep"], "VolumeToFlow"( $
"Abiquiu^RioGrande.Storage" [@"Previous Timestep"], @"Current Timestep" ) )
   ELSE
      "Min"( $ "Abiquiu^RioGrande.Storage" [@"Previous Timestep"] / "DeltaTimeToMarch"(  ),
"VolumeToFlow"( $ "Abiquiu^RioGrande.Storage" [@"Previous Timestep"], @"Current Timestep" ) )
   ENDIF;

    END;

    FUNCTION       "AbiquiuLockinCarryOver+5" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep + 5 Timesteps" >= @"July" AND @"Current Timestep + 5 Timesteps" <=
@"October" AND $ "ElVadoLocalInflow.Local Inflow" [@"Current Timestep + 5 Timesteps"] + $
"Embudo.Gage Inflow" [@"Current Timestep + 5 Timesteps"] < 1500.00000000 ["cfs"] )
   THEN
      IF ( $ "Abiquiu.Locked In" [@"Current Timestep + 4 Timesteps"] == 1.00000000 )
      THEN
         1.00000000
      ELSE
         IF ( $ "Abiquiu^RioGrande.Storage" [@"Current Timestep - 1 Timesteps"] + "FlowToVolume"(
"SumSlotOverTime"( "ElVadoLocalInflow.Local Inflow", @"Current Timestep", @"Current Timestep + 5
Timesteps" ) - ( 3.00000000 * $ "Abiquiu^RioGrande.Outflow" [@"Current Timestep - 1 Timesteps"]
), @"Current Timestep" ) > 0.00000000 ["acre-feet"] AND $ "CochitiData.FloodSpace" [@"Current
Timestep + 4 Timesteps"] > $ "CochitiData.MinFloodSpace" [0.00000000, 0.00000000] AND $
"Abiquiu^RioGrande.Storage" [@"Current Timestep - 1 Timesteps"] > 1000.00000000 ["acre-feet"] )
         THEN
            1.00000000
         ELSE
            0.00000000
         ENDIF
      ENDIF
   ELSE
      0.00000000
   ENDIF;

    END;

    FUNCTION       "AbiquiuRGCarryOverRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( @"Current Timestep" >= @"November" OR @"Current Timestep" <= @"March" ) AND ( $
"Abiquiu^RioGrande.Storage" [@"Previous Timestep"] > 0.00000000 ["acre-feet"] ) )
   THEN
      IF ( ( NOT IsNaN $ "AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"] AND $
"AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"] > 0.00000000 ["acre-feet"] ) OR ( $
"Abiquiu^RioGrande.Storage" [@"Previous Timestep"] > "AbiquiuData.MaxRGStorage" [0.00000000,
0.00000000] AND IsNaN $ "AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"] ) OR ( NOT IsNaN $
"Abiquiu.Carryover Content" [@"24:00:00 October 31, Current Year"] AND $ "Abiquiu.Carryover
Content" [@"24:00:00 October 31, Current Year"] > 0.00000000 ["acre-feet"] ) )
      THEN
         IF ( IsNaN $ "AbiquiuData.RGCarryOverRelease" [@"Current Timestep"] )
         THEN
            "AbiquiuConstantRGCarryOverRelease"(  )
         ELSE
            $ "AbiquiuData.RGCarryOverRelease" [@"Current Timestep"]
         ENDIF
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Abiquiu Account Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuRGOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "CorpsProjectsRGOutflow"( % "Abiquiu" );

    END;

    FUNCTION       "AbiquiuSJOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( $ "AbiquiuData.MaximumSJOutflow" ["SJOutflowRow"( % "Abiquiu", @"Current Timestep"
), 1.00000000], "TotalPotentialDestinationRelease"( % "Abiquiu" ) );

    END;

    FUNCTION       "AbiquiuRGStorageAdjustment" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "FloodCarryOverReleaseSeason"(  ) AND $ "AbiquiuData.RGCarryOverLeft" [@"Previous
Timestep"] > 10.00000000 ["acre-feet"] )
   THEN
      IF ( $ "Abiquiu.Incidental Content" [@"Previous Timestep"] - $
"AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"] < 0.00000000 ["acre-feet"] )
```

```
      THEN
         "Min"( "Abs"( "VolumeToFlow"( $ "Abiquiu.Incidental Content" [@"Previous Timestep"] - $
"AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"], @"Current Timestep" ) ),
"MaxIncidentalContentRelease"( % "Abiquiu" ) ) * - 1.00000000
      ELSE
         "Min"( "VolumeToFlow"( $ "Abiquiu.Incidental Content" [@"Previous Timestep"] - $
"AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"], @"Current Timestep" ),
"MaxIncidentalContentRelease"( % "Abiquiu" ) )
      ENDIF
   ELSE
      IF ( $ "Abiquiu.Incidental Content" [@"Previous Timestep"] < 0.00000000 ["acre-feet"] )
      THEN
         "Min"( "Abs"( "VolumeToFlow"( $ "Abiquiu.Incidental Content" [@"Previous Timestep"],
@"Current Timestep" ) ), "MaxIncidentalContentRelease"( % "Abiquiu" ) ) * - 1.00000000
      ELSE
         "Min"( "VolumeToFlow"( $ "Abiquiu.Incidental Content" [@"Previous Timestep"], @"Current
Timestep" ), "MaxIncidentalContentRelease"( % "Abiquiu" ) )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "AlbuquerquePaybackSupply" ( STRING account )
    RETURN_TYPE    STRING;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( account IN "ElVadoRootWaiverStorageAccounts"(  ) )
   THEN
      GET STRING @INDEX 0.00000000 FROM "MakeSupplyList"( { account },
"MakeAlbuquerqueAbiquiuPaybackAccountsList"( { account }, % "ElVado" ), % "ElVado", % "BlwElVado"
)
   ELSE
      GET STRING @INDEX 0.00000000 FROM "MakeSupplyList"( { account },
"MakeAlbuquerqueAbiquiuPaybackAccountsList"( { account }, % "Heron" ), % "Heron", %
"HeronSeepage" )
   ENDIF;

    END;

    FUNCTION       "FlowAbiquiuRGStorageMinusCarryOver" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "VolumeToFlow"( $ "Abiquiu^RioGrande.Storage" [@"Previous Timestep"] - $
"AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"], @"Current Timestep" );

    END;

    FUNCTION       "MaximumAlbuquerqueLoan" ( STRING account )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( account == "MRGCD" )
   THEN
      IF ( "Max"( "SumAccountStorageUpstreamOfAbiquiu"( { account } ) * ( 1.00000000 - "SJCLoss"(
% "ElVado", % "Abiquiu" ) ) - ( "GetAccountDebt"( "AlbuquerquePaybackSupply"( account ) ) +
"FlowToVolume"( "AlbuquerquePaybackSupply"( account ) [], @"Current Timestep" ) ) +
"RemainingMRGCDFromAlbuquerquePurchase"(  ), 0.00000000 ["acre-feet"] ) < 0.50000000 ["acre-
feet"] )
      THEN
         0.00000000 ["acre-feet"]
      ELSE
         "Max"( "SumAccountStorageUpstreamOfAbiquiu"( { account } ) * ( 1.00000000 - "SJCLoss"( %
"ElVado", % "Abiquiu" ) ) - ( "GetAccountDebt"( "AlbuquerquePaybackSupply"( account ) ) +
```

```
"FlowToVolume"( "AlbuquerquePaybackSupply"( account ) [], @"Current Timestep" ) ) +
"RemainingMRGCDFromAlbuquerquePurchase"(  ), 0.00000000 ["acre-feet"] )
      ENDIF
   ELSE
      IF ( "Max"( "SumAccountStorageUpstreamOfAbiquiu"( { account } ) * ( 1.00000000 - "SJCLoss"(
% "ElVado", % "Abiquiu" ) ) - ( "GetAccountDebt"( "AlbuquerquePaybackSupply"( account ) ) +
"FlowToVolume"( "AlbuquerquePaybackSupply"( account ) [], @"Current Timestep" ) ), 0.00000000
["acre-feet"] ) < 0.50000000 ["acre-feet"] )
      THEN
         0.00000000 ["acre-feet"]
      ELSE
         "Max"( "SumAccountStorageUpstreamOfAbiquiu"( { account } ) * ( 1.00000000 - "SJCLoss"( %
"ElVado", % "Abiquiu" ) ) - ( "GetAccountDebt"( "AlbuquerquePaybackSupply"( account ) ) +
"FlowToVolume"( "AlbuquerquePaybackSupply"( account ) [], @"Current Timestep" ) ), 0.00000000
["acre-feet"] )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "NoRGCarryoverInAbiquiu" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IsNaN $ "AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"];

    END;

    FUNCTION       "RGCarryoverInAbiquiu" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      NOT IsNaN $ "AbiquiuData.RGCarryOverLeft" [@"Previous Timestep"];

    END;

    FUNCTION       "EstimateAbiquiuRGloss" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( @"Current Timestep" == @"Start Timestep" )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "VolumeToFlow"( $ "Abiquiu^RioGrande.Gain Loss" [@"Previous Timestep"], @"Current Timestep"
)
   ENDIF;

    END;

    FUNCTION       "AbiquiuMRGCDDemand" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "AbiquiuData.MRGCDDemand" [date];

    END;

    FUNCTION       "MaxAbiquiuMRGCDSJRelease" (  )
```

```
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( ( "Max"( $ "Abiquiu.Inflow" [] - "CurrentRGInflow"( % "Abiquiu" ) -
"SumPreviousSupplies"( "AllElVadoAccountFillandPaybackSupplies"(  ) ) * ( 1.00000000 - "SJCLoss"(
% "ElVado", % "Abiquiu" ) ) - "SumPreviousSupplies"( "MakeSupplyList"(
"ElVadoThruCochitiFlowthruAccounts"(  ), "ElVadoThruCochitiFlowthruAccounts"(  ), % "ElVado", %
"BlwElVado" ) ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ), 0.00000000 ["cfs"] ) +
"AlbuquerqueAbiquiuToMRGCDAbiquiu.Supply" [] ) + "Max"( "VolumeToFlow"( "PreviousAccountStorage"(
"MRGCD", % "Abiquiu" ), @"Current Timestep" ), 0.00000000 ["cfs"] ) + "RGOutflow"( % "Abiquiu" )
+ "LetterWaterAdjustment"( % "Abiquiu", @"Current Timestep" ), "AbiquiuMRGCDDemand"( @"Current
Timestep" ) );

    END;

    FUNCTION       "AbiquiuMinFlowsSJRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "SlotInflow"( "AlbuquerqueAbiquiuToReclamationAbiquiu.Supply" ) + "VolumeToFlow"(
"PreviousAccountStorage"( "RioGrandeConservation", % "Abiquiu" ), @"Current Timestep" ) +
"VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", % "Abiquiu" ), @"Current Timestep" ), $
"AbiquiuData.MinFlowsDemand" [] );

    END;

    FUNCTION       "MaxDeliveryRequestCheck" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH NUMERIC amountStorageTransfers = "SumIntraReservoirTransfers"( "MakeSupplyList"(
"AlbuquerqueFiveTimesList"(  ), "AbiquiuAlbuquerqueLoanAccounts"(  ), % "Abiquiu", % "Abiquiu" )
) DO
      "Min"( "SumIntraReservoirTransfers"( "MakeSupplyList"( "AlbuquerqueFiveTimesList"(  ),
"AbiquiuAlbuquerqueLoanAccounts"(  ), % "Abiquiu", % "Abiquiu" ) ), "MaxDeliveryRequestRelease"(
) )
   ENDWITH;

    END;

    FUNCTION       "AbiquiuMinRGOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "CurrentRGInflow"( % "Abiquiu" ) + "VolumeToFlow"( "PreviousAccountGainLoss"(
"RioGrande", % "Abiquiu" ), @"Current Timestep" ) + "AbiquiuRGStorageAdjustment"(  ) +
"RGCarryOverRelease"( % "Abiquiu" ), $ "AbiquiuData.MinRGDemandRelease" [] );

    END;

    FUNCTION       "AbiquiuAlbuquerqueMRGCDLoan" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      IF ( "AlbuquerqueWillLoanToAccount"( "MRGCD" ) )
   THEN
      "VolumeToFlow"( "Max"( "FlowToVolume"( "AbiquiuMRGCDDemand"( @"Current Timestep" ),
@"Current Timestep" ) - "PreviousAccountStorage"( "MRGCD", % "Abiquiu" ), 0.00000000 ["acre-
feet"] ), @"Current Timestep" ) + "MRGCDAbiquiuSJDemandLoan"(  )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "AbiquiuMRGCDDemandMetWithSJ" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Min"( "MRGCDAbiquiuSJDemand"(  ), "AbiquiuAlbuquerqueMRGCDLoan"(  ) + "VolumeToFlow"(
"PreviousAccountStorage"( "MRGCD", % "Abiquiu" ), @"Current Timestep" ) + "SumPreviousSupplies"(
"AbiquiuMRGCDInflowSupplies"(  ) ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ) +
"SumSupplies"( "AbiquiuMRGCDOtherInflowSupplies"(  ) ) + "LetterWaterAdjustment"( % "Abiquiu",
@"Current Timestep" ) );

    END;

    FUNCTION       "InitialAbiquiuOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Abiquiu" );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      IF ( IsNaN "TotalOutflow"( % "Abiquiu" ) )
   THEN
      IF ( "IrrigationSeason"(  ) )
      THEN
         IF ( "PreviousOutflow"( % "ElVado" ) - "SumPreviousSupplies"(
"AllElVadoAccountFillandPaybackSupplies"(  ) ) >= $ "ElVadoData.MRGCDDemand" [@"Previous
Timestep"] )
         THEN
            "Max"( "AbiquiuMiddleValleyDemand"(  ), "Max"( "RGOutflow"( % "Abiquiu" ) +
"PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiElVadoToCochitiRecPoolHeronCochitiRecPoolCochitiBlwElVad
o.Supply" ) + "AbiquiuMinFlowsSJRelease"(  ), "MinOutflow"( % "Abiquiu" ) ) )
         ELSE
            "Max"( "Min"( "AbiquiuMiddleValleyDemand"(  ), "SJOutflow"( % "Abiquiu" ) +
"RGOutflow"( % "Abiquiu" ) ), "MinOutflow"( % "Abiquiu" ) )
         ENDIF
      ELSE
         "Max"( "SJOutflow"( % "Abiquiu" ) + "RGOutflow"( % "Abiquiu" ), "MinOutflow"( %
"Abiquiu" ) )
      ENDIF
   ELSE
      "TotalOutflow"( % "Abiquiu" )
   ENDIF;

    END;

    FUNCTION       "TotalFlowAtAbiquiuNeededToMeetMinTarget" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "AbiquiuData.TotalFlowNeededToMeetTarget" [];

    END;

  END;
```

```
  UTILITY_GROUP "Abiquiu Channel Capacity Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuChamitaChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "Abiquiu.Outflow" [] - ( $ "Chamita.Gage Inflow" [] - $ "AbiquiuData.ChannelCapacities"
["Capacity", "Chamita"] );

    END;

    FUNCTION       "AbiquiuOtowiChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "Abiquiu.Outflow" [] - ( $ "Otowi.Gage Inflow" [] - $ "AbiquiuData.ChannelCapacities"
["Capacity", "Otowi"] );

    END;

    FUNCTION       "AbiquiuReleaseForChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "This function returns the minimum of the three releases determined to not
exceed channel capacities below Abiquiu Dam.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "Min"( "AbiquiuReleaseForDSChannelCapacity"(  ),
"AbiquiuReleaseForChamitaChannelCapacity"(  ) ), "AbiquiuReleaseForOtowiChannelCapacity"(  ) );

    END;

    FUNCTION       "AbiquiuReleaseForDSChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function returns the release from Abiquiu to not exceed the channel
capacity below Abiquiu.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "AbiquiuData.ChannelCapacities" ["Capacity", "D/S"];

    END;

    FUNCTION       "AbiquiuReleaseForChamitaChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function returns the release from Abiquiu to not exceed the channel
capactiy at Chamita.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( ( $ "AbiquiuToChamitaLocalInflow.Local Inflow" [] > $ "AbiquiuData.ChannelCapacities"
["Capacity", "Chamita"] ) OR ( "AbiquiuChamitaChannelCapacity"(  ) < $
"AbiquiuData.ChannelCapacities" ["Capacity", "MinimumChamita"] ) )
   THEN
      $ "AbiquiuData.ChannelCapacities" ["Capacity", "MinimumChamita"]
   ELSE
```

```
      "AbiquiuChamitaChannelCapacity"(  )
   ENDIF;

    END;

    FUNCTION       "AbiquiuReleaseForOtowiChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function returns the release from Abiquiu to not exceed the channel
capacity at Otowi.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( ( $ "EmbudoToOtowiLocalInflow.Local Inflow" [] > $ "AbiquiuData.ChannelCapacities"
["Capacity", "Otowi"] ) OR ( "EmbudoLagAndLoss"(  ) + $ "EmbudoToOtowiLocalInflow.Local Inflow"
[] > $ "AbiquiuData.ChannelCapacities" ["Capacity", "Otowi"] ) OR (
"AbiquiuOtowiChannelCapacity"(  ) < $ "AbiquiuData.ChannelCapacities" ["Capacity",
"MinimumChamita"] ) )
   THEN
      $ "AbiquiuData.ChannelCapacities" ["Capacity", "MinimumChamita"]
   ELSE
      "AbiquiuOtowiChannelCapacity"(  )
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Abiquiu Flood Control Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuFCOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Abiquiu" );
    BEGIN

      IF ( "IsAbiquiuUnregulatedSpill<ChannelCapacity"(  ) )
   THEN
      "AbiquiuData.ChannelCapacities" [0.00000000, 0.00000000]
   ELSE
      "UnregulatedSpillWhenNoConduitFlow"( % "Abiquiu" )
   ENDIF;

    END;

    FUNCTION       "IsAbiquiuUnregulatedSpill<ChannelCapacity" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "UnregulatedSpillWhenNoConduitFlow"( % "Abiquiu" ) < "AbiquiuData.ChannelCapacities"
[0.00000000, 0.00000000];

    END;

  END;

  UTILITY_GROUP "AbiquiuPreEvacuationFunctions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuRemainingForecastVolume" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN $ "ForecastData.PercentForecastError" [] )
   THEN
      "SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow", @"Current Timestep", @"24:00:00
July 31, Current Year" ) + "SumFlowsToVolume"( $ "ElVadoToAbiquiuLocalInflow.Local Inflow",
@"Current Timestep", @"24:00:00 July 31, Current Year" ) - ( "ElevationToStorage"( % "ElVado", $
"ElVadoData.MaxAllowableElevation" ["March", "ElVado"] ) - "ElevationToStorage"( % "ElVado", $
"ElVado.Pool Elevation" [@"Previous Timestep"] ) ) - ( "ElevationToStorage"( % "Abiquiu", $
"AbiquiuData.PreEvacuationData" ["Abiquiu", "AllowableStorageElevation"] ) -
"ElevationToStorage"( % "Abiquiu", $ "Abiquiu.Pool Elevation" [@"Previous Timestep"] ) )
   ELSE
      ( "SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow", @"Current Timestep", @"24:00:00
July 31, Current Year" ) + "SumFlowsToVolume"( $ "ElVadoToAbiquiuLocalInflow.Local Inflow",
@"Current Timestep", @"24:00:00 July 31, Current Year" ) ) * ( 1.00000000 + $
"ForecastData.PercentForecastError" [] ) - ( "ElevationToStorage"( % "ElVado", $
"ElVadoData.MaxAllowableElevation" ["March", "ElVado"] ) - "ElevationToStorage"( % "ElVado", $
"ElVado.Pool Elevation" [@"Previous Timestep"] ) ) - ( "ElevationToStorage"( % "Abiquiu", $
"AbiquiuData.PreEvacuationData" ["Abiquiu", "AllowableStorageElevation"] ) -
"ElevationToStorage"( % "Abiquiu", $ "Abiquiu.Pool Elevation" [@"Previous Timestep"] ) )
   ENDIF;

    END;

    FUNCTION       "ComputePreEvacFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "AbiquiuRemainingForecastVolume"(  ) / ( @"24:00:00 August 1, Current Year" - @"Current
Timestep" );

    END;

    FUNCTION       "DeterminePreEvacFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ComputePreEvacFlow"(  ) >= $ "AbiquiuData.ChannelCapacities" ["Capacity", "D/S"] )
   THEN
      $ "AbiquiuData.ChannelCapacities" ["Capacity", "D/S"]
   ELSE
      IF ( "ComputePreEvacFlow"(  ) >= $ "AbiquiuData.SteppedReleaseData" ["Abiquiu",
"MaxThreshold"] + $ "AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxStepFactor"] )
      THEN
         $ "AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxThreshold"] + $
"AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxStepFactor"]
      ELSE
         IF ( "ComputePreEvacFlow"(  ) >= $ "AbiquiuData.SteppedReleaseData" ["Abiquiu",
"MaxThreshold"] )
         THEN
            $ "AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxThreshold"]
         ELSE
            IF ( "ComputePreEvacFlow"(  ) >= $ "AbiquiuData.SteppedReleaseData" ["Abiquiu",
"MaxThreshold"] - $ "AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxStepFactor"] )
            THEN
               $ "AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxThreshold"] - $
"AbiquiuData.SteppedReleaseData" ["Abiquiu", "MaxStepFactor"]
            ELSE
               $ "Abiquiu.Outflow" []
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;
```

```
    FUNCTION       "IsPreEvacuationRequired" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "IsPreEvacuationSeason"(  ) AND "AbiquiuRemainingForecastVolume"(  ) >= $
"AbiquiuData.PreEvacuationData" ["Abiquiu", "MinPreEvacStorage"];

    END;

    FUNCTION       "IsPreEvacuationSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" >= "OffsetDate"( @"24:00:00 December 31, Previous Year", $
"AbiquiuData.PreEvacuationData" ["Abiquiu", "PreEvacJulianDay"], "1 days" ) AND @"Current
Timestep" <= @"24:00:00 April 30, Current Year";

    END;

  END;

  UTILITY_GROUP "Account Lists";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuAlbuquerqueLoanAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "LosLunas" , "NambeFalls" , "RedRiver" , "Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "AbiquiuFlowthruAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" , "AlbuquerqueHeronRGOtowi" ,
"AlbuquerqueHeronAlbuquerqueEB" , "AlbuquerqueElVadoAlbuquerqueEB" , "BelenHeronRGOtowi" ,
"BernalilloHeronRGOtowi" , "CochitiRecPoolHeronCochitiRecPoolCochiti" , "EspanolaHeronRGOtowi" ,
"EspanolaElVadoRGOtowi" , "LosAlamosElVadoRGOtowi" , "LosAlamosHeronRGOtowi" ,
"LosLunasHeronRGOtowi" , "LosLunasElVadoRGOtowi" , "MRGCDHeronMRGCDMiddleValleyDiversions" ,
"NambeFallsHeronRGOtowi" , "NMISCHeronNMISCJemez" , "RedRiverHeronRGOtowi" ,
"SantaFeHeronRGOtowi" , "SantaFeElVadoRGOtowi" , "TaosHeronRGOtowi" , "TaosElVadoRGOtowi" ,
"TwiningHeronRGOtowi" , "TwiningElVadoRGOtowi" , "UncontractedHeronRGOtowi" };

    END;

    FUNCTION       "AbiquiuOtowiPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      { "Albuquerque" , "Bernalillo" , "Espanola" , "LosAlamos" , "SantaFe" , "Taos" , "Twining"
, "Belen" , "LosLunas" , "NambeFalls" , "RedRiver" , "Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "AbiquiuInitialSortList" ( STRING releaseType, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( releaseType == "NMISC" )
   THEN
      { { "NMISC" } , { "NMISCAbiquiuToNMISCAbiquiuNMISCJemezBlwAbiquiu.Supply" } , % "Abiquiu" ,
% "Abiquiu" , releaseType , available }
   ELSE
      IF ( releaseType == "OtowiPaybacks" )
      THEN
         { "PrioritizedAccounts"( % "Abiquiu", "AbiquiuOtowiPaybackAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Abiquiu", "AbiquiuOtowiPaybackAccounts"(  ) ),
"MakeOtowiPaybackAccountList"( % "Abiquiu", "PrioritizedAccounts"( % "Abiquiu",
"AbiquiuOtowiPaybackAccounts"(  ) ) ), % "Abiquiu", % "BlwAbiquiu" ) , % "Abiquiu" , % "Abiquiu"
, releaseType , available }
      ELSE
         IF ( releaseType == "AlbuquerqueLoan" )
         THEN
            { "PrioritizedAccounts"( % "Abiquiu", "AbiquiuAlbuquerqueLoanAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Abiquiu", "AbiquiuAlbuquerqueLoanAccounts"(  ) ),
"MakeOtowiPaybackAccountList"( % "Abiquiu", "PrioritizedAccounts"( % "Abiquiu",
"AbiquiuAlbuquerqueLoanAccounts"(  ) ) ), % "Abiquiu", % "BlwAbiquiu" ) , % "Abiquiu" , %
"Abiquiu" , releaseType , available }
         ELSE
            IF ( releaseType == "FlowThrough" )
            THEN
               { "AbiquiuFlowthruAccounts"(  ) , "MakeSupplyList"( "AbiquiuFlowthruAccounts"(  ),
"AbiquiuFlowthruAccounts"(  ), % "Abiquiu", % "BlwAbiquiu" ) , % "Abiquiu" , % "Abiquiu" ,
releaseType , available }
            ELSE
               IF ( releaseType == "MRGCD" )
               THEN
                  { { "MRGCD" } , {
"MRGCDAbiquiuToMRGCDAbiquiuMRGCDMiddleValleyDiversionsBlwAbiquiu.Supply" } , % "Abiquiu" , %
"Abiquiu" , releaseType , available }
               ELSE
                  IF ( releaseType == "Reclamation" )
                  THEN
                     { "PrioritizedAccounts"( % "Abiquiu", "AbiquiuMinFlowAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Abiquiu", "AbiquiuMinFlowAccounts"(  ) ), {
"RioGrande" , "RioGrande" }, % "Abiquiu", % "BlwAbiquiu" ) , % "Abiquiu" , % "Abiquiu" ,
releaseType , available }
                  ELSE
                     IF ( releaseType == "Albuquerque" )
                     THEN
                        { { "Albuquerque" } , {
"AlbuquerqueAbiquiuToAlbuquerqueAbiquiuAlbuquerqueMiddleValleyDiversionsBlwAbiquiu.Supply" } , %
"Abiquiu" , % "Abiquiu" , releaseType , available }
                     ELSE
                        {  }
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "AbiquiuReclamationLeaseList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      WITH LIST accounts = "AbiquiuWaterPurchaseAccounts"(  ) DO
      WITH LIST supplies = "MakeSupplyList"( accounts, "ReclamationEightTimesList"(  ), %
"Abiquiu", % "Abiquiu" ) DO
         FOR ( STRING supply IN supplies ) WITH LIST result = {  } DO
            IF ( ( GET STRING @INDEX LENGTH result FROM accounts ) ==
"AlbuquerqueAbiquiuReclamationAbiquiu" )
            THEN
               APPEND { "AlbuquerqueAbiquiuToAlbuquerqueAbiquiuReclamationAbiquiuAbiquiu.Supply"
, "AccountLeaseAmount"( "Albuquerque", % "Abiquiu" ) } ONTO result
            ELSE
               APPEND { supply , "AccountLeaseAmount"( GET STRING @INDEX LENGTH result FROM
accounts, % "Abiquiu" ) } ONTO result
            ENDIF
         ENDFOR
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "AbiquiuRootStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Bernalillo" , "Espanola" , "LosAlamos" , "MRGCD" , "Taos" , "SantaFe" ,
"Twining" , "Reclamation" };

    END;

    FUNCTION       "AbiquiuEXStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      { "Espanola" , "LosAlamos" , "SantaFe" , "Taos" , "Twining" };

    END;

    FUNCTION       "AbiquiuRootWaiverStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Bernalillo" , "Espanola" , "LosAlamos" , "MRGCD" , "Taos" , "SantaFe" ,
"Twining" };

    END;

    FUNCTION       "AlbuquerqueAbiquiuEXAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "Bernalillo" , "Espanola" , "LosAlamos" , "LosLunas" , "NambeFalls" ,
"RedRiver" , "SantaFe" , "Taos" , "Twining" , "Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "AbiquiuMinFlowAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Reclamation" , "RioGrandeConservation" };

    END;

    FUNCTION       "AlbuquerqueFiveTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" };

    END;

    FUNCTION       "AllAbiquiuSJSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      {
"AlbuquerqueHeronAlbuquerqueMiddleValleyDiversionsAbiquiuToAlbuquerqueHeronAlbuquerqueMiddleValle
yDiversionsBlwAbiquiu.Supply" ,
"AlbuquerqueHeronAlbuquerqueEBAbiquiuToAlbuquerqueHeronAlbuquerqueEBBlwAbiquiu.Supply" ,
"AlbuquerqueElVadoAlbuquerqueEBAbiquiuToAlbuquerqueElVadoAlbuquerqueEBBlwAbiquiu.Supply" ,
"CochitiRecPoolHeronCochitiRecPoolCochitiAbiquiuToCochitiRecPoolHeronCochitiRecPoolCochitiBlwAbiq
uiu.Supply" ,
"MRGCDElVadoMRGCDMiddleValleyDiversionsAbiquiuToMRGCDElVadoMRGCDMiddleValleyDiversionsBlwAbiquiu.
Supply" ,
"MRGCDHeronMRGCDMiddleValleyDiversionsAbiquiuToMRGCDHeronMRGCDMiddleValleyDiversionsBlwAbiquiu.Su
pply" , "NMISCHeronNMISCJemezAbiquiuToNMISCHeronNMISCJemezBlwAbiquiu.Supply" ,
"MRGCDAbiquiuToMRGCDAbiquiuMRGCDMiddleValleyDiversionsBlwAbiquiu.Supply" ,
"ReclamationAbiquiuToReclamationAbiquiuMRGCDMiddleValleyDiversionsBlwAbiquiu.Supply" ,
"AlbuquerqueAbiquiuToAlbuquerqueAbiquiuAlbuquerqueEBBlwAbiquiu.Supply" ,
"AlbuquerqueAbiquiuToAlbuquerqueAbiquiuAlbuquerqueMiddleValleyDiversionsBlwAbiquiu.Supply" ,
"CochitiRecPoolAbiquiuToCochitiRecPoolAbiquiuCochitiRecPoolCochitiBlwAbiquiu.Supply" ,
"NMISCAbiquiuToNMISCAbiquiuNMISCJemezBlwAbiquiu.Supply" };

    END;

    FUNCTION       "BlwAbiquiuOtowiPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN "AbiquiuOtowiPaybackAccounts"(  ) ) WITH LIST result = {  } DO
      APPEND account CONCAT "AbiquiuRGOtowi" ONTO result
   ENDFOR;

    END;

    FUNCTION       "CochitiFlowThroughAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueAbiquiuAlbuquerqueMiddleValleyDiversions" , "AlbuquerqueAbiquiuAlbuquerqueEB"
, "AlbuquerqueElVadoAlbuquerqueEB" , "AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" ,
"AlbuquerqueHeronAlbuquerqueEB" , "MRGCDAbiquiuMRGCDMiddleValleyDiversions" ,
"MRGCDElVadoMRGCDMiddleValleyDiversions" , "MRGCDHeronMRGCDMiddleValleyDiversions" ,
```

```
"NMISCAbiquiuNMISCJemez" , "NMISCHeronNMISCJemez" ,
"ReclamationAbiquiuMRGCDMiddleValleyDiversions" };

    END;

    FUNCTION       "CochitiMRGCDFlowThroughAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDAbiquiuMRGCDMiddleValleyDiversions" , "MRGCDElVadoMRGCDMiddleValleyDiversions" ,
"MRGCDHeronMRGCDMiddleValleyDiversions" };

    END;

    FUNCTION       "CochitiInitialSortList" ( STRING releaseType, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( releaseType == "FlowThrough" )
   THEN
      { "CochitiFlowThroughAccounts"(  ) , "MakeSupplyList"( "CochitiFlowThroughAccounts"(  ),
"CochitiFlowThroughAccounts"(  ), % "Cochiti", % "BlwCochitiDiversionsReach" ) , % "Cochiti" , %
"Cochiti" , releaseType , available }
   ELSE
      {  }
   ENDIF;

    END;

    FUNCTION       "DownstreamStorageAccounts" ( LIST accounts, OBJECT upstreamReservoir, OBJECT
downstreamReservoir )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT STRINGIFY upstreamReservoir CONCAT account CONCAT STRINGIFY
downstreamReservoir ONTO result
   ENDFOR;

    END;

    FUNCTION       "ElVadoAbiquiuCommonStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Espanola" , "LosAlamos" , "MRGCD" , "SantaFe" , "Taos" , "Twining" , "Albuquerque" ,
"Reclamation" };

    END;

    FUNCTION       "ElVadoFlowthruAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      { "AlbuquerqueHeronAlbuquerqueAbiquiu" ,
"AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" , "AlbuquerqueHeronAlbuquerqueEB" ,
"AlbuquerqueHeronRGOtowi" , "BelenHeronAlbuquerqueAbiquiu" , "BelenHeronRGOtowi" ,
"BernalilloHeronAlbuquerqueAbiquiu" , "BernalilloHeronBernalilloAbiquiu" ,
"BernalilloHeronRGOtowi" , "CochitiRecPoolHeronAlbuquerqueAbiquiu" ,
"CochitiRecPoolHeronCochitiRecPoolCochiti" , "EspanolaHeronAlbuquerqueAbiquiu" ,
"EspanolaHeronEspanolaAbiquiu" , "EspanolaHeronRGOtowi" , "LosAlamosHeronAlbuquerqueAbiquiu" ,
"LosAlamosHeronLosAlamosAbiquiu" , "LosAlamosHeronRGOtowi" , "LosLunasHeronAlbuquerqueAbiquiu" ,
"LosLunasHeronRGOtowi" , "MRGCDHeronAlbuquerqueAbiquiu" , "MRGCDHeronMRGCDAbiquiu" ,
"MRGCDHeronMRGCDMiddleValleyDiversions" , "NambeFallsHeronAlbuquerqueAbiquiu" ,
"NambeFallsHeronRGOtowi" , "RedRiverHeronAlbuquerqueAbiquiu" , "RedRiverHeronRGOtowi" ,
"SantaFeHeronRGOtowi" , "SantaFeHeronSantaFeAbiquiu" , "TaosHeronAlbuquerqueAbiquiu" ,
"TaosHeronRGOtowi" , "TaosHeronTaosAbiquiu" , "TwiningHeronAlbuquerqueAbiquiu" ,
"TwiningHeronRGOtowi" , "TwiningHeronTwiningAbiquiu" , "UncontractedHeronAlbuquerqueAbiquiu" ,
"UncontractedHeronRGOtowi" , "ReclamationHeronReclamationAbiquiu" ,
"ReclamationHeronAlbuquerqueAbiquiu" , "SantaFeHeronAlbuquerqueAbiquiu" ,
"ReclamationHeronMRGCDAbiquiu" , "AlbuquerqueHeronMRGCDAbiquiu" };

    END;

    FUNCTION       "ElVadoOtowiEXAccountList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      { "Espanola" , "LosAlamos" , "LosLunas" , "SantaFe" , "Taos" , "Twining" };

    END;

    FUNCTION       "ElVadoInitialSortList" ( STRING releaseType, NUMERIC remainingAvailable )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( releaseType == "AccountFill" )
   THEN
      { "PrioritizedAccounts"( % "ElVado", "ElVadoAbiquiuCommonWaiverStorageAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "ElVado", "ElVadoAbiquiuCommonWaiverStorageAccounts"(
) ), "MakeDestinationAccountList"( "PrioritizedAccounts"( % "ElVado",
"ElVadoAbiquiuCommonWaiverStorageAccounts"(  ) ), "PrioritizedAccounts"( % "ElVado",
"ElVadoAbiquiuCommonWaiverStorageAccounts"(  ) ), % "ElVado", % "Abiquiu" ), % "ElVado", %
"BlwElVado" ) , % "ElVado" , % "Abiquiu" , releaseType , remainingAvailable }
   ELSE
      IF ( releaseType == "OtowiPaybacks" )
      THEN
         { "PrioritizedAccounts"( % "ElVado", "ElVadoRootOtowiPaybackAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "ElVado", "ElVadoRootOtowiPaybackAccounts"(  ) ),
"MakeOtowiPaybackAccountList"( % "ElVado", "PrioritizedAccounts"( % "ElVado",
"ElVadoRootOtowiPaybackAccounts"(  ) ) ), % "ElVado", % "BlwElVado" ) , % "ElVado" , % "Abiquiu"
, releaseType , remainingAvailable }
      ELSE
         IF ( releaseType == "AlbuquerquePaybacks" )
         THEN
            { "PrioritizedAccounts"( % "ElVado", "ElVadoRootAlbuquerquePaybackAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "ElVado", "ElVadoRootAlbuquerquePaybackAccounts"(  )
), "MakeAlbuquerqueAbiquiuPaybackAccountsList"( "PrioritizedAccounts"( % "ElVado",
"ElVadoRootAlbuquerquePaybackAccounts"(  ) ), % "ElVado" ), % "ElVado", % "BlwElVado" ) , %
"ElVado" , % "Abiquiu" , releaseType , remainingAvailable }
         ELSE
            IF ( releaseType == "FlowThrough" )
            THEN
               { "ElVadoFlowthruAccounts"(  ) , "MakeSupplyList"( "ElVadoFlowthruAccounts"(  ),
"ElVadoFlowthruAccounts"(  ), % "ElVado", % "BlwElVado" ) , % "ElVado" , % "Abiquiu" ,
releaseType , remainingAvailable }
            ELSE
               IF ( releaseType == "MRGCDPaybacks" )
               THEN
                  { { "Reclamation" } , {
"ReclamationElVadoToReclamationElVadoMRGCDAbiquiuBlwElVado.Supply" } , % "ElVado" , % "Abiquiu" ,
releaseType , remainingAvailable }
               ELSE
```

```
                  IF ( releaseType == "Reclamation" )
                  THEN
                     { { "Reclamation" } , {
"ReclamationElVadoToReclamationElVadoReclamationAbiquiuBlwElVado.Supply" } , % "ElVado" , %
"Abiquiu" , releaseType , remainingAvailable }
                  ELSE
                     IF ( releaseType == "MRGCD" )
                     THEN
                        { { "MRGCD" } , {
"MRGCDElVadoToMRGCDElVadoMiddleValleyDiversionsBlwElVado.Supply" } , % "ElVado" , % "Abiquiu" ,
releaseType , remainingAvailable }
                     ELSE
                        {  }
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ElVadoReclamationLeaseList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST accounts = "ElVadoRootWaiverStorageAccounts"(  ) DO
      WITH LIST supplies = "MakeSupplyList"( "ElVadoRootWaiverStorageAccounts"(  ),
"ReclamationEightTimesList"(  ), % "ElVado", % "ElVado" ) DO
         FOR ( STRING supply IN supplies ) WITH LIST result = {  } DO
            APPEND { supply , "AccountLeaseAmount"( GET STRING @INDEX LENGTH result FROM
accounts, % "ElVado" ) } ONTO result
         ENDFOR
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "ElVadoStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCD" , "SantaFe" , "LosAlamos" , "Espanola" , "Taos" , "LosLunas" , "Twining" ,
"Albuquerque" , "Reclamation" };

    END;

    FUNCTION       "ElVadoRootWaiverStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCD" , "SantaFe" , "LosAlamos" , "Espanola" , "Taos" , "LosLunas" , "Twining" ,
"Albuquerque" };

    END;

    FUNCTION       "ElVadoRootOtowiPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
```

```
    BEGIN

      { "SantaFe" , "LosAlamos" , "Espanola" , "Taos" , "LosLunas" , "Twining" };

    END;

    FUNCTION       "ElVadoRootAlbuquerquePaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCD" , "SantaFe" , "LosAlamos" , "Espanola" , "Taos" , "LosLunas" , "Twining" };

    END;

    FUNCTION       "GetSortedPaybackAccounts" ( LIST accounts, OBJECT reservoir, LIST supplies,
NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      IF ( ( GET STRING @INDEX LENGTH alloc FROM accounts ) == "NMISC" )
      THEN
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "Min"( "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM supplies ),
@"Current Timestep" ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000
["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ), "VolumeToFlow"( "PreviousAccountStorage"( "Albuquerque", reservoir ),
@"Current Timestep" ) ) ) } ONTO alloc
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Abiquiu" AND ( GET STRING @INDEX LENGTH alloc
FROM accounts ) != "Albuquerque" )
         THEN
            APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "Min"( "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM supplies ),
@"Current Timestep" ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000
["cfs"] DO
               sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
            ENDFOR ), "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc
FROM accounts, reservoir ), @"Current Timestep" ) + "SupplyFlow"( "Albuquerque" CONCAT
"StringifyObject"( reservoir ) CONCAT "To" CONCAT ( GET STRING @INDEX LENGTH alloc FROM accounts
) CONCAT "StringifyObject"( reservoir ) CONCAT ".Supply" ) ) ) } ONTO alloc
         ELSE
            APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "Min"( "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM supplies ),
@"Current Timestep" ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000
["cfs"] DO
               sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
            ENDFOR ), "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc
FROM accounts, reservoir ), @"Current Timestep" ) ) ) } ONTO alloc
         ENDIF
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedMinFlowAccounts" ( LIST accounts, OBJECT reservoir, LIST supplies,
NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      IF ( ( GET STRING @INDEX LENGTH alloc FROM accounts ) == "Reclamation" )
      THEN
```

```
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", reservoir ), @"Current Timestep"
) + "AlbuquerqueAbiquiuToReclamationAbiquiu.Supply" [], available - FOR ( LIST alloc2 IN alloc )
WITH NUMERIC sum = 0.00000000 ["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ) ) } ONTO alloc
      ELSE
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts,
reservoir ), @"Current Timestep" ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum =
0.00000000 ["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ) ) } ONTO alloc
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedLoanAccounts" ( LIST upstreamSupplies, LIST downstreamSupplies, LIST
accounts, OBJECT reservoir )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH upstreamSupplies ) WITH LIST alloc = {  } DO
      IF ( "MaximumAlbuquerqueLoan"( GET STRING @INDEX LENGTH alloc FROM accounts ) +
"PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, % "Abiquiu" ) >=
"GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM downstreamSupplies ) )
      THEN
         IF ( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, reservoir )
> 0.00000000 ["acre-feet"] )
         THEN
            APPEND { GET STRING @INDEX LENGTH alloc FROM upstreamSupplies , "Min"( "Min"( "Max"(
"VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM downstreamSupplies ) -
"PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current
Timestep" ), 0.00000000 ["cfs"] ), "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH
alloc FROM downstreamSupplies ), @"Current Timestep" ) ), "MaxDeliveryRequestRelease"(  ) - FOR (
LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000 ["cfs"] DO
               sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
            ENDFOR ) } ONTO alloc
         ELSE
            APPEND { GET STRING @INDEX LENGTH alloc FROM upstreamSupplies , "Min"(
"VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM downstreamSupplies ),
@"Current Timestep" ), "MaxDeliveryRequestRelease"(  ) - FOR ( LIST alloc2 IN alloc ) WITH
NUMERIC sum = 0.00000000 ["cfs"] DO
               sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
            ENDFOR ) } ONTO alloc
         ENDIF
      ELSE
         APPEND { GET STRING @INDEX LENGTH alloc FROM upstreamSupplies , "Min"( "VolumeToFlow"(
"MaximumAlbuquerqueLoan"( GET STRING @INDEX LENGTH alloc FROM accounts ), @"Current Timestep" ),
"MaxDeliveryRequestRelease"(  ) - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000
["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ) } ONTO alloc
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedFlowThruAccounts" ( LIST accounts, OBJECT reservoir, LIST
upstreamSupplies, LIST downstreamSupplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH downstreamSupplies ) WITH LIST alloc = {  } DO
      IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
      THEN
         APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"( 0.00000000
["cfs"], "Min"( "PreviousSlotInflow"( GET STRING @INDEX LENGTH alloc FROM upstreamSupplies ) * (
```

```
1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ) + "VolumeToFlow"( "PreviousAccountStorage"(
GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current Timestep" ), available - FOR
( LIST slotValueList IN alloc ) WITH NUMERIC total = 0.00000000 ["cfs"] DO
            total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
         ENDFOR ) ) } ONTO alloc
      ELSE
         IF ( "StringifyObject"( reservoir ) == "ElVado" )
         THEN
            APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"( 0.00000000
["cfs"], "Min"( "SumSupplies"( "ElVadoFlowThroughAccountSupplies"( GET STRING @INDEX LENGTH alloc
FROM accounts ) ) + "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM
accounts, reservoir ), @"Current Timestep" ), available - FOR ( LIST slotValueList IN alloc )
WITH NUMERIC total = 0.00000000 ["cfs"] DO
               total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
            ENDFOR ) ) } ONTO alloc
         ELSE
            alloc
         ENDIF
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedAlbuquerqueLoanAccounts" ( LIST accounts, OBJECT reservoir, LIST
supplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"], "Min"(
"VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, reservoir
), @"Current Timestep" ) + "SupplyFlow"( "Albuquerque" CONCAT "StringifyObject"( reservoir )
CONCAT "To" CONCAT ( GET STRING @INDEX LENGTH alloc FROM accounts ) CONCAT "StringifyObject"(
reservoir ) CONCAT ".Supply" ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum =
0.00000000 ["cfs"] DO
         sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
      ENDFOR ) ) } ONTO alloc
   ENDWHILE;

    END;

    FUNCTION       "GetSortedMRGCDDeliveryAccounts" ( LIST accounts, OBJECT upstreamReservoir,
OBJECT downstreamReservoir, LIST supplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      IF ( "CurrentYearWaiverSwitch"(  ) == 0.00000000 )
      THEN
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "Min"( "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM
accounts, upstreamReservoir ), @"Current Timestep" ), "VolumeToFlow"( "AvailableAccountStorage"(
"MRGCD", % "ElVado" ), @"Current Timestep" ) * ( 1.00000000 + "SJCLoss"( upstreamReservoir,
downstreamReservoir ) ) ), available - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000
["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ) ) } ONTO alloc
      ELSE
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , 0.00000000 ["cfs"] } ONTO alloc
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedReclamationMRGCDLoans" ( NUMERIC loan )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST loansElVado = { "MRGCDElVadoReclamationHeronEX.Borrow" ,
"MRGCDElVadoReclamationElVadoEX.Borrow" , "MRGCDAbiquiuReclamationAbiquiuEX.Borrow" } DO
      WITH LIST loansAbiquiu = { "MRGCDAbiquiuReclamationHeronEX.Borrow" ,
"MRGCDAbiquiuReclamationElVadoEX.Borrow" , "MRGCDAbiquiuReclamationAbiquiuEX.Borrow" } DO
         WITH LIST suppliesElVado = {
"ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" ,
"ReclamationElVadoToMRGCDElVado.Supply" , "ReclamationAbiquiuToMRGCDAbiquiu.Supply" } DO
            WITH LIST suppliesAbiquiu = {
"ReclamationHeronToReclamationHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"ReclamationElVadoToReclamationElVadoMRGCDAbiquiuBlwElVado.Supply" ,
"ReclamationAbiquiuToMRGCDAbiquiu.Supply" } DO
               WITH LIST reservoirs = { % "Heron" , % "ElVado" , % "Abiquiu" } DO
                  IF ( "ElVadoIsPriority"(  ) )
                  THEN
                     WHILE ( ( LENGTH alloc ) < LENGTH loansElVado ) WITH LIST alloc = {  } DO
                        APPEND { GET STRING @INDEX LENGTH alloc FROM loansElVado , "Min"( "Max"(
"VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", GET OBJECT @INDEX LENGTH alloc FROM
reservoirs ) - "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM suppliesElVado ), @"Current
Timestep" ), 0.00000000 ["cfs"] ), "VolumeToFlow"( loan, @"Current Timestep" ) - FOR ( LIST
alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000 ["cfs"] DO
                           sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
                        ENDFOR ) } ONTO alloc
                     ENDWHILE
                  ELSE
                     WHILE ( ( LENGTH alloc ) < LENGTH loansAbiquiu ) WITH LIST alloc = {  } DO
                        IF ( "StringifyObject"( GET OBJECT @INDEX LENGTH alloc FROM reservoirs )
== "Heron" OR "StringifyObject"( GET OBJECT @INDEX LENGTH alloc FROM reservoirs ) == "ElVado" )
                        THEN
                           APPEND { GET STRING @INDEX LENGTH alloc FROM loansAbiquiu , "Min"(
"Max"( "VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", GET OBJECT @INDEX LENGTH alloc
FROM reservoirs ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ) - "GetAccountDebt"( GET
STRING @INDEX LENGTH alloc FROM suppliesAbiquiu ), @"Current Timestep" ), 0.00000000 ["cfs"] ),
"VolumeToFlow"( loan, @"Current Timestep" ) - FOR ( LIST alloc2 IN alloc ) WITH NUMERIC sum =
0.00000000 ["cfs"] DO
                              sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
                           ENDFOR ) } ONTO alloc
                        ELSE
                           APPEND { GET STRING @INDEX LENGTH alloc FROM loansAbiquiu , "Min"(
"Max"( "VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", GET OBJECT @INDEX LENGTH alloc
FROM reservoirs ) - "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM suppliesAbiquiu ),
@"Current Timestep" ), 0.00000000 ["cfs"] ), "VolumeToFlow"( loan, @"Current Timestep" ) - FOR (
LIST alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000 ["cfs"] DO
                              sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
                           ENDFOR ) } ONTO alloc
                        ENDIF
                     ENDWHILE
                  ENDIF
               ENDWITH
            ENDWITH
         ENDWITH
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "GetSortedStorageAccounts" ( LIST accounts, OBJECT upstreamReservoir, OBJECT
downstreamReservoir, LIST supplies, STRING releaseType, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      IF ( ( STRINGIFY upstreamReservoir ) == "Heron" AND releaseType == "AccountDelivery" AND
"CurrentYearIsWaiverYear"(  ) )
      THEN
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , 0.00000000 ["cfs"] } ONTO alloc
      ELSE
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"],
"Min"( "Min"( "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM
accounts, upstreamReservoir ), @"Current Timestep" ), "VolumeToFlow"( "AvailableAccountStorage"(
GET STRING @INDEX LENGTH alloc FROM accounts, downstreamReservoir ), @"Current Timestep" ) * (
```

```
1.00000000 + "SJCLoss"( upstreamReservoir, downstreamReservoir ) ) ), available - FOR ( LIST
alloc2 IN alloc ) WITH NUMERIC sum = 0.00000000 ["cfs"] DO
            sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
         ENDFOR ) ) } ONTO alloc
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "GetSortedWaiverAccounts" ( LIST accounts, OBJECT upstreamReservoir, OBJECT
downstreamReservoir, LIST supplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"], "Min"(
"Min"( "VolumeToFlow"( "PreviousWaiverBalance"( GET STRING @INDEX LENGTH alloc FROM accounts ),
@"Current Timestep" ), "VolumeToFlow"( "AvailableAccountStorage"( GET STRING @INDEX LENGTH alloc
FROM accounts, downstreamReservoir ), @"Current Timestep" ) * ( 1.00000000 + "SJCLoss"(
upstreamReservoir, downstreamReservoir ) ) ), available - FOR ( LIST alloc2 IN alloc ) WITH
NUMERIC sum = 0.00000000 ["cfs"] DO
         sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
      ENDFOR ) ) } ONTO alloc
   ENDWHILE;

    END;

    FUNCTION       "GetSortedAccounts" ( LIST list )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST accounts = GET LIST @INDEX 0.00000000 FROM list DO
      WITH LIST supplies = GET LIST @INDEX 1.00000000 FROM list DO
         WITH OBJECT upstreamReservoir = GET OBJECT @INDEX 2.00000000 FROM list DO
            WITH OBJECT downstreamReservoir = GET OBJECT @INDEX 3.00000000 FROM list DO
               WITH STRING releaseType = GET STRING @INDEX 4.00000000 FROM list DO
                  WITH NUMERIC available = GET NUMERIC @INDEX 5.00000000 FROM list DO
                     IF ( releaseType == "AccountDelivery" OR releaseType == "AccountFill" )
                     THEN
                        "GetSortedStorageAccounts"( accounts, upstreamReservoir,
downstreamReservoir, supplies, releaseType, available )
                     ELSE
                        IF ( releaseType == "ReleaseToMRGCD" )
                        THEN
                           "GetSortedMRGCDDeliveryAccounts"( accounts, upstreamReservoir,
downstreamReservoir, supplies, available )
                        ELSE
                           IF ( releaseType == "Waiver" )
                           THEN
                              "GetSortedWaiverAccounts"( accounts, upstreamReservoir,
downstreamReservoir, supplies, available )
                           ELSE
                              IF ( releaseType == "FlowThrough" AND "StringifyObject"(
upstreamReservoir ) == "ElVado" )
                              THEN
                                 "GetSortedFlowThruAccounts"( "ElVadoFlowthruAccounts"(  ), %
"ElVado", {  }, supplies, available )
                              ELSE
                                 IF ( releaseType == "FlowThrough" AND "StringifyObject"(
upstreamReservoir ) == "Abiquiu" )
                                 THEN
                                    "GetSortedFlowThruAccounts"( "AbiquiuFlowthruAccounts"(  ), %
"Abiquiu", "MakeSupplyList"( "AbiquiuFlowthruAccounts"(  ), "AbiquiuFlowthruAccounts"(  ), %
"BlwElVado", % "BlwElVadoToAbvAbiquiu" ), supplies, available )
                                 ELSE
                                    IF ( releaseType == "FlowThrough" AND "StringifyObject"(
upstreamReservoir ) == "Cochiti" )
                                    THEN
```

```
                                        "GetSortedCochitiFlowThroughAccounts"( accounts, %
"Cochiti", "MakeSupplyList"( accounts, accounts, % "BlwAbiquiu", % "BlwAbiquiuDiversionsReach" ),
supplies, available )
                                     ELSE
                                        IF ( releaseType == "CochitiRecPool" )
                                        THEN
                                           { { GET STRING @INDEX 0.00000000 FROM supplies , "Max"(
"Min"( "HeronCochitiRecPoolRelease"(  ), available ), 0.00000000 ["cfs"] ) } }
                                        ELSE
                                           IF ( releaseType == "AlbuquerquePaybacks" )
                                           THEN
                                              "GetSortedAlbuquerquePaybackAccounts"( accounts,
upstreamReservoir, downstreamReservoir, supplies, available )
                                           ELSE
                                              IF ( releaseType == "AlbuquerqueLoan" )
                                              THEN
                                                 "GetSortedAlbuquerqueLoanAccounts"( accounts,
upstreamReservoir, supplies, available )
                                              ELSE
                                                 IF ( releaseType == "Reclamation" AND ( STRINGIFY
upstreamReservoir ) == "ElVado" )
                                                 THEN
                                                    { { GET STRING @INDEX 0.00000000 FROM supplies
, "Max"( "Min"( "ElVadoReclamationRelease"(  ), available ), 0.00000000 ["cfs"] ) } }
                                                 ELSE
                                                    IF ( releaseType == "MRGCD" AND ( STRINGIFY
upstreamReservoir ) == "Abiquiu" )
                                                    THEN
                                                       { { GET STRING @INDEX 0.00000000 FROM
supplies , "MinItem"( { "VolumeToFlow"( "PreviousAccountStorage"( "MRGCD", % "Abiquiu" ),
@"Current Timestep" ) + "SumSupplies"( { "AlbuquerqueAbiquiuToMRGCDAbiquiu.Supply" ,
"ReclamationAbiquiuToMRGCDAbiquiu.Supply" } ) + "SumPreviousSupplies"(
"AllElVadoToMRGCDAbiquiuSupplies"(  ) ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ) ,
"Max"( $ "AbiquiuData.MRGCDDemand" [] - "RioGrandeAbiquiuToRioGrandeBlwAbiquiu.Supply" [] -
"LetterWaterAdjustment"( % "Abiquiu", @"Current Timestep" ), 0.00000000 ["cfs"] ) , available } )
} }
                                                    ELSE
                                                       IF ( releaseType == "Reclamation" AND (
STRINGIFY upstreamReservoir ) == "Abiquiu" )
                                                       THEN
                                                          "GetSortedMinFlowAccounts"( accounts,
upstreamReservoir, supplies, "Min"( $ "AbiquiuData.MinFlowsDemand" [], available ) )
                                                       ELSE
                                                          IF ( releaseType == "Albuquerque" AND (
STRINGIFY upstreamReservoir ) == "Abiquiu" )
                                                          THEN
                                                             { { GET STRING @INDEX 0.00000000 FROM
supplies , "Min"( 0.00000000 ["cfs"], available ) } }
                                                          ELSE
                                                             IF ( releaseType == "MRGCD" AND (
STRINGIFY upstreamReservoir ) == "ElVado" )
                                                             THEN
                                                                { { GET STRING @INDEX 0.00000000
FROM supplies , "MinItem"( { "Max"( $ "ElVadoData.ComputedMinMRGCDRelease" [] -
"RioGrandeElVadoToRioGrandeBlwElVado.Supply" [], 0.00000000 ["cfs"] ) , "VolumeToFlow"(
"PreviousAccountStorage"( "MRGCD", % "ElVado" ), @"Current Timestep" ) , available } ) } }
                                                             ELSE
                                                                "GetSortedPaybackAccounts"(
accounts, upstreamReservoir, supplies, available )
                                                             ENDIF
                                                          ENDIF
                                                       ENDIF
                                                    ENDIF
                                                 ENDIF
                                              ENDIF
                                           ENDIF
                                        ENDIF
                                     ENDIF
                                  ENDIF
                               ENDIF
                            ENDIF
                         ENDIF
                      ENDIF
                   ENDWITH
                ENDWITH
             ENDWITH
          ENDWITH
       ENDWITH
    ENDWITH;
```

```
    END;

    FUNCTION       "HeronDownStreamReservoirs" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "ElVado" , "Abiquiu" };

    END;

    FUNCTION       "HeronInitialSortList" ( STRING releaseType, STRING downstreamReservoir,
NUMERIC remainingAvailable )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( releaseType == "Reclamation" )
   THEN
      { "PrioritizedAccounts"( % "Heron", "HeronToReclamationAccountsList"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "HeronToReclamationAccountsList"(  ) ),
"ReclamationFourteenTimesList"(  ), % "Heron", % "HeronSeepage" ) , % "Heron" , % "Abiquiu" ,
releaseType , remainingAvailable }
   ELSE
      IF ( releaseType == "OtowiPaybacks" )
      THEN
         { "PrioritizedAccounts"( % "Heron", "HeronRootOtowiPaybackAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "HeronRootOtowiPaybackAccounts"(  ) ),
"MakeOtowiPaybackAccountList"( % "Heron", "PrioritizedAccounts"( % "Heron",
"HeronRootOtowiPaybackAccounts"(  ) ) ), % "Heron", % "HeronSeepage" ) , % "Heron" , % "Abiquiu"
, releaseType , remainingAvailable }
      ELSE
         IF ( releaseType == "AccountDelivery" AND downstreamReservoir == "Abiquiu" )
         THEN
            { "PrioritizedAccounts"( % "Heron", "AbiquiuRootStorageAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "AbiquiuRootStorageAccounts"(  ) ),
"PrioritizedReleaseTypeAccounts"( "AbiquiuRootStorageAccounts"(  ), "AccountDelivery", % "Heron",
% "Abiquiu" ), % "Heron", % "Heron" ) , % "Heron" , "ObjectifyString"( downstreamReservoir ) ,
releaseType , remainingAvailable }
         ELSE
            IF ( releaseType == "AccountDelivery" AND downstreamReservoir == "ElVado" )
            THEN
               { "PrioritizedAccounts"( % "Heron", "ElVadoStorageAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "ElVadoStorageAccounts"(  ) ),
"PrioritizedReleaseTypeAccounts"( "ElVadoStorageAccounts"(  ), "AccountDelivery", % "Heron", %
"ElVado" ), % "Heron", % "Heron" ) , % "Heron" , "ObjectifyString"( downstreamReservoir ) ,
releaseType , remainingAvailable }
            ELSE
               IF ( releaseType == "CochitiRecPool" )
               THEN
                  { { "CochitiRecPool" } , {
"CochitiRecPoolHeronToCochitiRecPoolHeronCochitiRecPoolCochitiHeronSeepage.Supply" } , % "Heron"
, % "Abiquiu" , releaseType , remainingAvailable }
               ELSE
                  IF ( releaseType == "AlbuquerquePaybacks" )
                  THEN
                     { "PrioritizedAccounts"( % "Heron", "HeronRootAlbuquerquePaybackAccounts"(
) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "HeronRootAlbuquerquePaybackAccounts"(
) ), "MakeAlbuquerqueAbiquiuPaybackAccountsList"( "PrioritizedAccounts"( % "Heron",
"HeronRootAlbuquerquePaybackAccounts"(  ) ), % "Heron" ), % "Heron", % "HeronSeepage" ) , %
"Heron" , % "Abiquiu" , releaseType , remainingAvailable }
                  ELSE
                     IF ( releaseType == "NMISC" )
                     THEN
                        { { "NMISC" } , {
"AlbuquerqueHeronToNMISCHeronNMISCJemezHeronSeepage.Supply" } , % "Heron" , % "Jemez" ,
releaseType , remainingAvailable }
                     ELSE
                        IF ( releaseType == "AccountFill" AND downstreamReservoir == "Abiquiu" )
                        THEN
```

```
                          { "PrioritizedAccounts"( % "Heron", "AbiquiuRootStorageAccounts"(  ) )
, "MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "AbiquiuRootStorageAccounts"(  ) ),
"PrioritizedReleaseTypeAccounts"( "AbiquiuRootStorageAccounts"(  ), "AccountFill", % "Heron", %
"Abiquiu" ), % "Heron", % "Heron" ) , % "Heron" , "ObjectifyString"( downstreamReservoir ) ,
releaseType , remainingAvailable }
                       ELSE
                          IF ( releaseType == "AccountFill" AND downstreamReservoir == "ElVado"
)
                          THEN
                             { "PrioritizedAccounts"( % "Heron", "ElVadoStorageAccounts"(  ) ) ,
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "ElVadoStorageAccounts"(  ) ),
"PrioritizedReleaseTypeAccounts"( "ElVadoStorageAccounts"(  ), "AccountFill", % "Heron", %
"ElVado" ), % "Heron", % "Heron" ) , % "Heron" , "ObjectifyString"( downstreamReservoir ) ,
releaseType , remainingAvailable }
                          ELSE
                             IF ( releaseType == "Waiver" AND downstreamReservoir == "Abiquiu" )
                             THEN
                                { "PrioritizedAccounts"( % "Heron",
"AbiquiuRootWaiverStorageAccounts"(  ) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"AbiquiuRootWaiverStorageAccounts"(  ) ), "PrioritizedReleaseTypeAccounts"(
"AbiquiuRootWaiverStorageAccounts"(  ), "Waiver", % "Heron", % "Abiquiu" ), % "Heron", % "Heron"
) , % "Heron" , "ObjectifyString"( downstreamReservoir ) , releaseType , remainingAvailable }
                             ELSE
                                IF ( releaseType == "Waiver" AND downstreamReservoir == "ElVado"
)
                                THEN
                                   { "PrioritizedAccounts"( % "Heron",
"ElVadoRootWaiverStorageAccounts"(  ) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"ElVadoRootWaiverStorageAccounts"(  ) ), "PrioritizedReleaseTypeAccounts"(
"ElVadoRootWaiverStorageAccounts"(  ), "Waiver", % "Heron", % "ElVado" ), % "Heron", % "Heron" )
, % "Heron" , "ObjectifyString"( downstreamReservoir ) , releaseType , remainingAvailable }
                                ELSE
                                   IF ( releaseType == "ReleaseToMRGCD" )
                                   THEN
                                      { "PrioritizedAccounts"( % "Heron",
"HeronToMRGCDAccountsList"(  ) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"HeronToMRGCDAccountsList"(  ) ), "MRGCDFifeteenTimesList"(  ), % "Heron", % "HeronSeepage" ) , %
"Heron" , % "ElVado" , releaseType , remainingAvailable }
                                   ELSE
                                      IF ( releaseType == "MRGCDPaybacks" )
                                      THEN
                                         { "PrioritizedAccounts"( % "Heron",
"HeronMRGCDPaybackAccounts"(  ) ) CONCAT "PrioritizedAccounts"( % "Heron",
"HeronMRGCDPaybackAccounts"(  ) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"HeronMRGCDPaybackAccounts"(  ) ), "MakeMRGCDPaybackAccountsList"( "PrioritizedAccounts"( %
"Heron", "HeronMRGCDPaybackAccounts"(  ) ), % "ElVado" ), % "Heron", % "HeronSeepage" ) CONCAT
"MakeSupplyList"( "PrioritizedAccounts"( % "Heron", "HeronMRGCDPaybackAccounts"(  ) ),
"MakeMRGCDPaybackAccountsList"( "PrioritizedAccounts"( % "Heron", "HeronMRGCDPaybackAccounts"(  )
), % "Abiquiu" ), % "Heron", % "HeronSeepage" ) , % "Heron" , "ReservoirInPriority"(  ) ,
releaseType , remainingAvailable }
                                      ELSE
                                         IF ( releaseType == "ElVadoDelivery" )
                                         THEN
                                            { "PrioritizedAccounts"( % "Heron",
"ElVadoRootWaiverStorageAccounts"(  ) ) , "MakeSupplyList"( "PrioritizedAccounts"( % "Heron",
"ElVadoRootWaiverStorageAccounts"(  ) ), "MakeHeronElVadoExAccountsLIst"( "PrioritizedAccounts"(
% "Heron", "ElVadoRootWaiverStorageAccounts"(  ) ) ), % "Heron", % "Heron" ) , % "Heron" , %
"ElVado" , releaseType , remainingAvailable }
                                         ELSE
                                            {  }
                                         ENDIF
                                      ENDIF
                                   ENDIF
                                ENDIF
                             ENDIF
                          ENDIF
                       ENDIF
                    ENDIF
                 ENDIF
              ENDIF
           ENDIF
        ENDIF
     ENDIF
   ENDIF;

    END;

    FUNCTION       "HeronMRGCDPaybackAccounts" (  )
    RETURN_TYPE    LIST;
```

```
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Reclamation" , "Albuquerque" };

    END;

    FUNCTION       "HeronNonDownstreamStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "CochitiRecPool" , "JicarillaApacheTribe" , "NambeFalls" , "RedRiver" ,
"Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "HeronRootAlbuquerquePaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "Bernalillo" , "NambeFalls" , "RedRiver" , "Uncontracted" , "Reclamation" ,
"SanJuanPueblo" };

    END;

    FUNCTION       "HeronRootAllAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Belen" , "Bernalillo" , "CochitiRecPool" , "Espanola" ,
"JicarillaApacheTribe" , "LosAlamos" , "LosLunas" , "MRGCD" , "NambeFalls" , "RedRiver" ,
"SantaFe" , "Taos" , "Twining" , "Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "HeronRootDownstreamStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Bernalillo" , "Espanola" , "LosAlamos" , "LosLunas" , "MRGCD" ,
"SantaFe" , "Taos" , "Twining" , "Reclamation" };

    END;

    FUNCTION       "HeronReclamationLeaseList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST accounts = "HeronToReclamationAccountsList"(  ) DO
```

```
      WITH LIST supplies = "MakeSupplyList"( "HeronToReclamationAccountsList"(  ),
"ReclamationFourteenTimesList"(  ), % "Heron", % "Heron" ) DO
         FOR ( STRING supply IN supplies ) WITH LIST result = {  } DO
            APPEND { supply , "AccountLeaseAmount"( GET STRING @INDEX LENGTH result FROM
accounts, % "Heron" ) } ONTO result
         ENDFOR
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "HeronDownstreamWaiverStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Bernalillo" , "Espanola" , "LosAlamos" , "LosLunas" , "MRGCD" ,
"SantaFe" , "Taos" , "Twining" };

    END;

    FUNCTION       "HeronRootOtowiPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Belen" , "Bernalillo" , "Espanola" , "LosAlamos" , "LosLunas" ,
"NambeFalls" , "RedRiver" , "SantaFe" , "Taos" , "Twining" , "SanJuanPueblo" };

    END;

    FUNCTION       "HeronOnlyRootOtowiPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "Bernalillo" , "NambeFalls" , "RedRiver" , "SanJuanPueblo" };

    END;

    FUNCTION       "RootStorageAccounts" ( OBJECT reservoir )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "StringifyObject"( reservoir ) == "Heron" )
   THEN
      { "Albuquerque" , "Belen" , "Bernalillo" , "CochitiRecPool" , "Espanola" ,
"JicarillaApacheTribe" , "LosAlamos" , "LosLunas" , "MRGCD" , "NambeFalls" , "RedRiver" ,
"SantaFe" , "Taos" , "Twining" , "Uncontracted" , "SanJuanPueblo" }
   ELSE
      IF ( "StringifyObject"( reservoir ) == "ElVado" )
      THEN
         { "MRGCD" , "SantaFe" , "LosAlamos" , "Espanola" , "Taos" , "LosLunas" , "Twining" ,
"Albuquerque" }
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
         THEN
            { "Espanola" , "LosAlamos" , "SantaFe" , "Taos" , "Twining" , "MRGCD" , "Albuquerque"
}
         ELSE
            IF ( "StringifyObject"( reservoir ) == "Cochiti" )
            THEN
```

```
            { "CochitiRecPool" }
         ELSE
            {  }
         ENDIF
      ENDIF
   ENDIF
ENDIF;

 END;

 FUNCTION       "HeronToMRGCDAccountsList" (  )
 RETURN_TYPE    LIST;
 SCALE_UNITS    "";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG FALSE;
 BEGIN

   { "Albuquerque" , "Belen" , "Bernalillo" , "Espanola" , "JicarillaApacheTribe" ,
"LosAlamos" , "LosLunas" , "NambeFalls" , "RedRiver" , "SantaFe" , "Taos" , "Twining" ,
"Uncontracted" , "MRGCD" , "SanJuanPueblo" };

 END;

 FUNCTION       "HeronToReclamationAccountsList" (  )
 RETURN_TYPE    LIST;
 SCALE_UNITS    "";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG FALSE;
 BEGIN

   { "Albuquerque" , "Belen" , "Bernalillo" , "Espanola" , "JicarillaApacheTribe" ,
"LosAlamos" , "LosLunas" , "MRGCD" , "RedRiver" , "SantaFe" , "Taos" , "Twining" , "Uncontracted"
, "SanJuanPueblo" };

 END;

 FUNCTION       "MakeAlbuquerqueAbiquiuPaybackAccountsList" ( LIST accounts, OBJECT source )
 RETURN_TYPE    LIST;
 SCALE_UNITS    "";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG FALSE;
 BEGIN

   FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
   APPEND account CONCAT STRINGIFY source CONCAT "AlbuquerqueAbiquiu" ONTO result
ENDFOR;

 END;

 FUNCTION       "MakeDestinationAccountList" ( LIST sourceAccounts, LIST destinationAccounts,
OBJECT sourceReservoir, OBJECT destinationReservoir )
 RETURN_TYPE    LIST;
 SCALE_UNITS    "";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG FALSE;
 BEGIN

   WITH LIST numbers = "CreateNumericList"( 0.00000000, ( LENGTH sourceAccounts ) -
1.00000000, 1.00000000 ) DO
   FOR ( NUMERIC number IN numbers ) WITH LIST result = {  } DO
      APPEND ( GET STRING @INDEX number FROM sourceAccounts ) CONCAT sourceReservoir CONCAT (
( GET STRING @INDEX number FROM destinationAccounts ) CONCAT destinationReservoir ) ONTO result
   ENDFOR
ENDWITH;

 END;

 FUNCTION       "MakeMRGCDPaybackAccountsList" ( LIST accounts, OBJECT destinationReservoir )
 RETURN_TYPE    LIST;
 SCALE_UNITS    "";
 DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "Heron" CONCAT "MRGCD" CONCAT "StringifyObject"( destinationReservoir
) ONTO result
   ENDFOR;

    END;

    FUNCTION       "MakeOtowiExchangeList" ( LIST accounts, STRING source )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND { account , "RGOtowi" CONCAT account CONCAT source CONCAT "EX" } ONTO result
   ENDFOR;

    END;

    FUNCTION       "MakeOtowiPaybackAccountList" ( OBJECT reservoir, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "StringifyObject"( reservoir ) CONCAT "RGOtowi" ONTO result
   ENDFOR;

    END;

    FUNCTION       "MRGCDFourteenTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" };

    END;

    FUNCTION       "PrepareSortList" ( STRING releaseType, OBJECT reservoir, STRING
downstreamReservoir, NUMERIC remainingAvailable )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "StringifyObject"( reservoir ) == "Heron" )
   THEN
      "HeronInitialSortList"( releaseType, downstreamReservoir, remainingAvailable )
   ELSE
      IF ( "StringifyObject"( reservoir ) == "ElVado" )
      THEN
         "ElVadoInitialSortList"( releaseType, remainingAvailable )
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
         THEN
```

```
            "AbiquiuInitialSortList"( releaseType, remainingAvailable )
         ELSE
            IF ( "StringifyObject"( reservoir ) == "Cochiti" )
            THEN
               "CochitiInitialSortList"( releaseType, remainingAvailable )
            ELSE
               {  }
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "PrioritizedReleaseTypeAccounts" ( LIST accounts, STRING releaseType, OBJECT
reservoir, OBJECT downstreamReservoir )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST prioritizedAccounts = "PrioritizedAccounts"( reservoir, accounts ) DO
      FOR ( STRING account IN prioritizedAccounts ) WITH LIST result = {  } DO
         APPEND account CONCAT "StringifyObject"( downstreamReservoir ) CONCAT releaseType ONTO
result
      ENDFOR
   ENDWITH;

    END;

    FUNCTION       "RGOtowiAlbuquerqueAbiquiuEXAccounts" ( LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "Abiquiu" CONCAT "RGOtowi" ONTO result
   ENDFOR;

    END;

    FUNCTION       "RGOtowiElVadoEXAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN "ElVadoOtowiEXAccountList"(  ) ) WITH LIST result = {  } DO
      APPEND account CONCAT "ElVado" CONCAT "RGOtowi" ONTO result
   ENDFOR;

    END;

    FUNCTION       "ReclamationFourteenTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" ,
"Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" ,
"Reclamation" , "Reclamation" , "Reclamation" };

    END;

    FUNCTION       "ReclamationEightTimesList" (  )
```

```
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" , "Reclamation" ,
"Reclamation" , "Reclamation" , "Reclamation" };

    END;

    FUNCTION       "ReleaseTypesList" ( OBJECT reservoir )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "StringifyObject"( reservoir ) == "Heron" )
   THEN
      { "ReleaseToMRGCD" , "OtowiPaybacks" , "Waiver" , "AccountFill" , "AccountDelivery" ,
"CochitiRecPool" , "AlbuquerquePaybacks" , "NMISC" , "MRGCDPaybacks" , "ElVadoDelivery" }
   ELSE
      IF ( "StringifyObject"( reservoir ) == "ElVado" )
      THEN
         { "OtowiPaybacks" , "AlbuquerquePaybacks" , "AccountFill" , "FlowThrough" ,
"MRGCDPaybacks" , "Reclamation" , "MRGCD" }
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
         THEN
            { "OtowiPaybacks" , "NMISC" , "FlowThrough" , "AlbuquerqueLoan" , "MRGCD" ,
"Reclamation" , "Albuquerque" }
         ELSE
            IF ( "StringifyObject"( reservoir ) == "Cochiti" )
            THEN
               { "FlowThrough" }
            ELSE
               {  }
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "StorageInHeronOnlyAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Belen" , "Bernalillo" , "NambeFalls" , "RedRiver" , "Uncontracted" , "SanJuanPueblo" };

    END;

    FUNCTION       "StorageInElVadoOnlyAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "LosLunas" };

    END;

    FUNCTION       "ZeroSortedAccounts" ( LIST list )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST supplies = GET LIST @INDEX 1.00000000 FROM list DO
      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
         APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , 0.00000000 ["cfs"] } ONTO alloc
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "StorageInAbiquiuOnlyAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Bernalillo" , "Reclamation" };

    END;

    FUNCTION       "GetSortedAlbuquerquePaybackAccounts" ( LIST accounts, OBJECT
upstreamReservoir, OBJECT downstreamReservoir, LIST supplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH supplies ) WITH LIST alloc = {  } DO
      APPEND { GET STRING @INDEX LENGTH alloc FROM supplies , "Max"( 0.00000000 ["cfs"], "Min"(
"Min"( "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM supplies ),
@"Current Timestep" ), "Min"( "VolumeToFlow"( "AvailableAccountStorage"( "Albuquerque", %
"Abiquiu" ), @"Current Timestep" ) * ( 1.00000000 + "SJCLoss"( upstreamReservoir,
downstreamReservoir ) ), "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc
FROM accounts, upstreamReservoir ), @"Current Timestep" ) ) ), available - FOR ( LIST alloc2 IN
alloc ) WITH NUMERIC sum = 0.00000000 ["cfs"] DO
         sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
      ENDFOR ) ) } ONTO alloc
   ENDWHILE;

    END;

    FUNCTION       "ElVadoAbiquiuCommonStorageAccountsLessAlb" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Espanola" , "LosAlamos" , "MRGCD" , "SantaFe" , "Taos" };

    END;

    FUNCTION       "CochitiAlbuquerqueFlowThroughAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueAbiquiuAlbuquerqueMiddleValleyDiversions" , "AlbuquerqueAbiquiuAlbuquerqueEB"
, "AlbuquerqueElVadoAlbuquerqueEB" , "AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" ,
"AlbuquerqueHeronAlbuquerqueEB" };

    END;

    FUNCTION       "CochitiReclamationFlowThroughAccounts" (  )
```

```
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "ReclamationAbiquiuMRGCDMiddleValleyDiversions" };

    END;

    FUNCTION       "GetSortedCochitiFlowThroughAccounts" ( LIST accounts, OBJECT reservoir, LIST
upstreamSupplies, LIST downstreamSupplies, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( ( LENGTH alloc ) < LENGTH downstreamSupplies ) WITH LIST alloc = {  } DO
      IF ( ( GET STRING @INDEX LENGTH alloc FROM accounts ) ==
"AlbuquerqueAbiquiuAlbuquerqueMiddleValleyDiversions" )
      THEN
         APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"( 0.00000000
["cfs"], "Min"( "PreviousSlotInflow"( GET STRING @INDEX LENGTH alloc FROM upstreamSupplies ) * (
1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) ) + "VolumeToFlow"( "PreviousAccountStorage"(
GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current Timestep" ), available - FOR
( LIST slotValueList IN alloc ) WITH NUMERIC total = 0.00000000 ["cfs"] DO
            total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
         ENDFOR ) ) } ONTO alloc
      ELSE
         IF ( ( GET STRING @INDEX LENGTH alloc FROM accounts ) ==
"MRGCDAbiquiuMRGCDMiddleValleyDiversions" )
         THEN
            APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"( 0.00000000
["cfs"], "Min"( "PreviousSlotInflow"( GET STRING @INDEX LENGTH alloc FROM upstreamSupplies ) * (
1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) ) + "VolumeToFlow"( "PreviousAccountStorage"(
GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current Timestep" ), available - FOR
( LIST slotValueList IN alloc ) WITH NUMERIC total = 0.00000000 ["cfs"] DO
               total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
            ENDFOR ) ) } ONTO alloc
         ELSE
            IF ( ( GET STRING @INDEX LENGTH alloc FROM accounts ) ==
"ReclamationAbiquiuMRGCDMiddleValleyDiversions" )
            THEN
               APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"(
0.00000000 ["cfs"], "Min"( "PreviousSlotInflow"( GET STRING @INDEX LENGTH alloc FROM
upstreamSupplies ) * ( 1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) ) + "VolumeToFlow"(
"PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current
Timestep" ), available - FOR ( LIST slotValueList IN alloc ) WITH NUMERIC total = 0.00000000
["cfs"] DO
                  total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
               ENDFOR ) ) } ONTO alloc
            ELSE
               APPEND { GET STRING @INDEX LENGTH alloc FROM downstreamSupplies , "Max"(
0.00000000 ["cfs"], "Min"( "PreviousSlotInflow"( GET STRING @INDEX LENGTH alloc FROM
upstreamSupplies ) * ( 1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) ) + "VolumeToFlow"(
"PreviousAccountStorage"( GET STRING @INDEX LENGTH alloc FROM accounts, reservoir ), @"Current
Timestep" ), available - FOR ( LIST slotValueList IN alloc ) WITH NUMERIC total = 0.00000000
["cfs"] DO
                  total + GET NUMERIC @INDEX 1.00000000 FROM slotValueList
               ENDFOR ) ) } ONTO alloc
            ENDIF
         ENDIF
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "CochitiRecPoolPaybackAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      { "MRGCD" , "Albuquerque" , "Reclamation" };

    END;

    FUNCTION       "ElVadoFlowThroughAccountSupplies" ( STRING account )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( account == "AlbuquerqueHeronAlbuquerqueAbiquiu" )
   THEN
      { "AlbuquerqueHeronToAlbuquerqueAbiquiuWaiverHeron.Supply" ,
"AlbuquerqueHeronToAlbuquerqueAbiquiuAccountFillHeron.Supply" ,
"AlbuquerqueHeronToAlbuquerqueAbiquiuAccountDeliveryHeron.Supply" }
   ELSE
      IF ( account == "AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" )
      THEN
         {
"AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueMiddleValleyDiversionsHeronSeepage.Supply" }
      ELSE
         IF ( account == "AlbuquerqueHeronAlbuquerqueEB" )
         THEN
            { "AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueEBHeronSeepage.Supply" }
         ELSE
            IF ( account == "AlbuquerqueHeronRGOtowi" )
            THEN
               { "AlbuquerqueHeronToAlbuquerqueHeronRGOtowiHeronSeepage.Supply" }
            ELSE
               IF ( account == "BelenHeronAlbuquerqueAbiquiu" )
               THEN
                  { "BelenHeronToBelenHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
               ELSE
                  IF ( account == "BelenHeronRGOtowi" )
                  THEN
                     { "BelenHeronToBelenHeronRGOtowiHeronSeepage.Supply" }
                  ELSE
                     IF ( account == "BernalilloHeronAlbuquerqueAbiquiu" )
                     THEN
                        { "BernalilloHeronToBernalilloHeronAlbuquerqueAbiquiuHeronSeepage.Supply"
}
                     ELSE
                        IF ( account == "BernalilloHeronBernalilloAbiquiu" )
                        THEN
                           { "BernalilloHeronToBernalilloAbiquiuWaiverHeron.Supply" ,
"BernalilloHeronToBernalilloAbiquiuAccountFillHeron.Supply" ,
"BernalilloHeronToBernalilloAbiquiuAccountDeliveryHeron.Supply" }
                        ELSE
                           IF ( account == "BernalilloHeronRGOtowi" )
                           THEN
                              { "BernalilloHeronToBernalilloHeronRGOtowiHeronSeepage.Supply" }
                           ELSE
                              IF ( account == "CochitiRecPoolHeronAlbuquerqueAbiquiu" )
                              THEN
                                 {
"CochitiRecPoolHeronToCochitiRecPoolHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                              ELSE
                                 IF ( account == "CochitiRecPoolHeronCochitiRecPoolCochiti" )
                                 THEN
                                    {
"CochitiRecPoolHeronToCochitiRecPoolHeronCochitiRecPoolCochitiHeronSeepage.Supply" }
                                 ELSE
                                    IF ( account == "EspanolaHeronAlbuquerqueAbiquiu" )
                                    THEN
                                       {
"EspanolaHeronToEspanolaHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                    ELSE
                                       IF ( account == "EspanolaHeronEspanolaAbiquiu" )
                                       THEN
                                          { "EspanolaHeronToEspanolaAbiquiuWaiverHeron.Supply" ,
"EspanolaHeronToEspanolaAbiquiuAccountFillHeron.Supply" ,
"EspanolaHeronToEspanolaAbiquiuAccountDeliveryHeron.Supply" }
                                       ELSE
                                          IF ( account == "EspanolaHeronRGOtowi" )
```

```
                                        THEN
                                           {
"EspanolaHeronToEspanolaHeronRGOtowiHeronSeepage.Supply" }
                                        ELSE
                                           IF ( account == "LosAlamosHeronAlbuquerqueAbiquiu" )
                                           THEN
                                              {
"LosAlamosHeronToLosAlamosHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                           ELSE
                                              IF ( account == "LosAlamosHeronLosAlamosAbiquiu"
)
                                              THEN
                                                 {
"LosAlamosHeronToLosAlamosAbiquiuWaiverHeron.Supply" ,
"LosAlamosHeronToLosAlamosAbiquiuAccountFillHeron.Supply" ,
"LosAlamosHeronToLosAlamosAbiquiuAccountDeliveryHeron.Supply" }
                                              ELSE
                                                 IF ( account == "LosAlamosHeronRGOtowi" )
                                                 THEN
                                                    {
"LosAlamosHeronToLosAlamosHeronRGOtowiHeronSeepage.Supply" }
                                                 ELSE
                                                    IF ( account ==
"LosLunasHeronAlbuquerqueAbiquiu" )
                                                    THEN
                                                       {
"LosLunasHeronToLosLunasHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                                    ELSE
                                                       IF ( account == "LosLunasHeronRGOtowi" )
                                                       THEN
                                                          {
"LosLunasHeronToLosLunasHeronRGOtowiHeronSeepage.Supply" }
                                                       ELSE
                                                          IF ( account ==
"MRGCDHeronAlbuquerqueAbiquiu" )
                                                          THEN
                                                             {
"MRGCDHeronToMRGCDHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                                          ELSE
                                                             IF ( account ==
"MRGCDHeronMRGCDAbiquiu" )
                                                             THEN
                                                                {
"MRGCDHeronToMRGCDAbiquiuWaiverHeron.Supply" , "MRGCDHeronToMRGCDAbiquiuAccountFillHeron.Supply"
, "MRGCDHeronToMRGCDAbiquiuAccountDeliveryHeron.Supply" }
                                                             ELSE
                                                                IF ( account ==
"MRGCDHeronMRGCDBlwCochitiDiversions" )
                                                                THEN
                                                                   {
"MRGCDHeronToMRGCDHeronMRGCDBlwCochitiDiversionsHeronSeepage.Supply" }
                                                                ELSE
                                                                   IF ( account ==
"MRGCDHeronMRGCDBlwIsletaDiversions" )
                                                                   THEN
                                                                      {
"MRGCDHeronToMRGCDHeronMRGCDBlwIsletaDiversionsHeronSeepage.Supply" }
                                                                   ELSE
                                                                      IF ( account ==
"MRGCDHeronMRGCDMiddleValleyDiversions" )
                                                                      THEN
                                                                         {
"MRGCDHeronToMRGCDHeronMRGCDMiddleValleyDiversionsHeronSeepage.Supply" }
                                                                      ELSE
                                                                         IF ( account ==
"NambeFallsHeronAlbuquerqueAbiquiu" )
                                                                         THEN
                                                                            {
"NambeFallsHeronToNambeFallsHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                                                         ELSE
                                                                            IF ( account ==
"NambeFallsHeronRGOtowi" )
                                                                            THEN
                                                                               {
"NambeFallsHeronToNambeFallsHeronRGOtowiHeronSeepage.Supply" }
                                                                            ELSE
                                                                               IF ( account ==
"RedRiverHeronAlbuquerqueAbiquiu" )
                                                                               THEN
```

```
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
"RedRiverHeronToRedRiverHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
{
ELSE
IF ( account
== "RedRiverHeronRGOtowi" )
THEN
{
"RedRiverHeronToRedRiverHeronRGOtowiHeronSeepage.Supply" }
ELSE
IF (
account == "SantaFeHeronRGOtowi" )
THEN
{
"SantaFeHeronToSantaFeHeronRGOtowiHeronSeepage.Supply" }
ELSE
IF (
account == "SantaFeHeronSantaFeAbiquiu" )
THEN
{
"SantaFeHeronToSantaFeAbiquiuWaiverHeron.Supply" ,
"SantaFeHeronToSantaFeAbiquiuAccountFillHeron.Supply" ,
"SantaFeHeronToSantaFeAbiquiuAccountDeliveryHeron.Supply" }
ELSE
IF (
account == "SantaFeHeronAlbuquerqueAbiquiu" )
THEN
{
"SantaFeHeronToSantaFeHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
ELSE
IF ( account == "TaosHeronAlbuquerqueAbiquiu" )
THEN
{ "TaosHeronToTaosHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
ELSE
IF ( account == "TaosHeronRGOtowi" )
THEN
{ "TaosHeronToTaosHeronRGOtowiHeronSeepage.Supply" }
ELSE
IF ( account == "TaosHeronTaosAbiquiu" )
THEN
{ "TaosHeronToTaosAbiquiuWaiverHeron.Supply" , "TaosHeronToTaosAbiquiuAccountFillHeron.Supply" ,
"TaosHeronToTaosAbiquiuAccountDeliveryHeron.Supply" }
ELSE
IF ( account == "TwiningHeronAlbuquerqueAbiquiu" )
THEN
{ "TwiningHeronToTwiningHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }
ELSE
IF ( account == "TwiningHeronRGOtowi" )
THEN
{ "TwiningHeronToTwiningHeronRGOtowiHeronSeepage.Supply" }
ELSE
IF ( account == "TwiningHeronTwiningAbiquiu" )
THEN
{ "TwiningHeronToTwiningAbiquiuWaiverHeron.Supply" ,
"TwiningHeronToTwiningAbiquiuAccountFillHeron.Supply" ,
"TwiningHeronToTwiningAbiquiuAccountDeliveryHeron.Supply" }
```

```
ELSE

IF ( account == "UncontractedHeronAlbuquerqueAbiquiu" )

THEN

{ "UncontractedHeronToUncontractedHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }

ELSE

IF ( account == "UncontractedHeronRGOtowi" )

THEN

{ "UncontractedHeronToUncontractedHeronRGOtowiHeronSeepage.Supply" }

ELSE

IF ( account == "ReclamationHeronReclamationAbiquiu" )

THEN

{ "ReclamationHeronToReclamationAbiquiuAccountFillHeron.Supply" ,
"ReclamationHeronToReclamationAbiquiuAccountDeliveryHeron.Supply" }

ELSE

IF ( account == "ReclamationHeronAlbuquerqueAbiquiu" )

THEN

{ "ReclamationHeronToReclamationHeronAlbuquerqueAbiquiuHeronSeepage.Supply" }

ELSE

{  }

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF
                                                
```

```
                              ENDIF
                            ENDIF
                          ENDIF
                        ENDIF
                      ENDIF
                    ENDIF
                  ENDIF
              ENDIF
          ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "AbiquiuWaterPurchaseAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueAbiquiuReclamationAbiquiu" , "Bernalillo" , "Espanola" , "LosAlamos" ,
"MRGCD" , "Taos" , "SantaFe" , "Twining" };

    END;

    FUNCTION       "GetSortedAlbuquerqueMRGCDLoans" ( NUMERIC loan )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST loans = { "MRGCDElVadoAlbuquerqueHeronEX.Borrow" ,
"MRGCDAbiquiuAlbuquerqueAbiquiuEX.Borrow" } DO
      WITH LIST supplies = { "AlbuquerqueHeronToAlbuquerqueHeronMRGCDElVadoHeronSeepage.Supply" ,
"AlbuquerqueAbiquiuToAlbuquerqueAbiquiuMRGCDAbiquiuAbiquiu.Supply" } DO
         WITH LIST reservoirs = { % "Heron" , % "Abiquiu" } DO
            WHILE ( ( LENGTH alloc ) < LENGTH loans ) WITH LIST alloc = {  } DO
               APPEND { GET STRING @INDEX LENGTH alloc FROM loans , "Min"( "Max"( "VolumeToFlow"(
"PreviousAccountStorage"( "Albuquerque", GET OBJECT @INDEX LENGTH alloc FROM reservoirs ) -
"GetAccountDebt"( GET STRING @INDEX LENGTH alloc FROM supplies ), @"Current Timestep" ),
0.00000000 ["cfs"] ), "VolumeToFlow"( loan, @"Current Timestep" ) - FOR ( LIST alloc2 IN alloc )
WITH NUMERIC sum = 0.00000000 ["cfs"] DO
                  sum + GET NUMERIC @INDEX 1.00000000 FROM alloc2
               ENDFOR ) } ONTO alloc
            ENDWHILE
         ENDWITH
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "MakeHeronElVadoExchangeList" ( LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "ElVado" CONCAT account CONCAT "HeronEX.Borrow" ONTO result
   ENDFOR;

    END;

    FUNCTION       "MakeHeronElVadoExAccountsLIst" ( LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "ElVadoExchange" ONTO result
   ENDFOR;

    END;

    FUNCTION       "ElVadoAbiquiuCommonWaiverStorageAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Espanola" , "LosAlamos" , "MRGCD" , "SantaFe" , "Taos" , "Twining" , "Albuquerque" };

    END;

    FUNCTION       "MRGCDThreeTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCD" , "MRGCD" , "MRGCD" };

    END;

    FUNCTION       "AbiquiuToCochitiMRGCDFlowThroughAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDAbiquiuMRGCDMiddleValleyDiversions" };

    END;

    FUNCTION       "AllElVadoToMRGCDAbiquiuSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDHeronMRGCDAbiquiuElVadoToMRGCDHeronMRGCDAbiquiuBlwElVado.Supply" ,
"ReclamationHeronMRGCDAbiquiuElVadoToReclamationHeronMRGCDAbiquiuBlwElVado.Supply" ,
"AlbuquerqueHeronMRGCDAbiquiuElVadoToAlbuquerqueHeronMRGCDAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoMRGCDAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoMiddleValleyDiversionsBlwElVado.Supply" };

    END;

    FUNCTION       "AllElVadoAccountFillandPaybackSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "BelenHeronAlbuquerqueAbiquiuElVadoToBelenHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"BernalilloHeronAlbuquerqueAbiquiuElVadoToBernalilloHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"BernalilloHeronBernalilloAbiquiuElVadoToBernalilloHeronBernalilloAbiquiuBlwElVado.Supply" ,
"EspanolaHeronAlbuquerqueAbiquiuElVadoToEspanolaHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"EspanolaHeronEspanolaAbiquiuElVadoToEspanolaHeronEspanolaAbiquiuBlwElVado.Supply" ,
"LosAlamosHeronAlbuquerqueAbiquiuElVadoToLosAlamosHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosAlamosHeronLosAlamosAbiquiuElVadoToLosAlamosHeronLosAlamosAbiquiuBlwElVado.Supply" ,
```

```
"LosLunasHeronAlbuquerqueAbiquiuElVadoToLosLunasHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"MRGCDHeronAlbuquerqueAbiquiuElVadoToMRGCDHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"MRGCDHeronMRGCDAbiquiuElVadoToMRGCDHeronMRGCDAbiquiuBlwElVado.Supply" ,
"NambeFallsHeronAlbuquerqueAbiquiuElVadoToNambeFallsHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"RedRiverHeronAlbuquerqueAbiquiuElVadoToRedRiverHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"SantaFeHeronSantaFeAbiquiuElVadoToSantaFeHeronSantaFeAbiquiuBlwElVado.Supply" ,
"TaosHeronAlbuquerqueAbiquiuElVadoToTaosHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosHeronTaosAbiquiuElVadoToTaosHeronTaosAbiquiuBlwElVado.Supply" ,
"TwiningHeronAlbuquerqueAbiquiuElVadoToTwiningHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TwiningHeronTwiningAbiquiuElVadoToTwiningHeronTwiningAbiquiuBlwElVado.Supply" ,
"UncontractedHeronAlbuquerqueAbiquiuElVadoToUncontractedHeronAlbuquerqueAbiquiuBlwElVado.Supply"
, "ReclamationHeronReclamationAbiquiuElVadoToReclamationHeronReclamationAbiquiuBlwElVado.Supply"
, "ReclamationHeronAlbuquerqueAbiquiuElVadoToReclamationHeronAlbuquerqueAbiquiuBlwElVado.Supply"
, "SantaFeHeronAlbuquerqueAbiquiuElVadoToSantaFeHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"ReclamationHeronMRGCDAbiquiuElVadoToReclamationHeronMRGCDAbiquiuBlwElVado.Supply" ,
"AlbuquerqueHeronMRGCDAbiquiuElVadoToAlbuquerqueHeronMRGCDAbiquiuBlwElVado.Supply" ,
"ReclamationElVadoToReclamationElVadoReclamationAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoMRGCDAbiquiuBlwElVado.Supply" ,
"AlbuquerqueElVadoToAlbuquerqueElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosElVadoToTaosElVadoTaosAbiquiuBlwElVado.Supply" ,
"EspanolaElVadoToEspanolaElVadoEspanolaAbiquiuBlwElVado.Supply" ,
"LosAlamosElVadoToLosAlamosElVadoLosAlamosAbiquiuBlwElVado.Supply" ,
"TwiningElVadoToTwiningElVadoTwiningAbiquiuBlwElVado.Supply" ,
"SantaFeElVadoToSantaFeElVadoSantaFeAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosElVadoToTaosElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"EspanolaElVadoToEspanolaElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosAlamosElVadoToLosAlamosElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosLunasElVadoToLosLunasElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TwiningElVadoToTwiningElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"SantaFeElVadoToSantaFeElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"ReclamationElVadoToReclamationElVadoMRGCDAbiquiuBlwElVado.Supply" };

    END;

    FUNCTION       "AlbuquerqueTwelveTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" ,
"Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" , "Albuquerque" ,
"Albuquerque" };

    END;

    FUNCTION       "MRGCDFifeteenTimesList" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" ,
"MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" , "MRGCDHeronMRGCDElVado" };

    END;

    FUNCTION       "AllElVadoSJSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueHeronRGOtowiElVadoToAlbuquerqueHeronRGOtowiBlwElVado.Supply" ,
"BelenHeronAlbuquerqueAbiquiuElVadoToBelenHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"BelenHeronRGOtowiElVadoToBelenHeronRGOtowiBlwElVado.Supply" ,
```

```
"BernalilloHeronAlbuquerqueAbiquiuElVadoToBernalilloHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"BernalilloHeronBernalilloAbiquiuElVadoToBernalilloHeronBernalilloAbiquiuBlwElVado.Supply" ,
"BernalilloHeronRGOtowiElVadoToBernalilloHeronRGOtowiBlwElVado.Supply" ,
"CochitiRecPoolHeronAlbuquerqueAbiquiuElVadoToCochitiRecPoolHeronAlbuquerqueAbiquiuBlwElVado.Supp
ly" ,
"CochitiRecPoolHeronCochitiRecPoolCochitiElVadoToCochitiRecPoolHeronCochitiRecPoolCochitiBlwElVad
o.Supply" ,
"EspanolaHeronAlbuquerqueAbiquiuElVadoToEspanolaHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"EspanolaHeronEspanolaAbiquiuElVadoToEspanolaHeronEspanolaAbiquiuBlwElVado.Supply" ,
"EspanolaHeronRGOtowiElVadoToEspanolaHeronRGOtowiBlwElVado.Supply" ,
"LosAlamosHeronAlbuquerqueAbiquiuElVadoToLosAlamosHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosAlamosHeronLosAlamosAbiquiuElVadoToLosAlamosHeronLosAlamosAbiquiuBlwElVado.Supply" ,
"LosAlamosHeronRGOtowiElVadoToLosAlamosHeronRGOtowiBlwElVado.Supply" ,
"LosLunasHeronAlbuquerqueAbiquiuElVadoToLosLunasHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosLunasHeronRGOtowiElVadoToLosLunasHeronRGOtowiBlwElVado.Supply" ,
"MRGCDHeronAlbuquerqueAbiquiuElVadoToMRGCDHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"MRGCDHeronMRGCDAbiquiuElVadoToMRGCDHeronMRGCDAbiquiuBlwElVado.Supply" ,
"MRGCDHeronMRGCDMiddleValleyDiversionsElVadoToMRGCDHeronMRGCDMiddleValleyDiversionsBlwElVado.Supp
ly" ,
"NambeFallsHeronAlbuquerqueAbiquiuElVadoToNambeFallsHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"NambeFallsHeronRGOtowiElVadoToNambeFallsHeronRGOtowiBlwElVado.Supply" ,
"RedRiverHeronAlbuquerqueAbiquiuElVadoToRedRiverHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"RedRiverHeronRGOtowiElVadoToRedRiverHeronRGOtowiBlwElVado.Supply" ,
"SantaFeHeronRGOtowiElVadoToSantaFeHeronRGOtowiBlwElVado.Supply" ,
"SantaFeHeronSantaFeAbiquiuElVadoToSantaFeHeronSantaFeAbiquiuBlwElVado.Supply" ,
"TaosHeronAlbuquerqueAbiquiuElVadoToTaosHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosHeronRGOtowiElVadoToTaosHeronRGOtowiBlwElVado.Supply" ,
"TaosHeronTaosAbiquiuElVadoToTaosHeronTaosAbiquiuBlwElVado.Supply" ,
"TwiningHeronAlbuquerqueAbiquiuElVadoToTwiningHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TwiningHeronRGOtowiElVadoToTwiningHeronRGOtowiBlwElVado.Supply" ,
"TwiningHeronTwiningAbiquiuElVadoToTwiningHeronTwiningAbiquiuBlwElVado.Supply" ,
"UncontractedHeronAlbuquerqueAbiquiuElVadoToUncontractedHeronAlbuquerqueAbiquiuBlwElVado.Supply"
, "UncontractedHeronRGOtowiElVadoToUncontractedHeronRGOtowiBlwElVado.Supply" ,
"ReclamationHeronReclamationAbiquiuElVadoToReclamationHeronReclamationAbiquiuBlwElVado.Supply" ,
"ReclamationHeronAlbuquerqueAbiquiuElVadoToReclamationHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"SantaFeHeronAlbuquerqueAbiquiuElVadoToSantaFeHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"ReclamationHeronMRGCDAbiquiuElVadoToReclamationHeronMRGCDAbiquiuBlwElVado.Supply" ,
"AlbuquerqueHeronMRGCDAbiquiuElVadoToAlbuquerqueHeronMRGCDAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoMiddleValleyDiversionsBlwElVado.Supply" ,
"ReclamationElVadoToReclamationElVadoReclamationAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoMRGCDAbiquiuBlwElVado.Supply" ,
"AlbuquerqueElVadoToAlbuquerqueElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosElVadoToTaosElVadoTaosAbiquiuBlwElVado.Supply" ,
"EspanolaElVadoToEspanolaElVadoEspanolaAbiquiuBlwElVado.Supply" ,
"LosAlamosElVadoToLosAlamosElVadoLosAlamosAbiquiuBlwElVado.Supply" ,
"TwiningElVadoToTwiningElVadoTwiningAbiquiuBlwElVado.Supply" ,
"SantaFeElVadoToSantaFeElVadoSantaFeAbiquiuBlwElVado.Supply" ,
"MRGCDElVadoToMRGCDElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TaosElVadoToTaosElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"EspanolaElVadoToEspanolaElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosAlamosElVadoToLosAlamosElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"LosLunasElVadoToLosLunasElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"TwiningElVadoToTwiningElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"SantaFeElVadoToSantaFeElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"ReclamationElVadoToReclamationElVadoMRGCDAbiquiuBlwElVado.Supply" ,
"TaosElVadoToTaosElVadoRGOtowiBlwElVado.Supply" ,
"EspanolaElVadoToEspanolaElVadoRGOtowiBlwElVado.Supply" ,
"LosAlamosElVadoToLosAlamosElVadoRGOtowiBlwElVado.Supply" ,
"LosLunasElVadoToLosLunasElVadoRGOtowiBlwElVado.Supply" ,
"TwiningElVadoToTwiningElVadoRGOtowiBlwElVado.Supply" ,
"SantaFeElVadoToSantaFeElVadoRGOtowiBlwElVado.Supply" ,
"AlbuquerqueHeronAlbuquerqueAbiquiuElVadoToAlbuquerqueHeronAlbuquerqueAbiquiuBlwElVado.Supply" ,
"AlbuquerqueElVadoToAlbuquerqueElVadoAlbuquerqueEBBlwElVado.Supply" ,
"AlbuquerqueHeronAlbuquerqueEBElVadoToAlbuquerqueHeronAlbuquerqueEBBlwElVado.Supply" ,
"AlbuquerqueHeronAlbuquerqueMiddleValleyDiversionsElVadoToAlbuquerqueHeronAlbuquerqueMiddleValley
DiversionsBlwElVado.Supply" ,
"JicarillaApacheTribeHeronAlbuquerqueAbiquiuElVadoToJicarillaApacheTribeHeronAlbuquerqueAbiquiuBl
wElVado.Supply" , "NMISCHeronNMISCJemezElVadoToNMISCHeronNMISCJemezBlwElVado.Supply" ,
"ReclamationElVadoToReclamationElVadoAlbuquerqueAbiquiuBlwElVado.Supply" ,
"SanJuanPuebloHeronAlbuquerqueAbiquiuElVadoToSanJuanPuebloAlbquerqueAbiquiuBlwElVado.Supply" ,
"SanJuanPuebloHeronRGOtowiElVadoToSanJuanPuebloRGOtowi BlwElVado.Supply" };

    END;

    FUNCTION       "AllHeronSJSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      {
"AlbuquerqueAbiquiuAccountDeliveryHeronToAlbuquerqueHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"AlbuquerqueAbiquiuAccountFillHeronToAlbuquerqueHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"AlbuquerqueAbiquiuWaiverHeronToAlbuquerqueHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"AlbuquerqueElVadoExchangeHeronToAlbuquerqueHeronAlbuquerqueElVadoHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueEBHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueElVadoHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronAlbuquerqueMiddleValleyDiversionsHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronMRGCDElVadoHeronSeepage.Supply" ,
"AlbuquerqueHeronToAlbuquerqueHeronRGOtowiHeronSeepage.Supply" ,
"AlbuquerqueHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"AlbuquerqueHeronToNMISCHeronNMISCJemezHeronSeepage.Supply" ,
"BelenHeronToBelenHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"BelenHeronToBelenHeronRGOtowiHeronSeepage.Supply" ,
"BelenHeronToBelenHeronSantaFeElVadoHeronSeepage.Supply" ,
"BelenHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"BernalilloAbiquiuAccountDeliveryHeronBernalilloHeronBernalilloAbiquiuHeronSeepage.Supply" ,
"BernalilloAbiquiuAccountFillHeronToBernalilloHeronBernalilloAbiquiuHeronSeepage.Supply" ,
"BernalilloAbiquiuWaiverHeronToBernalilloHeronBernalilloAbiquiuHeronSeepage.Supply" ,
"BernalilloHeronToBernalilloHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"BernalilloHeronToBernalilloHeronBernalilloAbiquiuHeronSeepage.Supply" ,
"BernalilloHeronToBernalilloHeronRGOtowiHeronSeepage.Supply" ,
"BernalilloHeronToBernalilloHeronSantaFeElVadoHeronSeepage.Supply" ,
"BernalilloHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"CochitiRecPoolHeronToCochitiRecPoolHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"CochitiRecPoolHeronToCochitiRecPoolHeronCochitiRecPoolCochitiHeronSeepage.Supply" ,
"CochitiRecPoolHeronToCochitiRecPoolHeronSantaFeElVadoHeronSeepage.Supply" ,
"EspanolaAbiquiuAccountDeliveryHeronToEspanolaHeronEspanolaAbiquiuHeronSeepage.Supply" ,
"EspanolaAbiquiuAccountFillHeronToEspanolaHeronEspanolaAbiquiuHeronSeepage.Supply" ,
"EspanolaAbiquiuWaiverHeronToEspanolaHeronEspanolaAbiquiuHeronSeepage.Supply" ,
"EspanolaElVadoAccountDeliveryHeronToEspanolaHeronEspanolaElVadoHeronSeepage.Supply" ,
"EspanolaElVadoAccountFillHeronToEspanolaHeronEspanolaElVadoHeronSeepage.Supply" ,
"EspanolaElVadoExchangeHeronToEspanolaHeronEspanolaElVadoHeronSeepage.Supply" ,
"EspanolaElVadoWaiverHeronToEspanolaHeronEspanolaElVadoHeronSeepage.Supply" ,
"EspanolaHeronToEspanolaHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"EspanolaHeronToEspanolaHeronEspanolaAbiquiuHeronSeepage.Supply" ,
"EspanolaHeronToEspanolaHeronEspanolaElVadoHeronSeepage.Supply" ,
"EspanolaHeronToEspanolaHeronRGOtowiHeronSeepage.Supply" ,
"EspanolaHeronToEspanolaHeronSantaFeElVadoHeronSeepage.Supply" ,
"EspanolaHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"JicarillaApacheTribeHeronToJicarillaApacheTribeHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"JicarillaApacheTribeHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"LosAlamosAbiqiuAccountDeliveryHeronToLosAlamosHeronLosAlamosAbiquiuHeronSeepage.Supply" ,
"LosAlamosAbiquiuAccountFillHeronToLosAlamosHeronLosAlamosAbiquiuHeronSeepage.Supply" ,
"LosAlamosAbiquiuWaiverHeronToLosAlamosHeronLosAlamosAbiquiuHeronSeepage.Supply" ,
"LosAlamosElVadoAccountDeliveryHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"LosAlamosElVadoAccountFillHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"LosAlamosElVadoExchangeHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"LosAlamosElVadoWaiverHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"LosAlamosHeronToLosAlamosHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"LosAlamosHeronToLosAlamosHeronLosAlamosAbiquiuHeronSeepage.Supply" ,
"LosAlamosHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"LosAlamosHeronToLosAlamosHeronRGOtowiHeronSeepage.Supply" ,
"LosAlamosHeronToLosAlamosHeronSantaFeElVadoHeronSeepage.Supply" ,
"LosAlamosHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"LosLunasElVadoAccountDeliveryHeronToLosLunasHeronLosLunasElVadoHeronSeepage.Supply" ,
"LosLunasElVadoAccountFillHeronToLosLunasHeronLosLunasElVadoHeronSeepage.Supply" ,
"LosLunasElVadoExchangeHeronToLosLunasHeronLosLunasElVadoHeronSeepage.Supply" ,
"LosLunasElVadoWaiverHeronToLosLunasHeronLosLunasElVadoHeronSeepage.Supply" ,
"LosLunasHeronToLosLunasHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"LosLunasHeronToLosLunasHeronLosLunasElVadoHeronSeepage.Supply" ,
"LosLunasHeronToLosLunasHeronRGOtowiHeronSeepage.Supply" ,
"LosLunasHeronToLosLunasHeronSantaFeElVadoHeronSeepage.Supply" ,
"LosLunasHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDAbiquiuAccountDeliveryHeronToMRGCDHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"MRGCDAbiquiuAccountFillHeronToMRGCDHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"MRGCDAbiquiuWaiverHeronToMRGCDHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"MRGCDElVadoAccountDeliveryHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDElVadoAccountFillHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDElVadoExchangeHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDElVadoWaiverHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDHeronToMRGCDHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"MRGCDHeronToMRGCDHeronMRGCDAbiquiuHeronSeepage.Supply" ,
```

```
"MRGCDHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"MRGCDHeronToMRGCDHeronMRGCDMiddleValleyDiversionsHeronSeepage.Supply" ,
"MRGCDHeronToMRGCDHeronSantaFeElVadoHeronSeepage.Supply" ,
"NambeFallsHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"NambeFallsHeronToNambeFallsHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"NambeFallsHeronToNambeFallsHeronRGOtowiHeronSeepage.Supply" ,
"NambeFallsHeronToNambeFallsHeronSantaFeElVadoHeronSeepage.Supply" ,
"ReclamationAbiquiuAccountDeliveryHeronToReclamationHeronReclamationAbiquiuHeronSeepage.Supply" ,
"ReclamationAbiquiuAccountFillHeronToReclamationHeronReclamationAbiquiuHeronSeepage.Supply" ,
"ReclamationElVadoAccountDeliveryHeronToReclamationHeronReclamationElVadoHeronSeepage.Supply" ,
"ReclamationElVadoAccountFillHeronToReclamationHeronReclamationElVadoHeronSeepage.Supply" ,
"ReclamationHeronToReclamationHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"ReclamationHeronToReclamationHeronMRGCDAbiquiuHeronSeepage.Supply" ,
"ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" ,
"RedRiverHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"RedRiverHeronToRedRiverHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"RedRiverHeronToRedRiverHeronRGOtowiHeronSeepage.Supply" ,
"RedRiverHeronToRedRiverHeronSantaFeElVadoHeronSeepage.Supply" ,
"SanJuanPuebloHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"SanJuanPuebloHeronToSanJuanPuebloHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"SanJuanPuebloHeronToReclamationHeronReclamationAbiquiuHeronSeepage.Supply" ,
"SanJuanPuebloHeronToSanJuanPuebloHeronRGOtowiHeronSeepage.Supply" ,
"SantaFeAbiquiuAccountDeliveryHeronToSantaFeHeronSantaFeAbiquiuHeronSeepage.Supply" ,
"SantaFeAbiquiuAccountFillHeronToSantaFeHeronSantaFeAbiquiuHeronSeepage.Supply" ,
"SantaFeAbiquiuWaiverHeronToSantaFeHeronSantaFeAbiquiuHeronSeepage.Supply" ,
"SantaFeElVadoAccountDeliveryHeronToSantaFeHeronSantaFeElVadoHeronSeepage.Supply" ,
"SantaFeElVadoAccountFilHeronToSantaFeHeronSantaFeElVadoHeronSeepage.Supply" ,
"SantaFeElVadoExchangeHeronToSantaFeHeronSantaFeElVadoHeronSeepage.Supply" ,
"SantaFeElVadoWaiverHeronToSantaFeHeronSantaFeElVadoHeronSeepage.Supply" ,
"SantaFeHeronToSantaFeHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"SantaFeHeronToSantaFeHeronSantaFeAbiquiuHeronSeepage.Supply" ,
"SantaFeHeronToSantaFeHeronSantaFeElVadoHeronSeepage.Supply" ,
"SantaFeHeronToSantaFeHeronRGOtowiHeronSeepage.Supply" ,
"SantaFeHeronToLosAlamosHeronLosAlamosElVadoHeronSeepage.Supply" ,
"SantaFeHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"TaosAbiquiuAccountDeliveryHeronToTaosHeronTaosAbiquiuHeronSeepage.Supply" ,
"TaosAbiquiuAccountFillHeronToTaosHeronTaosAbiquiuHeronSeepage.Supply" ,
"TaosAbiquiuWaiverHeronToTaosHeronTaosAbiquiuHeronSeepage.Supply" ,
"TaosElVadoAccountDeliveryHeronToTaosHeronTaosElVadoHeronSeepage.Supply" ,
"TaosElVadoAccountFillHeronToTaosHeronTaosElVadoHeronSeepage.Supply" ,
"TaosElVadoExchangeHeronToTaosHeronTaosElVadoHeronSeepage.Supply" ,
"TaosElVadoWaiverHeronToTaosHeronTaosElVadoHeronSeepage.Supply" ,
"TaosHeronToTaosHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"TaosHeronToTaosHeronTaosAbiquiuHeronSeepage.Supply" ,
"TaosHeronToTaosHeronTaosElVadoHeronSeepage.Supply" ,
"TaosHeronToTaosHeronRGOtowiHeronSeepage.Supply" ,
"TaosHeronToTaosHeronSantaFeElVadoHeronSeepage.Supply" ,
"TaosHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"TwiningAbiquiuAccountDeliveryHeronToTwiningHeronTwiningAbiquiuHeronSeepage.Supply" ,
"TwiningAbiquiuAccountFillHeronToTwiningHeronTwiningAbiquiuHeronSeepage.Supply" ,
"TwiningAbiquiuWaiverHeronToTwiningHeronTwiningAbiquiuHeronSeepage.Supply" ,
"TwiningElVadoAccountDeliveryHeronToTwiningHeronTwiningElVadoHeronSeepage.Supply" ,
"TwiningElVadoAccountFillHeronToTwiningHeronTwiningElVadoHeronSeepage.Supply" ,
"TwiningElVadoExchangeHeronToTwiningHeronTwiningElVadoHeronSeepage.Supply" ,
"TwiningElVadoWaiverHeronToTwiningHeronTwiningElVadoHeronSeepage.Supply" ,
"TwiningHeronToTwiningHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"TwiningHeronToTwiningHeronTwiningAbiquiuHeronSeepage.Supply" ,
"TwiningHeronToTwiningHeronTwiningElVadoHeronSeepage.Supply" ,
"TwiningHeronToTwiningHeronRGOtowiHeronSeepage.Supply" ,
"TwiningHeronToTwiningHeronSantaFeElVadoHeronSeepage.Supply" ,
"TwiningHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"UncontractedHeronToMRGCDHeronMRGCDElVadoHeronSeepage.Supply" ,
"UncontractedHeronToUncontractedHeronAlbuquerqueAbiquiuHeronSeepage.Supply" ,
"UncontractedHeronToUncontractedHeronRGOtowiHeronSeepage.Supply" ,
"UncontractedHeronToUncontractedHeronSantaFeElVadoHeronSeepage.Supply" };

    END;

    FUNCTION       "AbiquiuMRGCDInflowSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "MRGCDHeronMRGCDAbiquiuElVadoToAbiquiuLocalInflowToMRGCDAbiquiu.Supply" ,
"MRGCDElVadoMRGCDAbiquiuElVadoToAbiquiuLocalInflowToMRGCDAbiquiu.Supply" ,
```

```
"ReclamationHeronMRGCDAbiquiuElVadoToAbiquiuLocalInflowToMRGCDAbiquiu.Supply" ,
"ReclamationElVadoMRGCDAbiquiuElVadoToAbiquiuLocalInflowToMRGCDAbiquiu.Supply" ,
"AlbuquerqueHeronMRGCDAbiquiuElVadoToAbiquiuLocalInflowToMRGCDAbiquiu.Supply" };

    END;

    FUNCTION       "AbiquiuMRGCDOtherInflowSupplies" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "ReclamationAbiquiuToMRGCDAbiquiu.Supply" };

    END;

    FUNCTION       "ElVadoThruCochitiFlowthruAccounts" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      { "AlbuquerqueHeronAlbuquerqueMiddleValleyDiversions" , "AlbuquerqueHeronAlbuquerqueEB" ,
"CochitiRecPoolHeronCochitiRecPoolCochiti" };

    END;

  END;

  UTILITY_GROUP "Account Summing Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "Average3DayRioGrandeInflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" < @"Start Timestep" + 2.00000000 ["day"] )
   THEN
      "PreviousAccountInflow"( "RioGrande", reservoir )
   ELSE
      IF ( ( STRINGIFY reservoir ) == "ElVado" )
      THEN
         "ForecastElVadoInflow"( @"Current Timestep", "DatePlusXTimesteps"( @"Current Timestep",
2.00000000 ["day"] ) ) / 3.00000000 ["day"]
      ELSE
         IF ( ( STRINGIFY reservoir ) == "Abiquiu" )
         THEN
            IF ( "IrrigationSeason"(  ) )
            THEN
               "RoutedRGFlowElVadoToAbiquiu"( @"Current Timestep", $ "ElVado.Outflow" [] )
            ELSE
               ( $ "Abiquiu^RioGrande.Inflow" [@"Previous Timestep"] +
"RoutedRGFlowElVadoToAbiquiu"( @"Current Timestep", $ "ElVado.Outflow" [] ) * 2.00000000 ) /
3.00000000
            ENDIF
         ELSE
            IF ( ( STRINGIFY reservoir ) == "Cochiti" )
            THEN
               IF ( "IrrigationSeason"(  ) )
               THEN
                  "RoutedRGFlowAbiquiuToCochiti"(  )
               ELSE
                  ( ( "Cochiti^RioGrande.Inflow" [@"Previous Timestep - 1 Timesteps"] +
"Cochiti^RioGrande.Inflow" [@"Previous Timestep"] ) + "RoutedRGFlowAbiquiuToCochiti"(  ) ) /
3.00000000
               ENDIF
```

```
            ELSE
               IF ( ( STRINGIFY reservoir ) == "Jemez" )
               THEN
                  "SumFlowsToVolume"( "SlotifyString"( "Jemez^RioGrande.Inflow" ), @"Previous
Timestep - 2 Timesteps", @"Previous Timestep" ) / 3.00000000 ["day"]
               ELSE
                  0.00000000 ["cfs"]
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "SumAccountStorageUpstreamOfAbiquiu" ( LIST accounts )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      IF ( account IN "ElVadoStorageAccounts"(  ) )
      THEN
         "PreviousAccountStorage"( account, % "Heron" ) + "PreviousAccountStorage"( account, %
"ElVado" ) + result
      ELSE
         "PreviousAccountStorage"( account, % "Heron" ) + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "SumFlowThruAccounts" ( LIST accounts, OBJECT upstreamObject, OBJECT
downstreamObject )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( accounts, accounts, upstreamObject,
downstreamObject ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      IF ( IsNaN supply [@"Previous Timestep"] )
      THEN
         result + 0.00000000 ["cfs"]
      ELSE
         result + supply [@"Previous Timestep"]
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "SumFlowThruAccountStorage" ( LIST accounts, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      result + "PreviousAccountStorage"( account, reservoir )
   ENDFOR;

    END;

    FUNCTION       "SumAccountStorage" ( OBJECT reservoir, LIST accounts )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      result + "PreviousAccountStorage"( account, reservoir )
   ENDFOR;

    END;

    FUNCTION       "SumIntraReservoirTransfers" ( LIST supplies )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      IF ( IsNaN supply [] )
      THEN
         result
      ELSE
         result + supply []
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "TotalAccountStorage" ( LIST accounts )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      IF ( account IN "ElVadoAbiquiuCommonStorageAccounts"(  ) )
      THEN
         "PreviousAccountStorage"( account, % "Heron" ) + "PreviousAccountStorage"( account, %
"ElVado" ) + "PreviousAccountStorage"( account, % "Abiquiu" ) + result
      ELSE
         IF ( account IN "StorageInElVadoOnlyAccounts"(  ) )
         THEN
            "PreviousAccountStorage"( account, % "Heron" ) + "PreviousAccountStorage"( account, %
"ElVado" ) + result
         ELSE
            IF ( account IN "StorageInAbiquiuOnlyAccounts"(  ) )
            THEN
               "PreviousAccountStorage"( account, % "Heron" ) + "PreviousAccountStorage"(
account, % "Abiquiu" ) + result
            ELSE
               "PreviousAccountStorage"( account, % "Heron" ) + result
            ENDIF
         ENDIF
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "TotalWaiverBalance" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST accounts = "HeronDownstreamWaiverStorageAccounts"(  ) DO
      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
         "HeronData." CONCAT account CONCAT "WaiverBalance" [@"Previous Timestep"] + result
      ENDFOR
   ENDWITH;

    END;
```

```
    FUNCTION       "SumSupplies" ( LIST supplies )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      IF ( IsNaN supply [] )
      THEN
         0.00000000 ["cfs"] + result
      ELSE
         supply [] + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "SumPreviousSupplies" ( LIST supplies )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      IF ( IsNaN supply [@"Previous Timestep"] )
      THEN
         0.00000000 ["cfs"] + result
      ELSE
         supply [@"Previous Timestep"] + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "SumAvailableAccountStorage" ( LIST accounts, OBJECT upstreamReservoir, OBJECT
downstreamReservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      "Max"( "Min"( "PreviousAccountStorage"( account, upstreamReservoir ),
"AvailableAccountStorage"( account, downstreamReservoir ) ), 0.00000000 ["acre-feet"] ) + result
   ENDFOR;

    END;

    FUNCTION       "SumCurrentFlowThruAccounts" ( LIST upstreamAccounts, LIST downstreamAccounts,
OBJECT upstreamObject, OBJECT downstreamObject )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( upstreamAccounts, downstreamAccounts,
upstreamObject, downstreamObject ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      IF ( IsNaN supply [] )
      THEN
         result + 0.00000000 ["cfs"]
      ELSE
         result + supply []
      ENDIF
   ENDFOR;

    END;
```

```
    FUNCTION       "SumAccountStorageNonNegative" ( OBJECT reservoir, LIST accounts )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      result + "Max"( "PreviousAccountStorage"( account, reservoir ), 0.00000000 ["acre-feet"] )
   ENDFOR;

    END;

  END;

  UTILITY_GROUP "Caballo Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CaballoChannelCapacity" ( STRING beginLocation, STRING endLocation, SLOT
slot, NUMERIC No.OfDaysDS )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( beginLocation CONCAT "To"
CONCAT endLocation, slot, 0.00000000 ["cfs"], "ComputeMaxOutflow"( % "Caballo" ), {  },
"ObjectifyString"( endLocation ) & "Gage Inflow", "OffsetDate"( @"Current Timestep", No.OfDaysDS,
"1 days" ), $ "CaballoData.ChannelCapacities" ["Capacity", "ElPaso"], $
"CaballoData.ChannelCapacities" ["Tolerance", "ElPaso"], 20.00000000 );

    END;

    FUNCTION       "ComputeCaballoFloodRelease" ( OBJECT reservoir, NUMERIC inflow, NUMERIC
outflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "ComputeMaxOutflow"( reservoir );
    BEGIN

      IF ( "IsReservoirRising"( reservoir ) )
   THEN
      IF ( inflow <= 5000.00000000 ["cfs"] )
      THEN
         "Min"( "Max"( inflow, ( STRINGIFY reservoir ) CONCAT ".Outflow" [@"Previous Timestep"]
), "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $ "BlwCaballo.Gage Inflow", $
"CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] ) )
      ELSE
         "Max"( ( STRINGIFY reservoir ) CONCAT ".Outflow" [@"Previous Timestep"],
"ComputeCaballoFloodReleaseBasedOnElevation"( reservoir, inflow, outflow ) )
      ENDIF
   ELSE
      "ComputeCaballoFloodReleaseBasedOnElevation"( reservoir, inflow, outflow )
   ENDIF;

    END;

    FUNCTION       "ComputeCaballoFloodReleaseBasedOnElevation" ( OBJECT reservoir, NUMERIC
inflow, NUMERIC outflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "ComputeMaxOutflow"( reservoir );
```

```
    BEGIN

      IF ( "StorageToElevationBasedOnCurrentInflow"( reservoir, inflow, outflow ) > ( STRINGIFY
reservoir ) CONCAT "Data.PoolLevels" ["Max", "Elevation"] )
   THEN
      "ComputeMaxOutflow"( reservoir )
   ELSE
      IF ( "StorageToElevationBasedOnCurrentInflow"( reservoir, inflow, outflow ) > 4180.00000000
["feet"] )
      THEN
         "Max"( "Min"( "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $ "BlwCaballo.Gage
Inflow", $ "CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] ), 10000.00000000 ["cfs"] ),
"RoundFlow"( inflow * 0.50000000, 500.00000000 ["cfs"] ) )
      ELSE
         IF ( "StorageToElevationBasedOnCurrentInflow"( reservoir, inflow, outflow ) >
4177.00000000 ["feet"] )
         THEN
            "Max"( "Min"( "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $ "BlwCaballo.Gage
Inflow", $ "CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] ), 5000.00000000 ["cfs"] ),
"RoundFlow"( inflow * 0.30000000, 500.00000000 ["cfs"] ) )
         ELSE
            "Max"( "Min"( "CaballoChannelCapacity"( "BlwCaballo", "ElPaso", $ "BlwCaballo.Gage
Inflow", $ "CaballoData.ApproxNoOfDaysDS" ["ElPaso", "Days"] ), "Min"( 2500.00000000 ["cfs"],
"OutflowToGetBelowFloodPool"( reservoir ) ) ), "RoundFlow"( inflow * 0.20000000, 500.00000000
["cfs"] ) )
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "RoundFlow" ( NUMERIC input, NUMERIC round )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "Fraction"( input, round ) >= 0.50000000 ["cfs"] )
   THEN
      "Ceiling"( input, round )
   ELSE
      "Floor"( input, round )
   ENDIF;

    END;

    FUNCTION       "CheckReservourOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "MinItem"( { "ComputeMaxOutflow"( reservoir ) , "ComputeOutflowToGetToZeroStorage"(
reservoir ) , "GetMaxReleaseGivenInflow"( reservoir, reservoir & "Inflow" [], @"Current Timestep"
) } );

    END;

  END;

  UTILITY_GROUP "Cochiti Account Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CochitiSJOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      ( ( "SumPreviousSupplies"( "AllAbiquiuSJSupplies"(  ) ) - "PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiAbiquiuToCochitiRecPoolHeronCochitiRecPoolCochitiBlwAbiq
uiu.Supply" ) ) - "PreviousSlotInflow"(
"CochitiRecPoolAbiquiuToCochitiRecPoolAbiquiuCochitiRecPoolCochitiBlwAbiquiu.Supply" ) ) * (
1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) );

    END;

    FUNCTION       "CochitiRGStorageAdjustment" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( $ "Cochiti.Incidental Content" [@"Previous Timestep"] < 0.00000000 ["acre-feet"] )
   THEN
      "Min"( "Abs"( "VolumeToFlow"( $ "Cochiti.Incidental Content" [@"Previous Timestep"],
@"Current Timestep" ) ), "MaxIncidentalContentRelease"( % "Cochiti" ) ) * - 1.00000000
   ELSE
      "Min"( "VolumeToFlow"( $ "Cochiti.Incidental Content" [@"Previous Timestep"], @"Current
Timestep" ), "MaxIncidentalContentRelease"( % "Cochiti" ) )
   ENDIF;

    END;

    FUNCTION       "CochitiRGOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "CorpsProjectsRGOutflow"( % "Cochiti" );

    END;

    FUNCTION       "MaxCochitiRecPoolVolume" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( % "Cochiti", "TableInterpolation"( $ "Cochiti.Elevation Area Table",
1.00000000, 0.00000000, 1200.00000000 ["acre"], @"Current Timestep" ) );

    END;

    FUNCTION       "ComputeCochitiMinimumFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "MaxItem"( { "ZeroNaNs"( $ "MiddleValleyDemands.MinReleaseForCentral", @"Current Timestep"
) , "ZeroNaNs"( $ "MiddleValleyDemands.MinReleaseForIsleta", @"Current Timestep" ) , "ZeroNaNs"(
$ "MiddleValleyDemands.MinReleaseForSanAcacia", @"Current Timestep" ) , "ZeroNaNs"( $
"MiddleValleyDemands.MinReleaseForSanMarcial", @"Current Timestep" ) , $
"MiddleValleyDemands.MRGCD" [] } );

    END;

    FUNCTION       "InitialCochitiOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG FALSE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Cochiti" );
    BEGIN

      IF ( IsNaN "TotalOutflow"( % "Cochiti" ) )
   THEN
      IF ( "PreviousOutflow"( % "Abiquiu" ) >= $ "AbiquiuData.MRGCDDemand" [@"Previous Timestep"]
+ $ "AbiquiuData.MinFlowsDemand" [@"Previous Timestep"] AND $ "Cochiti.Inflow" [] < 1500.00000000
["cfs"] AND "IrrigationSeason"(  ) )
      THEN
         "Max"( "SolveOutflow"( % "Cochiti", $ "Cochiti.Inflow" [], "PreviousAccountStorage"(
"CochitiRecPool", % "Cochiti" ) + "PreviousAccountGainLoss"( "CochitiRecPool", % "Cochiti" ) +
"FlowToVolume"( "PreviousSlotInflow"(
"CochitiRecPoolAbiquiuToCochitiRecPoolAbiquiuCochitiRecPoolCochitiBlwAbiquiu.Supply" ), @"Current
Timestep" ) + "FlowToVolume"( "PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiAbiquiuToCochitiRecPoolHeronCochitiRecPoolCochitiBlwAbiq
uiu.Supply" ), @"Current Timestep" ), "PreviousAccountStorage"( "CochitiRecPool", % "Cochiti" ),
@"Current Timestep" ) + "CochitiRGStorageAdjustment"(  ), "SJOutflow"( % "Cochiti" ) +
"MinRGOutflow"( % "Cochiti" ) )
      ELSE
         IF ( "PreviousOutflow"( % "Abiquiu" ) - "PreviousSlotInflow"(
"CochitiRecPoolAbiquiuToCochitiRecPoolAbiquiuCochitiRecPoolCochitiBlwAbiquiu.Supply" ) -
"PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiAbiquiuToCochitiRecPoolHeronCochitiRecPoolCochitiBlwAbiq
uiu.Supply" ) == $ "AbiquiuData.MRGCDDemand" [@"Previous Timestep"] AND "IrrigationSeason"(  ) )
         THEN
            $ "MiddleValleyDemands.MRGCD" []
         ELSE
            "Max"( "SolveOutflow"( % "Cochiti", $ "Cochiti.Inflow" [], "PreviousAccountStorage"(
"CochitiRecPool", % "Cochiti" ) + "PreviousAccountGainLoss"( "CochitiRecPool", % "Cochiti" ) +
"FlowToVolume"( "PreviousSlotInflow"(
"CochitiRecPoolAbiquiuToCochitiRecPoolAbiquiuCochitiRecPoolCochitiBlwAbiquiu.Supply" ), @"Current
Timestep" ) + "FlowToVolume"( "PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiAbiquiuToCochitiRecPoolHeronCochitiRecPoolCochitiBlwAbiq
uiu.Supply" ), @"Current Timestep" ), "PreviousAccountStorage"( "CochitiRecPool", % "Cochiti" ),
@"Current Timestep" ) + "CochitiRGStorageAdjustment"(  ), "SJOutflow"( % "Cochiti" ) +
"MinRGOutflow"( % "Cochiti" ) )
         ENDIF
      ENDIF
   ELSE
      "TotalOutflow"( % "Cochiti" )
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Cochiti Channel Capacity Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CochitiReleaseForCentralChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BlwCochitiToCentral",
$ "BlwCochitiDiversionsReach.Outflow", "MinOutflow"( % "Cochiti" ), "MaxOutflow"( % "Cochiti" ),
"EstimatedJemezReleaseList"( 0.00000000 ["day"] ), $ "Central.Gage Inflow", @"Current Timestep" +
0.00000000 ["day"], "ChannelCapacity"( % "Cochiti", "Central" ), "ChannelCapacityTolerance"( %
"Cochiti", "Central" ), 20.00000000 ), GET NUMERIC @INDEX 1.00000000 FROM
"HypTargetSimWithStatus"( "BlwCochitiToCentral", $ "BlwCochitiDiversionsReach.Outflow",
"MinOutflow"( % "Cochiti" ), "MaxOutflow"( % "Cochiti" ), "EstimatedJemezReleaseList"( 1.00000000
["day"] ), $ "Central.Gage Inflow", @"Current Timestep" + 1.00000000 ["day"], "ChannelCapacity"(
% "Cochiti", "Central" ), "ChannelCapacityTolerance"( % "Cochiti", "Central" ), 20.00000000 ) );

    END;

    FUNCTION       "CochitiToCentralLocalInflows" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "Galisteo.Gage Inflow" [] + $ "BlwCochitiToSanFelipeLocalInflow.Local Inflow" [] + (
"CorpsProjectsRGOutflow"( % "Jemez" ) + "SJOutflow"( % "Jemez" ) ) + $ "NorthFloodwayChannel.Gage
Inflow" [] + $ "SanFelipeToCentralLocalInflow.Local Inflow" [];

    END;

    FUNCTION       "CochitiToCentralPreviousLocalInflows" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "Galisteo.Gage Inflow" [@"Previous Timestep"] + $ "BlwCochitiToSanFelipeLocalInflow.Local
Inflow" [@"Previous Timestep"] + $ "Jemez.Outflow" [@"Previous Timestep"] + $
"NorthFloodwayChannel.Gage Inflow" [@"Previous Timestep"] + $
"SanFelipeToCentralLocalInflow.Local Inflow" [@"Previous Timestep"];

    END;

    FUNCTION       "CochitiToElephantButteLocalInflows" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "SanMarcialToElephantButteLocalInflow.Local Inflow" [@"Current Timestep + 2 Timesteps"] +
"CochitiToSanMarcialLocalInflows"(  );

    END;

    FUNCTION       "CochitiToSanMarcialLocalInflows" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "SouthDiversionChannel.Gage Inflow" [] + $ "CentralToBernardoLocalInflow.Local Inflow"
[@"Next Timestep"] + $ "RioPuerco.Gage Inflow" [@"Next Timestep"] + $
"BernardoToSanAcaciaLocalInflow.Local Inflow" [@"Next Timestep"] + $
"SanAcaciaToSanMarcialLocalInflow.Local Inflow" [@"Next Timestep"] +
"CochitiToCentralLocalInflows"(  );

    END;

    FUNCTION       "CochitiReleaseForBlwElephantButteChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "Central.Gage Inflow" [] < "ChannelCapacity"( % "Cochiti", "Central" ) )
   THEN
      $ "Cochiti.Outflow" []
   ELSE
      IF ( ( "CochitiToElephantButteLocalInflows"(  ) > $ "CochitiData.ChannelCapacities"
["Capacity", "BlwElephantButte"] ) OR ( "CochitiBlwElephantButteChannelCapacity"(  ) < $
"CochitiData.ChannelCapacities" ["Capacity", "Minimum"] ) )
      THEN
         $ "CochitiData.ChannelCapacities" ["Capacity", "Minimum"]
      ELSE
         "CochitiBlwElephantButteChannelCapacity"(  )
      ENDIF
   ENDIF;
```

```
    END;

    FUNCTION       "CochitiBlwElephantButteChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "Cochiti.Outflow" [] - ( "CochitiToElephantButteLocalInflows"(  ) - $
"CochitiData.ChannelCapacities" ["Capacity", "BlwElephantButte"] );

    END;

    FUNCTION       "CochitiFlowToMatchCentralChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "Central.Gage Inflow" [] > "ChannelCapacity"( % "Cochiti", "Central" ) )
   THEN
      IF ( "IsReleaseToMatchChannelCapacity<0"( % "Cochiti", "Central", $ "Central.Gage Inflow"
[] ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez", "Central", $ "Central.Gage Inflow" [] )
)
      THEN
         "ChannelCapacity"( % "Cochiti", "Minimum" )
      ELSE
         IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "Central", $ "Central.Gage
Inflow" [] ) AND "IsReleaseToMatchChannelCapacity>=0"( % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) )
         THEN
            $ "Cochiti.Outflow" []
         ELSE
            IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "Central", $ "Central.Gage
Inflow" [] ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) )
            THEN
               "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $
"Central.Gage Inflow" [] ) - $ "Jemez.Outflow" []
            ELSE
               $ "Cochiti.Outflow" []
            ENDIF
         ENDIF
      ENDIF
   ELSE
      "FlowToMatchChannelCapacity"( % "Cochiti", "Central", $ "Central.Gage Inflow" [] )
   ENDIF;

    END;

    FUNCTION       "CochitiReleaseForChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( $ "CochitiData.MaxReleaseForCentralChannelCap" [], $
"CochitiData.MaxReleaseForSanMarcialChannelCap" [] );

    END;

    FUNCTION       "CochitiReleaseForSanMarcialChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      "Min"( GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"(
"BlwCochitiToSanMarcial", $ "BlwCochitiDiversionsReach.Outflow", "MinOutflow"( % "Cochiti" ),
"MaxOutflow"( % "Cochiti" ), "EstimatedJemezReleaseList"( 1.00000000 ["day"] ), $
"SanMarcialFloodway.Gage Inflow", @"Current Timestep" + 1.00000000 ["day"], "ChannelCapacity"( %
"Cochiti", "SanMarcialFloodway" ), "ChannelCapacityTolerance"( % "Cochiti", "SanMarcialFloodway"
), 20.00000000 ), GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"(
"BlwCochitiToSanMarcial", $ "BlwCochitiDiversionsReach.Outflow", "MinOutflow"( % "Cochiti" ),
"MaxOutflow"( % "Cochiti" ), "EstimatedJemezReleaseList"( 2.00000000 ["day"] ), $
"SanMarcialFloodway.Gage Inflow", @"Current Timestep" + 2.00000000 ["day"], "ChannelCapacity"( %
"Cochiti", "SanMarcialFloodway" ), "ChannelCapacityTolerance"( % "Cochiti", "SanMarcialFloodway"
), 20.00000000 ) );

    END;

  END;

  UTILITY_GROUP "Cochiti Flood Control Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CochitiFCOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "Cochiti.Unregulated Spill" [] < "ChannelCapacity"( % "Cochiti", "Central" ) )
   THEN
      "CochitiFlowToMatchCentralChannelCapacity"(  )
   ELSE
      $ "Cochiti.Outflow" []
   ENDIF;

    END;

    FUNCTION       "CochitiFloodSpace" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( % "Cochiti", $ "CochitiData.PoolLevels" ["Elevation",
"ElevAt7KcfsOverSpillway"] ) - $ "Cochiti.Hold Pool" [@"Previous Timestep"];

    END;

    FUNCTION       "CochitiFloodSpaceAvailable" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( "ElevationToStorage"( % "Cochiti", $ "CochitiData.PoolLevels" ["Elevation",
"SummerFloodPool"] ) + $ "CochitiData.MinimumFloodSpaceForFloodCarryOver" [0.00000000,
0.00000000] ) - "EstimatedRGStorageLookAhead"( % "Cochiti" );

    END;

    FUNCTION       "IfCochitiFloodSpaceAvailable" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      "CochitiFloodSpaceAvailable"(  ) > $ "CochitiData.MinimumFloodSpaceForFloodCarryOver"
[0.00000000, 0.00000000];

    END;

    FUNCTION       "IfFloodCarryOverAtCochiti" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "IfCochitiFloodSpaceAvailable"(  ) AND ( "EstimatedRGStorageLookAhead"( % "Cochiti" ) >
"MinRGCarryOverStorage"( reservoir ) );

    END;

    FUNCTION       "CochitiComputedMaxOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "CochitiReleaseForChannelCapacity"(  ), "DetermineSteppedRelease"( % "Cochiti" ) );

    END;

  END;

  UTILITY_GROUP "Cochiti Jemez WCM Balance Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "IfCochitiJemezBalancedOperation" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "PreviousStorage"( % "Cochiti" ) > "MinCochitiStorageForBalancedOperation"(  ) ) AND (
"PreviousStorage"( % "Jemez" ) > "MinJemezStorageForBalancedOperation"(  ) );

    END;

    FUNCTION       "CochitiJemezStorageDifferentialRatio" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( "CochitiFloodSpace"(  ) - ( "PreviousStorage"( % "Cochiti" ) - $ "Cochiti.Hold Pool"
[@"Previous Timestep"] ) ) / "CochitiFloodSpace"(  ) - ( "JemezFloodSpace"(  ) - (
"PreviousStorage"( % "Jemez" ) - "ElevationToStorage"( % "Jemez", $ "JemezData.PoolLevels"
["Elevation", "TopOfFloodControlPool"] ) ) ) / "JemezFloodSpace"(  );

    END;

    FUNCTION       "CochitiBalancedOperation" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Cochiti" );
    BEGIN
```

```
      IF ( "CochitiJemezStorageDifferentialRatio"(  ) < 0.00000000 )
   THEN
      ( "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) - ( "CochitiJemezStorageDifferentialRatio"(  ) *
"CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $ "Central.Gage Inflow"
[] ) ) ) / 2.00000000
   ELSE
      IF ( "CochitiJemezStorageDifferentialRatio"(  ) > 0.00000000 )
      THEN
         "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) - "JemezFlowToMatchCentralChannelCapacity"(  )
      ELSE
         $ "Cochiti.Outflow" []
      ENDIF
   ENDIF;

    END;

    FUNCTION       "JemezBalancedOperation" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Jemez" );
    BEGIN

      IF ( "CochitiJemezStorageDifferentialRatio"(  ) < 0.00000000 )
   THEN
      "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) - "CochitiFlowToMatchCentralChannelCapacity"(  )
   ELSE
      IF ( "CochitiJemezStorageDifferentialRatio"(  ) > 0.00000000 )
      THEN
         ( "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $
"Central.Gage Inflow" [] ) + ( "CochitiJemezStorageDifferentialRatio"(  ) *
"CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $ "Central.Gage Inflow"
[] ) ) ) / 2.00000000
      ELSE
         $ "Jemez.Outflow" []
      ENDIF
   ENDIF;

    END;

    FUNCTION       "MinCochitiStorageForBalancedOperation" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "CochitiFloodSpace"(  ) * $ "CochitiData.PercentOfFloodSpace" ["Cochiti&Jemez",
"Percent"] ) + $ "Cochiti.Hold Pool" [@"Previous Timestep"];

    END;

    FUNCTION       "MinJemezStorageForBalancedOperation" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "JemezFloodSpace"(  ) * $ "CochitiData.PercentOfFloodSpace" ["Cochiti&Jemez", "Percent"]
) + "ElevationToStorage"( % "Jemez", $ "JemezData.PoolLevels" ["Elevation", "SedimentPool"] );

    END;

  END;

  UTILITY_GROUP "Date Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
```

```
BEGIN

  FUNCTION       "DeltaTimeToMarch" (  )
  RETURN_TYPE    NUMERIC;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG FALSE;
  BEGIN

    IF ( @"Current Month" <= @"March" )
 THEN
    @"24:00:00 March 31, Current Year" - @"Current Timestep"
 ELSE
    @"24:00:00 March 31, Next Year" - @"Current Timestep"
 ENDIF;

  END;

  FUNCTION       "DatePlusXTimesteps" ( DATETIME date, NUMERIC timesteps )
  RETURN_TYPE    DATETIME;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG TRUE;
  BEGIN

    "DateMin"( date + timesteps, @"Finish Timestep" );

  END;

  FUNCTION       "FloodCarryOverReleaseSeason" (  )
  RETURN_TYPE    BOOLEAN;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG TRUE;
  BEGIN

    ( @"Current Timestep" >= @"November" OR @"Current Timestep" <= @"March" );

  END;

  FUNCTION       "FloodCarryOverSeason" (  )
  RETURN_TYPE    BOOLEAN;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG TRUE;
  BEGIN

    @"Current Timestep" >= @"July" AND @"Current Timestep" <= @"October";

  END;

  FUNCTION       "FloodCarryOverSeasonLookAhead" (  )
  RETURN_TYPE    BOOLEAN;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG FALSE;
  BEGIN

    "LookAhead"(  ) >= @"July" AND "LookAhead"(  ) <= @"October";

  END;

  FUNCTION       "GetDayOfWeekAsStringGivenDate" ( DATETIME date )
  RETURN_TYPE    STRING;
  SCALE_UNITS    "";
  DESCRIPTION    "";
  ACTIVE         TRUE;
  PRE_EXEC_DIAG  FALSE;
  POST_EXEC_DIAG FALSE;
```

```
    BEGIN

      IF ( date == @"Friday" )
   THEN
      "Friday"
   ELSE
      IF ( date == @"Saturday" )
      THEN
         "Saturday"
      ELSE
         IF ( date == @"Sunday" )
         THEN
            "Sunday"
         ELSE
            IF ( date == @"Monday" )
            THEN
               "Monday"
            ELSE
               IF ( date == @"Tuesday" )
               THEN
                  "Tuesday"
               ELSE
                  IF ( date == @"Wednesday" )
                  THEN
                     "Wednesday"
                  ELSE
                     "Thursday"
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "IrrigationSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( @"Current Timestep" >= @"24:00:00 March 1, Current Year" AND @"Current Timestep" <=
@"24:00:00 October 31, Current Year" );

    END;

    FUNCTION       "IsEndOfMonth" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetDayOfMonth"( @"Current Timestep" ) == "GetDaysInMonth"( @"Current Timestep" );

    END;

    FUNCTION       "IsFirstDayOfMonth" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetDayOfMonth"( @"Current Timestep" ) == 1.00000000 ["day"];

    END;

    FUNCTION       "IsFirstHalfOfMonth" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetDayOfMonth"( @"Current Timestep" ) <= 15.00000000 ["day"];

    END;

    FUNCTION       "IsSecondHalfOfMonth" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetDayOfMonth"( @"Current Timestep" ) > 15.00000000 ["day"];

    END;

    FUNCTION       "LookAhead" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "OffsetDate"( @"Current Timestep", $ "FloodCarryOverData.LookAhead" [0.00000000,
0.00000000], "1 days" );

    END;

    FUNCTION       "LookAhead-1" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "OffsetDate"( @"Current Timestep", $ "FloodCarryOverData.LookAhead" [0.00000000,
0.00000000] - 1.00000000, "1 days" );

    END;

    FUNCTION       "LookAhead-2" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "OffsetDate"( @"Current Timestep", $ "FloodCarryOverData.LookAhead" [0.00000000,
0.00000000] - 2.00000000, "1 days" );

    END;

    FUNCTION       "LookBehindForSteppedRelease" ( OBJECT reservoir )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" - "SteppedReleaseData"( reservoir, "DaysAtMaxStep" );

    END;

    FUNCTION       "NonIrrigationSeason" (  )
```

```
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" >= @"24:00:00 November 1, Current Year" OR @"Current Timestep" <
@"24:00:00 March 1, Current Year";

    END;

    FUNCTION       "SummerEvapSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      @"Current Timestep" >= @"April" AND @"Current Timestep" <= @"October";

    END;

    FUNCTION       "SummerFloodSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" >= @"24:00:00 June 1, Current Year" AND @"Current Timestep" <=
@"24:00:00 October 31, Current Year";

    END;

    FUNCTION       "WinterFloodSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      @"Current Timestep" < @"24:00:00 June 1, Current Year" OR @"Current Timestep" > @"24:00:00
October 31, Current Year";

    END;

    FUNCTION       "EndOfMonthOrRun" ( DATETIME date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "DateMin"( @"24:00:00 Current Month Max DayOfMonth, Current Year", @"Finish Timestep" );

    END;

    FUNCTION       "MinDate" ( DATETIME date, DATETIME date1 )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( date < date1 )
   THEN
      date
```

```
   ELSE
      date1
   ENDIF;

    END;

    FUNCTION       "MaxDate" ( DATETIME date, DATETIME date1 )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( date < date1 )
   THEN
      date1
   ELSE
      date
   ENDIF;

    END;

    FUNCTION       "EndOfSeasonDate" ( OBJECT reservoir, STRING slot )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( @"Current Timestep" ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data." CONCAT slot ["JulianDay", "GetSeasonColumn"( reservoir, slot )] + 1.00000000, "1 days" )
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data." CONCAT slot ["JulianDay", "GetSeasonColumn"( reservoir, slot )], "1 days" )
   ENDIF;

    END;

    FUNCTION       "DateMinusXTimesteps" ( DATETIME date, NUMERIC timesteps )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "DateMax"( date - timesteps, @"Start Timestep" );

    END;

  END;

  UTILITY_GROUP "Demand Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuMinFlowsDemand" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BelowAbiquiu", $
"BlwAbiquiu.Gage Inflow", 0.00000000 ["cfs"], 1800.00000000 ["cfs"], {  }, $
"OtowiToCochitiLocalInflow.Outflow", "DatePlusXTimesteps"( @"Current Timestep", lookAheadDays ),
$ "MiddleValleyDemands.MinimumFlow" ["DatePlusXTimesteps"( @"Current Timestep", lookAheadDays )],
5.00000000 ["cfs"], 20.00000000 );
```

```
    END;

    FUNCTION       "AbiquiuComputedMRGCDDemand" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BelowAbiquiu", $
"BlwAbiquiu.Gage Inflow", 0.00000000 ["cfs"], 1800.00000000 ["cfs"], {  }, $
"OtowiToCochitiLocalInflow.Outflow", "DatePlusXTimesteps"( @"Current Timestep", lookAheadDays ),
$ "MiddleValleyDemands.MRGCD" ["DatePlusXTimesteps"( @"Current Timestep", lookAheadDays )],
2.00000000 ["cfs"], 30.00000000 );

    END;

    FUNCTION       "AlbuquerqueRemainingMVDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      IF ( "MRGCDRemainingDemand"(  ) == 0.00000000 ["cfs"] )
   THEN
      "Min"( "Max"( $ "MiddleValleyDemands.MinimumFlow" [] - "Max"( "ForecastCochitiRGOutflow"(
) - $ "MiddleValleyDemands.MRGCD" [], 0.00000000 ["cfs"] ), 0.00000000 ["cfs"] ), "VolumeToFlow"(
"PreviousAccountStorage"( "Albuquerque", % "Abiquiu" ), @"Current Timestep" ) )
   ELSE
      "Min"( $ "MiddleValleyDemands.MinimumFlow" [], "VolumeToFlow"( "PreviousAccountStorage"(
"Albuquerque", % "Abiquiu" ), @"Current Timestep" ) )
   ENDIF;

    END;

    FUNCTION       "MRGCDRemainingDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH NUMERIC maxLoan = "MaximumAlbuquerqueLoan"( "MRGCD" ) DO
      WITH NUMERIC estimateOutflow = "EstimateMRGCDCochitiOutflow"(  ) DO
         IF ( maxLoan > estimateOutflow )
         THEN
            estimateOutflow
         ELSE
            "Min"( estimateOutflow, maxLoan )
         ENDIF
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "SubbasinDiversionRequirement" ( STRING subbasin, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 0.00000000 FROM "SolveSubBasinDiversions"( "ListSubbasin"( subbasin ),
date );

    END;

    FUNCTION       "SubbasinOutflow" ( STRING subbasin, DATETIME date )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "SolveSubBasinDiversions"( "ListSubbasin"( subbasin ),
date );

    END;

    FUNCTION       "LetterWaterAdjustment" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "Abiquiu" )
   THEN
      "Min"( "VolumeToFlow"( "GetObjectDebt"( % "Abiquiu", @"Current Timestep" ), @"Current
Timestep" ), "MaxDeliveryRequestRelease"(  ) )
   ELSE
      IF ( ( STRINGIFY reservoir ) == "ElVado" )
      THEN
         "Min"( FOR ( STRING supply IN "MakeSupplyList"( "ElVadoOtowiEXAccountList"(  ),
"MakeOtowiPaybackAccountList"( % "ElVado", "ElVadoOtowiEXAccountList"(  ) ), % "ElVado", %
"BlwElVado" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
            result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
         ENDFOR, "MaximumSJOutflow"( % "ElVado" ) ) + "Min"( FOR ( STRING supply IN
"MakeSupplyList"( "HeronOnlyRootOtowiPaybackAccounts"(  ), "MakeOtowiPaybackAccountList"( %
"Heron", "HeronOnlyRootOtowiPaybackAccounts"(  ) ), % "Heron", % "HeronSeepage" ) ) WITH NUMERIC
result = 0.00000000 ["cfs"] DO
            result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
         ENDFOR, "MaximumSJOutflow"( % "Heron" ) )
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "MRGCDAbiquiuSJDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( $ "AbiquiuData.MRGCDDemand" [] - "RGOutflow"( % "Abiquiu" ) -
"LetterWaterAdjustment"( % "Abiquiu", @"Current Timestep" ), 0.00000000 ["cfs"] );

    END;

    FUNCTION       "MRGCDAbiquiuSJDemandLoan" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "PreviousAccountStorage"( "MRGCD", % "Abiquiu" ) >= "FlowToVolume"(
"MRGCDAbiquiuSJDemand"(  ), @"Current Timestep" ) )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "MRGCDAbiquiuSJDemand"(  )
   ENDIF;

    END;

    FUNCTION       "AbiquiuMinRGDemand" ( DATETIME date )
```

```
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST dates = date TO "DatePlusXTimesteps"( date, 2.00000000 ["day"] ) DO
      FOR ( DATETIME sumDate IN dates ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         "SubbasinDiversionRequirement"( "BlwAbiquiuToConfluence", sumDate ) / ( LENGTH dates ) +
result
      ENDFOR
   ENDWITH;

    END;

  END;

  UTILITY_GROUP "Elephant Butte Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EBPrudentRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( "NonIrrigationSeason"(  ) AND "StorageToElevationBasedOnCurrentInflow"( %
"ElephantButte", $ "ElephantButte.Inflow" [], $ "ElephantButte.Outflow" [] ) >
"ComputeEBPrudentElevation"( "Winter" ) ) )
   THEN
      "Max"( "OutflowToGetBelowPrudentPool"( % "ElephantButte", "Winter" ), "DownstreamDemands"(
% "ElephantButte" ) )
   ELSE
      IF ( "IrrigationSeason"(  ) AND "StorageToElevationBasedOnCurrentInflow"( %
"ElephantButte", $ "ElephantButte.Inflow" [], $ "ElephantButte.Outflow" [] ) >
"ComputeEBPrudentElevation"( "Summer" ) )
      THEN
         "Max"( "OutflowToGetBelowPrudentPool"( % "ElephantButte", "Summer" ),
"DownstreamDemands"( % "ElephantButte" ) )
      ELSE
         $ "ElephantButte.Outflow" []
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ElephantButteComputedMaxOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "ElephantButte" );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      "Min"( "ElephantButteReleaseForDSChannelCapacity"(  ), "Min"( "EBPrudentRelease"(  ),
"EBReleaseForCaballo"(  ) ) );

    END;

    FUNCTION       "ElephantButteReleaseForDSChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ChannelCapacity"( % "ElephantButte", "D/S" );
```

```
    END;

    FUNCTION       "EBReleaseForCaballo" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( ( "WinterFloodSeason"(  ) AND "EstimatedCurrentCaballoElevation"(  ) > $
"CaballoData.PoolLevels" ["BottomOfFlood", "Elevation"] ) AND "PreviousElevation"( %
"ElephantButte" ) < "ComputeEBPrudentElevation"( "Winter" ) ) OR ( "SummerFloodSeason"(  ) AND (
"PreviousElevation"( % "ElephantButte" ) < "ComputeEBPrudentElevation"( "Summer" ) AND
"EstimatedCurrentCaballoElevation"(  ) > $ "CaballoData.PoolLevels" ["BottomOfFlood",
"Elevation"] ) ) )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "ElephantButteReleaseForDSChannelCapacity"(  )
   ENDIF;

    END;

    FUNCTION       "ComputeEBPrudentElevation" ( STRING season )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StorageToElevation"( % "ElephantButte", "ElevationToStorage"( % "ElephantButte", $
"ElephantButteData.PoolLevels" ["TOC", 0.00000000] ) - $
"ElephantButteData.PrudentAvailableStorage" [season, 0.00000000] );

    END;

    FUNCTION       "EstimatedCurrentCaballoElevation" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "StorageToElevationBasedOnCurrentInflow"( % "Caballo", $
"ElephantButteToCaballoLocalInflow.Local Inflow" [] + "DownstreamDemands"( % "ElephantButte" ),
"DownstreamDemands"( % "Caballo" ) );

    END;

  END;

  UTILITY_GROUP "ElVado Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ElVadoAccountStorageAdjustment" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" <= "TargetFillDate"( % "ElVado" ) )
   THEN
      IF ( "ElVadoIsPriority"(  ) )
      THEN
         "SumAccountSlotsByWaterType"( % "ElVado", "SanJuan", "Storage", @"Previous Timestep" ) +
"GetAccountDebt"( "AlbuquerqueHeronToAlbuquerqueHeronMRGCDElVadoHeronSeepage.Supply" ) +
"GetAccountDebt"( "ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" ) + "Max"(
```

```
"AccountFillMaxVolume"( % "Heron" ) - $ "HeronData.CumulativeAccountFillRelease" [@"Previous
Timestep"], 0.00000000 ["acre-feet"] )
      ELSE
         "SumAccountSlotsByWaterType"( % "ElVado", "SanJuan", "Storage", @"Previous Timestep" ) +
"GetAccountDebt"( "AlbuquerqueHeronToAlbuquerqueHeronMRGCDElVadoHeronSeepage.Supply" ) +
"GetAccountDebt"( "ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" )
      ENDIF
   ELSE
      IF ( "CurrentYearIsWaiverYear"(  ) )
      THEN
         "SumAccountSlotsByWaterType"( % "ElVado", "SanJuan", "Storage", @"Previous Timestep" ) +
"SumSlot"( $ "HeronData.AlbuquerqueFromMRGCDLoan", @"Current Timestep", "TargetElevationDate"( %
"ElVado", @"Current Timestep" ) )
      ELSE
         "SumAccountSlotsByWaterType"( % "ElVado", "SanJuan", "Storage", @"Previous Timestep" ) +
"SumSlot"( $ "HeronData.AlbuquerqueFromMRGCDLoan", @"Current Timestep", "TargetElevationDate"( %
"ElVado", @"Current Timestep" ) ) + "SumAccountStorage"( % "Heron",
"HeronNonDownstreamStorageAccounts"(  ) ) - "SumAccountStorage"( % "Heron", { "CochitiRecPool" ,
"NambeFalls" } )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "PeliminaryRaftingRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "ComputedElVadoRaftingSchedule"(  ) - "RGOutflow"( % "ElVado" ), 0.00000000 ["cfs"]
);

    END;

    FUNCTION       "IndianStorageRequirementRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "ElVadoData.IndianStorageReqRelAllSwitch" [0.00000000, 0.00000000] == 1.00000000 )
   THEN
      "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
"IndianStorageRequirement"(  ) + "PreviousAccountGainLoss"( "RioGrande", % "ElVado" ) +
"IndianCallStorageAdjustment"(  ), @"Current Timestep" ) + "CurrentRGInflow"( % "ElVado" ),
0.00000000 ["cfs"] ) + "IndianCall"(  )
   ELSE
      "Max"( "Min"( "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
"IndianStorageRequirement"(  ) + "PreviousAccountGainLoss"( "RioGrande", % "ElVado" ) +
"IndianCallStorageAdjustment"(  ), @"Current Timestep" ) + "CurrentRGInflow"( % "ElVado" ),
0.00000000 ["cfs"] ), ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) +
"ForecastElVadoInflow"( @"Current Timestep", "EndOfMonthOrRun"( @"Current Timestep" ) ) -
"IndianStorageRequirement"(  ) + "VolumeOfRelease"( "VolumeToFlow"( "PreviousAccountGainLoss"(
"RioGrande", % "ElVado" ), @"Current Timestep" ), @"Current Timestep", "EndOfMonthOrRun"(
@"Current Timestep" ) ) ) / ( ( "EndOfMonthOrRun"( @"Current Timestep" ) - @"Current Timestep" )
+ 1.00000000 ["day"] ) ), 0.00000000 ["cfs"] ) + "IndianCall"(  )
   ENDIF;

    END;

    FUNCTION       "ComputeElVadoMRGCDDemand" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BelowElVado", $
"BlwElVado.Gage Inflow", 0.00000000 ["cfs"], 5000.00000000 ["cfs"], {  }, $
"ElVadoToAbiquiuLocalInflow.Outflow", "DatePlusXTimesteps"( @"Current Timestep", lookAheadDays ),
```

```
"GetAbiquiuMRGCDDemand"( "DatePlusXTimesteps"( @"Current Timestep", lookAheadDays ) ), 2.00000000
["cfs"], 20.00000000 );

    END;

  END;

  UTILITY_GROUP "ElVado Account Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ElVadoAlbuquerquePaybackRelease" ( LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( accounts,
"MakeAlbuquerqueAbiquiuPaybackAccountsList"( accounts, % "ElVado" ), % "ElVado", % "BlwElVado" )
) WITH LIST result = { { 0.00000000 ["acre-feet"] , 0.00000000 } } DO
      APPEND { "Min"( "Min"( "PreviousAccountStorage"( GET STRING @INDEX GET NUMERIC @INDEX
1.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM result FROM accounts, %
"ElVado" ), "GetAccountDebt"( supply ) ), "Max"( "AccountStorageAvailable"( "Albuquerque", %
"Abiquiu" ) - GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000
FROM result, 0.00000000 ["acre-feet"] ) ) + GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX (
LENGTH result ) - 1.00000000 FROM result , ( GET NUMERIC @INDEX 1.00000000 FROM GET LIST @INDEX (
LENGTH result ) - 1.00000000 FROM result ) + 1.00000000 } ONTO result
   ENDFOR;

    END;

    FUNCTION       "ElVadoRGWinterRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
10.00000000 ["acre-feet"], @"Current Timestep" ), 0.00000000 ["cfs"] );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      "Max"( "ComputeElVadoTargetRelease"(  ), "MinRGOutflow"( % "ElVado" ) ) + "TexasCall"(  );

    END;

    FUNCTION       "ElVadoOtowiDebt" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( NOT "ElVadoRaftingSeason"(  ) AND @"Current Timestep" >= @"24:00:00 May 15, Current
Year" AND @"Current Timestep" <= @"24:00:00 September 10, Current Year" )
   THEN
      0.00000000 ["cfs"]
   ELSE
      FOR ( STRING supply IN "MakeSupplyList"( "ElVadoRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "ElVado", "ElVadoRootOtowiPaybackAccounts"(  ) ), % "ElVado", %
"BlwElVado" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) + result
      ENDFOR
   ENDIF;

    END;

    FUNCTION       "ElVadoOtowiDebtRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( "ElVadoRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "ElVado", "ElVadoRootOtowiPaybackAccounts"(  ) ), % "ElVado", %
"BlwElVado" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) + result
   ENDFOR;

    END;

    FUNCTION       "ElVadoRGRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ElVadoSpringRunoff "(  ) AND NOT "ElVadoRaftingSeason"(  ) )
   THEN
      IF ( "CompactVIIIsInEffect"(  ) )
      THEN
         "Max"( "CurrentRGInflow"( % "ElVado" ), "ElVadoRGSpringRunoffRelease"(  ) )
      ELSE
         "ElVadoRGSpringRunoffRelease"(  )
      ENDIF
   ELSE
      IF ( "ElVadoSummerIrrigationSeason"(  ) )
      THEN
         IF ( "CompactVIIIsInEffect"(  ) )
         THEN
            "Max"( "CurrentRGInflow"( % "ElVado" ), "ElVadoRGSummerRelease"(  ) )
         ELSE
            "ElVadoRGSummerRelease"(  )
         ENDIF
      ELSE
         IF ( "CompactVIIIsInEffect"(  ) )
         THEN
            "Max"( "CurrentRGInflow"( % "ElVado" ), "ElVadoRGWinterRelease"(  ) )
         ELSE
            "ElVadoRGWinterRelease"(  )
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ElVadoRGSummerRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
10.00000000 ["acre-feet"], @"Current Timestep" ), 0.00000000 ["cfs"] );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      WITH NUMERIC minOutflow = "MinRGOutflow"( % "ElVado" ) DO
      WITH NUMERIC targetRelease = "ComputeElVadoTargetRelease"(  ) DO
         "MaxItem"( { minOutflow , targetRelease , "MinComputedElVadoMRGCDRelease"(  ) } ) +
"TexasCall"(  ) + "IndianCall"(  )
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "ElVadoSJStorageAccountsRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      IF ( "ElVadoRaftingSeason"(  ) )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "PeliminaryElVadoSJStorageAccountsRelease"(  )
   ENDIF;

    END;

    FUNCTION       "ElVadoSpringRunoff " (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      @"Current Timestep" >= @"24:00:00 March 15, Current Year" AND @"Current Timestep" <=
"TargetFillDate"( % "ElVado" );

    END;

    FUNCTION       "ElVadoSummerIrrigationSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      @"Current Timestep" > "TargetFillDate"( % "ElVado" ) AND @"Current Month" <= @"October" OR
"ElVadoRaftingSeason"(  );

    END;

    FUNCTION       "ElVadoRGSpringRunoffRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) -
10.00000000 ["acre-feet"], @"Current Timestep" ), 0.00000000 ["cfs"] );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      "Max"( "ComputeElVadoTargetRelease"(  ), "Max"( "MinRGOutflow"( % "ElVado" ),
"MinComputedElVadoMRGCDRelease"(  ) ) );

    END;

    FUNCTION       "TotalElVadoSJRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    " [cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      "Min"( "TotalElVadoFlowThruAccounts"(  ) + "ElVadoOtowiDebtRelease"(  ) +
"ElVadoSJStorageAccountsRelease"(  ) + "ElVadoRaftingRelease"(  ) + "ElVadoMRGCDDebtRelease"(  )
+ "ElVadoReclamationRelease"(  ), "MaximumSJOutflow"( % "ElVado" ) );

    END;

    FUNCTION       "TotalElVadoFlowThruAccounts" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      WITH LIST supplies = FOR ( STRING account IN "ElVadoFlowthruAccounts"(  ) ) WITH LIST list
= {  } DO
      "ElVadoFlowThroughAccountSupplies"( account ) CONCAT list
   ENDFOR DO
      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         result + "SupplyFlow"( supply )
      ENDFOR + "VolumeToFlow"( "SumAccountStorage"( % "ElVado", "ElVadoFlowthruAccounts"(  ) ),
@"Current Timestep" )
   ENDWITH;

    END;

    FUNCTION       "PeliminaryElVadoSJStorageAccountsRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST result = "ElVadoAlbuquerquePaybackRelease"(
"ElVadoRootAlbuquerquePaybackAccounts"(  ) ) DO
      ( GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM
result ) / "Max"( "EndOfSeasonDate"( % "ElVado", "ReleaseTypePriority" ) - @"Current Timestep",
1.00000000 ["day"] )
   ENDWITH;

    END;

    FUNCTION       "ElVadoMRGCDDebtRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", % "ElVado" ),
@"Current Timestep" ), 0.00000000 ["cfs"] ), "Max"( "VolumeToFlow"( "GetAccountDebt"(
"ReclamationElVadoToReclamationElVadoMRGCDAbiquiuBlwElVado.Supply" ), @"Current Timestep" ) -
"SupplyFlow"( "ReclamationHeronToReclamationHeronMRGCDAbiquiuHeronSeepage.Supply" ), 0.00000000
["cfs"] ) );

    END;

    FUNCTION       "ElVadoReclamationRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "PreviousAccountStorage"( "Reclamation", % "ElVado" ) > 0.00000000 ["acre-feet"] )
   THEN
      "MinList"( { "VolumeToFlow"( "PreviousAccountStorage"( "Reclamation", % "ElVado" ),
@"Current Timestep" ) , "VolumeToFlow"( "AvailableAccountStorage"( "Reclamation", % "Abiquiu" ),
@"Current Timestep" ) , $ "ElVadoData.MaximumSJOutflow" ["GetSeasonRow"( % "ElVado",
"MaximumSJOutflow" ), "MaxSJRelease"] } )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "ComputeMinElVadoRGDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "IsIrrigationSeason"(  ) )
```

```
   THEN
      IF ( "ElVadoRaftingSeason"(  ) AND "ComputedElVadoRaftingSchedule"(  ) > 0.00000000 ["cfs"]
)
      THEN
         "Min"( "ElVadoMRGCDRelease"(  ), "ComputedElVadoRaftingSchedule"(  ) )
      ELSE
         "ElVadoMRGCDRelease"(  )
      ENDIF
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "ElVadoMRGCDRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "PreviousAccountStorage"( "MRGCD", % "Abiquiu" ) < "MaxAccountStorage"( "MRGCD", %
"Abiquiu" ) + "MaxAccountStorageTolerance"( % "Abiquiu" ) )
   THEN
      $ "ElVadoData.MRGCDDemand" []
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "MaxElVadoMRGCDSJRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( ( "Max"( "VolumeToFlow"( "PreviousAccountStorage"( "MRGCD", % "ElVado" ), @"Current
Timestep" ), 0.00000000 ["cfs"] ) - "RGOutflow"( % "ElVado" ) ), 0.00000000 ["cfs"] );

    END;

  END;

  UTILITY_GROUP "ElVado Compact Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CompactVIIIsInEffect" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "PreviousAccountStorage"( "RioGrande", % "ElephantButte" ) - "NMDebit"(  ) -
"CODebit"(  ), 0.00000000 ["acre-feet"] ) + "PreviousStorage"( % "Caballo" ) <
"CompactMinStorage"(  );

    END;

    FUNCTION       "ElVadoIndianMet" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "Indian.Indian Call" [] > "MaxOutflow"( % "ElVado" ) )
```

```
   THEN
      "MaxOutflow"( % "ElVado" )
   ELSE
      $ "Indian.Indian Call" []
   ENDIF;

    END;

    FUNCTION       "ElVadoRGCompactVIIndianMet" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) < "NMDebit"(  ) AND
"PreviousAccountStorage"( "RioGrande", % "ElVado" ) > "ElVadoIndianStorage"(  ) )
   THEN
      "Min"( "VolumeToFlow"( "ElVadoIndianStorage"(  ), @"Current Timestep" ), "IndianCall"(  ) )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "ElVadoRGCompactVITexasMet" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) < "NMDebit"(  ) AND
"PreviousAccountStorage"( "RioGrande", % "ElVado" ) > "ElVadoIndianStorage"(  ) )
   THEN
      IF ( "Min"( "TexasCall"(  ), "VolumeToFlow"( "NMDebit"(  ), @"Current Timestep" ) ) >
0.00000000 ["cfs"] )
      THEN
         IF ( "Min"( "TexasCall"(  ), "VolumeToFlow"( "NMDebit"(  ), @"Current Timestep" ) ) >
"VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ), @"Current Timestep" ) -
"ElVadoIndianStorage"(  ) )
         THEN
            "Min"( "VolumeToFlow"( "PreviousAccountStorage"( "RioGrande", % "ElVado" ), @"Current
Timestep" ) - "ElVadoIndianStorage"(  ), "VolumeToFlow"( "NMDebit"(  ), @"Current Timestep" ) )
         ELSE
            "Min"( "TexasCall"(  ), "VolumeToFlow"( "NMDebit"(  ), @"Current Timestep" ) )
         ENDIF
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "ElVadoTexasMet" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "Indian.Indian Call" [] + $ "RioGrandeCompactData.Texas Call" [] > "MaxOutflow"( %
"ElVado" ) )
   THEN
      IF ( $ "Indian.Indian Call" [] > "MaxOutflow"( % "ElVado" ) )
      THEN
         0.00000000 ["cfs"]
      ELSE
         "MaxOutflow"( % "ElVado" ) - $ "Indian.Indian Call" []
      ENDIF
   ELSE
```

```
      $ "RioGrandeCompactData.Texas Call" []
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Estimated Inflow Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EstimatedAbiquiuInflowLookAhead" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   THEN
      "SumSlot"( $ "ElVadoLocalInflow.Local Inflow", @"Current Timestep", "LookAhead"(  ) ) +
"SumSlot"( $ "ElVadoToAbiquiuLocalInflow.Local Inflow", @"Current Timestep", "LookAhead"(  ) )
   ELSE
      ( "SumSlot"( $ "ElVadoLocalInflow.Local Inflow", @"Current Timestep", "LookAhead"(  ) ) +
"SumSlot"( $ "ElVadoToAbiquiuLocalInflow.Local Inflow", @"Current Timestep", "LookAhead"(  ) ) )
* ( 1.00000000 + $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   ENDIF;

    END;

    FUNCTION       "EstimatedCochitiInflowLookAhead" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   THEN
      "EstimatedRGOutflowLookAhead"( % "Abiquiu" ) + "SumSlot"( $ "Embudo.Gage Inflow", @"Current
Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "SantaFeRiverAbvCochiti.Gage Inflow", @"Current
Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "EmbudoToOtowiLocalInflow.Local Inflow", @"Current
Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "OtowiToCochitiLocalInflow.Local Inflow", @"Current
Timestep", "LookAhead"(  ) )
   ELSE
      ( "EstimatedRGOutflowLookAhead"( % "Abiquiu" ) + "SumSlot"( $ "Embudo.Gage Inflow",
@"Current Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "SantaFeRiverAbvCochiti.Gage Inflow",
@"Current Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "EmbudoToOtowiLocalInflow.Local Inflow",
@"Current Timestep", "LookAhead"(  ) ) + "SumSlot"( $ "OtowiToCochitiLocalInflow.Local Inflow",
@"Current Timestep", "LookAhead"(  ) ) ) * ( 1.00000000 + $ "ForecastData.PercentForecastError"
["EndOfMonth"(  )] )
   ENDIF;

    END;

    FUNCTION       "ForecastCochitiRGOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      IF ( IsNaN $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   THEN
      ( ( ( $ "Abiquiu^RioGrande.Outflow" [@"Previous Timestep"] - ( $
"BlwAbiquiuDiversions.Total Diversion Requested" [] - $ "BlwAbiquiuDiversions.Total Depletion
Requested" [] ) ) * ( 1.00000000 + "LossRate"( % "BlwAbiquiuToChamita" ) ) - ( $
"AbvConfluenceDiversions.Total Diversion Requested" [] - $ "AbvConfluenceDiversions.Total
Depletion Requested" [] ) + $ "AbiquiuToChamitaLocalInflow.Local Inflow" [] - ( $
"BlwConfluenceDiversions.Total Diversion Requested" [] - $ "BlwConfluenceDiversions.Total
Depletion Requested" [] ) - ( $ "BlwChamitaDiversions.Total Diversion Requested" [] - $
```

```
"BlwChamitaDiversions.Total Depletion Requested" [] ) + $ "Embudo.Gage Outflow" [@"Previous
Timestep"] * ( 1.00000000 + "LossRate"( % "EmbudoToConfluence" ) ) ) * ( 1.00000000 + "LossRate"(
% "ConfluenceToOtowi" ) ) + $ "EmbudoToOtowiLocalInflow.Local Inflow" [] ) * ( 1.00000000 +
"LossRate"( % "OtowiToCochiti" ) ) + $ "SantaFeRiverAbvCochiti.Gage Inflow" [] + $
"OtowiToCochitiLocalInflow.Local Inflow" []
   ELSE
      ( ( ( ( $ "Abiquiu^RioGrande.Outflow" [@"Previous Timestep"] - ( $
"BlwAbiquiuDiversions.Total Diversion Requested" [] - $ "BlwAbiquiuDiversions.Total Depletion
Requested" [] ) ) * ( 1.00000000 + "LossRate"( % "BlwAbiquiuToChamita" ) ) - ( $
"AbvConfluenceDiversions.Total Diversion Requested" [] - $ "AbvConfluenceDiversions.Total
Depletion Requested" [] ) + $ "AbiquiuToChamitaLocalInflow.Local Inflow" [] - ( $
"BlwConfluenceDiversions.Total Diversion Requested" [] - $ "BlwConfluenceDiversions.Total
Depletion Requested" [] ) - ( $ "BlwChamitaDiversions.Total Diversion Requested" [] - $
"BlwChamitaDiversions.Total Depletion Requested" [] ) + $ "Embudo.Gage Outflow" [@"Previous
Timestep"] * ( 1.00000000 + "LossRate"( % "EmbudoToConfluence" ) ) ) * ( 1.00000000 + "LossRate"(
% "ConfluenceToOtowi" ) ) + $ "EmbudoToOtowiLocalInflow.Local Inflow" [] ) * ( 1.00000000 +
"LossRate"( % "OtowiToCochiti" ) ) + $ "SantaFeRiverAbvCochiti.Gage Inflow" [] + $
"OtowiToCochitiLocalInflow.Local Inflow" [] ) * ( 1.00000000 + $
"ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   ENDIF;

    END;

    FUNCTION       "ForecastElVadoInflow" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN $ "ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   THEN
      "Max"( "SumFlowsToVolume"( $ "AzoteaWillow.Inflow2", startDate, endDate ) +
"PreviousAccountStorage"( "RioGrande", % "Heron" ), 0.00000000 ["acre-feet"] ) +
"SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow", startDate, endDate )
   ELSE
      ( "Max"( "SumFlowsToVolume"( $ "AzoteaWillow.Inflow2", startDate, endDate ) +
"PreviousAccountStorage"( "RioGrande", % "Heron" ), 0.00000000 ["acre-feet"] ) +
"SumFlowsToVolume"( $ "ElVadoLocalInflow.Local Inflow", startDate, endDate ) ) * ( 1.00000000 + $
"ForecastData.PercentForecastError" ["EndOfMonth"(  )] )
   ENDIF;

    END;

    FUNCTION       "EstimateElVadoInflow" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "SumFlowsToVolume"( $ "AzoteaWillow.Inflow2", startDate, endDate ) + "SumFlowsToVolume"( $
"ElVadoLocalInflow.Local Inflow", startDate, endDate );

    END;

  END;

  UTILITY_GROUP "Estimated Loss Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EstimateElVadoEvapLosses" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( startDate >= @"April" AND endDate <= @"October" )
   THEN
      "EstimateElVadoSummerEvapLosses"( startDate, endDate )
```

```
   ELSE
      IF ( startDate < @"April" AND ( endDate >= @"April" AND endDate <= @"October" ) )
      THEN
         "EstimateElVadoWinterEvapLosses"( startDate, @"24:00:00 March 31, Current Year" ) +
"EstimateElVadoSummerEvapLosses"( @"24:00:00 April 1, Current Year", endDate )
      ELSE
         IF ( ( startDate >= @"April" AND startDate <= @"October" ) AND endDate > @"October" )
         THEN
            "EstimateElVadoSummerEvapLosses"( startDate, @"24:00:00 October 31, Current Year" ) +
"EstimateElVadoWinterEvapLosses"( @"24:00:00 November 1, Current Year", endDate )
         ELSE
            "EstimateElVadoWinterEvapLosses"( startDate, endDate )
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "EstimateElVadoSummerEvapLosses" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "SumPanEvap"( % "ElVado", startDate, endDate ) - "SumPrecip"( % "ElVado", startDate,
endDate ) ) * "AverageSurfaceArea"( % "ElVado", "PreviousPoolElevation"( % "ElVado" ),
"TargetElevation"( % "ElVado", @"Current Timestep" ) );

    END;

    FUNCTION       "EstimateElVadoWinterEvapLosses" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "SumWinterEvapInFeet"( % "ElVado", startDate, endDate ) - "SumPrecip"( % "ElVado",
startDate, endDate ) ) * "AverageSurfaceArea"( % "ElVado", $ "ElVado.Pool Elevation" [@"Previous
Timestep"], "TargetElevation"( % "ElVado", @"24:00:00 February Max DayOfMonth, Current Year" ) );

    END;

    FUNCTION       "SumPanEvap" ( OBJECT reservoir, DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SumSlot"( reservoir & "Pan Evaporation", startDate, endDate ) * 1.00000000 ["day"] *
reservoir & "Pan Evaporation Coefficient" [0.00000000, 0.00000000];

    END;

    FUNCTION       "SumPrecip" ( OBJECT reservoir, DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SumSlot"( reservoir & "Precipitation Rate", startDate, endDate ) * 1.00000000 ["day"];

    END;

    FUNCTION       "SumWinterEvapInFeet" ( OBJECT reservoir, DATETIME startDate, DATETIME endDate
)
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( DATETIME date IN startDate TO endDate ) WITH NUMERIC result = 0.00000000 ["feet"] DO
      result + "AverageAirTempTimesKFactor"( reservoir, date ) * ( 1.00000000 -
"SurfaceIceCoverage"( reservoir, date ) )
   ENDFOR;

    END;

    FUNCTION       "AverageSurfaceArea" ( OBJECT reservoir, NUMERIC startElevation, NUMERIC
endElevation )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "ElevationToArea"( reservoir, startElevation ) + "ElevationToArea"( reservoir,
endElevation ) ) / 2.00000000;

    END;

    FUNCTION       "MaxAirTemp" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Max Air Temperature" [date];

    END;

    FUNCTION       "MinAirTemp" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Min Air Temperature" [date];

    END;

    FUNCTION       "SurfaceIceCoverage" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Surface Ice Coverage" [date];

    END;

    FUNCTION       "AverageAirTempTimesKFactor1" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( reservoir & "Max Air Temperature" [date] + reservoir & "Min Air Temperature" [date] ) /
2.00000000 * reservoir & "K Factor" [date];
```

```
    END;

    EXTERNAL_FUNCTION "AverageAirTempTimesKFactor" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE       NUMERIC;
    SCALE_UNITS       "feet";
    DESCRIPTION       "";
    ACTIVE            TRUE;
    PRE_EXEC_DIAG     FALSE;
    POST_EXEC_DIAG    TRUE;
    BEGIN
# This function is used because of a problem with Riverware using temperature units

   set maxAirTemp [C_GetValue  "$reservoir.Max Air Temperature" 1.0 F $date]
   set minAirTemp  [C_GetValue  "$reservoir.Min Air Temperature" 1.0 F $date]
   set aveAirTemp  [expr ($maxAirTemp + $minAirTemp) / 2]

   set KFactor [C_GetValue  "$reservoir.K Factor" 1.0 "ft/F" $date]

   set result [C_Max [expr $aveAirTemp * $KFactor] 0]

   return "$result \[\"feet\"\]"



    END;

  END;

  UTILITY_GROUP "EstimatedOutflowFunctions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EstimateMRGCDCochitiOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( $ "MiddleValleyDemands.MRGCD" [] - "ForecastCochitiRGOutflow"(  ), 0.00000000
["cfs"] );

    END;

  END;

  UTILITY_GROUP "Exchange Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AlbuquerqueLoanEXs" ( LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( LIST accountLoanList IN "MakeOtowiExchangeList"( accounts, "Abiquiu" ) ) WITH LIST
result = {  } DO
      APPEND { GET STRING @INDEX 1.00000000 FROM accountLoanList , "DeliveryRequestFor"( GET
STRING @INDEX 0.00000000 FROM accountLoanList ) } ONTO result
   ENDFOR;

    END;

    FUNCTION       "AlbuquerqueWillLoanToAccount" ( STRING account )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      "AlbuquerqueLoanSwitch"( account ) == 1.00000000;

    END;

    FUNCTION       "AlbuquerqueWillLoan" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "PreviousAccountStorage"( "Albuquerque", % "Abiquiu" ) > "MinLoanPool"( "Albuquerque", %
"Abiquiu" ) + "MaxAccountStorageTolerance"( % "Abiquiu" );

    END;

    FUNCTION       "CurrentTotalObjectDebtFlow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
   THEN
      "Min"( FOR ( STRING supply IN "MakeSupplyList"( "AbiquiuOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "Abiquiu", "AbiquiuOtowiPaybackAccounts"(  ) ), % "Abiquiu", %
"BlwAbiquiu" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
      ENDFOR, "MaxDeliveryRequestCheck"(  ) ) + "VolumeToFlow"( "GetAccountDebt"(
"NMISCAbiquiuToNMISCAbiquiuNMISCJemezBlwAbiquiu.Supply" ), @"Current Timestep" ) +
"SumIntraReservoirTransfers"( "MakeSupplyList"( "AlbuquerqueFiveTimesList"(  ),
"AbiquiuAlbuquerqueLoanAccounts"(  ), % "Abiquiu", % "Abiquiu" ) )
   ELSE
      IF ( "StringifyObject"( reservoir ) == "ElVado" )
      THEN
         FOR ( STRING supply IN "MakeSupplyList"( "ElVadoRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "ElVado", "ElVadoRootOtowiPaybackAccounts"(  ) ), % "ElVado", %
"BlwElVado" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
            result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
         ENDFOR + FOR ( STRING supply IN "MakeSupplyList"(
"ElVadoRootAlbuquerquePaybackAccounts"(  ), "MakeAlbuquerqueAbiquiuPaybackAccountsList"(
"ElVadoRootAlbuquerquePaybackAccounts"(  ), % "ElVado" ), % "ElVado", % "BlwElVado" ) ) WITH
NUMERIC result = 0.00000000 ["cfs"] DO
            result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
         ENDFOR
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Heron" )
         THEN
            FOR ( STRING supply IN "MakeSupplyList"( "HeronRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "Heron", "HeronRootOtowiPaybackAccounts"(  ) ), % "Heron", %
"HeronSeepage" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
               result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
            ENDFOR + FOR ( STRING supply IN "MakeSupplyList"(
"HeronRootAlbuquerquePaybackAccounts"(  ), "MakeAlbuquerqueAbiquiuPaybackAccountsList"(
"HeronRootAlbuquerquePaybackAccounts"(  ), % "Heron" ), % "Heron", % "HeronSeepage" ) ) WITH
NUMERIC result = 0.00000000 ["cfs"] DO
               result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
            ENDFOR + FOR ( STRING supply IN "MakeSupplyList"( "ElVadoRootWaiverStorageAccounts"(
), "MakeHeronElVadoExAccountsLIst"( "ElVadoRootWaiverStorageAccounts"(  ) ), % "Heron", % "Heron"
) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
               result + "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" )
            ENDFOR + "VolumeToFlow"( "GetAccountDebt"(
"ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" ), @"Current Timestep" ) +
"VolumeToFlow"( "GetAccountDebt"(
"ReclamationHeronToReclamationHeronMRGCDAbiquiuHeronSeepage.Supply" ), @"Current Timestep" ) +
"VolumeToFlow"( "GetAccountDebt"( "AlbuquerqueHeronToNMISCHeronNMISCJemezHeronSeepage.Supply" ),
@"Current Timestep" )
         ELSE
            0.00000000 ["cfs"]
         ENDIF
      ENDIF
   ENDIF;
```

```
    END;

    FUNCTION       "GetAccountDebt" ( STRING supply )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetPaybackDebt"( supply, @"Current Timestep" );

    END;

    FUNCTION       "JemezDebt" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetPaybackDebt"( "NMISCAbiquiuToNMISCAbiquiuNMISCJemezBlwAbiquiu.Supply", @"Current
Timestep" );

    END;

    FUNCTION       "HeronElVadoEXs" (  )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST accounts = "ElVadoRootWaiverStorageAccounts"(  ) DO
      WITH LIST exchanges = "MakeHeronElVadoExchangeList"( accounts ) DO
         WITH LIST RGOtowiSupplies = "MakeSupplyList"( accounts, "MakeOtowiPaybackAccountList"( %
"ElVado", accounts ), % "ElVado", % "BlwElVado" ) DO
            WITH LIST AlbuquerquePaybackSupplies = "MakeSupplyList"( accounts,
"MakeAlbuquerqueAbiquiuPaybackAccountsList"( accounts, % "ElVado" ), % "ElVado", % "BlwElVado" )
DO
               WITH LIST exchangeSupplies = "MakeSupplyList"( accounts,
"MakeHeronElVadoExAccountsLIst"( accounts ), % "Heron", % "Heron" ) DO
                  FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
                     IF ( account == "Albuquerque" )
                     THEN
                        APPEND { GET STRING @INDEX LENGTH result FROM exchanges , 0.00000000
["cfs"] } ONTO result
                     ELSE
                        IF ( account == "MRGCD" )
                        THEN
                           IF ( "PreviousStorage"( % "ElVado" ) < $
"ElVadoData.MinimumTotalStorage" [0.00000000, 0.00000000] AND @"Current Timestep" >
"TargetFillDate"( % "ElVado" ) )
                           THEN
                              APPEND { GET STRING @INDEX LENGTH result FROM exchanges , "Min"(
"VolumeToFlow"( "Max"( "PreviousAccountStorage"( "MRGCD", % "Heron" ) - "GetAccountDebt"( GET
STRING @INDEX LENGTH result FROM exchangeSupplies ), 0.00000000 ["acre-feet"] ), @"Current
Timestep" ), "VolumeToFlow"( $ "ElVadoData.MinimumTotalStorage" [0.00000000, 0.00000000] -
"PreviousStorage"( % "ElVado" ), @"Current Timestep" ) ) } ONTO result
                           ELSE
                              APPEND { GET STRING @INDEX LENGTH result FROM exchanges , "Min"(
"Max"( "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH result FROM accounts, %
"Heron" ), @"Current Timestep" ) - "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH
result FROM exchangeSupplies ), @"Current Timestep" ), 0.00000000 ["cfs"] ), "Max"(
"VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH result FROM AlbuquerquePaybackSupplies
) - ( "PreviousAccountStorage"( account, % "ElVado" ) + "GetAccountDebt"( GET STRING @INDEX
LENGTH result FROM exchangeSupplies ) ), @"Current Timestep" ), 0.00000000 ["cfs"] ) ) } ONTO
result
                           ENDIF
                        ELSE
                           APPEND { GET STRING @INDEX LENGTH result FROM exchanges , "MinItem"( {
"Max"( "VolumeToFlow"( "PreviousAccountStorage"( GET STRING @INDEX LENGTH result FROM accounts, %
"Heron" ), @"Current Timestep" ) - "VolumeToFlow"( "GetAccountDebt"( GET STRING @INDEX LENGTH
```

```
result FROM exchangeSupplies ), @"Current Timestep" ), 0.00000000 ["cfs"] ) , "VolumeToFlow"(
"AccountStorageAvailable"( account, % "ElVado" ), @"Current Timestep" ) , "Max"( "VolumeToFlow"(
"GetAccountDebt"( GET STRING @INDEX LENGTH result FROM AlbuquerquePaybackSupplies ) +
"GetAccountDebt"( GET STRING @INDEX LENGTH result FROM RGOtowiSupplies ) - (
"PreviousAccountStorage"( account, % "ElVado" ) + "GetAccountDebt"( GET STRING @INDEX LENGTH
result FROM exchangeSupplies ) ), @"Current Timestep" ), 0.00000000 ["cfs"] ) } ) } ONTO result
                        ENDIF
                      ENDIF
                  ENDFOR
               ENDWITH
            ENDWITH
         ENDWITH
      ENDWITH
   ENDWITH;

    END;

  END;

  UTILITY_GROUP "ForecastErrorFunctions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ComputeForecastError" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "GetMonth"( @"Current Timestep" ) <= 5.00000000 )
   THEN
      $ "ForecastData.ForecastCoefficients" ["GetMonthAsString"( @"Current Timestep" ), "c1"] *
"EstimateElVadoInflow"( @"24:00:00 March 1, Current Year", @"24:00:00 July 31, Current Year" ) +
$ "ForecastData.ForecastCoefficients" ["GetMonthAsString"( @"Current Timestep" ), "c2"] *
"PreviousForecastError"(  ) + $ "ForecastData.ForecastCoefficients" ["GetMonthAsString"(
@"Current Timestep" ), "c3"] + $ "ForecastData.ForecastCoefficients" ["GetMonthAsString"(
@"Current Timestep" ), "sde"] * ( 0.33333330 * "RanDev"( 1.00000000 ) )
   ELSE
      IF ( "GetMonth"( @"Current Timestep" ) == 6.00000000 )
      THEN
         0.50000000 * % "ForecastData" & "ForecastError" [@"24:00:00 Previous Month Max
DayOfMonth, Current Year"]
      ELSE
         IF ( "GetMonth"( @"Current Timestep" ) == 7.00000000 )
         THEN
            0.25000000 * % "ForecastData" & "ForecastError" [@"24:00:00 Previous Month Max
DayOfMonth, Current Year"]
         ELSE
            0.00000000 ["acre-feet"]
         ENDIF
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Flood Carry Over Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ComputeIsNaNRGCarryOverLeft" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverLeft" [@"Previous Timestep"] -
"FlowToVolume"( "RGCarryOverRelease"( reservoir ), @"Current Timestep" );

    END;
```

```
    FUNCTION       "ComputeNOTIsNaNRGCarryOverLeft" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverLeft" [@"Previous Timestep"] -
"FlowToVolume"( ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [], @"Current Timestep"
);

    END;

    FUNCTION       "ConstantRGCarryOverRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( NOT IsNaN "PreviousRGCarryOverRelease"( reservoir ) AND "PreviousRGCarryOverRelease"(
reservoir ) > 0.00000000 ["cfs"] )
   THEN
      "Min"( "PreviousRGCarryOverRelease"( reservoir ), "VolumeToFlow"( "PreviousRGStorage"(
reservoir ), @"Current Timestep" ) )
   ELSE
      "Min"( "PreviousRGStorage"( reservoir ) / "DeltaTimeToMarch"(  ), "VolumeToFlow"(
"PreviousRGStorage"( reservoir ), @"Current Timestep" ) )
   ENDIF;

    END;

    FUNCTION       "EstimatedRGStorageLookAhead" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "PreviousRGStorage"( reservoir ) + "FlowToVolume"( "EstimatedRGInflowLookAhead"( reservoir
) - "EstimatedRGOutflowLookAhead"( reservoir ), @"Current Timestep" );

    END;

    FUNCTION       "EstimatedRGOutflowLookAhead" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function estimates the amount of flow that will be released while the
release is shutting back to match inflow when locking in carryover storage.
";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "Abiquiu" )
   THEN
      ( ( ( "GetJulianDate"( "LookAhead"(  ) ) / "GetJulianDate"( "LookAhead"(  ) ) ) *
3.50000000 ) * "PreviousRGOutflow"( reservoir ) )
   ELSE
      ( ( ( "GetJulianDate"( "LookAhead"(  ) ) / "GetJulianDate"( "LookAhead"(  ) ) ) *
5.00000000 ) * "PreviousRGOutflow"( reservoir ) )
   ENDIF;

    END;

    FUNCTION       "EstimatedRGInflowLookAhead" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "Abiquiu" )
   THEN
      "EstimatedAbiquiuInflowLookAhead"(  )
   ELSE
      "EstimatedCochitiInflowLookAhead"(  )
   ENDIF;

    END;

    FUNCTION       "LockedIn" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Locked In" [];

    END;

    FUNCTION       "LockedInLookAhead" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Locked In" ["LookAhead"(  )];

    END;

    FUNCTION       "LockinCarryOver" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "IfLockinCarryOver"(  ) AND "PreviousRGStorage"( reservoir ) > ( STRINGIFY reservoir )
CONCAT "Data.MinRGCarryOverStorage" [0.00000000, 0.00000000] )
   THEN
      IF ( "PreviousLockedIn"( reservoir ) == 1.00000000 )
      THEN
         1.00000000
      ELSE
         IF ( "PreviousRGStorage"( reservoir ) > ( STRINGIFY reservoir ) CONCAT
"Data.MinRGCarryOverStorage" [0.00000000, 0.00000000] )
         THEN
            1.00000000
         ELSE
            0.00000000
         ENDIF
      ENDIF
   ELSE
      0.00000000
   ENDIF;

    END;

    FUNCTION       "LockinCarryOverLookAhead" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "IfLockinCarryOverLookAhead"(  ) AND "EstimatedRGStorageLookAhead"( reservoir ) >
"MinRGCarryOverStorage"( reservoir ) )
```

```
   THEN
      IF ( reservoir & "Locked In" ["LookAhead-1"(  )] == 1.00000000 )
      THEN
         1.00000000
      ELSE
         IF ( ( ( "EstimatedRGStorageLookAhead"( reservoir ) > "MinRGCarryOverStorage"( reservoir
) ) AND "IfCochitiFloodSpaceAvailable"(  ) ) OR "IfFloodCarryOverAtCochiti"( reservoir ) )
         THEN
            1.00000000
         ELSE
            0.00000000
         ENDIF
      ENDIF
   ELSE
      0.00000000
   ENDIF;

    END;

    FUNCTION       "IfLockinCarryOver" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "FloodCarryOverSeason"(  ) AND "Otowi"(  ) < "FloodCarryOverData.MinLockinFlow"
[0.00000000, 0.00000000];

    END;

    FUNCTION       "IfLockinCarryOverLookAhead" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "FloodCarryOverSeasonLookAhead"(  ) AND "OtowiLookAhead"(  ) <
"FloodCarryOverData.MinLockinFlow" [0.00000000, 0.00000000];

    END;

    FUNCTION       "IfRGCarryOverRelease" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( NOT IsNaN "PreviousRGCarryOverLeft"( reservoir ) AND "PreviousRGCarryOverLeft"( reservoir
) > 0.00000000 ["acre-feet"] ) OR ( "PreviousRGStorage"( reservoir ) > ( STRINGIFY reservoir )
CONCAT "Data.MinRGCarryOverStorage" [0.00000000, 0.00000000] AND IsNaN "PreviousRGCarryOverLeft"(
reservoir ) ) OR ( NOT IsNaN reservoir & "Carryover Content" [@"24:00:00 October 31, Current
Year"] AND reservoir & "Carryover Content" [@"24:00:00 October 31, Current Year"] > 0.00000000
["acre-feet"] );

    END;

    FUNCTION       "MinRGCarryOverStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MinRGCarryOverStorage" [0.00000000, 0.00000000];

    END;

    FUNCTION       "PreviousCarryOverContent" ( OBJECT reservoir )
```

```
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Carryover Content" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousLockedIn" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Locked In" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousRGCarryOverLeft" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverLeft" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousRGCarryOverRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Previous Timestep"];

    END;

    FUNCTION       "RGCarryOverLeft" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "RGCarryOverIsNaNBoolean"( reservoir ) AND "ConstantCOReleaseBoolean"( reservoir ) )
   THEN
      IF ( "PreviousRGStorage"( reservoir ) > 0.00000000 ["acre-feet"] )
      THEN
         IF ( NOT IsNaN "PreviousRGCarryOverLeft"( reservoir ) AND "PreviousRGCarryOverLeft"(
reservoir ) > 0.00000000 ["acre-feet"] )
         THEN
            "PreviousRGCarryOverLeft"( reservoir ) - "FlowToVolume"( "RGCarryOverRelease"(
reservoir ), @"Current Timestep" )
         ELSE
            "PreviousRGStorage"( reservoir ) - "FlowToVolume"( "RGCarryOverRelease"( reservoir ),
@"Current Timestep" )
         ENDIF
      ELSE
         0.00000000 ["acre-feet"]
      ENDIF
   ELSE
      IF ( "RGCarryOverNOTNaNBoolean"( reservoir ) AND "ConstantCOReleaseBoolean"( reservoir ) )
      THEN
```

```
         IF ( "PreviousRGStorage"( reservoir ) > 0.00000000 ["acre-feet"] )
         THEN
            "PreviousRGStorage"( reservoir ) - "FlowToVolume"( ( STRINGIFY reservoir ) CONCAT
"Data.RGCarryOverRelease" [], @"Current Timestep" )
         ELSE
            0.00000000 ["acre-feet"]
         ENDIF
      ELSE
         IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [] )
         THEN
            IF ( ( NOT IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverLeft" [@"Previous
Timestep"] AND "ComputeIsNaNRGCarryOverLeft"( reservoir ) <= 0.00000000 ["acre-feet"] ) OR
"RGCarryOverRelease"( reservoir ) == 0.00000000 ["cfs"] )
            THEN
               0.00000000 ["acre-feet"]
            ELSE
               "ComputeIsNaNRGCarryOverLeft"( reservoir )
            ENDIF
         ELSE
            IF ( ( "ComputeNOTIsNaNRGCarryOverLeft"( reservoir ) <= 0.00000000 ["acre-feet"] ) OR
( "RGCarryOverRelease"( reservoir ) >= "VolumeToFlow"( "PreviousRGStorage"( reservoir ),
@"Current Timestep" ) ) )
            THEN
               0.00000000 ["acre-feet"]
            ELSE
               "ComputeNOTIsNaNRGCarryOverLeft"( reservoir )
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "RGCarryOverRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "FloodCarryOverReleaseSeason"(  ) AND ( "PreviousRGStorage"( reservoir ) > 0.00000000
["acre-feet"] ) AND ( NOT IsNaN $ "Abiquiu.Locked In" [] AND $ "Abiquiu.Locked In" [] ==
0.00000000 ) )
   THEN
      IF ( "IfRGCarryOverRelease"( reservoir ) )
      THEN
         IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Current
Timestep"] )
         THEN
            "ConstantRGCarryOverRelease"( reservoir )
         ELSE
            ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Current Timestep"]
         ENDIF
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Flood Carry Over Boolean Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ConstantCOReleaseBoolean" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      IF ( "FloodCarryOverReleaseSeason"(  ) AND ( "PreviousRGStorage"( reservoir ) > 0.00000000
["acre-feet"] ) AND ( NOT IsNaN $ "Abiquiu.Locked In" [] AND $ "Abiquiu.Locked In" [] ==
0.00000000 ) )
   THEN
      IF ( "IfRGCarryOverRelease"( reservoir ) )
      THEN
         IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Current
Timestep"] )
         THEN
            TRUE
         ELSE
            FALSE
         ENDIF
      ELSE
         FALSE
      ENDIF
   ELSE
      FALSE
   ENDIF;

    END;

    FUNCTION       "NOTNaNLockedInLookAheadAndValue=OneTest" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      NOT IsNaN "LockedInLookAhead"( reservoir ) AND "LockedInLookAhead"( reservoir ) ==
1.00000000;

    END;

    FUNCTION       "NOTNaNPreviousCarryoverContentAndValue>Test" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      NOT IsNaN "PreviousCarryOverContent"( reservoir ) AND "PreviousCarryOverContent"( reservoir
) > 0.00000000 ["acre-feet"];

    END;

    FUNCTION       "NOTNanPreviousRGCarryOverLeftAndValue=Test" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( NOT IsNaN "PreviousRGCarryOverLeft"( reservoir ) AND ( "PreviousRGCarryOverLeft"(
reservoir ) == 0.00000000 ["acre-feet"] ) );

    END;

    FUNCTION       "RGCarryOverIsNaNBoolean" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "FloodCarryOverReleaseSeason"(  ) AND ( "PreviousRGStorage"( reservoir ) > 0.00000000
["acre-feet"] ) )
   THEN
      IF ( "IfRGCarryOverRelease"( reservoir ) )
      THEN
```

```
         IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Current
Timestep"] )
         THEN
            TRUE
         ELSE
            FALSE
         ENDIF
      ELSE
         FALSE
      ENDIF
   ELSE
      FALSE
   ENDIF;

    END;

    FUNCTION       "RGCarryOverNOTNaNBoolean" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "FloodCarryOverReleaseSeason"(  ) AND ( "PreviousRGStorage"( reservoir ) > 0.00000000
["acre-feet"] ) )
   THEN
      IF ( "IfRGCarryOverRelease"( reservoir ) )
      THEN
         IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGCarryOverRelease" [@"Current
Timestep"] )
         THEN
            FALSE
         ELSE
            TRUE
         ENDIF
      ELSE
         FALSE
      ENDIF
   ELSE
      FALSE
   ENDIF;

    END;

  END;

  UTILITY_GROUP "General Account Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AllRGPossible" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "PreviousRGInflow"( reservoir ) + "RGCarryOverRelease"( reservoir ) + "VolumeToFlow"(
"PreviousRGStorage"( reservoir ) - "PreviousRGCarryOverLeft"( reservoir ), @"Current Timestep" );

    END;

    FUNCTION       "AvailableAccountStorage" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    MIN_CONSTRAINT 0.00000000 ["acre-feet"];
    BEGIN

      "Max"( "MaxAccountStorage"( account, reservoir ) - "PreviousAccountStorage"( account,
reservoir ), 0.00000000 ["acre-feet"] );
```

```
    END;

    FUNCTION       "CorpsProjectsRGOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( reservoir ) - "SJOutflow"( reservoir );
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      WITH NUMERIC RGInflow = "Average3DayRioGrandeInflow"( reservoir ) DO
      WITH NUMERIC previousAccountStorage = "PreviousAccountStorage"( "RioGrande", reservoir ) DO
         IF ( RGInflow <= 0.00000000 ["cfs"] AND previousAccountStorage <= 0.00000000 ["acre-
feet"] )
         THEN
            0.00000000 ["cfs"]
         ELSE
            IF ( "FloodCarryOverReleaseSeason"(  ) AND "NOTJemez"( reservoir ) )
            THEN
               IF ( previousAccountStorage > 0.00000000 ["acre-feet"] )
               THEN
                  IF ( IsNaN "PreviousRGCarryOverLeft"( reservoir ) OR (
"NOTNanPreviousRGCarryOverLeftAndValue=Test"( reservoir ) AND
"NOTNaNPreviousCarryoverContentAndValue>Test"( reservoir ) ) OR ( NOT IsNaN
"PreviousRGCarryOverLeft"( reservoir ) AND ( ( previousAccountStorage -
"PreviousRGCarryOverLeft"( reservoir ) < 0.00000000 ["acre-feet"] ) AND ( RGInflow +
"VolumeToFlow"( previousAccountStorage - "PreviousRGCarryOverLeft"( reservoir ), @"Current
Timestep" ) >= 0.00000000 ["cfs"] ) ) ) OR ( "RGCarryOverRelease"( reservoir ) >= "VolumeToFlow"(
previousAccountStorage, @"Current Timestep" ) ) )
                  THEN
                     RGInflow + "RGCarryOverRelease"( reservoir )
                  ELSE
                     IF ( ( RGInflow + "VolumeToFlow"( previousAccountStorage -
"PreviousRGCarryOverLeft"( reservoir ), @"Current Timestep" ) < 0.00000000 ["cfs"] AND
"RGCarryOverRelease"( reservoir ) > 0.00000000 ["cfs"] ) )
                     THEN
                        "RGCarryOverRelease"( reservoir )
                     ELSE
                        "AllRGPossible"( reservoir )
                     ENDIF
                  ENDIF
               ELSE
                  IF ( ( ( previousAccountStorage + "FlowToVolume"( RGInflow, @"Current Timestep"
) < 0.00000000 ["acre-feet"] ) AND RGInflow > 0.00000000 ["cfs"] ) )
                  THEN
                     ( STRINGIFY reservoir ) CONCAT "Data.PercentRGReleaseWhenNegRGStorage"
[0.00000000, 0.00000000] * RGInflow
                  ELSE
                     RGInflow + "VolumeToFlow"( previousAccountStorage, @"Current Timestep" )
                  ENDIF
               ENDIF
            ELSE
               IF ( NOT IsNaN "PreviousCarryOverContent"( reservoir ) AND previousAccountStorage
> "PreviousCarryOverContent"( reservoir ) )
               THEN
                  IF ( "NOTJemez"( reservoir ) AND "RGStorageIsLockedIn"( reservoir ) )
                  THEN
                     RGInflow
                  ELSE
                     IF ( IsNaN "PreviousIncidentalContent"( reservoir ) )
                     THEN
                        RGInflow
                     ELSE
                        RGInflow + "IncidentalContentRelease"( reservoir )
                     ENDIF
                  ENDIF
               ELSE
                  IF ( IsNaN "PreviousIncidentalContent"( reservoir ) )
                  THEN
                     RGInflow
                  ELSE
                     IF ( "PreviousIncidentalContent"( reservoir ) + "FlowToVolume"( RGInflow,
@"Current Timestep" ) < 0.00000000 ["acre-feet"] AND RGInflow > 0.00000000 ["cfs"] )
                     THEN
```

```
                        ( STRINGIFY reservoir ) CONCAT "Data.PercentRGReleaseWhenNegRGStorage"
[0.00000000, 0.00000000] * RGInflow
                     ELSE
                        IF ( "PreviousIncidentalContent"( reservoir ) + "FlowToVolume"( RGInflow,
@"Current Timestep" ) <= 0.00000000 ["acre-feet"] AND RGInflow <= 0.00000000 ["cfs"] )
                        THEN
                           0.00000000 ["cfs"]
                        ELSE
                           RGInflow + "IncidentalContentRelease"( reservoir )
                        ENDIF
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDWITH
   ENDWITH;

    END;

    FUNCTION       "PreviousAccountInflow" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Inflow" [@"Previous
Timestep"] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Inflow" [@"Previous Timestep"]
   ENDIF;

    END;

    FUNCTION       "PreviousAccountStorage" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Storage" [@"Previous
Timestep"] )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Storage" [@"Previous Timestep"]
   ENDIF;

    END;

    FUNCTION       "PreviousRGStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "^RioGrande.Storage" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousRGInflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      reservoir ~ "RioGrande.Inflow" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousRGOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir ~ "RioGrande.Outflow" [@"Previous Timestep"];

    END;

    FUNCTION       "ReconcileRGOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "Cochiti" OR ( STRINGIFY reservoir ) == "Abiquiu" )
   THEN
      "Max"( reservoir & "Outflow" [] - "SJOutflow"( reservoir ), "MinRGOutflow"( reservoir ) )
   ELSE
      IF ( "RGOutflow"( reservoir ) > reservoir & "Outflow" [] )
      THEN
         reservoir & "Outflow" []
      ELSE
         IF ( reservoir & "Outflow" [] > "RGOutflow"( reservoir ) + "SJOutflow"( reservoir ) )
         THEN
            "Max"( reservoir & "Outflow" [] - "SJOutflow"( reservoir ), "MinRGOutflow"( reservoir
) )
         ELSE
            "RGOutflow"( reservoir )
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ReconcileRGOuflowAtGivenDate" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "RGOutflowAtGivenDate"( reservoir, date ) > reservoir & "Outflow" [date] )
   THEN
      reservoir & "Outflow" [date]
   ELSE
      IF ( reservoir & "Outflow" [date] > "RGOutflowAtGivenDate"( reservoir, date ) +
"SJOutflowAtGivenDate"( reservoir, date ) )
      THEN
         "Max"( reservoir & "Outflow" [date] - "SJOutflowAtGivenDate"( reservoir, date ),
0.00000000 ["cfs"] )
      ELSE
         "RGOutflowAtGivenDate"( reservoir, date )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ReconcileSJOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
```

```
 BEGIN

   reservoir & "Outflow" [] - "ReconcileRGOutflow"( reservoir );

 END;

 FUNCTION       "ReconcileSJOutflowAtGivenDate" ( OBJECT reservoir, DATETIME date )
 RETURN_TYPE    NUMERIC;
 SCALE_UNITS    "cfs";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG TRUE;
 BEGIN

   reservoir & "Outflow" [date] - "ReconcileRGOuflowAtGivenDate"( reservoir, date );

 END;

 FUNCTION       "RGOutflow" ( OBJECT reservoir )
 RETURN_TYPE    NUMERIC;
 SCALE_UNITS    "cfs";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG TRUE;
 BEGIN

   IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.RGOutflow" [] )
THEN
   0.00000000 ["cfs"]
ELSE
   ( STRINGIFY reservoir ) CONCAT "Data.RGOutflow" []
ENDIF;

 END;

 FUNCTION       "RGOutflowAtGivenDate" ( OBJECT reservoir, DATETIME date )
 RETURN_TYPE    NUMERIC;
 SCALE_UNITS    "cfs";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG TRUE;
 BEGIN

   ( STRINGIFY reservoir ) CONCAT "Data.RGOutflow" [date];

 END;

 FUNCTION       "SJOutflow" ( OBJECT reservoir )
 RETURN_TYPE    NUMERIC;
 SCALE_UNITS    "cfs";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG TRUE;
 BEGIN

   IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.SJOutflow" [] )
THEN
   0.00000000 ["cfs"]
ELSE
   ( STRINGIFY reservoir ) CONCAT "Data.SJOutflow" []
ENDIF;

 END;

 FUNCTION       "SJOutflowAtGivenDate" ( OBJECT reservoir, DATETIME date )
 RETURN_TYPE    NUMERIC;
 SCALE_UNITS    "cfs";
 DESCRIPTION    "";
 ACTIVE         TRUE;
 PRE_EXEC_DIAG  FALSE;
 POST_EXEC_DIAG TRUE;
 BEGIN

   ( STRINGIFY reservoir ) CONCAT "Data.SJOutflow" [date];
```

```
    END;

    FUNCTION       "PrioritizedAccountReleases" ( OBJECT reservoir, NUMERIC available )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING releaseType IN "PrioritizedReleaseTypes"( reservoir, "ReleaseTypesList"(
reservoir ) ) ) WITH LIST result = {  } DO
      WITH NUMERIC totUsed = "SumListOfLists1"( result ) DO
         IF ( available - totUsed > 0.01000000 ["cfs"] )
         THEN
            IF ( "StringifyObject"( reservoir ) == "Heron" AND ( ( releaseType ==
"AccountDelivery" OR releaseType == "AccountFill" ) OR releaseType == "Waiver" ) )
            THEN
               FOR ( STRING downstreamReservoir IN "PrioritizedReservoirs"( reservoir,
"HeronDownStreamReservoirs"(  ) ) ) WITH LIST result2 = result DO
                  WITH NUMERIC used = "SumListOfLists1"( result2 ) DO
                     result2 CONCAT "GetSortedAccounts"( "PrepareSortList"( releaseType,
reservoir, downstreamReservoir, available - used ) )
                  ENDWITH
               ENDFOR
            ELSE
               result CONCAT "GetSortedAccounts"( "PrepareSortList"( releaseType, reservoir,
"Abiquiu", available - totUsed ) )
            ENDIF
         ELSE
            result CONCAT "ZeroSortedAccounts"( "PrepareSortList"( releaseType, reservoir,
"Abiquiu", 0.00000000 ["cfs"] ) )
         ENDIF
      ENDWITH
   ENDFOR;

    END;

    FUNCTION       "IncidentalContentRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "PreviousIncidentalContent"( reservoir ) < 0.00000000 ["acre-feet"] )
   THEN
      "Min"( "VolumeToFlow"( "Abs"( "PreviousIncidentalContent"( reservoir ) ), @"Current
Timestep" ), "MaxIncidentalContentRelease"( reservoir ) ) * - 1.00000000
   ELSE
      IF ( "PreviousIncidentalContent"( reservoir ) > 3000.00000000 ["acre-feet"] )
      THEN
         "VolumeToFlow"( "PreviousIncidentalContent"( reservoir ), @"Current Timestep" )
      ELSE
         "Min"( "VolumeToFlow"( "PreviousIncidentalContent"( reservoir ), @"Current Timestep" ),
"MaxIncidentalContentRelease"( reservoir ) )
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "General Boolean Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CurrentYearIsWaiverYear" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      "CurrentYearWaiverSwitch"(  ) == 1.00000000;

    END;

    FUNCTION       "ElVadoIsPriority" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( GET STRING @INDEX 0.00000000 FROM "PrioritizedReservoirs"( % "Heron",
"HeronDownStreamReservoirs"(  ) ) ) == "ElVado";

    END;

    FUNCTION       "IsElevationBasedOnInflowAndOutflow>TOC" ( OBJECT reservoir, NUMERIC inflow,
NUMERIC outflow )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StorageToElevationBasedOnCurrentInflow"( reservoir, inflow, outflow ) >
"JustBelowTopOfConservation"( reservoir );

    END;

    FUNCTION       "IsFloodReleaseRequired" ( OBJECT reservoir, NUMERIC Inflow, NUMERIC Outflow )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( NOT IsNaN Outflow AND "StorageToElevation"( reservoir, "SolveStorage"( reservoir, Inflow,
Outflow, "PreviousStorage"( reservoir ), @"Current Timestep" ) ) > ( STRINGIFY reservoir ) CONCAT
"Data.PoolLevels" ["BottomOfFlood", "Elevation"] );

    END;

    FUNCTION       "IsInflow>FloodRelease" ( OBJECT reservoir, NUMERIC inflow )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      inflow > "FloodRelease"( reservoir, inflow );

    END;

    FUNCTION       "IsLeapYear" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetDaysInMonth"( @"24:00:00 February Max DayOfMonth, Current Year" ) == 29.00000000
["day"];

    END;

    FUNCTION       "IsMaxCapacityExceeded" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Inflow" [] > ( STRINGIFY reservoir ) CONCAT "Data.DownstreamDemand" [] AND
reservoir & "Pool Elevation" [] > ( STRINGIFY reservoir ) CONCAT "Data.PoolLevels" ["Max",
"Elevation"];

    END;

    FUNCTION       "IsNOTFirstTimestep" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" != @"Start Timestep";

    END;

    FUNCTION       "IsPreviousElevation>TOC" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "PreviousElevation"( reservoir ) > ( STRINGIFY reservoir ) CONCAT "Data.PoolLevels"
["BottomOfFlood", "Elevation"];

    END;

    FUNCTION       "Is5000cfs>Inflow" ( OBJECT reservoir, NUMERIC inflow )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.FloodReleaseTable" [1.00000000, 3.00000000] > inflow;

    END;

    FUNCTION       "Is5000cfs>OutflowToGetBelowFloodPool" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.FloodReleaseTable" [1.00000000, 3.00000000] >
"OutflowToGetBelowFloodPool"( reservoir );

    END;

    FUNCTION       "NOTJemez" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) != "Jemez";

    END;
```

```
    FUNCTION       "OutflowIsDecreasing" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Outflow" [] < "PreviousOutflow"( reservoir );

    END;

    FUNCTION       "OutflowIsIncreasing" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Outflow" [] > reservoir & "Outflow" [@"Previous Timestep"];

    END;

    FUNCTION       "ReservoirIsSpillingUnreg" ( OBJECT res )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      res & "Unregulated Spill" [] > 0.00000000 ["cfs"];

    END;

    FUNCTION       "RGStorageIsLockedIn" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( ( IsNaN "LockedInLookAhead"( reservoir ) AND "LockinCarryOverLookAhead"( reservoir ) ==
1.00000000 AND "LockedIn"( reservoir ) == 0.00000000 ) OR (
"NOTNaNLockedInLookAheadAndValue=OneTest"( reservoir ) AND "LockedIn"( reservoir ) == 0.00000000
) ) OR ( "LockedIn"( reservoir ) == 1.00000000 AND "PreviousLockedIn"( reservoir ) == 0.00000000
);

    END;

    FUNCTION       "PreviousYearIsWaiverYear" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "PreviousYearWaiverSwitch"(  ) == 1.00000000;

    END;

    FUNCTION       "WaiversAreInEffect" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "PreviousYearIsWaiverYear"(  ) AND @"Current Timestep" <= "WaiverDate"(  );
```

```
    END;

    FUNCTION       "IsIrrigationSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      @"Current Timestep" >= @"24:00:00 March 1, Current Year" AND @"Current Timestep" <=
@"24:00:00 October 31, Current Year";

    END;

    FUNCTION       "IsReservoirRising" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Pool Elevation" [] > "PreviousElevation"( reservoir );

    END;

  END;

  UTILITY_GROUP "General Channel Capacity Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ChannelCapacity" ( OBJECT reservoir, STRING location )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.ChannelCapacities" [0.00000000, location];

    END;

    FUNCTION       "CombinedFlowToMatchChannelCapacity" ( OBJECT reservoir1, OBJECT reservoir2,
STRING location, NUMERIC value )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( reservoir1 & "Outflow" [] + reservoir2 & "Outflow" [] ) - ( value - "ChannelCapacity"(
reservoir1, location ) );

    END;

    FUNCTION       "FlowToMatchChannelCapacity" ( OBJECT reservoir, STRING location, NUMERIC
value )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Outflow" [] - ( value - "ChannelCapacity"( reservoir, location ) );

    END;
```

```
    FUNCTION       "IsReleaseToMatchChannelCapacity<0" ( OBJECT reservoir, STRING location,
NUMERIC value )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "FlowToMatchChannelCapacity"( reservoir, location, value ) < 0.00000000 ["cfs"] );

    END;

    FUNCTION       "IsReleaseToMatchChannelCapacity>=0" ( OBJECT reservoir, STRING location,
NUMERIC value )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "FlowToMatchChannelCapacity"( reservoir, location, value ) >= 0.00000000 ["cfs"] );

    END;

  END;

  UTILITY_GROUP "General Compute Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AccountStorageAvailable" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "MaxAccountStorage"( account, reservoir ) - "PreviousAccountStorage"( account,
reservoir ), 0.00000000 ["acre-feet"] );

    END;

    FUNCTION       "ComputeDeltaPoolElev" ( OBJECT Res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN Res & "Pool Elevation" [@"Current Timestep"] )
   THEN
      "Abs"( Res & "Pool Elevation" [@"Previous Timestep"] - Res & "Pool Elevation" [@"Previous
Timestep"] )
   ELSE
      "Abs"( Res & "Pool Elevation" [@"Previous Timestep"] - Res & "Pool Elevation" [@"Current
Timestep"] )
   ENDIF;

    END;

    FUNCTION       "ComputeDeltaOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Abs"( reservoir & "Outflow" [] - reservoir & "Outflow" [@"Previous Timestep"] );
```

```
    END;

    FUNCTION       "ComputeMaxOutflow" ( OBJECT Res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetMaxOutflowGivenInflow"( Res, Res & "Inflow" [], @"Current Timestep" );

    END;

    FUNCTION       "ComputeOutflowAtGivenStorage" ( OBJECT Res, NUMERIC storage )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SolveOutflow"( Res, Res & "Inflow" [@"Current Timestep"], storage, Res & "Storage"
[@"Previous Timestep"], @"Current Timestep" );

    END;

    FUNCTION       "ComputeReachLoss" ( OBJECT object, DATETIME date, NUMERIC inflow, NUMERIC
previousInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ReachLaggedFlow"( object, date, inflow, previousInflow ) * "SeasonGainLossCoeff"( object,
date ) + "SeasonGainLoss"( object, date );

    END;

    FUNCTION       "ComputeReachVariableLagLossOutflow" ( OBJECT object, DATETIME date, NUMERIC
inflow, NUMERIC previousInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ReachLaggedFlow"( object, date, inflow, previousInflow ) + "ComputeReachLoss"( object,
date, inflow, previousInflow );

    END;

    FUNCTION       "ComputeElVadoTargetRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "MaxOutflow"( % "ElVado" );
    MIN_CONSTRAINT "MinRGOutflow"( % "ElVado" );
    BEGIN

      IF ( @"Current Timestep" < "TargetFillDate"( % "ElVado" ) )
   THEN
      "Average3DayRioGrandeInflow"( % "ElVado" ) * "PercentElVadoRGRelease"(  )
   ELSE
      ( ( "ForecastElVadoInflow"( @"Current Timestep", "TargetElevationDate"( % "ElVado",
@"Current Timestep" ) ) + ( "PreviousAccountStorage"( "RioGrande", % "ElVado" ) - (
"TargetStorage"( % "ElVado", @"Current Timestep" ) - "ElVadoAccountStorageAdjustment"(  ) ) ) ) -
"EstimateElVadoEvapLosses"( @"Current Timestep", "TargetElevationDate"( % "ElVado", @"Current
```

```
Timestep" ) ) ) / ( ( "TargetElevationDate"( % "ElVado", @"Current Timestep" ) - @"Current
Timestep" ) + 1.00000000 ["day"] )
   ENDIF;

    END;

    FUNCTION       "FlowFraction" ( OBJECT object, NUMERIC inflow, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      1.00000000 - ( "VariableLagTime"( object, inflow, date ) / "GetTimestep"( @"Current
Timestep" ) - "VariableLagTime"( object, inflow, date ) DIV "GetTimestep"( @"Current Timestep" )
);

    END;

    FUNCTION       "JustBelowTopOfConservation" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.PoolLevels" ["BottomOfFlood", "Elevation"] -
0.01000000 ["ft"];

    END;

    FUNCTION       "OutflowToGetBelowFloodPool" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SolveOutflow"( reservoir, "CurrentInflow"( reservoir ), "ElevationToStorage"( reservoir, (
STRINGIFY reservoir ) CONCAT "Data.PoolLevels" ["BottomOfFlood", "Elevation"] - 0.01000000
["feet"] ), "PreviousStorage"( reservoir ), @"Current Timestep" );

    END;

    FUNCTION       "OutflowToGetBelowPrudentPool" ( OBJECT reservoir, STRING season )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SolveOutflow"( reservoir, "CurrentInflow"( reservoir ), "ElevationToStorage"( reservoir, (
STRINGIFY reservoir ) CONCAT "Data.PrudentPoolElevations" [season, 0.00000000] - 0.01000000
["feet"] ), "PreviousStorage"( reservoir ), @"Current Timestep" );

    END;

    FUNCTION       "ComputePercentElVadoRGRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      1.00000000 - ( "TargetStorage"( % "ElVado", @"Current Timestep" ) -
"ElVadoAccountStorageAdjustment"(  ) - "PreviousAccountStorage"( "RioGrande", % "ElVado" ) ) /
"ForecastElVadoInflow"( @"Current Timestep", "TargetElevationDate"( % "ElVado", @"Current
Timestep" ) );
```

```
    END;

    FUNCTION       "ReachLaggedFlow" ( OBJECT object, DATETIME date, NUMERIC inflow, NUMERIC
previousInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "FlowFraction"( object, inflow, date ) * inflow + ( 1.00000000 - "FlowFraction"( object,
previousInflow, date - 1.00000000 ["day"] ) ) * previousInflow;

    END;

    FUNCTION       "CurrentRGInflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "ElVado" )
   THEN
      IF ( IsNaN $ "ElVado.Inflow" [] )
      THEN
         "RGOutflow"( % "Heron" ) + $ "ElVadoLocalInflow.Local Inflow" [] + IF ( NOT IsNaN %
"Heron" & "Seepage" [@"Previous Timestep"] )
         THEN
            % "Heron" & "Seepage" [@"Previous Timestep"]
         ELSE
            0.00000000 ["cfs"]
         ENDIF
      ELSE
         $ "ElVado.Inflow" [] - "SumSupplies"( "AllHeronSJSupplies"(  ) )
      ENDIF
   ELSE
      IF ( ( STRINGIFY reservoir ) == "Abiquiu" )
      THEN
         $ "Abiquiu.Inflow" [] - "SumPreviousSupplies"( "AllElVadoSJSupplies"(  ) ) * (
1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) )
      ELSE
         IF ( ( STRINGIFY reservoir ) == "Cochiti" )
         THEN
            $ "Cochiti.Inflow" [] - "SumPreviousSupplies"( "AllAbiquiuSJSupplies"(  ) ) * (
1.00000000 - "SJCLoss"( % "Abiquiu", % "Cochiti" ) )
         ELSE
            IF ( ( STRINGIFY reservoir ) == "Jemez" )
            THEN
               $ "Jemez.Inflow" []
            ELSE
               0.00000000 ["cfs"]
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "SolveStorageBasedOnCurrentInflow" ( OBJECT reservoir, NUMERIC inflow, NUMERIC
outflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SolveStorage"( reservoir, inflow, outflow, "PreviousStorage"( reservoir ), @"Current
Timestep" );

    END;
```

```
    FUNCTION       "StorageToElevationBasedOnCurrentInflow" ( OBJECT reservoir, NUMERIC inflow,
NUMERIC outflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StorageToElevation"( reservoir, "SolveStorageBasedOnCurrentInflow"( reservoir, inflow,
outflow ) );

    END;

    FUNCTION       "TotalAccountStorageAvailable" ( LIST accounts, OBJECT upstreamReservoir,
OBJECT downstreamReservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      result + "Min"( "Max"( "PreviousAccountStorage"( account, upstreamReservoir ), 0.00000000
["acre-feet"] ), "AccountStorageAvailable"( account, downstreamReservoir ) )
   ENDFOR;

    END;

    FUNCTION       "TotalPotentialDestinationRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "StringifyObject"( reservoir ) == "Heron" )
   THEN
      "Max"( "Min"( "VolumeToFlow"( "HeronAvailableDownstreamAccountStorage"(  ), @"Current
Timestep" ) + "CurrentTotalObjectDebtFlow"( % "Heron" ) + "HeronCochitiRecPoolRelease"(  ),
"VolumeToFlow"( "SumAccountStorageNonNegative"( % "Heron", "HeronToMRGCDAccountsList"(  ) ),
@"Current Timestep" ) ), "CurrentTotalObjectDebtFlow"( % "Heron" ) +
"HeronCochitiRecPoolRelease"(  ) )
   ELSE
      IF ( "StringifyObject"( reservoir ) == "ElVado" )
      THEN
         IF ( "ElVadoRaftingSeason"(  ) )
         THEN
            "Min"( "MaxElVadoMRGCDSJRelease"(  ) + "CurrentTotalObjectDebtFlow"( % "ElVado" ) +
"TotalElVadoFlowThruAccounts"(  ), "ElVadoRaftingRelease"(  ) + "TotalElVadoFlowThruAccounts"(  )
)
         ELSE
            "MaxElVadoMRGCDSJRelease"(  ) + "CurrentTotalObjectDebtFlow"( % "ElVado" ) +
"TotalElVadoFlowThruAccounts"(  )
         ENDIF
      ELSE
         IF ( "StringifyObject"( reservoir ) == "Abiquiu" )
         THEN
            IF ( "IrrigationSeason"(  ) )
            THEN
               "Max"( "MaxAbiquiuMRGCDSJRelease"(  ) - "RGOutflow"( % "Abiquiu" ), 0.00000000
["cfs"] ) + "AbiquiuMinFlowsSJRelease"(  ) + "SumPreviousSupplies"( "MakeSupplyList"(
"ElVadoThruCochitiFlowthruAccounts"(  ), "ElVadoThruCochitiFlowthruAccounts"(  ), % "ElVado", %
"BlwElVado" ) ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) ) +
"LetterWaterAdjustment"( % "Abiquiu", @"Current Timestep" )
            ELSE
               "CurrentTotalObjectDebtFlow"( % "Abiquiu" ) + "SumFlowThruAccounts"(
"AbiquiuFlowthruAccounts"(  ), % "BlwElVado", % "BlwElVadoToAbvAbiquiu" ) * ( 1.00000000 -
"SJCLoss"( % "ElVado", % "Abiquiu" ) ) + "VolumeToFlow"( "SumFlowThruAccountStorage"(
"AbiquiuFlowthruAccounts"(  ), % "Abiquiu" ), @"Current Timestep" ) + "AbiquiuMinFlowsSJRelease"(
)
            ENDIF
         ELSE
            0.00000000 ["cfs"]
```

```
        ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ZeroNaNs" ( SLOT slot, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN slot [date] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      slot [date]
   ENDIF;

    END;

    FUNCTION       "VolumeOfRelease" ( NUMERIC flow, DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( DATETIME date IN startDate TO endDate ) WITH NUMERIC result = 0.00000000 ["acre-
feet"] DO
      result + "FlowToVolume"( flow, date )
   ENDFOR;

    END;

    FUNCTION       "SumElVadoWeeklyDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( DATETIME date IN @"Current Timestep" TO @"Current Timestep + 6 Timesteps" ) WITH
NUMERIC result = 0.00000000 ["cfs"] DO
      $ "ElVadoData.MRGCDDemand" [date] + result
   ENDFOR;

    END;

    FUNCTION       "ComputeOutflowToGetToZeroStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SolveOutflow"( reservoir, reservoir & "Inflow" [], 0.10000000 ["acre-feet"], reservoir &
"Storage" [@"Previous Timestep"], @"Current Timestep" );

    END;

  END;

  UTILITY_GROUP "General Tcl Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "Units NONE" ( NUMERIC days )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      days / 1.00000000 ["day"];

    END;

    FUNCTION       "StringifyObject" ( OBJECT object )
    RETURN_TYPE    STRING;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      STRINGIFY object;

    END;

    FUNCTION       "ObjectifyString" ( STRING string )
    RETURN_TYPE    OBJECT;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetObject"( string );

    END;

    FUNCTION       "SlotifyString" ( STRING slot )
    RETURN_TYPE    SLOT;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "GetSlot"( slot );

    END;

    FUNCTION       "round" ( NUMERIC number, NUMERIC factor )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "Ceiling"( number, factor ) == "Ceiling"( number + factor / 2.00000000, factor ) )
   THEN
      "Floor"( number, factor ) * factor
   ELSE
      "Ceiling"( number, factor ) * factor
   ENDIF;

    END;

    EXTERNAL_FUNCTION "round10cfs" ( NUMERIC flow )
    RETURN_TYPE       NUMERIC;
    SCALE_UNITS       "";
    DESCRIPTION       "";
    ACTIVE            TRUE;
    PRE_EXEC_DIAG     FALSE;
    POST_EXEC_DIAG    FALSE;
    BEGIN
# convert metric flow values to english units (cfs)
set flow [expr $flow * 35.3146667214886]
```

```
if {$flow < 0.0} {
   set flow [expr $flow - 5.0000]
} else {
   set flow [expr $flow + 5.0000]
}

set flow [expr $flow / 10]
set iflow [expr int($flow)]
set flow [expr $iflow * 10]

return "$flow \[\"cfs\"\]"
    END;

    EXTERNAL_FUNCTION "StringifyAggObjectElement" ( OBJECT object )
    RETURN_TYPE       STRING;
    SCALE_UNITS       "";
    DESCRIPTION       "";
    ACTIVE            TRUE;
    PRE_EXEC_DIAG     FALSE;
    POST_EXEC_DIAG    TRUE;
    BEGIN
set newList [split $object :]
if {[llength $newList] > 1} {
   set returnString [lindex $newList 1]
} else {
   set returnString [lindex $newList 0]
}
return \"$returnString\"
    END;

  END;

  UTILITY_GROUP "General List Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "CreateNumericList" ( NUMERIC startNumber, NUMERIC endNumber, NUMERIC offset )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( GET NUMERIC @INDEX ( LENGTH result ) - 1.00000000 FROM result ) < endNumber )
WITH LIST result = { startNumber } DO
      IF ( ( GET NUMERIC @INDEX ( LENGTH result ) - 1.00000000 FROM result ) + offset <=
endNumber )
      THEN
         APPEND ( GET NUMERIC @INDEX ( LENGTH result ) - 1.00000000 FROM result ) + offset ONTO
result
      ELSE
         APPEND endNumber ONTO result
      ENDIF
   ENDWHILE;

    END;

    FUNCTION       "MaxList" ( LIST list )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( NUMERIC index IN list ) WITH NUMERIC result = GET NUMERIC @INDEX 0.00000000 FROM list
DO
      IF ( index > result )
      THEN
         index
      ELSE
         result
      ENDIF
   ENDFOR;
```

```
    END;

    FUNCTION       "MinList" ( LIST list )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( NUMERIC index IN list ) WITH NUMERIC result = GET NUMERIC @INDEX 0.00000000 FROM list
DO
      IF ( index < result )
      THEN
         index
      ELSE
         result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "NumericListElement" ( NUMERIC number, LIST list )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( WHILE ( number != GET NUMERIC @INDEX result FROM list ) WITH NUMERIC result =
0.00000000 DO
      result + 1.00000000
   ENDWHILE == ( LENGTH list ) - 1.00000000 AND number != GET NUMERIC @INDEX ( LENGTH list ) -
1.00000000 FROM list )
   THEN
      - 1.00000000
   ELSE
      WHILE ( number != GET NUMERIC @INDEX result FROM list ) WITH NUMERIC result = 0.00000000 DO
         result + 1.00000000
      ENDWHILE
   ENDIF;

    END;

    FUNCTION       "SumList" ( LIST values )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST numbers = "CreateNumericList"( 1.00000000, ( LENGTH values ) - 1.00000000,
1.00000000 ) DO
      FOR ( NUMERIC index IN numbers ) WITH NUMERIC result = GET NUMERIC @INDEX 0.00000000 FROM
values DO
         result + GET NUMERIC @INDEX index FROM values
      ENDFOR
   ENDWITH;

    END;

    FUNCTION       "SumListOfLists1" ( LIST values )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( ( LENGTH values ) == 0.00000000 )
   THEN
      0.00000000 ["cfs"]
   ELSE
```

```
      FOR ( LIST value IN values ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         result + GET NUMERIC @INDEX 1.00000000 FROM value
      ENDFOR
   ENDIF;

    END;

    FUNCTION       "NoNaNColumnList" ( SLOT slot, NUMERIC startRow, NUMERIC endRow, NUMERIC
column )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( NUMERIC index IN "CreateNumericList"( startRow, endRow, 1.00000000 ) ) WITH LIST
result = {  } DO
      IF ( IsNaN slot [index, column] )
      THEN
         result
      ELSE
         APPEND slot [index, column] ONTO result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "MonthlyDATETIMEList" ( DATETIME startDate, DATETIME endDate )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( GET DATETIME @INDEX ( LENGTH result ) - 1.00000000 FROM result ) < endDate ) WITH
LIST result = { "OffsetDate"( startDate, "Units NONE"( "GetDaysInMonth"( startDate ) - 1.00000000
["day"] ), "1 days" ) } DO
      APPEND "OffsetDate"( ( GET DATETIME @INDEX ( LENGTH result ) - 1.00000000 FROM result ) +
1.00000000 ["day"], "Units NONE"( "GetDaysInMonth"( ( GET DATETIME @INDEX ( LENGTH result ) -
1.00000000 FROM result ) + 1.00000000 ["day"] ) - 1.00000000 ["day"] ), "1 days" ) ONTO result
   ENDWHILE;

    END;

  END;

  UTILITY_GROUP "General Reach And Diversion Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EstimatedLagAndLossLookAhead" ( OBJECT reach, NUMERIC inflow, NUMERIC
previousinflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( "EstimatedVariableTimeLagLookAhead"( reach, inflow, previousinflow ) *
"VariableLossCoeff"( reach ) ) + "EstimatedVariableTimeLagLookAhead"( reach, inflow,
previousinflow );

    END;

    FUNCTION       "EstimatedLagAndLoss" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      ( "EstimatedVariableTimeLag"( reach ) * "VariableLossCoeff"( reach ) ) +
"EstimatedVariableTimeLag"( reach );

    END;

    FUNCTION       "EstimatedVariableTimeLagLookAhead" ( OBJECT routreach, NUMERIC inflow,
NUMERIC previousinflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( previousinflow * ( "TableInterpolation"( routreach & "Variable LagTime Table",
1.00000000, 2.00000000, previousinflow, @"Current Timestep" ) / 1.00000000 ["day"] ) ) + ( inflow
* ( ( 1.00000000 ["day"] - "TableInterpolation"( routreach & "Variable LagTime Table",
1.00000000, 2.00000000, inflow, @"Current Timestep" ) ) / 1.00000000 ["day"] ) );

    END;

    FUNCTION       "EstimatedVariableTimeLag" ( OBJECT routreach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function returns the estimated lagged flow using the VariableTimeLag
method.  It is an estimate because only one of the lag times for a given flow range is used in
the computation
(the same computation as the TimeLag method).  Use of the function requires the routing reach
object and flow range (integer) as arguments.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN routreach & "Inflow" [@"Previous Timestep"] )
   THEN
      routreach & "Inflow" []
   ELSE
      ( routreach & "Inflow" [@"Previous Timestep"] * ( "TableInterpolation"( routreach &
"Variable LagTime Table", 1.00000000, 2.00000000, routreach & "Inflow" [@"Previous Timestep"],
@"Current Timestep" ) / 1.00000000 ["day"] ) ) + ( routreach & "Inflow" [] * ( ( 1.00000000
["day"] - "TableInterpolation"( routreach & "Variable LagTime Table", 1.00000000, 2.00000000,
routreach & "Inflow" [], @"Current Timestep" ) ) / 1.00000000 ["day"] ) )
   ENDIF;

    END;

    FUNCTION       "InflowLookAhead" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reach & "Inflow" ["LookAhead"(  )];

    END;

    FUNCTION       "InflowLookAhead-1" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reach & "Inflow" ["LookAhead-1"(  )];

    END;

    FUNCTION       "InflowLookAhead-2" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reach & "Inflow" ["LookAhead-2"(  )];

    END;

    FUNCTION       "RoutedRGFlowAbiquiuToCochiti" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeReachVariableLagLossOutflow"( % "OtowiToCochiti", @"Current Timestep",
"ComputeReachVariableLagLossOutflow"( % "ConfluenceToOtowi", @"Current Timestep",
"ComputeReachVariableLagLossOutflow"( % "BlwAbiquiuToChamita", @"Current Timestep", $
"Abiquiu.Outflow" [] - $ "AbvAbiquiuDiversions.Total Depletion Requested" [],
"PreviousReachInflow"( % "BlwAbiquiuToChamita" ) ) - $ "AbvConfluenceDiversions.Total Depletion
Requested" [] + $ "AbiquiuToChamitaLocalInflow.Local Inflow" [] - $
"BlwConfluenceDiversions.Total Depletion Requested" [] - $ "BlwChamitaDiversions.Total Depletion
Requested" [] + $ "RioGrandeChama.Inflow1" [], "PreviousReachInflow"( % "ConfluenceToOtowi" ) ) +
$ "EmbudoToOtowiLocalInflow.Local Inflow" [], "PreviousReachInflow"( % "OtowiToCochiti" ) ) + $
"RioGrandeSantaFeRiver.Inflow2" [] + $ "OtowiToCochitiLocalInflow.Local Inflow" [] -
"ReconcileSJOutflowAtGivenDate"( % "Abiquiu", @"Previous Timestep" ) * ( 1.00000000 - "SJCLoss"(
% "Abiquiu", % "Cochiti" ) );

    END;

    FUNCTION       "RoutedRGFlowElVadoToAbiquiu" ( DATETIME date, NUMERIC inflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeReachVariableLagLossOutflow"( % "BlwElVadoToAbvAbiquiu", date, inflow,
"PreviousOutflow"( % "ElVado" ) ) - $ "AbvAbiquiuDiversions.Total Depletion Requested" [date] + $
"ElVadoToAbiquiuLocalInflow.Local Inflow" [date] - "ReconcileSJOutflowAtGivenDate"( % "ElVado",
date - 1.00000000 ["day"] ) * ( 1.00000000 - "SJCLoss"( % "ElVado", % "Abiquiu" ) );

    END;

    FUNCTION       "VariableLossCoeff" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function returns the loss coefficient for a reach object.  When calling
the function, an object name must be placed in the argument list.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Month" == @"January" )
   THEN
      reach & "Variable GainLoss Coeff Table" ["Season 1", "Flow Range 1"]
   ELSE
      IF ( @"Current Month" == @"February" )
      THEN
         reach & "Variable GainLoss Coeff Table" ["Season 2", "Flow Range 1"]
      ELSE
         IF ( @"Current Month" == @"March" )
         THEN
            reach & "Variable GainLoss Coeff Table" ["Season 3", "Flow Range 1"]
         ELSE
            IF ( @"Current Month" == @"April" )
            THEN
               reach & "Variable GainLoss Coeff Table" ["Season 4", "Flow Range 1"]
            ELSE
               IF ( @"Current Month" == @"May" )
               THEN
                  reach & "Variable GainLoss Coeff Table" ["Season 5", "Flow Range 1"]
               ELSE
```

```
                  IF ( @"Current Month" == @"June" )
                  THEN
                     reach & "Variable GainLoss Coeff Table" ["Season 6", "Flow Range 1"]
                  ELSE
                     IF ( @"Current Month" == @"July" )
                     THEN
                        reach & "Variable GainLoss Coeff Table" ["Season 7", "Flow Range 1"]
                     ELSE
                        IF ( @"Current Month" == @"August" )
                        THEN
                           reach & "Variable GainLoss Coeff Table" ["Season 8", "Flow Range 1"]
                        ELSE
                           IF ( @"Current Month" == @"September" )
                           THEN
                              reach & "Variable GainLoss Coeff Table" ["Season 9", "Flow Range
1"]
                           ELSE
                              IF ( @"Current Month" == @"October" )
                              THEN
                                 reach & "Variable GainLoss Coeff Table" ["Season 10", "Flow
Range 1"]
                              ELSE
                                 IF ( @"Current Month" == @"November" )
                                 THEN
                                    reach & "Variable GainLoss Coeff Table" [10.00000000, "Flow
Range 1"]
                                 ELSE
                                    reach & "Variable GainLoss Coeff Table" [11.00000000, "Flow
Range 1"]
                                 ENDIF
                              ENDIF
                           ENDIF
                        ENDIF
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Get Data Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AccountFillDate" ( OBJECT reservoir )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( "OffsetDate"( @"24:00:00 December 31, Previous
Year", "StringifyObject"( reservoir ) CONCAT "Data.AccountFillMaxVolume" ["GetSeasonRow"(
reservoir, "AccountFillMaxVolume" ), 0.00000000], "1 days" ) ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", "StringifyObject"( reservoir ) CONCAT
"Data.AccountFillMaxVolume" ["GetSeasonRow"( reservoir, "AccountFillMaxVolume" ), 0.00000000], "1
days" ) + 1.00000000 ["day"]
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", "StringifyObject"( reservoir ) CONCAT
"Data.AccountFillMaxVolume" ["GetSeasonRow"( reservoir, "AccountFillMaxVolume" ), 0.00000000], "1
days" )
   ENDIF;

    END;

    FUNCTION       "AccountLeaseAmount" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data." CONCAT account CONCAT "WaterLease" [];

    END;

    FUNCTION       "AccountPriorityList" ( OBJECT object, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING account IN accounts ) WITH LIST result = {  } DO
      APPEND "Priority"( object, account ) ONTO result
   ENDFOR;

    END;

    FUNCTION       "CompactMinStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "RioGrandeCompactData.MinimumStorage" ["EButteCaballo", "MinStorage"];

    END;

    FUNCTION       "ComputedElVadoRaftingSchedule" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN $ "ElVadoData.ComputedRaftingSchedule" [] )
   THEN
      $ "ElVadoData.RaftingSchedule" ["GetDayOfWeekAsStringGivenDate"( @"Current Timestep" ),
"Release"]
   ELSE
      $ "ElVadoData.ComputedRaftingSchedule" []
   ENDIF;

    END;

    FUNCTION       "CurrentInflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Inflow" [];

    END;

    FUNCTION       "CurrentYearWaiverSwitch" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "HeronData.WaiverSwitch" [@"24:00:00 December 31, Current Year"];
```

```
    END;

    FUNCTION       "DeliveryRequestFor" ( STRING account )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      % "DeliveryRequests" & account [];

    END;

    FUNCTION       "DownstreamDemands" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "CheckReservourOutflow"( reservoir );
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.DownstreamDemand" [];

    END;

    FUNCTION       "ElevationCurveColumn" ( OBJECT reservoir, NUMERIC peakInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "NONE";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data.ElevationCurveTable"
["PeakInflowRow"( reservoir, peakInflow ), result] AND "PreviousElevation"( reservoir ) >= (
STRINGIFY reservoir ) CONCAT "Data.ElevationCurveTable" ["PeakInflowRow"( reservoir, peakInflow
), result] AND result <= 10.00000000 ) WITH NUMERIC result = 1.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "FloodRelease" ( OBJECT reservoir, NUMERIC peakInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetFloodRelease"( reservoir, peakInflow );

    END;

    FUNCTION       "FlowRangeColumn" ( OBJECT object, NUMERIC inflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN object & "Flow Range" [0.00000000, result] AND inflow > object & "Flow
Range" [0.00000000, result] AND result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "GetFloodRelease" ( OBJECT reservoir, NUMERIC peakInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "StringifyObject"( reservoir ) CONCAT "Data.FloodReleaseTable" ["PeakInflowRow"( reservoir,
peakInflow ), "ElevationCurveColumn"( reservoir, peakInflow )];

    END;

    FUNCTION       "GetIndianStorageRequirement" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN $ "ElVadoData.IndianStorageReq" [date] )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      $ "ElVadoData.IndianStorageReq" [date]
   ENDIF;

    END;

    FUNCTION       "IndianCall" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "Indian.Indian Call" [];

    END;

    FUNCTION       "VariableLagTime" ( OBJECT object, NUMERIC inflow, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "TableInterpolation"( object & "Variable LagTime Table", 1.00000000, 2.00000000, inflow,
date );

    END;

    FUNCTION       "LastTargetElevationRow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data.TargetElevation" [result,
0.00000000] AND result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "LockedInStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
```

```
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.LockedInStorage" [];

    END;

    FUNCTION       "LossRate" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      object & "Variable GainLoss Coeff Table" ["GetMonth"( @"Current Timestep" ) - 1.00000000,
"Flow Range 1"];

    END;

    FUNCTION       "MaxAccountStorage" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MaxAccountStorage" ["GetSeasonRow"( reservoir,
"MaxAccountStorage" ), account];

    END;

    FUNCTION       "MaxAllowableElevation" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MaxAllowableElevation" ["GetMonthAsString"( @"Current
Timestep" ), STRINGIFY reservoir];

    END;

    FUNCTION       "MaxElevation" ( OBJECT res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY res CONCAT "Data.ReservoirData" ) [STRINGIFY res, "MaxElevation"];

    END;

    FUNCTION       "MaxOutflow" ( OBJECT res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY res CONCAT "Data.ReservoirData" ) [STRINGIFY res, "maxRelease"];

    END;

    FUNCTION       "MaxStorage" ( OBJECT res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( res, "MaxElevation"( res ) );

    END;

    FUNCTION       "MaxTargetElevationRow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "NumericListElement"( "MaxList"( "NoNaNColumnList"( "ObjectifyString"( "StringifyObject"(
reservoir ) CONCAT "Data" ) & "TargetElevation", 0.00000000, 11.00000000, 1.00000000 ) ),
"NoNaNColumnList"( "ObjectifyString"( "StringifyObject"( reservoir ) CONCAT "Data" ) &
"TargetElevation", 0.00000000, 11.00000000, 1.00000000 ) );

    END;

    FUNCTION       "MaximumSJOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "TotalPotentialDestinationRelease"( reservoir ), "StringifyObject"( reservoir )
CONCAT "Data.MaximumSJOutflow" ["GetSeasonRow"( reservoir, "MaximumSJOutflow" ), 1.00000000] );

    END;

    FUNCTION       "MinLoanPool" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data." CONCAT "MinPool" CONCAT ( account CONCAT "Loan" )
[0.00000000, 0.00000000];

    END;

    FUNCTION       "MinElevation" ( OBJECT res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY res CONCAT "Data.ReservoirData" ) [STRINGIFY res, "MinElevation"];

    END;

    FUNCTION       "MinElVadoRGStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MinElVadoRGStorage" [0.00000000, 0.00000000];

    END;

    FUNCTION       "MinOutflow" ( OBJECT res )
```

```
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY res ) == "ElVado" )
   THEN
      "Min"( "Max"( "CurrentRGInflow"( res ), 0.01000000 ["cfs"] ), ( STRINGIFY res CONCAT
"Data.MinRelease" ) ["GetSeasonRow"( res, "MinRelease" ), "MinRelease"] )
   ELSE
      IF ( ( STRINGIFY res ) == "Abiquiu" )
      THEN
         ( STRINGIFY res CONCAT "Data.MinRelease" ) ["GetSeasonRow"( res, "MinRelease" ),
"MinRelease"]
      ELSE
         ( STRINGIFY res CONCAT "Data.MinRelease" ) ["GetSeasonRow"( res, "MinRelease" ),
"MinRelease"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "MinRGOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY reservoir ) == "ElVado" )
   THEN
      "Min"( "Max"( "CurrentRGInflow"( reservoir ), 0.01000000 ["cfs"] ),
"ElVadoData.MinRGOutflow" ["GetSeasonRow"( reservoir, "MinRGOutflow" ), "MinRGOutflow"] )
   ELSE
      IF ( ( STRINGIFY reservoir ) == "Abiquiu" OR ( STRINGIFY reservoir ) == "Cochiti" )
      THEN
         ( STRINGIFY reservoir ) CONCAT "Data.MinRGOutflow" ["GetSeasonRow"( reservoir,
"MinRGOutflow" ), "MinRGOutflow"]
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "MinStorage" ( OBJECT res )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( res, "MinElevation"( res ) );

    END;

    FUNCTION       "MinimumMonthlyFlows" ( OBJECT reservoir, STRING location )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MinimumFlows" ["GetMonthAsString"( @"Current Timestep"
), location];

    END;

    FUNCTION       "MinimumMRGCDStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StringifyObject"( reservoir ) CONCAT "Data.MinimumMRGCDStorage" [0.00000000, 0.00000000];

    END;

    FUNCTION       "NMDebit" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Min"( $ "RioGrandeCompactData.NMCreditDebit" [@"24:00:00 December 31, Previous Year"],
"MaxCompactCredit"( "NM" ) );

    END;

    FUNCTION       "PeakInflowRow" ( OBJECT reservoir, NUMERIC peakInflow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "NONE";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data.ElevationCurveTable" [result,
0.00000000] AND peakInflow >= ( STRINGIFY reservoir ) CONCAT "Data.ElevationCurveTable" [result,
0.00000000] AND result <= 249.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "PreviousAccountGainLoss" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Gain Loss"
[@"Previous Timestep"] )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      ( STRINGIFY reservoir ) CONCAT "^" CONCAT account CONCAT ".Gain Loss" [@"Previous
Timestep"]
   ENDIF;

    END;

    FUNCTION       "PreviousElevation" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Pool Elevation" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousForecastError" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( @"Current Timestep" == @"Start Timestep" )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      IF ( "GetMonth"( @"Current Timestep" ) == 1.00000000 )
      THEN
         0.00000000 ["acre-feet"]
      ELSE
         % "ForecastData" & "ForecastError" [@"24:00:00 Previous Month Max DayOfMonth, Current
Year"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "PreviousIncidentalContent" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Incidental Content" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousLockedInStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN ( STRINGIFY reservoir ) CONCAT "Data.LockedInStorage" [@"Previous Timestep"] )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      ( STRINGIFY reservoir ) CONCAT "Data.LockedInStorage" [@"Previous Timestep"]
   ENDIF;

    END;

    FUNCTION       "PreviousOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Outflow" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousReachInflow" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      object & "Inflow" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      reservoir & "Storage" [@"Previous Timestep"];

    END;

    FUNCTION       "PreviousYearWaiverSwitch" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "HeronData.WaiverSwitch" [@"24:00:00 December 31, Previous Year"];

    END;

    FUNCTION       "PreviousWaiverBalance" ( STRING account )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" == @"Start Timestep" AND @"Current Timestep" == @"24:00:00 January
1, Current Year" )
   THEN
      "Heron^" CONCAT account CONCAT ".Carry Over" []
   ELSE
      "HeronData." CONCAT account CONCAT "WaiverBalance" [@"Previous Timestep"]
   ENDIF;

    END;

    FUNCTION       "Priority" ( OBJECT object, STRING account )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object ) CONCAT "Data.AccountReleasePriority" [account, "SeasonColumn"( object
)];

    END;

    FUNCTION       "ReachInflow" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN object & "Inflow" [] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      object & "Inflow" []
   ENDIF;

    END;

    FUNCTION       "ReleasePriority" ( OBJECT object, STRING type )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object ) CONCAT "Data.ReleaseTypePriority" [type, "ReleaseTypeColumn"( object
)];

    END;

    FUNCTION       "ReleasePriorityList" ( OBJECT object )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING type IN "ReleaseTypesList"( object ) ) WITH LIST result = {  } DO
      APPEND "ReleasePriority"( object, type ) ONTO result
   ENDFOR;

    END;

    FUNCTION       "ReleaseTypeColumn" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( object ) CONCAT "Data.ReleaseTypePriority"
[0.00000000, result] AND "SeasonDayOfYear"( @"Current Timestep" ) > "StringifyObject"( object )
CONCAT "Data.ReleaseTypePriority" [0.00000000, result] AND result <= 11.00000000 ) WITH NUMERIC
result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "ReservoirPriority" ( OBJECT object, STRING reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object ) CONCAT "Data.ReservoirPriority" [reservoir, "ReservoirSeasonColumn"(
object )];

    END;

    FUNCTION       "ReservoirPriorityList" ( OBJECT object, LIST reservoirs )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING reservoir IN reservoirs ) WITH LIST result = {  } DO
      APPEND "ReservoirPriority"( object, reservoir ) ONTO result
   ENDFOR;

    END;

    FUNCTION       "ReservoirSeasonColumn" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
```

```
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( object ) CONCAT "Data.ReservoirPriority" [0.00000000,
result] AND "SeasonDayOfYear"( @"Current Timestep" ) > "StringifyObject"( object ) CONCAT
"Data.ReservoirPriority" [0.00000000, result] AND result <= 11.00000000 ) WITH NUMERIC result =
0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "SeasonColumn" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( object ) CONCAT "Data.AccountReleasePriority"
[0.00000000, result] AND "SeasonDayOfYear"( @"Current Timestep" ) > "StringifyObject"( object )
CONCAT "Data.AccountReleasePriority" [0.00000000, result] AND result <= 11.00000000 ) WITH
NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "SeasonDayOfYear" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( date ) > 60.00000000 ["day"] )
   THEN
      "Units NONE"( "GetDayOfYear"( date ) ) - 1.00000000
   ELSE
      "Units NONE"( "GetDayOfYear"( date ) )
   ENDIF;

    END;

    FUNCTION       "SeasonRow" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data.TargetElevation" [result,
0.00000000] AND "SeasonDayOfYear"( date ) > "StringifyObject"( reservoir ) CONCAT
"Data.TargetElevation" [result, 0.00000000] AND result <= 11.00000000 ) WITH NUMERIC result =
0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "SJCLoss" ( OBJECT fromReservoir, OBJECT toReservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "SanJuanChamaRules.Losses" [STRINGIFY fromReservoir, STRINGIFY toReservoir];

    END;

    FUNCTION       "SJOutflowRow" ( OBJECT reservoir, DATETIME date )
```

```
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data.MaximumSJOutflow" [result,
0.00000000] AND "SeasonDayOfYear"( date ) > "StringifyObject"( reservoir ) CONCAT
"Data.MaximumSJOutflow" [result, 0.00000000] AND result <= 11.00000000 ) WITH NUMERIC result =
0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "SupplyFlow" ( STRING slot )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN slot [] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      slot []
   ENDIF;

    END;

    FUNCTION       "TargetElevation" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH DATETIME targetElevDate = "ComputeTargetElevationDate"( reservoir, date ) DO
      IF ( targetElevDate > @"Finish Timestep" )
      THEN
         "InterpolateTargetElevation"( reservoir, @"Finish Timestep" )
      ELSE
         "StringifyObject"( reservoir ) CONCAT "Data.TargetElevation" ["GetSeasonRowGivenDate"(
"SlotifyString"( "StringifyObject"( reservoir ) CONCAT "Data.TargetElevation" ), date ),
1.00000000]
      ENDIF
   ENDWITH;

    END;

    FUNCTION       "TargetElevationDate" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH DATETIME targetDate = "ComputeTargetElevationDate"( reservoir, date ) DO
      IF ( targetDate > @"Finish Timestep" )
      THEN
         @"Finish Timestep"
      ELSE
         targetDate
      ENDIF
   ENDWITH;

    END;

    FUNCTION       "TargetFillDate" ( OBJECT reservoir )
    RETURN_TYPE    DATETIME;
```

```
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( "OffsetDate"( @"24:00:00 December 31, Previous
Year", ( STRINGIFY reservoir ) CONCAT "Data.TargetElevation" ["MaxTargetElevationRow"( reservoir
), 0.00000000], "1 days" ) ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["MaxTargetElevationRow"( reservoir ), 0.00000000] + 1.00000000, "1 days"
)
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["MaxTargetElevationRow"( reservoir ), 0.00000000], "1 days" )
   ENDIF;

    END;

    FUNCTION       "TargetStorage" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( reservoir, "TargetElevation"( reservoir, date ) );

    END;

    FUNCTION       "TexasCall" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "RioGrandeCompactData.Texas Call" [];

    END;

    FUNCTION       "WaiverDay" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "HeronData.WaiverDate" [0.00000000, 0.00000000];

    END;

    FUNCTION       "WaiverDate" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "OffsetDate"( @"24:00:00 December 31, Previous Year", "WaiverDay"(  ), "1 days" );

    END;

    FUNCTION       "GainSeasonRow" ( OBJECT object, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN "StringifyObject"( object ) CONCAT ".Date Range" ["End Day", result] AND
"SeasonDayOfYear"( date ) > "StringifyObject"( object ) CONCAT ".Date Range" ["End Day", result]
AND result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "SeasonGainLoss" ( OBJECT object, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      object & "Variable GainLoss Table" ["GainSeasonRow"( object, date ), "FlowRangeColumn"(
object, "ReachInflow"( object ) )];

    END;

    FUNCTION       "SeasonGainLossCoeff" ( OBJECT object, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      object & "Variable GainLoss Coeff Table" ["GainSeasonRow"( object, date ),
"FlowRangeColumn"( object, "ReachInflow"( object ) )];

    END;

    FUNCTION       "AccountFillMaxVolume" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StringifyObject"( reservoir ) CONCAT "Data.AccountFillMaxVolume" ["GetSeasonRow"(
reservoir, "AccountFillMaxVolume" ), 1.00000000];

    END;

    FUNCTION       "GetSeasonRow" ( OBJECT reservoir, STRING slot )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH NUMERIC dayOfYear = "SeasonDayOfYear"( @"Current Timestep" ) DO
      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data." CONCAT slot [result,
0.00000000] AND dayOfYear > "StringifyObject"( reservoir ) CONCAT "Data." CONCAT slot [result,
0.00000000] AND result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
         result + 1.00000000
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "IndianStorageRequirement" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      IF ( "EndOfMonth"(  ) > @"Finish Timestep" )
   THEN
      "InterpolateIndianStorageReq"(  )
   ELSE
      "GetIndianStorageRequirement"( "EndOfMonth"(  ) )
   ENDIF;

    END;

    FUNCTION       "PreviousSlotInflow" ( STRING slot )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN slot [@"Previous Timestep"] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      slot [@"Previous Timestep"]
   ENDIF;

    END;

    FUNCTION       "TotalOutflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.TotalOutflow" [];

    END;

    FUNCTION       "PercentElVadoRGRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "ElVadoData.PercentRGRelease" [];

    END;

    FUNCTION       "AlbuquerqueLoanSwitch" ( STRING account )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "AbiquiuData.AlbuquerqueLoanSwitch" [account, "GetSeasonColumn"( % "Abiquiu",
"AlbuquerqueLoanSwitch" )];

    END;

    FUNCTION       "GetSeasonColumn" ( OBJECT reservoir, STRING slot )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH NUMERIC dayOfYear = "SeasonDayOfYear"( @"Current Timestep" ) DO
```

```
      WHILE ( NOT IsNaN "StringifyObject"( reservoir ) CONCAT "Data." CONCAT slot [0.00000000,
result] AND dayOfYear > "StringifyObject"( reservoir ) CONCAT "Data." CONCAT slot [0.00000000,
result] AND result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
         result + 1.00000000
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "MaintenanceSwitch" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object ) CONCAT "Data.MaintenanceSwitch" ["GetSeasonRow"( object,
"MaintenanceSwitch" ), 1.00000000];

    END;

    FUNCTION       "SlotInflow" ( STRING slot )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( IsNaN slot [] )
   THEN
      0.00000000 ["cfs"]
   ELSE
      slot []
   ENDIF;

    END;

    FUNCTION       "AccountFillDateGivenDate" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( "OffsetDate"( @"24:00:00 December 31, Previous
Year", "StringifyObject"( reservoir ) CONCAT "Data.AccountFillMaxVolume"
["GetSeasonRowGivenDate"( "SlotifyString"( "StringifyObject"( reservoir ) CONCAT
"Data.AccountFillMaxVolume" ), date ), 0.00000000], "1 days" ) ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", "StringifyObject"( reservoir ) CONCAT
"Data.AccountFillMaxVolume" ["GetSeasonRowGivenDate"( "SlotifyString"( "StringifyObject"(
reservoir ) CONCAT "Data.AccountFillMaxVolume" ), date ), 0.00000000], "1 days" ) + 1.00000000
["day"]
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", "StringifyObject"( reservoir ) CONCAT
"Data.AccountFillMaxVolume" ["GetSeasonRowGivenDate"( "SlotifyString"( "StringifyObject"(
reservoir ) CONCAT "Data.AccountFillMaxVolume" ), date ), 0.00000000], "1 days" )
   ENDIF;

    END;

    FUNCTION       "GetSeasonRowGivenDate" ( SLOT slot, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH NUMERIC dayOfYear = "SeasonDayOfYear"( date ) DO
      WHILE ( NOT IsNaN slot [result, 0.00000000] AND dayOfYear > slot [result, 0.00000000] AND
result <= 11.00000000 ) WITH NUMERIC result = 0.00000000 DO
```

```
        result + 1.00000000
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "MaxAccountStorageTolerance" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object ) CONCAT "Data.MaxAccountStorageTolerance" [0.00000000, 0.00000000];

    END;

    FUNCTION       "MaxIncidentalContentRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MaxIncidentalContentRelease" [0.00000000, 0.00000000];

    END;

    FUNCTION       "MinComputedElVadoMRGCDRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Max"( $ "ElVadoData.ComputedMinMRGCDRelease" [] - "LetterWaterAdjustment"( % "ElVado",
"DatePlusXTimesteps"( @"Current Timestep", 1.00000000 ["day"] ) ), 0.00000000 ["cfs"] );

    END;

    FUNCTION       "MaxDeliveryRequestRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "AbiquiuData.MaxDeliveryRequestRelease" [0.00000000, 0.00000000];

    END;

    FUNCTION       "IndianDemand" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "Indian.IndianDemand" ["GetMonthAsString"( date ), 0.00000000];

    END;

    FUNCTION       "ComputeTargetElevationDate" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( date ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["SeasonRow"( reservoir, date ), 0.00000000] + 1.00000000, "1 days" )
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["SeasonRow"( reservoir, date ), 0.00000000], "1 days" )
   ENDIF;

    END;

    FUNCTION       "InterpolateTargetElevation" ( OBJECT reservoir, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.TargetElevation" ["Max"( "GetSeasonRowGivenDate"(
"SlotifyString"( ( STRINGIFY reservoir ) CONCAT "Data.TargetElevation" ), date ) - 1.00000000,
0.00000000 ), 1.00000000] + ( "SeasonDayOfYear"( date ) - ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["Max"( "GetSeasonRowGivenDate"( "SlotifyString"( ( STRINGIFY reservoir )
CONCAT "Data.TargetElevation" ), date ) - 1.00000000, 0.00000000 ), 0.00000000] ) / ( ( STRINGIFY
reservoir ) CONCAT "Data.TargetElevation" ["GetSeasonRowGivenDate"( "SlotifyString"( ( STRINGIFY
reservoir ) CONCAT "Data.TargetElevation" ), date ), 0.00000000] - ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["Max"( "GetSeasonRowGivenDate"( "SlotifyString"( ( STRINGIFY reservoir )
CONCAT "Data.TargetElevation" ), date ) - 1.00000000, 0.00000000 ), 0.00000000] ) * ( ( STRINGIFY
reservoir ) CONCAT "Data.TargetElevation" ["GetSeasonRowGivenDate"( "SlotifyString"( ( STRINGIFY
reservoir ) CONCAT "Data.TargetElevation" ), date ), 1.00000000] - ( STRINGIFY reservoir ) CONCAT
"Data.TargetElevation" ["Max"( "GetSeasonRowGivenDate"( "SlotifyString"( ( STRINGIFY reservoir )
CONCAT "Data.TargetElevation" ), date ) - 1.00000000, 0.00000000 ), 1.00000000] );

    END;

    FUNCTION       "InterpolateIndianStorageReq" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "ElVadoData.IndianStorageReq" ["EndOfPreviousMonth"(  )] + "GetDayOfMonth"( @"Finish
Timestep" ) / "GetDaysInMonth"( @"Finish Timestep" ) * ( "GetIndianStorageRequirement"(
"EndOfMonth"(  ) ) - $ "ElVadoData.IndianStorageReq" ["EndOfPreviousMonth"(  )] );

    END;

    FUNCTION       "EndOfPreviousMonth" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "OffsetDate"( @"24:00:00 Current Month Max DayOfMonth, Current Year", - 1.00000000, "1
Months" );

    END;

    FUNCTION       "PreviousPoolElevation" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      reservoir & "Pool Elevation" [@"Previous Timestep"];

    END;
```

```
    FUNCTION       "RaftingSeasonStartDate" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( @"Current Timestep" ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", $ "ElVadoData.RaftingSeasonDates"
[0.00000000, "StartDay"] + 1.00000000, "1 days" )
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", $ "ElVadoData.RaftingSeasonDates"
[0.00000000, "StartDay"], "1 days" )
   ENDIF;

    END;

    FUNCTION       "RaftingSeasonEndDate" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsLeapYear"(  ) AND "GetDayOfYear"( @"Current Timestep" ) > 60.00000000 ["day"] )
   THEN
      "OffsetDate"( @"24:00:00 December 31, Previous Year", $ "ElVadoData.RaftingSeasonDates"
[0.00000000, "EndDay"] + 1.00000000, "1 days" )
   ELSE
      "OffsetDate"( @"24:00:00 December 31, Previous Year", $ "ElVadoData.RaftingSeasonDates"
[0.00000000, "EndDay"], "1 days" )
   ENDIF;

    END;

    FUNCTION       "RemainingMRGCDFromAlbuquerquePurchase" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "SumSlot"( $ "HeronData.AlbuquerqueFromMRGCDLoan", @"Current Timestep", "MinDate"(
@"24:00:00 December 31, Current Year", @"Finish Timestep" ) );

    END;

    FUNCTION       "GetAbiquiuMRGCDDemand" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN $ "AbiquiuData.MRGCDDemand" [date] )
   THEN
      $ "AbiquiuData.MRGCDDemand" []
   ELSE
      $ "AbiquiuData.MRGCDDemand" [date]
   ENDIF;

    END;

    FUNCTION       "AbiquiuMiddleValleyDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( % "Abiquiu" );
    BEGIN

      $ "AbiquiuData.MRGCDDemand" [] + "PreviousSlotInflow"(
"CochitiRecPoolHeronCochitiRecPoolCochitiElVadoToCochitiRecPoolHeronCochitiRecPoolCochitiBlwElVad
o.Supply" ) + "AbiquiuMinFlowsSJRelease"(  );

    END;

    FUNCTION       "ZeroNaNTableSlot" ( SLOT slot, NUMERIC row, STRING column )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( IsNaN slot [row, column] )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      slot [row, column]
   ENDIF;

    END;

    FUNCTION       "MaxPercentOutflowDifference" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.MaxPercentOutflowDifference" [0.00000000, 0.00000000];

    END;

    FUNCTION       "NumDaysDownstream" ( OBJECT reservoir, STRING location )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.ApproxNoOfDaysDS" [location, 0.00000000];

    END;

    FUNCTION       "MinTargetFlow" ( STRING location, NUMERIC numDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "MiddleValleyDemands.MinTargetFlows" ["GetSeasonRowGivenDate"( $
"MiddleValleyDemands.MinTargetFlows", "DatePlusXTimesteps"( @"Current Timestep", numDays ) ),
location];

    END;

    FUNCTION       "MaxCompactCredit" ( STRING state )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      % "RioGrandeCompactData" & "Max" CONCAT state CONCAT "Credit" [0.00000000, 0.00000000];
```

```
    END;

    FUNCTION       "CODebit" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Min"( $ "RioGrandeCompactData.COCreditDebit" [@"24:00:00 December 31, Previous Year"],
"MaxCompactCredit"( "CO" ) );

    END;

    FUNCTION       "ChannelCapacityTolerance" ( OBJECT reservoir, STRING location )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.ChannelCapacities" ["Tolerance", location];

    END;

    FUNCTION       "AvailableStorage" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Max"( "MaxStorage"( reservoir ) - "PreviousStorage"( reservoir ), 0.00000000 ["acre-feet"]
);

    END;

    FUNCTION       "UnregulatedSpillWhenNoConduitFlow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetMinSpillGivenInflowRelease"( reservoir, reservoir & "Inflow" [], 0.00000000 ["cfs"],
@"Current Timestep" );

    END;

  END;

  UTILITY_GROUP "Heron Account Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ComputeHeronSJOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "MaximumSJOutflow"( % "Heron" );
    BEGIN

      IF ( NOT "ElVadoRaftingSeason"(  ) AND @"Current Timestep" <= @"24:00:00 July 14, Current
Year" )
   THEN
```

```
      "Max"( "HeronAccountFillRelease"( "AccountFillDate"( % "Heron" ) ) +
"HeronOtowiAndJemezPaybackRelease"(  ) + "MRGCDPaybackRelease"(  ) + "HeronWaiverRelease"(  ) +
"VolumeToFlow"( "HeronAlbuquerquePaybackVolume"( "HeronRootAlbuquerquePaybackAccounts"(  ) ),
@"Current Timestep" ) + "VolumeToFlow"( "HeronCochitiRecPoolPaybackVolume"(  ), @"Current
Timestep" ), "MinHeronSJOutflow"(  ) ) + "ElVadoDeliveryRelease"(  )
   ELSE
      IF ( NOT "ElVadoRaftingSeason"(  ) AND @"Current Timestep" > @"24:00:00 July 14, Current
Year" AND @"Current Timestep" <= @"24:00:00 August 31, Current Year" )
      THEN
         "MinHeronSJOutflow"(  ) + "MRGCDPaybackRelease"(  ) + "ElVadoDeliveryRelease"(  )
      ELSE
         IF ( "ElVadoRaftingSeason"(  ) )
         THEN
            "HeronRaftingSeasonSJRelease"(  ) + "MRGCDPaybackRelease"(  ) +
"ElVadoDeliveryRelease"(  )
         ELSE
            IF ( "CurrentYearIsWaiverYear"(  ) )
            THEN
               "Max"( "HeronAlbuquerqueAbiquiuDebtRelease"(  ) + "HeronCochitiRecPoolRelease"(  )
+ "HeronOtowiAndJemezPaybackRelease"(  ), "MinHeronSJOutflow"(  ) ) + "MRGCDPaybackRelease"(  ) +
"ElVadoDeliveryRelease"(  )
            ELSE
               "Max"( "SumAccountStorageNonNegative"( % "Heron", "HeronRootAllAccounts"(  ) ) / (
@"24:00:00 January 1, Next Year" - @"Current Timestep" ), "MinHeronSJOutflow"(  ) ) +
"MRGCDPaybackRelease"(  )
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "HeronAccountFillRelease" ( DATETIME endDate )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "HeronAvailableDownstreamAccountStorage"(  ), "Max"( "AccountFillMaxVolume"( %
"Heron" ) - $ "HeronData.CumulativeAccountFillRelease" [@"Previous Timestep"], 0.00000000 ["acre-
feet"] ) ) / "Max"( endDate + 1.00000000 ["day"] - @"Current Timestep", 1.00000000 ["day"] );

    END;

    FUNCTION       "HeronAlbuquerqueAbiquiuDebtRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "HeronAlbuquerquePaybackVolume"( "HeronRootAlbuquerquePaybackAccounts"(  ) ) / ( @"24:00:00
January 1, Next Year" - @"Current Timestep" );

    END;

    FUNCTION       "HeronAlbuquerquePaybackVolume" ( LIST accounts )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST result = "HeronAlbuquerquePaybackVolumeList"( accounts ) DO
      GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM
result
   ENDWITH;

    END;

    FUNCTION       "HeronAlbuquerquePaybackVolumeList" ( LIST accounts )
    RETURN_TYPE    LIST;
```

```
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( accounts,
"MakeAlbuquerqueAbiquiuPaybackAccountsList"( accounts, % "Heron" ), % "Heron", % "HeronSeepage" )
) WITH LIST result = { { 0.00000000 ["acre-feet"] , 0.00000000 } } DO
      APPEND { "Min"( "Min"( "PreviousAccountStorage"( GET STRING @INDEX GET NUMERIC @INDEX
1.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM result FROM accounts, %
"Heron" ), "GetAccountDebt"( supply ) ), "Max"( "AccountStorageAvailable"( "Albuquerque", %
"Abiquiu" ) - GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000
FROM result, 0.00000000 ["acre-feet"] ) ) + GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX (
LENGTH result ) - 1.00000000 FROM result , ( GET NUMERIC @INDEX 1.00000000 FROM GET LIST @INDEX (
LENGTH result ) - 1.00000000 FROM result ) + 1.00000000 } ONTO result
   ENDFOR;

    END;

    FUNCTION       "HeronAvailableAbiquiuOnlyStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN "StorageInAbiquiuOnlyAccounts"(  ) ) WITH NUMERIC result =
0.00000000 ["acre-feet"] DO
      IF ( "Min"( "AvailableAccountStorage"( account, % "Abiquiu" ), "PreviousAccountStorage"(
account, % "Heron" ) ) >= 100.00000000 ["acre-feet"] )
      THEN
         "Min"( "AvailableAccountStorage"( account, % "Abiquiu" ), "PreviousAccountStorage"(
account, % "Heron" ) ) + result
      ELSE
         0.00000000 ["acre-feet"] + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "HeronAvailableElVadoAbiquiuStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN "ElVadoAbiquiuCommonStorageAccounts"(  ) ) WITH NUMERIC result =
0.00000000 ["acre-feet"] DO
      IF ( "Min"( "AvailableAccountStorage"( account, % "ElVado" ) + "AvailableAccountStorage"(
account, % "Abiquiu" ), "PreviousAccountStorage"( account, % "Heron" ) ) >= 100.00000000 ["acre-
feet"] )
      THEN
         "Min"( "AvailableAccountStorage"( account, % "ElVado" ) + "AvailableAccountStorage"(
account, % "Abiquiu" ), "PreviousAccountStorage"( account, % "Heron" ) ) + result
      ELSE
         0.00000000 ["acre-feet"] + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "HeronAvailableElVadoOnlyStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN "StorageInElVadoOnlyAccounts"(  ) ) WITH NUMERIC result =
0.00000000 ["acre-feet"] DO
```

```
      IF ( "Min"( "AvailableAccountStorage"( account, % "ElVado" ), "PreviousAccountStorage"(
account, % "Heron" ) ) >= 100.00000000 ["acre-feet"] )
      THEN
         "Min"( "AvailableAccountStorage"( account, % "ElVado" ), "PreviousAccountStorage"(
account, % "Heron" ) ) + result
      ELSE
         0.00000000 ["acre-feet"] + result
      ENDIF
   ENDFOR;

    END;

    FUNCTION       "HeronAvailableDownstreamAccountStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( NOT "CurrentYearIsWaiverYear"(  ) )
   THEN
      "HeronAvailableElVadoAbiquiuStorage"(  ) + "HeronAvailableElVadoOnlyStorage"(  ) +
"HeronAvailableAbiquiuOnlyStorage"(  ) + "Min"( "SumAccountStorageNonNegative"( % "Heron",
"HeronToMRGCDAccountsList"(  ) ), "AvailableAccountStorage"( "MRGCD", % "ElVado" ) ) +
"PreviousAccountStorage"( "CochitiRecPool", % "Heron" )
   ELSE
      "HeronAvailableElVadoAbiquiuStorage"(  ) + "HeronAvailableElVadoOnlyStorage"(  ) +
"HeronAvailableAbiquiuOnlyStorage"(  )
   ENDIF;

    END;

    FUNCTION       "HeronCochitiRecPoolRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "PreviousAccountStorage"( "CochitiRecPool", % "Heron" ), "Max"(
"MaxCochitiRecPoolVolume"(  ) - "PreviousAccountStorage"( "CochitiRecPool", % "Cochiti" ),
0.00000000 ["acre-feet"] ) ) / ( @"24:00:00 January 1, Next Year" - @"Current Timestep" );

    END;

    FUNCTION       "HeronCochitiRecPoolPaybackVolume" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Min"( "PreviousAccountStorage"( "CochitiRecPool", % "Heron" ), "Max"(
"MaxCochitiRecPoolVolume"(  ) - "PreviousAccountStorage"( "CochitiRecPool", % "Cochiti" ),
0.00000000 ["acre-feet"] ) );

    END;

    FUNCTION       "HeronOtowiAndJemezPaybackRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( "HeronRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "Heron", "HeronRootOtowiPaybackAccounts"(  ) ), % "Heron", %
"HeronSeepage" ) ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
      "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) + result
   ENDFOR + "VolumeToFlow"( "GetAccountDebt"(
"AlbuquerqueHeronToNMISCHeronNMISCJemezHeronSeepage.Supply" ), @"Current Timestep" );
```

```
    END;

    FUNCTION       "HeronOtowiAndJemezPaybackVolume" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( STRING supply IN "MakeSupplyList"( "HeronRootOtowiPaybackAccounts"(  ),
"MakeOtowiPaybackAccountList"( % "Heron", "HeronRootOtowiPaybackAccounts"(  ) ), % "Heron", %
"HeronSeepage" ) ) WITH NUMERIC result = 0.00000000 ["acre-feet"] DO
      "GetAccountDebt"( supply ) + result
   ENDFOR + "GetAccountDebt"( "AlbuquerqueHeronToNMISCHeronNMISCJemezHeronSeepage.Supply" );

    END;

    FUNCTION       "HeronRaftingSeasonSJRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST result = "ElVadoAlbuquerquePaybackRelease"(
"ElVadoRootAlbuquerquePaybackAccounts"(  ) ) DO
      IF ( "VolumeToFlow"( GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result, @"Current Timestep" ) + "ElVadoOtowiDebt"(  ) + "VolumeToFlow"(
"TotalAccountStorageAvailable"( "ElVadoAbiquiuCommonStorageAccountsLessAlb"(  ), % "ElVado", %
"Abiquiu" ), @"Current Timestep" ) >= "Max"( "ComputedElVadoRaftingSchedule"(  ) - "RGOutflow"( %
"ElVado" ), 0.00000000 ["cfs"] ) )
      THEN
         0.00000000 ["cfs"]
      ELSE
         "Max"( "ComputedElVadoRaftingSchedule"(  ) - "RGOutflow"( % "ElVado" ) - ( "Max"(
"VolumeToFlow"( GET NUMERIC @INDEX 0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000
FROM result, @"Current Timestep" ), 0.00000000 ["cfs"] ) + "ElVadoOtowiDebt"(  ) +
"VolumeToFlow"( "TotalAccountStorageAvailable"( "ElVadoAbiquiuCommonStorageAccountsLessAlb"(  ),
% "ElVado", % "Abiquiu" ), @"Current Timestep" ) ), 0.00000000 ["cfs"] )
      ENDIF
   ENDWITH;

    END;

    FUNCTION       "HeronWaiverRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" <= "WaiverDate"(  ) AND "PreviousYearIsWaiverYear"(  ) AND
"TotalWaiverBalance"(  ) > 0.00000000 ["acre-feet"] )
   THEN
      "TotalWaiverBalance"(  ) / ( "WaiverDate"(  ) + 1.00000000 ["day"] - @"Current Timestep" )
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "MinHeronSJOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "MinOutflow"( % "ElVado" ) - "ElVadoOtowiDebtRelease"(  ) -
"ElVadoSJStorageAccountsRelease"(  ) - "RGOutflow"( % "ElVado" ), 0.00000000 ["cfs"] );
```

```
    END;

    FUNCTION       "MRGCDPaybackRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST supplies = "MakeSupplyList"( "HeronMRGCDPaybackAccounts"(  ),
"MakeMRGCDPaybackAccountsList"( "HeronMRGCDPaybackAccounts"(  ), % "ElVado" ), % "Heron", %
"HeronSeepage" ) CONCAT "MakeSupplyList"( "HeronMRGCDPaybackAccounts"(  ),
"MakeMRGCDPaybackAccountsList"( "HeronMRGCDPaybackAccounts"(  ), % "Abiquiu" ), % "Heron", %
"HeronSeepage" ) DO
      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         IF ( supply == "ReclamationHeronToReclamationHeronMRGCDElVadoHeronSeepage.Supply" )
         THEN
            "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) -
"ReclamationElVadoToMRGCDElVado.Supply" [] + result
         ELSE
            "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) + result
         ENDIF
      ENDFOR
   ENDWITH;

    END;

  END;

  UTILITY_GROUP "Heron Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "HeronEndOfMonthRGRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ElevationToUnregulatedSpill"( % "Heron", "StorageToElevation"( % "Heron",
"SolveStorage"( % "Heron", $ "AzoteaWillow.Inflow2" [], 0.00000000 ["cfs"], "PreviousStorage"( %
"Heron" ), @"Current Timestep" ) ), @"Current Timestep" ) > 0.00000000 ["cfs"] )
   THEN
      "ElevationToUnregulatedSpill"( % "Heron", "StorageToElevation"( % "Heron", "SolveStorage"(
% "Heron", $ "AzoteaWillow.Inflow2" [], 0.00000000 ["cfs"], "PreviousStorage"( % "Heron" ),
@"Current Timestep" ) ), @"Current Timestep" )
   ELSE
      IF ( "IsEndOfMonth"(  ) )
      THEN
         "Max"( "VolumeToFlow"( $ "Heron^RioGrande.Storage" [@"Previous Timestep"], @"Current
Timestep" ), 0.00000000 ["cfs"] )
      ELSE
         IF ( @"Current Timestep" == @"Monday" )
         THEN
            "Max"( $ "Heron^RioGrande.Storage" [@"Previous Timestep"] / 4.00000000 ["day"],
0.00000000 ["cfs"] )
         ELSE
            IF ( @"Current Timestep" == @"Thursday" )
            THEN
               "Max"( $ "Heron^RioGrande.Storage" [@"Previous Timestep"] / 3.00000000 ["day"],
0.00000000 ["cfs"] )
            ELSE
               $ "Heron^RioGrande.Outflow" [@"Previous Timestep"]
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "HeronFirstOfMonthRGRelease" (  )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ElevationToUnregulatedSpill"( % "Heron", "StorageToElevation"( % "Heron",
"SolveStorage"( % "Heron", $ "AzoteaWillow.Inflow2" [], 0.00000000 ["cfs"], "PreviousStorage"( %
"Heron" ), @"Current Timestep" ) ), @"Current Timestep" ) > 0.00000000 ["cfs"] )
   THEN
      "ElevationToUnregulatedSpill"( % "Heron", "StorageToElevation"( % "Heron", "SolveStorage"(
% "Heron", $ "AzoteaWillow.Inflow2" [], 0.00000000 ["cfs"], "PreviousStorage"( % "Heron" ),
@"Current Timestep" ) ), @"Current Timestep" )
   ELSE
      IF ( "IsFirstDayOfMonth"(  ) AND @"Current Timestep" != @"Monday" )
      THEN
         0.00000000 ["cfs"]
      ELSE
         IF ( @"Current Timestep" == @"Monday" )
         THEN
            "Max"( "HeronRGBaseFlow"(  ) + $ "Heron^RioGrande.Storage" [@"Previous Timestep"] /
7.00000000 ["day"], 0.00000000 ["cfs"] )
         ELSE
            IF ( "Heron.Storage" [@"Previous Timestep"] > "MaxStorage"( % "Heron" ) AND NOT
@"Current Timestep" == @"Monday" )
            THEN
               "Min"( "VolumeToFlow"( "Heron.Storage" [@"Previous Timestep"] - "MaxStorage"( %
"Heron" ), @"Current Timestep" ), "GetMaxOutflowGivenHW"( % "Heron", "MaxElevation"( % "Heron" ),
@"Current Timestep" ) )
            ELSE
               $ "Heron^RioGrande.Outflow" [@"Previous Timestep"] - $ "Heron.Seepage" [@"Previous
Timestep"]
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "HeronHasFullIceCover" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Heron.Surface Ice Coverage" [] >= "HeronData.IceCoverageThreshold" [0.00000000,
0.00000000];

    END;

    FUNCTION       "HeronLimitDeltaStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "VolumeToFlow"( "Heron.Previous Storage" [] - "ElevationToStorage"( % "Heron", "Heron.Pool
Elevation" [@"Previous Timestep"] - $ "HeronData.maxDeltaPoolElev" [0.00000000, 0.00000000] ),
@"Current Timestep" );

    END;

    FUNCTION       "HeronMaxRGStorageToRel" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      "Max"( "VolumeToFlow"( $ "Heron^RioGrande.Storage" [@"Previous Timestep"], @"Current
Timestep" ), 0.00000000 ["cfs"] );

    END;

    FUNCTION       "HeronMustSpill" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "VolumeToFlow"( "Heron.Storage" [@"Previous Timestep"] - "MaxStorage"( % "Heron" ),
@"Current Timestep" ), 0.00000000 ["cfs"] ) + "Heron.Inflow" [] - "AzoteaWillow.Inflow1" [] >
"GetMaxOutflowGivenHW"( % "Heron", "MaxElevation"( % "Heron" ), @"Current Timestep" );

    END;

    FUNCTION       "HeronRGBaseFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "VolumeToFlow"( $ "HeronData.RGBaseFlow" ["GetMonthAsString"( @"Current Timestep" ),
"BaseFlow"] * 1.00000000 ["day"] / "GetDaysInMonth"( @"Current Timestep" ), @"Current Timestep"
);

    END;

    FUNCTION       "HeronRGOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      IF ( "IsHeronRGRelGreaterThanZero"(  ) AND "IsFirstHalfOfMonth"(  ) )
   THEN
      "HeronFirstOfMonthRGRelease"(  )
   ELSE
      IF ( "IsHeronRGRelGreaterThanZero"(  ) AND "IsSecondHalfOfMonth"(  ) )
      THEN
         "HeronEndOfMonthRGRelease"(  )
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "HeronSJOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "HeronHasFullIceCover"(  ) OR "HeronMustSpill"(  ) )
   THEN
      0.00000000 ["cfs"]
   ELSE
      "ComputeHeronSJOutflow"(  )
   ENDIF;

    END;

    FUNCTION       "ElVadoDeliveryRelease" (  )
    RETURN_TYPE    NUMERIC;
```

```
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST supplies = "MakeSupplyList"( "ElVadoRootWaiverStorageAccounts"(  ),
"MakeHeronElVadoExAccountsLIst"( "ElVadoRootWaiverStorageAccounts"(  ) ), % "Heron", % "Heron" )
DO
      FOR ( STRING supply IN supplies ) WITH NUMERIC result = 0.00000000 ["cfs"] DO
         "VolumeToFlow"( "GetAccountDebt"( supply ), @"Current Timestep" ) + result
      ENDFOR
   ENDWITH;

    END;

    FUNCTION       "IsHeronRGRelGreaterThanZero" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SJOutflow"( % "Heron" ) > 0.00000000 ["cfs"] OR $ "Heron^RioGrande.Storage" [@"Previous
Timestep"] >= $ "HeronData.HeronMinRGStorageForRel" [0.00000000, 0.00000000] OR $
"Heron^RioGrande.Storage" [@"Previous Timestep"] > $ "HeronData.MaxRGStorage" [0.00000000,
0.00000000] OR "ElevationToUnregulatedSpill"( % "Heron", "StorageToElevation"( % "Heron",
"SolveStorage"( % "Heron", $ "AzoteaWillow.Inflow2" [], 0.00000000 ["cfs"], "PreviousStorage"( %
"Heron" ), @"Current Timestep" ) ), @"Current Timestep" ) > 0.00000000 ["cfs"] AND NOT
"HeronHasFullIceCover"(  );

    END;

  END;

  UTILITY_GROUP "HypotheticalSimulation Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "PredictedFlowAtLocation" ( STRING beginLocation, STRING endLocation, SLOT
slot, NUMERIC No.OfDaysDS )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 0.00000000 FROM "GeneralHypotheticalSimulation"( beginLocation,
endLocation, slot, No.OfDaysDS );

    END;

    FUNCTION       "PredictedFlowAtLocationFromCochitiAndJemez" ( STRING beginLocation, STRING
endLocation, SLOT slot1, SLOT slot2, NUMERIC No.OfDaysDS )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 0.00000000 FROM "GeneralHypotheticalSimulationFromCochitiAndJemez"(
beginLocation, endLocation, slot1, slot2, No.OfDaysDS );

    END;

    FUNCTION       "GeneralHypotheticalSimulation" ( STRING beginLocation, STRING endLocation,
SLOT slot, NUMERIC No.OfDaysDS )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "HypotheticalSimulation"( beginLocation CONCAT "To" CONCAT endLocation, { { slot , slot []
, @"Current Timestep" } , { slot , slot [] , @"Current Timestep + 1 Timesteps" } , { slot , slot
[] , @"Current Timestep + 2 Timesteps" } , { slot , slot [] , @"Current Timestep + 3 Timesteps" }
}, { { "ObjectifyString"( endLocation ) & "Gage Inflow" , "OffsetDate"( @"Current Timestep",
No.OfDaysDS, "1 days" ) } , { "ObjectifyString"( endLocation ) & "Gage Inflow" , "OffsetDate"(
@"Current Timestep", No.OfDaysDS + 1.00000000, "1 days" ) } } );

    END;

    FUNCTION       "GeneralHypotheticalSimulationFromCochitiAndJemez" ( STRING beginLocation,
STRING endLocation, SLOT slot1, SLOT slot2, NUMERIC No.OfDaysDS )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "HypotheticalSimulation"( beginLocation CONCAT "To" CONCAT endLocation, { { slot1 , slot1
[] , @"Current Timestep" } , { slot1 , slot1 [] , @"Current Timestep + 1 Timesteps" } , { slot1 ,
slot1 [] , @"Current Timestep + 2 Timesteps" } , { slot1 , slot1 [] , @"Current Timestep + 3
Timesteps" } , { slot2 , slot2 [] , @"Current Timestep" } , { slot2 , slot2 [] , @"Current
Timestep + 1 Timesteps" } , { slot2 , slot2 [] , @"Current Timestep + 2 Timesteps" } , { slot2 ,
slot2 [] , @"Current Timestep + 3 Timesteps" } }, { { "ObjectifyString"( endLocation ) & "Gage
Inflow" , "OffsetDate"( @"Current Timestep", No.OfDaysDS, "1 days" ) } , { "ObjectifyString"(
endLocation ) & "Gage Inflow" , "OffsetDate"( @"Current Timestep", No.OfDaysDS + 1.00000000, "1
days" ) } } );

    END;

    FUNCTION       "EstimatedJemezReleaseList" ( NUMERIC numOfDays )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( GET DATETIME @INDEX 2.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result ) < @"Current Timestep" + numOfDays ) WITH LIST result = { { $
"BlwJemez.Gage Inflow" , 0.00000000 ["cfs"] , @"Current Timestep" } } DO
      APPEND { $ "BlwJemez.Gage Inflow" , 0.00000000 ["cfs"] , ( GET DATETIME @INDEX 2.00000000
FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM result ) + 1.00000000 ["day"] } ONTO
result
   ENDWHILE;

    END;

    FUNCTION       "EstimatedCochitiReleaseList" ( NUMERIC numOfDays )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( ( GET DATETIME @INDEX 2.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result ) < @"Current Timestep" + numOfDays ) WITH LIST result = { { $
"BlwCochitiDiversionsReach.Inflow" , $ "Cochiti.Outflow" [] , @"Current Timestep" } } DO
      APPEND { $ "BlwCochitiDiversionsReach.Inflow" , $ "Cochiti.Outflow" [] , ( GET DATETIME
@INDEX 2.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM result ) + 1.00000000
["day"] } ONTO result
   ENDWHILE;

    END;

    FUNCTION       "ReleaseToMeetMinFlowAtCentral" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BlwCochitiToCentral", $
"BlwCochitiDiversionsReach.Inflow", 0.00000000 ["cfs"], "MaxOutflow"( % "Cochiti" ),
"JemezInflowInputs"( lookAheadDays ), $ "Central.Gage Outflow", "DatePlusXTimesteps"( @"Current
Timestep", lookAheadDays ), "MinTargetFlow"( "Central", lookAheadDays ), 10.00000000 ["cfs"],
20.00000000 );

    END;

    FUNCTION       "ReleaseToMeetMinFlowAtIsleta" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BlwCochitiToSanAcacia", $
"BlwCochitiDiversionsReach.Inflow", 0.00000000 ["cfs"], "MaxOutflow"( % "Cochiti" ),
"JemezInflowInputs"( lookAheadDays ), $ "BlwIsletaDiversionsReach.Outflow", "DatePlusXTimesteps"(
@"Current Timestep", lookAheadDays ), "MinTargetFlow"( "Isleta", lookAheadDays ), 10.00000000
["cfs"], 30.00000000 );

    END;

    FUNCTION       "ReleaseToMeetMinFlowAtSanAcacia" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BlwCochitiToSanAcacia", $
"BlwCochitiDiversionsReach.Inflow", 0.00000000 ["cfs"], "MaxOutflow"( % "Cochiti" ),
"JemezInflowInputs"( lookAheadDays ), $ "SanAcaciaFloodway.Gage Outflow", "DatePlusXTimesteps"(
@"Current Timestep", lookAheadDays ), "MinTargetFlow"( "SanAcacia", lookAheadDays ), 10.00000000
["cfs"], 30.00000000 );

    END;

    FUNCTION       "ReleaseToMeetMinFlowAtSanMarcial" ( NUMERIC lookAheadDays )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      GET NUMERIC @INDEX 1.00000000 FROM "HypTargetSimWithStatus"( "BlwCochitiToSanMarcial", $
"BlwCochitiDiversionsReach.Inflow", 0.00000000 ["cfs"], "MaxOutflow"( % "Cochiti" ),
"JemezInflowInputs"( lookAheadDays ), $ "SanMarcialFloodway.Gage Outflow", "DatePlusXTimesteps"(
@"Current Timestep", lookAheadDays ), "MinTargetFlow"( "SanMarcial", lookAheadDays ), 10.00000000
["cfs"], 30.00000000 );

    END;

  END;

  UTILITY_GROUP "Indian Storage Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "IndianCallStorageAdjustment" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( @"Current Timestep" == @"24:00:00 Current Month 1, Current Year" )
```

```
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      "SumFlowsToVolume"( $ "Indian.Indian Call", @"24:00:00 Current Month 1, Current Year",
@"Previous Timestep" )
   ENDIF;

    END;

    FUNCTION       "ChangeInRioChamaRGStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "ElVado^RioGrande.Storage" [@"24:00:00 Previous Month Max DayOfMonth, Current Year"] + $
"Abiquiu^RioGrande.Storage" [@"24:00:00 Previous Month Max DayOfMonth, Current Year"] + $
"Abiquiu^RioGrandeConservation.Storage" [@"24:00:00 Previous Month Max DayOfMonth, Current Year"]
- ( ( $ "ElVado^RioGrande.Storage" ["MaxDate"( @"24:00:00 February Max DayOfMonth, Current Year",
@"Start Timestep - 1 Timesteps" )] + $ "Abiquiu^RioGrande.Storage" ["MaxDate"( @"24:00:00
February Max DayOfMonth, Current Year", @"Start Timestep - 1 Timesteps" )] ) + $
"Abiquiu^RioGrandeConservation.Storage" ["MaxDate"( @"24:00:00 February Max DayOfMonth, Current
Year", @"Start Timestep - 1 Timesteps" )] );

    END;

    FUNCTION       "ComputeIndianStorageReq" ( DATETIME dateIn )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( DATETIME date IN "MonthlyDATETIMEList"( dateIn, @"24:00:00 October Max DayOfMonth,
Current Year" ) ) WITH LIST result = {  } DO
      APPEND "ComputeMonthlyIndianStorageReq"( date ) ONTO result
   ENDFOR;

    END;

    FUNCTION       "ComputeElVadoRunoffLeft" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( @"Current Month" == @"March" )
   THEN
      "InputElVadoForecast"(  ) * "ElVadoForecastFactor"(  )
   ELSE
      IF ( @"Current Month" >= @"April" AND @"Current Month" <= @"July" )
      THEN
         ( "InputElVadoForecast"(  ) - "RealizedOtowiForecast"(  ) ) * "ElVadoForecastFactor"(  )
      ELSE
         0.00000000 ["acre-feet"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ComputeMonthlyIndianStorageReq" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      "Max"( "IndianDemand"( date ) - "ElVadoUsableFlow"( date ), 0.00000000 ["acre-feet"] ) /
"Indian.IndianStorageAdjFactor" [0.00000000, 0.00000000] * ( 1.00000000 + $
"Indian.FuturePossiblitiesIndianStorageRequirementAdjustment" [0.00000000, 0.00000000] );

    END;

    FUNCTION       "ComputeSupplyAtOtowi" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "GetMonthAsString"( @"Current Timestep" ) == "March" )
   THEN
      "Indian.LowRecordLateSummer" ["GetMonthAsString"( date ), "Otowi Flows"] +
"ElVadoRunoffLeft"(  ) * "GetMonthPercent"( date ) - "Indian.AnticipStorage" ["GetMonthAsString"(
date ), "Anticipated Storage"]
   ELSE
      IF ( "GetMonthAsString"( @"Current Timestep" ) == "April" )
      THEN
         "Indian.LowRecordLateSummer" ["GetMonthAsString"( date ), "Otowi Flows"] +
"ElVadoRunoffLeft"(  ) * "GetMonthPercent"( date ) - "Indian.AnticipStorage" ["GetMonthAsString"(
date ), "Anticipated Storage"]
      ELSE
         "Indian.LowRecordLateSummer" ["GetMonthAsString"( date ), "Otowi Flows"] +
"ElVadoRunoffLeft"(  ) * "GetMonthPercent"( date ) - "Indian.AnticipStorage" ["GetMonthAsString"(
date ), "Anticipated Storage"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ElVadoForecastFactor" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "ElVadoData.ForecastFactors" ["GetMonthAsString"( @"Current Timestep" ), "ElVado"];

    END;

    FUNCTION       "InputElVadoForecast" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "Indian.OtowiForecast" [0.00000000, 0.00000000];

    END;

    FUNCTION       "RealizedOtowiForecast" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "MaxDate"( @"24:00:00 March 1, Current Year", @"Start Timestep" ) == @"Start Timestep"
AND "MaxDate"( @"24:00:00 Previous Month Max DayOfMonth, Current Year", @"Start Timestep" ) ==
@"Start Timestep" )
   THEN
      "ZeroNaNTableSlot"( $ "Indian.InitialRealizedForecast", 0.00000000, "April" ) +
"ZeroNaNTableSlot"( $ "Indian.InitialRealizedForecast", 0.00000000, "May" )
   ELSE
      "SumFlowsToVolume"( $ "Otowi.Gage Inflow", "MaxDate"( @"24:00:00 March 1, Current Year",
@"Start Timestep" ), "MaxDate"( @"24:00:00 Previous Month Max DayOfMonth, Current Year", @"Start
```

```
Timestep" ) ) + "ChangeInRioChamaRGStorage"(  ) + ( "ZeroNaNTableSlot"( $
"Indian.InitialRealizedForecast", 0.00000000, "April" ) + "ZeroNaNTableSlot"( $
"Indian.InitialRealizedForecast", 0.00000000, "May" ) ) - "SumFlowsToVolume"( $
"AccountingCheck.AbiquiuSJOutflow", "MaxDate"( @"24:00:00 March 1, Current Year", @"Start
Timestep + 2 Timesteps" ), "MaxDate"( @"24:00:00 Previous Month Max DayOfMonth, Current Year - 2
Timesteps", @"Start Timestep + 2 Timesteps" ) )
   ENDIF;

    END;

    FUNCTION       "ElVadoIndianStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ElVadoSpringRunoff "(  ) OR "ElVadoSummerIrrigationSeason"(  ) )
   THEN
      "ElVadoData.IndianStorageReq" [@"24:00:00 Current Month Max DayOfMonth, Current Year"]
   ELSE
      0.00000000 ["acre-feet"]
   ENDIF;

    END;

    FUNCTION       "ElVadoRunoffLeft" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeElVadoRunoffLeft"(  );

    END;

    FUNCTION       "ElVadoUsableFlow" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-ft";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeSupplyAtOtowi"( date ) * "LookupUsuableFlowFactor"( "ComputeSupplyAtOtowi"( date )
);

    END;

    FUNCTION       "GetMonthPercent" ( DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElVadoData.MonthPercent" ["GetMonthAsString"( date ), "GetMonthAsString"( @"Current
Timestep" )];

    END;

    FUNCTION       "LookupUsuableFlowFactor" ( NUMERIC supply )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( supply <= "Indian.UsuableFlowFactor" [0.00000000, "Flow"] )
```

```
   THEN
      "Indian.UsuableFlowFactor" [0.00000000, "Factor"]
   ELSE
      IF ( supply <= "Indian.UsuableFlowFactor" [1.00000000, "Flow"] )
      THEN
         "Indian.UsuableFlowFactor" [1.00000000, "Factor"]
      ELSE
         "Indian.UsuableFlowFactor" [2.00000000, "Factor"]
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "JemezFunctions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ComputeJemezExchangeWhenRGStorage" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( "JemezAccumulatedDeliveryRequestInCFS"(  ) - "JemezAccumulatedSJCaptureInCFS"(  ) ) / $
"JemezData.ExchangeData" ["Jemez", "Divisor"];

    END;

    FUNCTION       "EstimateJemezInflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeReachVariableLagLossOutflow"( % "NrJemezToJemez", @"Current Timestep",
"ReachInflow"( % "NrJemezToJemez" ), "PreviousReachInflow"( % "NrJemezToJemez" ) ) + $
"JemezLocalInflow.Local Inflow" [];

    END;

    FUNCTION       "EstimateJemezOutflowForExchange" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( NOT "IrrigationSeason"(  ) )
   THEN
      0.00000000 ["cfs"]
   ELSE
      IF ( ( "EstimateJemezInflow"(  ) > $ "JemezData.ReservoirData" ["Jemez",
"MinNonWinterEXRelease"] ) OR ( "PreviousRGStorage"( % "Jemez" ) > "FlowToVolume"( $
"JemezData.ReservoirData" ["Jemez", "MinNonWinterEXRelease"], @"Current Timestep" ) ) )
      THEN
         $ "JemezData.ReservoirData" ["Jemez", "MinNonWinterEXRelease"]
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "JemezExchange" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "IsNOTFirstTimestep"(  ) AND "JemezAccumulatedSJCaptureInACRE-FT"(  ) <
"JemezAccumulatedDeliveryRequestInACRE-FT"(  ) AND ( "PreviousRGStorage"( % "Jemez" ) >
"JemezAccumulatedDeliveryRequestInACRE-FT"(  ) - "JemezAccumulatedSJCaptureInACRE-FT"(  ) ) AND (
"JemezAccumulatedDeliveryRequestInCFS"(  ) - "JemezAccumulatedSJCaptureInCFS"(  ) > $
"JemezData.ExchangeData" ["Jemez", "minExchangeRGFlow"] ) AND "JemezIsStoring"(  ) )
   THEN
      "Min"( "EstimateJemezInflow"(  ) - "EstimateJemezOutflowForExchange"(  ), "Max"(
"ComputeJemezExchangeWhenRGStorage"(  ), $ "JemezData.ExchangeData" ["Jemez",
"minExchangeRGFlow"] ) )
   ELSE
      IF ( "IsNOTFirstTimestep"(  ) AND "JemezIsStoring"(  ) AND
"JemezAccumulatedSJCaptureInACRE-FT"(  ) < "JemezAccumulatedDeliveryRequestInACRE-FT"(  ) )
      THEN
         "Min"( "EstimateJemezInflow"(  ) - "EstimateJemezOutflowForExchange"(  ), "Max"(
"JemezAccumulatedDeliveryRequestInCFS"(  ) - "JemezAccumulatedSJCaptureInCFS"(  ), 0.00000000
["cfs"] ) )
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "JemezExchangeSumStartDate" (  )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( @"Start Year" == @"Current Year" )
   THEN
      @"Start Timestep"
   ELSE
      @"24:00:00 January 1, Current Year"
   ENDIF;

    END;

    FUNCTION       "JemezIsStoring" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "EstimateJemezInflow"(  ) - "EstimateJemezOutflowForExchange"(  ) > 0.00000000 ["cfs"];

    END;

    FUNCTION       "JemezAccumulatedDeliveryRequestInCFS" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "VolumeToFlow"( "SumSlot"( $ "DeliveryRequests.NMISCJemez", "JemezExchangeSumStartDate"(
), @"Current Timestep" ), @"Current Timestep" );

    END;

    FUNCTION       "JemezAccumulatedDeliveryRequestInACRE-FT" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      "SumSlot"( $ "DeliveryRequests.NMISCJemez", "JemezExchangeSumStartDate"(  ), @"Current
Timestep" );

    END;

    FUNCTION       "JemezAccumulatedSJCaptureInCFS" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SumSlot"( $ "RioGrandeJemezToRioGrandeJemezJemezSedimentPoolJemez.Supply",
"JemezExchangeSumStartDate"(  ), @"Previous Timestep" );

    END;

    FUNCTION       "JemezAccumulatedSJCaptureInACRE-FT" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "acre-feet";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" > @"24:00:00 January 1, Current Year" )
   THEN
      "SumFlowsToVolume"( $ "RioGrandeJemezToRioGrandeJemezJemezSedimentPoolJemez.Supply",
"JemezExchangeSumStartDate"(  ), @"Previous Timestep" )
   ELSE
      0.00000000 ["acre-feet"]
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Jemez Flood Control Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "JemezFloodSpace" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function determines the amount of flood space in Jemez Canyon Reservoir.
";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ElevationToStorage"( % "Jemez", $ "JemezData.PoolLevels" ["Elevation",
"TopOfFloodControlPool"] ) - "ElevationToStorage"( % "Jemez", $ "JemezData.PoolLevels"
["Elevation", "SedimentPool"] );

    END;

    FUNCTION       "JemezComputedMaxOutflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "JemezReleaseForChannelCapacity"(  ), "DetermineSteppedRelease"( % "Jemez" ) );

    END;

    FUNCTION       "JemezFloodRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "Min"( "AvailableStorage"( % "Jemez" ) / ( "AvailableStorage"( % "Cochiti" ) +
"AvailableStorage"( % "Jemez" ) ) * "Min"( $ "CochitiData.MaxReleaseForSanMarcialChannelCap" [],
$ "CochitiData.MaxReleaseForCentralChannelCap" [] ), "VolumeToFlow"( "PreviousAccountStorage"(
"RioGrande", % "Jemez" ), @"Current Timestep" ) + $ "Jemez.Inflow" [] );

    END;

  END;

  UTILITY_GROUP "Jemez Channel Capacity Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "JemezFlowToMatchCentralChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( $ "Central.Gage Inflow" [] > "ChannelCapacity"( % "Cochiti", "Central" ) )
   THEN
      IF ( "IsReleaseToMatchChannelCapacity<0"( % "Cochiti", "Central", $ "Central.Gage Inflow"
[] ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez", "Central", $ "Central.Gage Inflow" [] )
)
      THEN
         "ChannelCapacity"( % "Jemez", "Minimum" )
      ELSE
         IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "Central", $ "Central.Gage
Inflow" [] ) AND "IsReleaseToMatchChannelCapacity>=0"( % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) )
         THEN
            "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $
"Central.Gage Inflow" [] ) - $ "Cochiti.Outflow" []
         ELSE
            IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "Central", $ "Central.Gage
Inflow" [] ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez", "Central", $ "Central.Gage
Inflow" [] ) )
            THEN
               $ "Jemez.Outflow" []
            ELSE
               "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "Central", $
"Central.Gage Inflow" [] ) - $ "Cochiti.Outflow" []
            ENDIF
         ENDIF
      ENDIF
   ELSE
      "FlowToMatchChannelCapacity"( % "Jemez", "Central", $ "Central.Gage Inflow" [] )
   ENDIF;

    END;

    FUNCTION       "JemezReleaseForChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "JemezReleaseForCentralChannelCapacity"(  ), $
"JemezData.MaxReleaseForSanMarcialChannelCap" [] );

    END;

    FUNCTION       "JemezReleaseForCentralChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
```

```
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( "CochitiToCentralLocalInflows"(  ) > "ChannelCapacity"( % "Cochiti", "Central" ) )
OR ( "JemezFlowToMatchCentralChannelCapacity"(  ) < "ChannelCapacity"( % "Cochiti", "Minimum" ) )
)
   THEN
      "ChannelCapacity"( % "Jemez", "Minimum" )
   ELSE
      "JemezFlowToMatchCentralChannelCapacity"(  )
   ENDIF;

    END;

    FUNCTION       "JemezFlowToMatchSanMarcialFloodwayChannelCapacity" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach",
"SanMarcialFloodway", $ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $
"CochitiData.ApproxNoOfDaysDS" ["SanMarcialFloodway", "Days"] ) > "ChannelCapacity"( % "Cochiti",
"SanMarcialFloodway" ) )
   THEN
      IF ( "IsReleaseToMatchChannelCapacity<0"( % "Cochiti", "SanMarcialFloodway",
"PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach", "SanMarcialFloodway",
$ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $ "CochitiData.ApproxNoOfDaysDS"
["SanMarcialFloodway", "Days"] ) ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez",
"SanMarcialFloodway", "PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach",
"SanMarcialFloodway", $ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $
"CochitiData.ApproxNoOfDaysDS" ["SanMarcialFloodway", "Days"] ) ) )
      THEN
         "ChannelCapacity"( % "Jemez", "Minimum" )
      ELSE
         IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "SanMarcialFloodway",
"PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach", "SanMarcialFloodway",
$ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $ "CochitiData.ApproxNoOfDaysDS"
["SanMarcialFloodway", "Days"] ) ) AND "IsReleaseToMatchChannelCapacity>=0"( % "Jemez",
"SanMarcialFloodway", "PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach",
"SanMarcialFloodway", $ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $
"CochitiData.ApproxNoOfDaysDS" ["SanMarcialFloodway", "Days"] ) ) )
         THEN
            "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez", "SanMarcialFloodway",
"PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach", "SanMarcialFloodway",
$ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $ "CochitiData.ApproxNoOfDaysDS"
["SanMarcialFloodway", "Days"] ) ) - $ "Cochiti.Outflow" []
         ELSE
            IF ( "IsReleaseToMatchChannelCapacity>=0"( % "Cochiti", "SanMarcialFloodway",
"PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach", "SanMarcialFloodway",
$ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $ "CochitiData.ApproxNoOfDaysDS"
["SanMarcialFloodway", "Days"] ) ) AND "IsReleaseToMatchChannelCapacity<0"( % "Jemez",
"SanMarcialFloodway", "PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach",
"SanMarcialFloodway", $ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $
"CochitiData.ApproxNoOfDaysDS" ["SanMarcialFloodway", "Days"] ) ) )
            THEN
               $ "Jemez.Outflow" []
            ELSE
               "CombinedFlowToMatchChannelCapacity"( % "Cochiti", % "Jemez",
"SanMarcialFloodway", "PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach",
"SanMarcialFloodway", $ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $
"CochitiData.ApproxNoOfDaysDS" ["SanMarcialFloodway", "Days"] ) ) - $ "Cochiti.Outflow" []
            ENDIF
         ENDIF
      ENDIF
   ELSE
      "FlowToMatchChannelCapacity"( % "Jemez", "SanMarcialFloodway",
"PredictedFlowAtLocationFromCochitiAndJemez"( "BlwCochitiDiversionsReach", "SanMarcialFloodway",
$ "BlwCochitiDiversionsReach.Outflow", $ "BlwJemez.Gage Inflow", $ "CochitiData.ApproxNoOfDaysDS"
["SanMarcialFloodway", "Days"] ) )
   ENDIF;

    END;
```

```
  END;

  UTILITY_GROUP "Loss Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AdjustGWLoss" ( OBJECT reach, NUMERIC offset, STRING season )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reach ) CONCAT "Data.GWLoss" ["OffsetDate"( @"Current Timestep", offset, "1
days" )] * ( STRINGIFY reach ) CONCAT "Data.SeepLossandInterceptFactors" [season, "Factor"];

    END;

    FUNCTION       "GetPanEvapForReach" ( OBJECT reach )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "in/day";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( ( STRINGIFY reach ) == "SanAcaciaToSanMarcialLosses" ) OR ( ( STRINGIFY reach ) ==
"SanMarcialToElephantButteLosses" ) )
   THEN
      $ "ElephantButte.Pan Evaporation" []
   ELSE
      IF ( "SummerEvapSeason"(  ) )
      THEN
         IF ( ( ( STRINGIFY reach ) == "BlwCochitiToSanFelipeLosses" ) OR ( ( STRINGIFY reach )
== "SanFelipeToCentralLosses" ) )
         THEN
            $ "Cochiti.Pan Evaporation" []
         ELSE
            IF ( ( ( STRINGIFY reach ) == "CentralToBernardoLosses" ) OR ( ( STRINGIFY reach ) ==
"BernardoToSanAcaciaLosses" ) )
            THEN
               $ "Jemez.Pan Evaporation" []
            ELSE
               0.00000000 ["in/day"]
            ENDIF
         ENDIF
      ELSE
         IF ( ( ( STRINGIFY reach ) == "BlwCochitiToSanFelipeLosses" ) OR ( ( STRINGIFY reach )
== "SanFelipeToCentralLosses" ) )
         THEN
            $ "Cochiti.Evaporation" [] / $ "Cochiti.Surface Area" [] * 12.00000000 ["in/day-ft"]
         ELSE
            IF ( ( ( STRINGIFY reach ) == "CentralToBernardoLosses" ) OR ( ( STRINGIFY reach ) ==
"BernardoToSanAcaciaLosses" ) )
            THEN
               $ "Jemez.Evaporation" [] / $ "Jemez.Surface Area" [] * 12.00000000 ["in/day-ft"]
            ELSE
               0.00000000 ["in/day"]
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    EXTERNAL_FUNCTION "ComputeLoss" ( STRING reach, NUMERIC flow, NUMERIC pan )
    RETURN_TYPE       NUMERIC;
    SCALE_UNITS       "cfs";
    DESCRIPTION       "";
    ACTIVE            TRUE;
    PRE_EXEC_DIAG     FALSE;
    POST_EXEC_DIAG    TRUE;
    BEGIN
# convert metric flow values to english units (cfs)
set flow [expr $flow * 35.3146667214886]
```

```
# convert metric pan evap values to english units (in/day)
# set pan  [expr $pan * 3401574.80315]
# convert metric pan evap values to english units (ft/day)
set pan [expr $pan * 283478.4]

if {$reach == "BlwCochitiToSanFelipeLosses"} {
   if {$flow >= 5650} {
      set loss [expr $pan * (111 * pow ($flow, 0.2))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (111 * pow ($flow, 0.2)) + 0.25 * $pan * (625 - (111 * pow ($flow,
0.2)))]
   }
}

if {$reach == "SanFelipeToCentralLosses"} {
   if {$flow >= 4820} {
      set loss [expr $pan * (84 * pow ($flow, 0.41))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (84 * pow ($flow, 0.41)) + 0.25 * $pan * (2718- (84 * pow ($flow,
0.41)))]
   }
}

if {$reach == "CentralToBernardoLosses"} {
   if {$flow >= 4820} {
      set loss [expr $pan * (124 * pow ($flow, 0.44))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (124 * pow ($flow, 0.44)) + 0.25 * $pan * (5175 - (124 * pow ($flow,
0.44)))]
   }
}

if {$reach == "BernardoToSanAcaciaLosses"} {
   if {$flow >= 4000} {
      set loss [expr $pan * (13 * pow ($flow, 0.53))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (13 * pow ($flow, 0.53)) + 0.25 * $pan * (1054 - (13 * pow ($flow,
0.53)))]
   }
}

if {$reach == "SanAcaciaToSanMarcialLosses"} {
   if {$flow >= 9100} {
      set loss [expr $pan * (158 * pow ($flow, 0.32))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (158 * pow ($flow, 0.32)) + 0.25 * $pan * (2913 - (158 * pow ($flow,
0.32)))]
   }
}
if {$reach == "SanMarcialToElephantButteLosses"} {
   if {$flow >= 2400} {
      set loss [expr $pan * (60 * pow ($flow, 0.13))]
   } elseif {$flow <= 0} {
      set loss 0
   } else {
      set loss [expr $pan * (60 * pow ($flow, 0.13)) + 0.25 * $pan * (166 - (60 * pow ($flow,
0.13)))]
   }
}

set losscnv [expr $loss/1.983471]

return "$losscnv \[\"cfs\"\]"
    END;

  END;

  UTILITY_GROUP "Mainstem Reach Functions";
  DESCRIPTION   "";
```

```
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "EmbudoLagAndLoss" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "This function estimates the routed flow, including losses, for the Embudo to
Confluence reach.  The lagged flow is based on the 5000 to 15000 cfs flow range.";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "EstimatedLagAndLoss"( % "EmbudoToConfluence" );

    END;

    FUNCTION       "OtowiLookAhead" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "ElVadoLocalInflow.Local Inflow" ["LookAhead"(  )] + $ "Embudo.Gage Inflow" ["LookAhead"(
)];

    END;

    FUNCTION       "Otowi" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      $ "ElVadoLocalInflow.Local Inflow" [] + $ "Embudo.Gage Inflow" [];

    END;

  END;

  UTILITY_GROUP "Maintenance Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "MaintenanceFlow" ( OBJECT object )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      ( STRINGIFY object CONCAT "Data.MaintenanceFlow" ) [STRINGIFY object, "MaintenanceFlow"];

    END;

    FUNCTION       "IsMaintenanceSwitchOn" ( OBJECT object )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "MaintenanceSwitch"( object ) == 1.00000000;

    END;

    FUNCTION       "EndOfMonth" (  )
    RETURN_TYPE    DATETIME;
```

```
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      @"24:00:00 Current Month Max DayOfMonth, Current Year";

    END;

  END;

  UTILITY_GROUP "Priority Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "PrioritizedAccountList" ( OBJECT object, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST scratch = "AccountPriorityList"( object, accounts ) DO
      WHILE ( ( LENGTH result ) < LENGTH scratch ) WITH LIST result = { { "NumericListElement"(
"MinList"( scratch ), scratch ) , SUB 2500.00000000 @INDEX "NumericListElement"( "MinList"(
scratch ), scratch ) OF scratch } } DO
         APPEND { "NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ) , SUB 2500.00000000 @INDEX
"NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result
) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result ) OF GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result } ONTO result
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "PrioritizedAccounts" ( OBJECT object, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( LIST priorityScratch IN "PrioritizedAccountList"( object, accounts ) ) WITH LIST
result = {  } DO
      APPEND GET STRING @INDEX GET NUMERIC @INDEX 0.00000000 FROM priorityScratch FROM accounts
ONTO result
   ENDFOR;

    END;

    FUNCTION       "PrioritizedReleaseTypeList" ( OBJECT object, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST scratch = "ReleasePriorityList"( object ) DO
      WHILE ( ( LENGTH result ) < LENGTH scratch ) WITH LIST result = { { "NumericListElement"(
"MinList"( scratch ), scratch ) , SUB 2500.00000000 @INDEX "NumericListElement"( "MinList"(
scratch ), scratch ) OF scratch } } DO
         APPEND { "NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ) , SUB 2500.00000000 @INDEX
"NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result
) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
```

```
1.00000000 FROM result ) OF GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result } ONTO result
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "PrioritizedReleaseTypes" ( OBJECT object, LIST accounts )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( LIST priorityScratch IN "PrioritizedReleaseTypeList"( object, accounts ) ) WITH LIST
result = {  } DO
      APPEND GET STRING @INDEX GET NUMERIC @INDEX 0.00000000 FROM priorityScratch FROM accounts
ONTO result
   ENDFOR;

    END;

    FUNCTION       "PrioritizedReservoirList" ( OBJECT object, LIST reservoirs )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      WITH LIST scratch = "ReservoirPriorityList"( object, reservoirs ) DO
      WHILE ( ( LENGTH result ) < LENGTH scratch ) WITH LIST result = { { "NumericListElement"(
"MinList"( scratch ), scratch ) , SUB 2500.00000000 @INDEX "NumericListElement"( "MinList"(
scratch ), scratch ) OF scratch } } DO
         APPEND { "NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST
@INDEX ( LENGTH result ) - 1.00000000 FROM result ) , SUB 2500.00000000 @INDEX
"NumericListElement"( "MinList"( GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result
) - 1.00000000 FROM result ), GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result ) OF GET LIST @INDEX 1.00000000 FROM GET LIST @INDEX ( LENGTH result ) -
1.00000000 FROM result } ONTO result
      ENDWHILE
   ENDWITH;

    END;

    FUNCTION       "PrioritizedReservoirs" ( OBJECT object, LIST reservoirs )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      FOR ( LIST priorityScratch IN "PrioritizedReservoirList"( object, reservoirs ) ) WITH LIST
result = {  } DO
      APPEND GET STRING @INDEX GET NUMERIC @INDEX 0.00000000 FROM priorityScratch FROM reservoirs
ONTO result
   ENDFOR;

    END;

    FUNCTION       "ReservoirInPriority" (  )
    RETURN_TYPE    OBJECT;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( "ElVadoIsPriority"(  ) )
   THEN
      % "ElVado"
   ELSE
```

```
      % "Abiquiu"
   ENDIF;

    END;

  END;

  UTILITY_GROUP "Rafting Release Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ElVadoRaftingRelease" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "ElVadoRaftingSeason"(  ) )
   THEN
      WITH LIST result = "ElVadoAlbuquerquePaybackRelease"(
"ElVadoRootAlbuquerquePaybackAccounts"(  ) ) DO
         "Min"( "PeliminaryRaftingRelease"(  ), "Max"( "VolumeToFlow"( GET NUMERIC @INDEX
0.00000000 FROM GET LIST @INDEX ( LENGTH result ) - 1.00000000 FROM result, @"Current Timestep" )
+ "ElVadoOtowiDebt"(  ) + "VolumeToFlow"( "TotalAccountStorageAvailable"(
"ElVadoAbiquiuCommonStorageAccountsLessAlb"(  ), % "ElVado", % "Abiquiu" ), @"Current Timestep" )
+ "ElVadoMRGCDDebtRelease"(  ) + "TotalElVadoFlowThruAccounts"(  ), 0.00000000 ["cfs"] ) )
      ENDWITH
   ELSE
      0.00000000 ["cfs"]
   ENDIF;

    END;

    FUNCTION       "ElVadoRaftingSeason" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  TRUE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current DayOfWeek" == @"Friday" )
   THEN
      IF ( "RaftingSeasonStartDate"(  ) <= @"Current Timestep" AND @"Current Timestep" <=
"RaftingSeasonEndDate"(  ) )
      THEN
         TRUE
      ELSE
         FALSE
      ENDIF
   ELSE
      IF ( @"Previous DayOfWeek" == @"Friday" )
      THEN
         IF ( "RaftingSeasonStartDate"(  ) + 1.00000000 ["day"] <= @"Current Timestep" AND
@"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
         THEN
            TRUE
         ELSE
            FALSE
         ENDIF
      ELSE
         IF ( @"Current DayOfWeek" == @"Sunday" )
         THEN
            IF ( "RaftingSeasonStartDate"(  ) + 2.00000000 ["day"] <= @"Current Timestep" AND
@"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
            THEN
               TRUE
            ELSE
               FALSE
            ENDIF
         ELSE
            IF ( @"Current DayOfWeek" == @"Monday" )
            THEN
```

```
               IF ( "RaftingSeasonStartDate"(  ) + 3.00000000 ["day"] <= @"Current Timestep" AND
@"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
               THEN
                  TRUE
               ELSE
                  FALSE
               ENDIF
            ELSE
               IF ( @"Current DayOfWeek" == @"Tuesday" )
               THEN
                  IF ( "RaftingSeasonStartDate"(  ) + 4.00000000 ["day"] <= @"Current Timestep"
AND @"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
                  THEN
                     TRUE
                  ELSE
                     FALSE
                  ENDIF
               ELSE
                  IF ( @"Current DayOfWeek" == @"Wednesday" )
                  THEN
                     IF ( "RaftingSeasonStartDate"(  ) + 5.00000000 ["day"] <= @"Current
Timestep" AND @"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
                     THEN
                        TRUE
                     ELSE
                        FALSE
                     ENDIF
                  ELSE
                     IF ( @"Current DayOfWeek" == @"Thursday" )
                     THEN
                        IF ( "RaftingSeasonStartDate"(  ) + 6.00000000 ["day"] <= @"Current
Timestep" AND @"Current Timestep" <= "RaftingSeasonEndDate"(  ) )
                        THEN
                           TRUE
                        ELSE
                           FALSE
                        ENDIF
                     ELSE
                        FALSE
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "San Juan Diversion Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AnnualDiversionCalc" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "Capacity"( % "AzoteaTunnelInlet" );
    BEGIN

      "VolumeToFlow"( $ "SanJuanChamaRules.MaxAnnualDiv" [0.00000000, 0.00000000] - "EOYTotal"(
), @"Current Timestep" );

    END;

    FUNCTION       "AnnualDiversionExceeded" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
```

```
    BEGIN

      "AnnualDiversionCalc"(  ) < "TotalAvailableForDiversion"(  );

    END;

    FUNCTION       "AvailableForDiversion" ( OBJECT diversion )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY diversion ) == "BlancoDiversion" )
   THEN
      $ "RioBlanco.Available For Diversion" []
   ELSE
      IF ( ( STRINGIFY diversion ) == "LittleOsoDiversion" )
      THEN
         $ "LittleNavajoRiver.Available For Diversion" []
      ELSE
         $ "NavajoRiver.Available For Diversion" []
      ENDIF
   ENDIF;

    END;

    FUNCTION       "Capacity" ( OBJECT diversion )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      diversion & "Max Diversion" [0.00000000, 0.00000000];

    END;

    FUNCTION       "DecadeDiversionCalc" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "Capacity"( % "AzoteaTunnelInlet" );
    BEGIN

      "VolumeToFlow"( $ "SanJuanChamaRules.MaxDecadeDiv" [0.00000000, 0.00000000] - "EODTotal"(
), @"Current Timestep" );

    END;

    FUNCTION       "DecadeDiversionExceeded" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "DecadeDiversionCalc"(  ) < "TotalAvailableForDiversion"(  );

    END;

    FUNCTION       "DiversionIsLimited" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      "AnnualDiversionExceeded"(  ) OR "DecadeDiversionExceeded"(  ) OR "HeronSpaceIsLimited"(
);

    END;

    FUNCTION       "EODTotal" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[acre-feet]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "EOYTotal"(  ) + "SumSlotOverTime"( "SanJuanChamaDiversions.AnnualDiversion", @"24:00:00
December 31, Current Year - 9 Year", @"24:00:00 December 31, Previous Year" );

    END;

    FUNCTION       "EOYTotal" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[acre-feet]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" == @"24:00:00 January 1, Current Year" )
   THEN
      0.00000000 ["acre-feet"]
   ELSE
      "FlowToVolume"( "SumSlotOverTime"( "SanJuanChamaDiversions.Total", @"24:00:00 January 1,
Current Year", @"Previous Timestep" ), @"Current Timestep" )
   ENDIF;

    END;

    FUNCTION       "HeronMaximumInflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    MAX_CONSTRAINT "Capacity"( % "AzoteaTunnelInlet" );
    BEGIN

      "SolveInflow"( % "Heron", "RGOutflow"( % "Heron" ) + "SJOutflow"( % "Heron" ),
"ElevationToStorage"( % "Heron", "MaxElevation"( % "Heron" ) ), "PreviousStorage"( % "Heron" ),
@"Current Timestep" ) / ( 1.00000000 - $ "AzoteaToHeronLoss.GainLoss Coefficient" [0.00000000,
0.00000000] ) - $ "AzoteaWillow.Inflow2" [];

    END;

    FUNCTION       "HeronSpaceIsLimited" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      "HeronMaximumInflow"(  ) < "TotalAvailableForDiversion"(  );

    END;

    FUNCTION       "LargeDivCalc" ( OBJECT diversion, NUMERIC targetDiversion )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN
```

```
      IF ( ( STRINGIFY diversion ) == "BlancoDiversion" )
   THEN
      "Min"( targetDiversion, "MaxDiversion"( % "BlancoDiversion" ) )
   ELSE
      IF ( ( STRINGIFY diversion ) == "LittleOsoDiversion" )
      THEN
         "Min"( targetDiversion - "LargeDivCalc"( % "BlancoDiversion", targetDiversion ),
"MaxDiversion"( % "LittleOsoDiversion" ) )
      ELSE
         IF ( ( STRINGIFY diversion ) == "OsoDiversion" )
         THEN
            "Min"( targetDiversion - "LargeDivCalc"( % "BlancoDiversion", targetDiversion ) -
"LargeDivCalc"( % "LittleOsoDiversion", targetDiversion ), "MaxDiversion"( % "OsoDiversion" ) )
         ELSE
            "Dummy.NaN" []
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "MaxDiversion" ( OBJECT diversion )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( STRINGIFY diversion ) == "BlancoDiversion" )
   THEN
      "Min"( "Capacity"( diversion ), "AvailableForDiversion"( diversion ) )
   ELSE
      IF ( ( STRINGIFY diversion ) == "LittleOsoDiversion" )
      THEN
         "Min"( "Min"( "Capacity"( diversion ), "AvailableForDiversion"( diversion ) ), "Max"(
"Capacity"( % "OsoTunnelInlet" ) - "MaxDiversion"( % "BlancoDiversion" ), 0.00000000 ["cfs"] ) )
      ELSE
         IF ( ( STRINGIFY diversion ) == "OsoDiversion" )
         THEN
            "Min"( "Min"( "Capacity"( diversion ), "AvailableForDiversion"( diversion ) ), "Max"(
"Capacity"( % "AzoteaTunnelInlet" ) - "MaxDiversion"( % "BlancoDiversion" ) - "MaxDiversion"( %
"LittleOsoDiversion" ), 0.00000000 ["cfs"] ) )
         ELSE
            "Dummy.NaN" []
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "SmallDivCalc" ( OBJECT diversion, NUMERIC targetDiversion )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      IF ( ( STRINGIFY diversion ) == "OsoDiversion" )
   THEN
      "Min"( "MaxDiversion"( % "OsoDiversion" ), targetDiversion )
   ELSE
      IF ( ( STRINGIFY diversion ) == "LittleOsoDiversion" )
      THEN
         "Min"( targetDiversion - "SmallDivCalc"( % "OsoDiversion", targetDiversion ),
"MaxDiversion"( % "LittleOsoDiversion" ) )
      ELSE
         IF ( ( STRINGIFY diversion ) == "BlancoDiversion" )
         THEN
            "Min"( targetDiversion - "SmallDivCalc"( % "OsoDiversion", targetDiversion ) -
"SmallDivCalc"( % "LittleOsoDiversion", targetDiversion ), "MaxDiversion"( % "BlancoDiversion" )
)
         ELSE
            "Dummy.NaN" []
         ENDIF
      ENDIF
```

```
   ENDIF;

    END;

    FUNCTION       "TargetTotalDiversion" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "[cfs]";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Min"( "HeronMaximumInflow"(  ), "Min"( "AnnualDiversionCalc"(  ), "DecadeDiversionCalc"(
) ) );

    END;

    FUNCTION       "TotalAvailableForDiversion" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "MaxDiversion"( % "BlancoDiversion" ) + "MaxDiversion"( % "LittleOsoDiversion" ) +
"MaxDiversion"( % "OsoDiversion" );

    END;

  END;

  UTILITY_GROUP "Scale Diversion Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "GetDiversionRequested" ( STRING diversion, OBJECT dataobject, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( NOT IsNaN ( STRINGIFY dataobject ) CONCAT ".ScaleDiversions" [diversion,
"FromJulianDay"] ) AND NOT IsNaN ( STRINGIFY dataobject ) CONCAT ".ScaleDiversions" [diversion,
"ToJulianDay"] AND date >= "GetStartScaleDiversionDate"( diversion, dataobject, date ) AND date
<= "GetEndScaleDiversionDate"( diversion, dataobject, date ) )
   THEN
      ( STRINGIFY dataobject ) CONCAT "." CONCAT diversion [date] * ( 1.00000000 + ( STRINGIFY
dataobject ) CONCAT ".ScaleDiversions" [diversion, "ScaleAdjustment"] )
   ELSE
      ( STRINGIFY dataobject ) CONCAT "." CONCAT diversion [date]
   ENDIF;

    END;

    FUNCTION       "GetDepletionRequested" ( STRING depletion, OBJECT dataobject, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetDiversionRequested"( depletion, dataobject, date ) * ( 1.00000000 - "GetMonthlyData"(
dataobject, "FractionalReturnFlows", depletion, date ) );

    END;

    FUNCTION       "GetStartScaleDiversionDate" ( STRING diversion, OBJECT dataobject, DATETIME
date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
```

```
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "OffsetDate"( "OffsetDate"( @"24:00:00 December 31, Start Year", ( ( "GetYear"( date ) -
"GetYear"( @"Start Timestep" ) ) - 1.00000000 ) * 12.00000000, "1 months" ), ( STRINGIFY
dataobject ) CONCAT ".ScaleDiversions" [diversion, "FromJulianDay"], "1 days" );

    END;

    FUNCTION       "GetEndScaleDiversionDate" ( STRING diversion, OBJECT dataobject, DATETIME
date )
    RETURN_TYPE    DATETIME;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "OffsetDate"( "OffsetDate"( @"24:00:00 December 31, Start Year", ( ( "GetYear"( date ) -
"GetYear"( @"Start Timestep" ) ) - 1.00000000 ) * 12.00000000, "1 months" ), ( STRINGIFY
dataobject ) CONCAT ".ScaleDiversions" [diversion, "ToJulianDay"], "1 days" );

    END;

    FUNCTION       "GetMonthlyData" ( OBJECT object, STRING slot, STRING column, DATETIME date )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      object & slot ["GetMonth"( date ) - 1.00000000, column];

    END;

  END;

  UTILITY_GROUP "Silvery Minnow Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "JemezInflowInputs" ( NUMERIC lookAheadDays )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WITH LIST dates = @"Current Timestep" TO "DatePlusXTimesteps"( @"Current Timestep",
lookAheadDays ) DO
      FOR ( DATETIME date IN dates ) WITH LIST result = {  } DO
         APPEND { $ "RioGrandeJemez.Inflow2" , $ "Jemez.Inflow" [date] , date } ONTO result
      ENDFOR
   ENDWITH;

    END;

  END;

  UTILITY_GROUP "Stepped Release Functions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "ComputeOutflowDecrSteppedRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
```

```
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "SteppedReleaseTest1"( reservoir ) AND "SteppedReleaseLessThanMinRelease"( reservoir )
)
   THEN
      "PreviousOutflow"( reservoir ) / "SteppedReleaseData"( reservoir, "MinStepFactor" )
   ELSE
      IF ( "SteppedReleaseTest1"( reservoir ) )
      THEN
         "PreviousOutflow"( reservoir ) - "GetSteppedReleaseFactor"( reservoir,
"PreviousOutflow"( reservoir ) )
      ELSE
         "PreviousOutflow"( reservoir ) / "SteppedReleaseData"( reservoir, "MinStepFactor" )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "ComputeOutflowIncrSteppedRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "SteppedReleaseFactorIsMaximum"( reservoir ) AND NOT "SteppedOutflowNeedsToChange"(
reservoir ) AND "HasMaxReleaseIncreased"( reservoir ) AND "IsRGWaterLeft"( reservoir ) AND NOT
"IsMaxBelowMaxThreshold"( reservoir ) )
   THEN
      "PreviousOutflow"( reservoir )
   ELSE
      IF ( "IsMaxBelowMaxThreshold"( reservoir ) AND "IsMaxThresholdExceeded"( reservoir ) )
      THEN
         "SteppedReleaseData"( reservoir, "MaxThreshold" )
      ELSE
         IF ( "SteppedReleaseTest1"( reservoir ) )
         THEN
            "PreviousOutflow"( reservoir ) + "GetSteppedReleaseFactor"( reservoir,
"PreviousOutflow"( reservoir ) )
         ELSE
            IF ( "PreviousOutflow"( reservoir ) < "SteppedReleaseData"( reservoir,
"MaxReleaseFrom0Flow" ) AND ( reservoir & "Outflow" [] <= "SteppedReleaseData"( reservoir,
"MaxReleaseFrom0Flow" ) ) )
            THEN
               reservoir & "Outflow" []
            ELSE
               IF ( "PreviousOutflow"( reservoir ) < "SteppedReleaseData"( reservoir,
"MaxReleaseFrom0Flow" ) AND ( reservoir & "Outflow" [] > "SteppedReleaseData"( reservoir,
"MaxReleaseFrom0Flow" ) ) AND ( "SteppedReleaseData"( reservoir, "MinStepFactor" ) *
"PreviousOutflow"( reservoir ) <= "SteppedReleaseData"( reservoir, "MaxReleaseFrom0Flow" ) ) )
               THEN
                  "SteppedReleaseData"( reservoir, "MaxReleaseFrom0Flow" )
               ELSE
                  "SteppedReleaseData"( reservoir, "MinStepFactor" ) * "PreviousOutflow"(
reservoir )
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF;

    END;

    FUNCTION       "DetermineSteppedRelease" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MAX_CONSTRAINT "ComputeMaxOutflow"( reservoir );
    BEGIN

      IF ( "OutflowIsIncreasing"( reservoir ) OR ( ( "SteppedReleaseFactorIsMaximum"( reservoir )
AND "HasMaxReleaseIncreased"( reservoir ) ) AND "OutflowIsIncreasing"( reservoir ) ) )
   THEN
```

```
      "ComputeOutflowIncrSteppedRelease"( reservoir )
   ELSE
      IF ( "OutflowIsDecreasing"( reservoir ) )
      THEN
         IF ( "ComputeOutflowDecrSteppedRelease"( reservoir ) > "SteppedReleaseData"( reservoir,
"ShutoffFlow" ) )
         THEN
            "ComputeOutflowDecrSteppedRelease"( reservoir )
         ELSE
            0.00000000 ["cfs"]
         ENDIF
      ELSE
         reservoir & "Outflow" []
      ENDIF
   ENDIF;

    END;

    FUNCTION       "HasMaxReleaseIncreased" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( ( @"Current Timestep" >= ( @"Start Timestep" + "SteppedReleaseData"( reservoir,
"DaysAtMaxStep" ) ) ) AND "MaxRelease#DaysAgo"( reservoir ) < "MaxReleaseYesterday"( reservoir )
)
   THEN
      TRUE
   ELSE
      FALSE
   ENDIF;

    END;

    FUNCTION       "IsMaxBelowMaxThreshold" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "MaxReleaseYesterday"( reservoir ) < "SteppedReleaseData"( reservoir, "MaxThreshold" )
)
   THEN
      TRUE
   ELSE
      FALSE
   ENDIF;

    END;

    FUNCTION       "IsMaxThresholdExceeded" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( "PreviousOutflow"( reservoir ) + "GetSteppedReleaseFactor"( reservoir, "PreviousOutflow"(
reservoir ) ) > "SteppedReleaseData"( reservoir, "MaxThreshold" ) );

    END;

    FUNCTION       "IsRGWaterLeft" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN
```

```
      "VolumeToFlow"( "PreviousRGStorage"( reservoir ), @"Current Timestep" ) +
"PreviousRGInflow"( reservoir ) > "PreviousRGOutflow"( reservoir );

    END;

    FUNCTION       "SteppedOutflowNeedsToChange" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "PreviousOutflow"( reservoir ) - reservoir & "Outflow" [@"Current Timestep" -
"SteppedReleaseData"( reservoir, "DaysAtMaxStep" )] == 0.00000000 ["cfs"];

    END;

    FUNCTION       "SteppedReleaseData" ( OBJECT reservoir, STRING column )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      ( STRINGIFY reservoir ) CONCAT "Data.SteppedReleaseData" [STRINGIFY reservoir, column];

    END;

    FUNCTION       "SteppedReleaseFactorIsMaximum" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Abs"( "GetSteppedReleaseFactor"( reservoir, "PreviousOutflow"( reservoir ) ) -
"SteppedReleaseData"( reservoir, "MaxStepFactor" ) ) < 0.01000000 ["cfs"];

    END;

    FUNCTION       "SteppedReleaseFactorIsMinimum" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetSteppedReleaseFactor"( reservoir, "PreviousOutflow"( reservoir ) ) == 0.00000000
["cfs"];

    END;

    FUNCTION       "SteppedReleaseIsNeeded" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "SteppedReleaseTest1"( reservoir ) OR "SteppedReleaseTest2"( reservoir );

    END;

    FUNCTION       "SteppedReleaseLessThanMinRelease" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
```

```
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "PreviousOutflow"( reservoir ) - "GetSteppedReleaseFactor"( reservoir, "PreviousOutflow"(
reservoir ) ) < "SteppedReleaseData"( reservoir, "MinimumRelease" );

    END;

    FUNCTION       "SteppedReleaseTest1" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "ComputeDeltaOutflow"( reservoir ) >= "GetSteppedReleaseFactor"( reservoir,
"PreviousOutflow"( reservoir ) ) AND NOT "SteppedReleaseFactorIsMinimum"( reservoir );

    END;

    FUNCTION       "SteppedReleaseTest2" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( "OutflowIsIncreasing"( reservoir ) )
   THEN
      "ComputeDeltaOutflow"( reservoir ) >= "SteppedReleaseData"( reservoir, "MinStepFactor" ) *
"PreviousOutflow"( reservoir )
   ELSE
      "ComputeDeltaOutflow"( reservoir ) >= "PreviousOutflow"( reservoir ) /
"SteppedReleaseData"( reservoir, "MinStepFactor" )
   ENDIF;

    END;

    FUNCTION       "MaxReleaseYesterday" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" == @"24:00:00 January Min DayOfMonth, Current Year" OR @"Current
Timestep" == @"Start Timestep" )
   THEN
      "PreviousOutflow"( reservoir )
   ELSE
      GET NUMERIC @INDEX 1.00000000 FROM GET LIST @INDEX 0.00000000 FROM
"MaxTimestepsForEachObject"( STRINGIFY reservoir, "Outflow", "ALL", FALSE, @"24:00:00 January Min
DayOfMonth, Current Year", @"Previous Timestep" )
   ENDIF;

    END;

    FUNCTION       "MaxRelease#DaysAgo" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( @"Current Timestep" == @"24:00:00 January Min DayOfMonth, Current Year" OR @"Current
Timestep" == @"Start Timestep" )
   THEN
      "PreviousOutflow"( reservoir )
   ELSE
```

```
      IF ( @"Current Timestep" > @"24:00:00 January Min DayOfMonth, Current Year" +
"SteppedReleaseData"( reservoir, "DaysAtMaxStep" ) )
      THEN
         GET NUMERIC @INDEX 1.00000000 FROM GET LIST @INDEX 0.00000000 FROM
"MaxTimestepsForEachObject"( STRINGIFY reservoir, "Outflow", "ALL", FALSE, @"24:00:00 January Min
DayOfMonth, Current Year", "LookBehindForSteppedRelease"( reservoir ) )
      ELSE
         GET NUMERIC @INDEX 1.00000000 FROM GET LIST @INDEX 0.00000000 FROM
"MaxTimestepsForEachObject"( STRINGIFY reservoir, "Outflow", "ALL", FALSE, @"24:00:00 January Min
DayOfMonth, Current Year", @"Previous Timestep" )
      ENDIF
   ENDIF;

    END;

    FUNCTION       "GetFlowRow" ( SLOT slot, NUMERIC flow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN slot [result, 0.00000000] AND flow >= slot [result, 0.00000000] AND
result <= 5.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

    FUNCTION       "GetSteppedReleaseFactor" ( OBJECT reservoir, NUMERIC flow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "GetObject"( ( STRINGIFY reservoir ) CONCAT "Data" ) & "SteppedReleaseTable" ["GetFlowRow"(
"GetObject"( ( STRINGIFY reservoir ) CONCAT "Data" ) & "SteppedReleaseTable", flow ), "Factor"];

    END;

    FUNCTION       "GetFlowRow1" ( SLOT slot, NUMERIC flow )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         FALSE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      WHILE ( NOT IsNaN slot [result, 0.00000000] AND flow > slot [result, 0.00000000] AND result
<= 5.00000000 ) WITH NUMERIC result = 0.00000000 DO
      result + 1.00000000
   ENDWHILE;

    END;

  END;

  UTILITY_GROUP "Supply Lists";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "GetStringFromList" ( LIST list, NUMERIC index )
    RETURN_TYPE    STRING;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG FALSE;
    BEGIN

      GET STRING @INDEX index FROM list;
```

```
    END;

    FUNCTION       "MakeSupplyList" ( LIST upstreamAccounts, LIST downstreamAccounts, OBJECT
upstreamObject, OBJECT downstreamObject )
    RETURN_TYPE    LIST;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      FOR ( STRING account IN upstreamAccounts ) WITH LIST result = {  } DO
      APPEND account CONCAT "StringifyObject"( upstreamObject ) CONCAT "To" CONCAT
"GetStringFromList"( downstreamAccounts, LENGTH result ) CONCAT "StringifyObject"(
downstreamObject ) CONCAT ".Supply" ONTO result
   ENDFOR;

    END;

    FUNCTION       "WaiverSupply" ( STRING account, OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      IF ( account IN "AbiquiuRootStorageAccounts"(  ) AND "StringifyObject"( reservoir ) ==
"Abiquiu" )
   THEN
      IF ( IsNaN account CONCAT "HeronTo" CONCAT account CONCAT "StringifyObject"( reservoir )
CONCAT "WaiverHeron.Supply" [] )
      THEN
         0.00000000 ["cfs"]
      ELSE
         account CONCAT "HeronTo" CONCAT account CONCAT "StringifyObject"( reservoir ) CONCAT
"WaiverHeron.Supply" []
      ENDIF
   ELSE
      IF ( account IN "ElVadoStorageAccounts"(  ) AND "StringifyObject"( reservoir ) == "ElVado"
)
      THEN
         IF ( IsNaN account CONCAT "HeronTo" CONCAT account CONCAT "StringifyObject"( reservoir )
CONCAT "WaiverHeron.Supply" [] )
         THEN
            0.00000000 ["cfs"]
         ELSE
            account CONCAT "HeronTo" CONCAT account CONCAT "StringifyObject"( reservoir ) CONCAT
"WaiverHeron.Supply" []
         ENDIF
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

  END;

  UTILITY_GROUP "RGConservationFunctions";
  DESCRIPTION   "";
  ACTIVE        TRUE;
  BEGIN

    FUNCTION       "AbiquiuEasementSpaceAvailableAsFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "VolumeToFlow"( "ElevationToStorage"( % "Abiquiu", $ "AbiquiuData.PoolLevels" ["Elevation",
"SJCEasement"] ), @"Current Timestep" ) - "VolumeToFlow"( "PreviousStorage"( % "Abiquiu" ),
@"Current Timestep" );
```

```
    END;

    FUNCTION       "AbiquiuRGConsSpaceAvailableAsFlow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "VolumeToFlow"( $ "AbiquiuData.RGConservationSpaceAvailable" ["Storage", "RGConservation"],
@"Current Timestep" ) - "VolumeToFlow"( "PreviousAccountStorage"( "RioGrandeConservation", %
"Abiquiu" ), @"Current Timestep" );

    END;

    FUNCTION       "AbiquiuRGOutflowWRGConsTransfer" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "Max"( "Max"( "AbiquiuDSDemand"(  ), "MinRGOutflow"( % "Abiquiu" ) ), $
"AbiquiuData.MRGCDDemand" [] );

    END;

    FUNCTION       "RGConsInflow" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "CurrentRGInflow"( reservoir );

    END;

    FUNCTION       "RGConsInflowWPrevGainLoss" ( OBJECT reservoir )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "CurrentRGInflow"( reservoir ) + "VolumeToFlow"( "PreviousAccountGainLoss"( "RioGrande",
reservoir ), @"Current Timestep" );

    END;

    FUNCTION       "AbiquiuDSDemand" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "AbiquiuData.MinRGFlowToAllowConsStorage" ["GetSeasonRow"( % "Abiquiu",
"MinRGFlowToAllowConsStorage" ), 1.00000000];

    END;

    FUNCTION       "IsRGConsSpaceAvailable" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
```

```
    BEGIN

      "PreviousAccountStorage"( "RioGrandeConservation", % "Abiquiu" ) < $
"AbiquiuData.RGConservationSpaceAvailable" ["Storage", "RGConservation"] AND $
"AbiquiuData.RGConservationSpaceAvailable" ["Storage", "RGConservation"] > 0.00000000 ["acre-
feet"];

    END;

    FUNCTION       "IsEasementSpaceAvailable" ( OBJECT reservoir )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "StorageToElevation"( reservoir, "PreviousStorage"( reservoir ) ) < $
"AbiquiuData.PoolLevels" ["Elevation", "SJCEasement"];

    END;

    FUNCTION       "IsRGFlowEnoughToMeetMRGCDDemand" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "CurrentRGInflow"( % "Abiquiu" ) >= $ "AbiquiuData.MRGCDDemand" [];

    END;

    FUNCTION       "IsRGFlowEnoughToMeetDSDemand" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      "AbiquiuMinRGOutflow"(  ) > "AbiquiuDSDemand"(  );

    END;

    FUNCTION       "ComputeAbiquiuRGConsInflow" (  )
    RETURN_TYPE    NUMERIC;
    SCALE_UNITS    "cfs";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    MIN_CONSTRAINT 0.00000000 ["cfs"];
    BEGIN

      IF ( NOT IsNaN ( STRINGIFY % "Abiquiu" ) CONCAT "^" CONCAT "RioGrande" CONCAT ".Gain Loss"
[@"Previous Timestep"] AND "PreviousAccountGainLoss"( "RioGrande", % "Abiquiu" ) < 0.00000000
["acre-feet"] )
   THEN
      IF ( "IsRGFlowEnoughToMeetMRGCDDemand"(  ) AND "IsEasementSpaceAvailable"( % "Abiquiu" )
AND "IsRGConsSpaceAvailable"( % "Abiquiu" ) AND NOT "CompactVIIIsInEffect"(  ) AND
"IsRGFlowEnoughToMeetDSDemand"(  ) AND NOT "IsMinFlowsDemandRequired"(  ) )
      THEN
         IF ( "PreviousRGStorage"( % "Abiquiu" ) < 0.00000000 ["acre-feet"] )
         THEN
            "Min"( "RGConsInflowWPrevGainLoss"( % "Abiquiu" ) -
"AbiquiuRGOutflowWRGConsTransfer"(  ) + "VolumeToFlow"( "PreviousRGStorage"( % "Abiquiu" ),
@"Current Timestep" ), "Min"( "AbiquiuEasementSpaceAvailableAsFlow"(  ),
"AbiquiuRGConsSpaceAvailableAsFlow"(  ) ) )
         ELSE
            "Min"( "RGConsInflowWPrevGainLoss"( % "Abiquiu" ) -
"AbiquiuRGOutflowWRGConsTransfer"(  ), "Min"( "AbiquiuEasementSpaceAvailableAsFlow"(  ),
"AbiquiuRGConsSpaceAvailableAsFlow"(  ) ) )
         ENDIF
      ELSE
```

```
         0.00000000 ["cfs"]
      ENDIF
   ELSE
      IF ( @"Current Timestep" == @"Start Timestep" )
      THEN
         IF ( "IsRGFlowEnoughToMeetMRGCDDemand"(  ) AND "IsEasementSpaceAvailable"( % "Abiquiu" )
AND "IsRGConsSpaceAvailable"( % "Abiquiu" ) AND NOT "CompactVIIIsInEffect"(  ) AND
"IsRGFlowEnoughToMeetDSDemand"(  ) AND NOT "IsMinFlowsDemandRequired"(  ) )
         THEN
            "Min"( "RGConsInflow"( % "Abiquiu" ) - "AbiquiuRGOutflowWRGConsTransfer"(  ), "Min"(
"AbiquiuEasementSpaceAvailableAsFlow"(  ), "AbiquiuRGConsSpaceAvailableAsFlow"(  ) ) )
         ELSE
            0.00000000 ["cfs"]
         ENDIF
      ELSE
         0.00000000 ["cfs"]
      ENDIF
   ENDIF;

    END;

    FUNCTION       "IsMinFlowsDemandRequired" (  )
    RETURN_TYPE    BOOLEAN;
    SCALE_UNITS    "";
    DESCRIPTION    "";
    ACTIVE         TRUE;
    PRE_EXEC_DIAG  FALSE;
    POST_EXEC_DIAG TRUE;
    BEGIN

      $ "AbiquiuData.ComputedMinFlowsDemand" [] > 0.00000000 ["cfs"];

    END;

  END;

END
```