MODSIM 8.1: River Basin Management Decision Support System

Tutorials and Example Networks

contact:

John W. Labadie
Department of Civil and Environmental Engineering
Colorado State University
Ft. Collins, CO 80523-1372
email: labadie@engr.colostate.edu

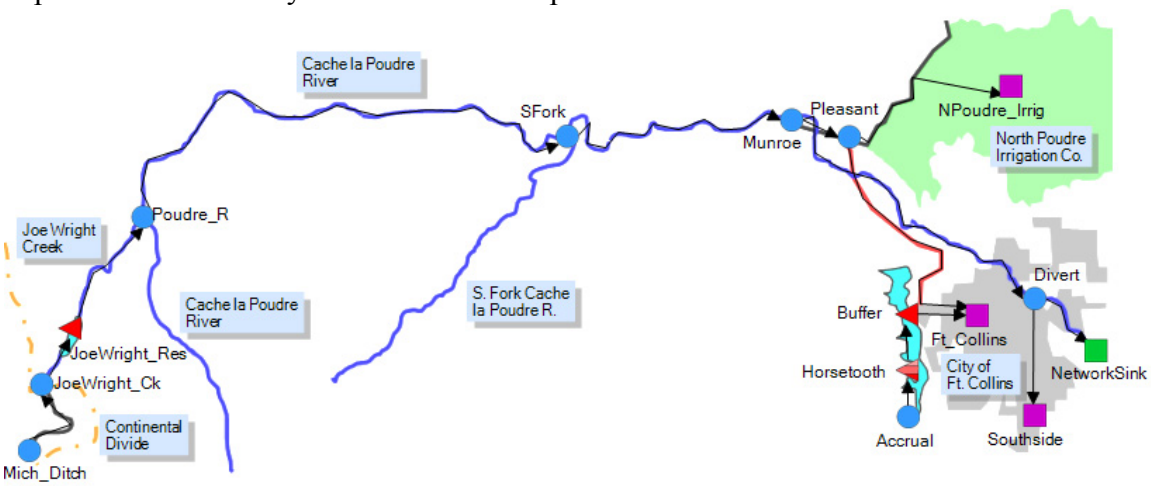
2010

TABLE OF CONTENTS

TUTORIAL A
Conjunctive Use of Surface Water and Groundwater

A.1 Introduction

This Tutorial provides an introduction to the capabilities and functionality of MODSIM through a simplified example of water supply analysis in the Poudre River Basin, Colorado under water rights, storage accounts, exchanges, conjunctive use of surface and groundwater, and a *Plan for Augmentation*. The City of Ft. Collins has senior direct flow water rights on the Poudre River, with diversion via the Pleasant Valley Pipeline to the expanded Soldier Canyon water treatment plant next to Horsetooth Reservoir.



All inflows to Horsetooth Reservoir originate from the Western Slope via the Colorado-Big Thompson project, providing supplemental water for:

- M&I water supply augmentation
- Irrigation water supply

The North Poudre Irrigation Co. has junior direct flow rights on the Poudre River, but owns shares in Horsetooth Reservoir (1 share = 1 acre-foot), even though it is unable to physically withdraw water from Horsetooth Reservoir. Although Ft. Collins receives diversions from the Poudre River via the Pleasant Valley Pipeline, for purposes of this Tutorial, the limited capacity of the pipeline is assumed to be insufficient to meet future projected demands. Although, in reality, Ft. Collins also owns shares in Horsetooth Reservoir, it will be assumed that these shares are insufficient to meet future demands.

To assure satisfaction of future water demands, Ft. Collins needs to enter into an Exchange Agreement with North Poudre Irrigation Co., allowing North Poudre to divert Poudre River flow owned by Ft. Collins, and in exchange, allowing Ft. Collins to receive shares from North Poudre directly from Horsetooth Reservoir. Total seasonal diversion of Ft. Collins water by North Poudre should equal total release from Horsetooth.

Ft. Collins is possibly interested in purchasing the Southside Ditch water rights, the oldest and highest priority in the basin, as farm lands south of Ft. Collins are converted to urban use. In order to provide future water supply, Ft. Collins was interested in building a new reservoir—Joe Wright Reservoir. Since it was new, it had the lowest storage right in the basin—i.e., the right to divert water for storage. Ft. Collins needed to determine how large to build the reservoir. Releases from Joe Wright also enhance the Exchange Agreement with North Poudre Irrigation Co.

The North Poudre Irrigation Co. is considering development of a well field for irrigation water supply from groundwater as a supplement to their water supply. However, since they have the lowest priority in the basin, they must insure that groundwater pumping does not deplete the river and injure senior water rights due to stream-aquifer interactions.

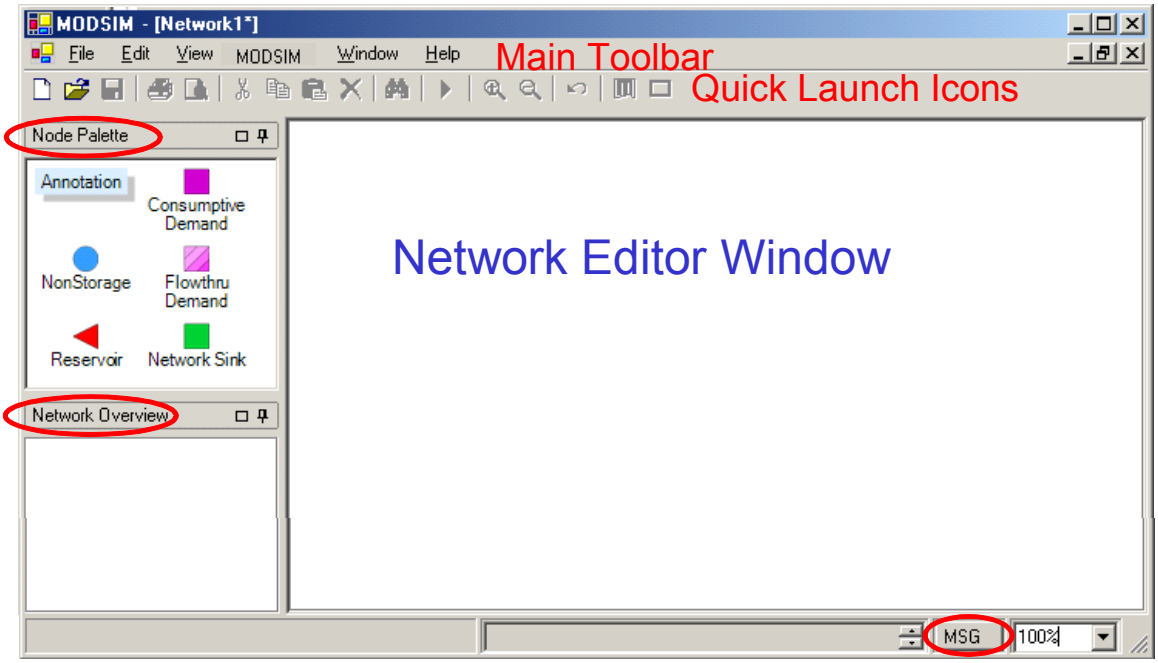
Means of increasing water supply to Ft. Collins are ranked in order of increasing cost as:

- Enter into an Exchange Agreement with North Poudre Irrigation Company
- Purchase the Southside Ditch water rights
- Construct Joe Wright Reservoir at minimum capacity needed to satisfy projected demands
- For North Poudre Irrigation Co., determine the minimum groundwater pumping capacity (within safe yield limits) to meet their demands.

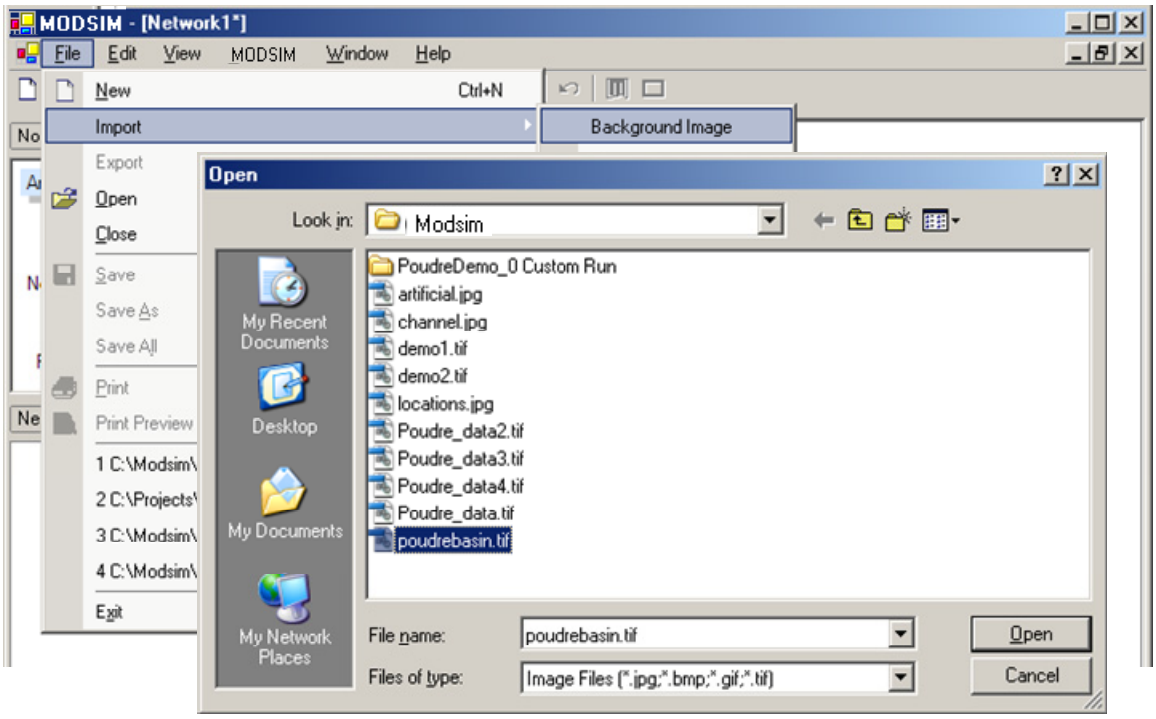
A.2 Network Creation

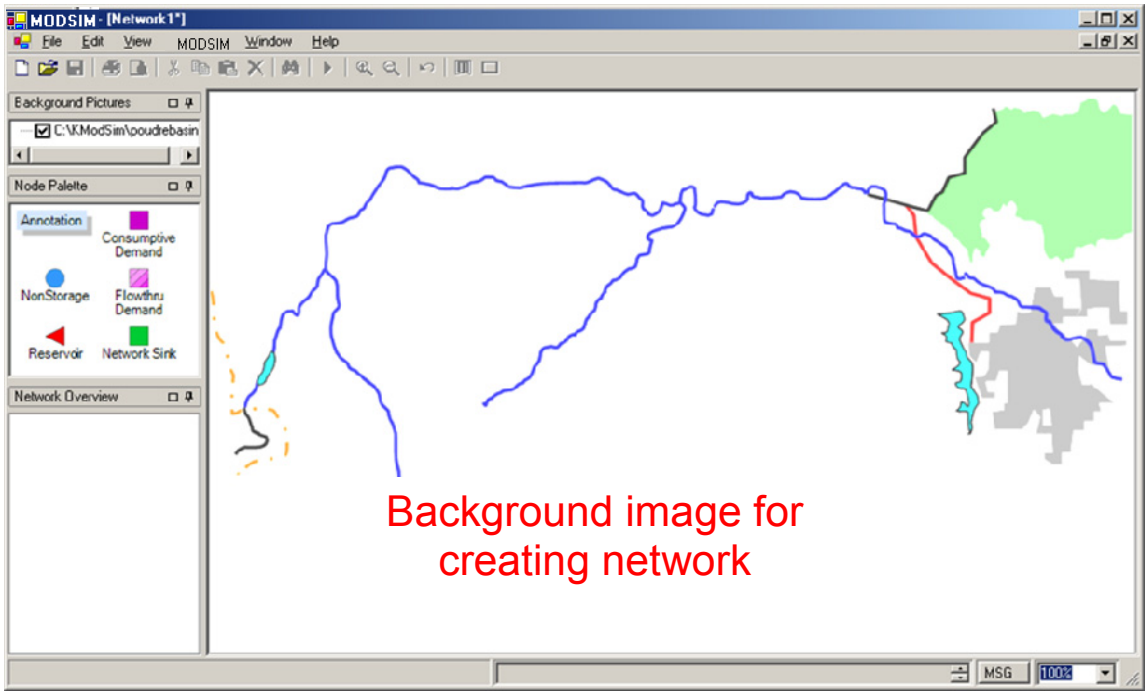
Executing MODSIM 8.1 opens the graphical user interface, revealing the six main areas:

- The Main Toolbar
- Toolbar with Quick Launch Icons
- The Network Editor Window
- The Node Palette Menu
- The Network Overview Window
- The Message Button

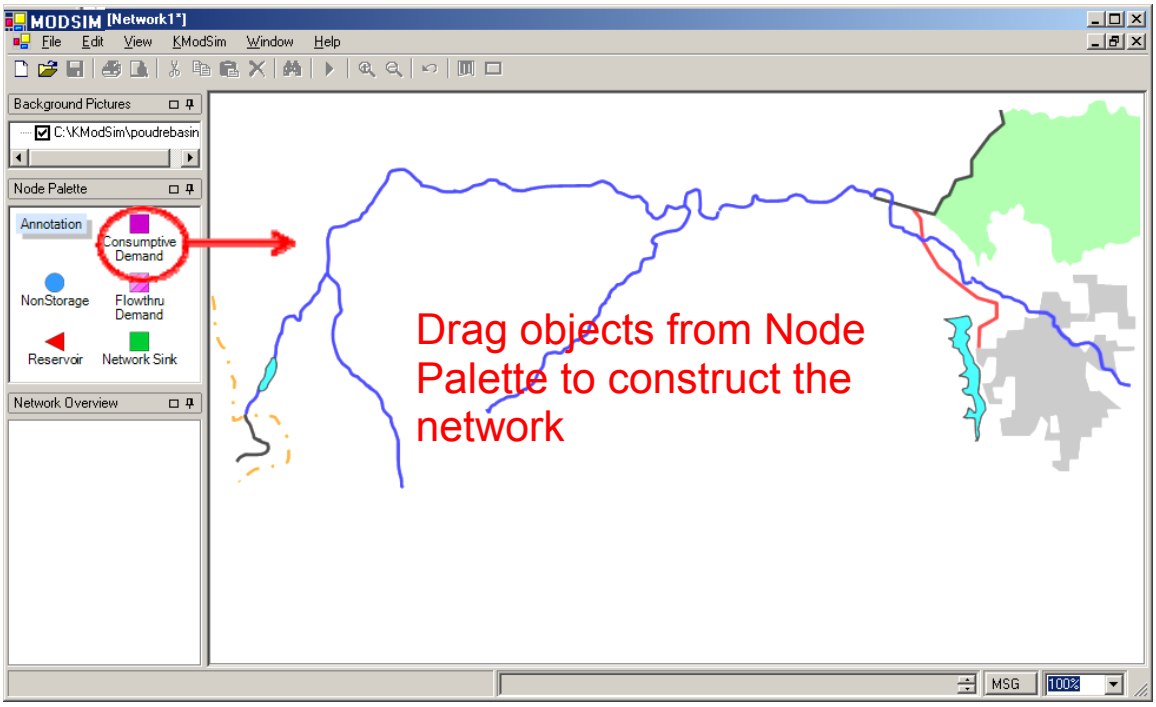


In order to provide an aid in developing geo-referenced network topologies, it is useful to import a background image of the study area. Image files may be in *.jpg, *.bmp, *.gif, or *.tif formats, and can be created by saving a geographic information system (GIS) layer as an image file, downloading from the Internet, or other sources. For this Tutorial, the image file poudrebasin.tif is included and is accessed by selecting **File > Import > Background image** from the Main Menu, and then browsing to the location of the image file, which is included in the installation of MODSIM.

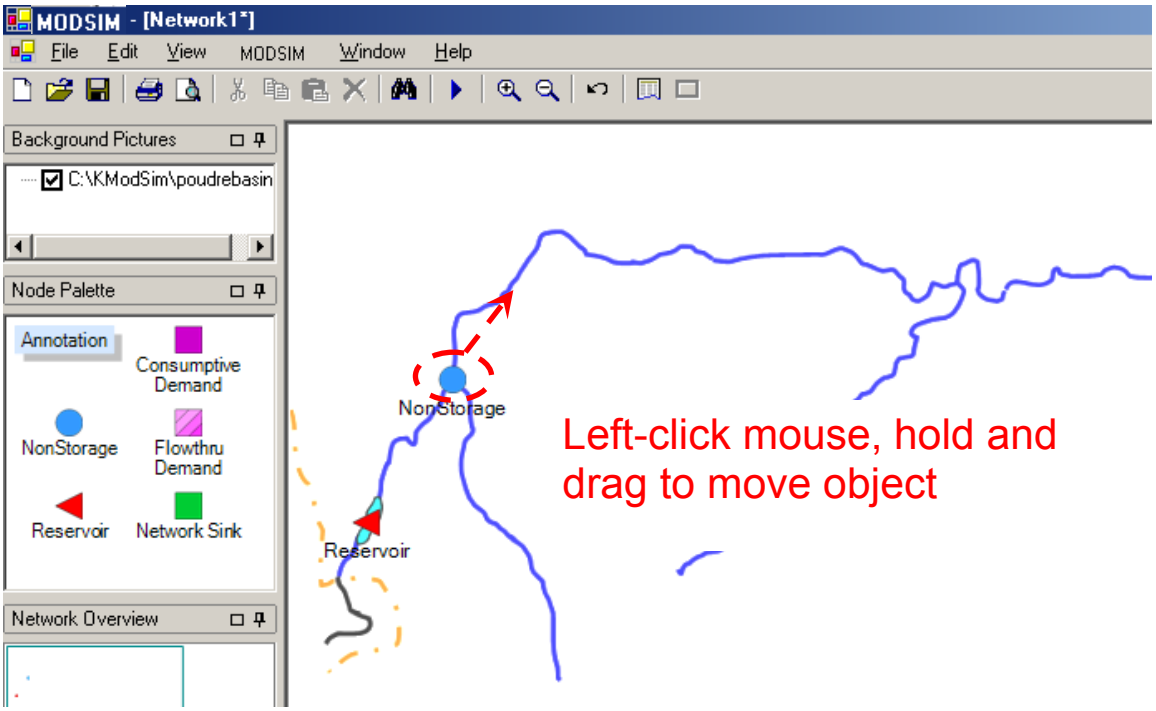




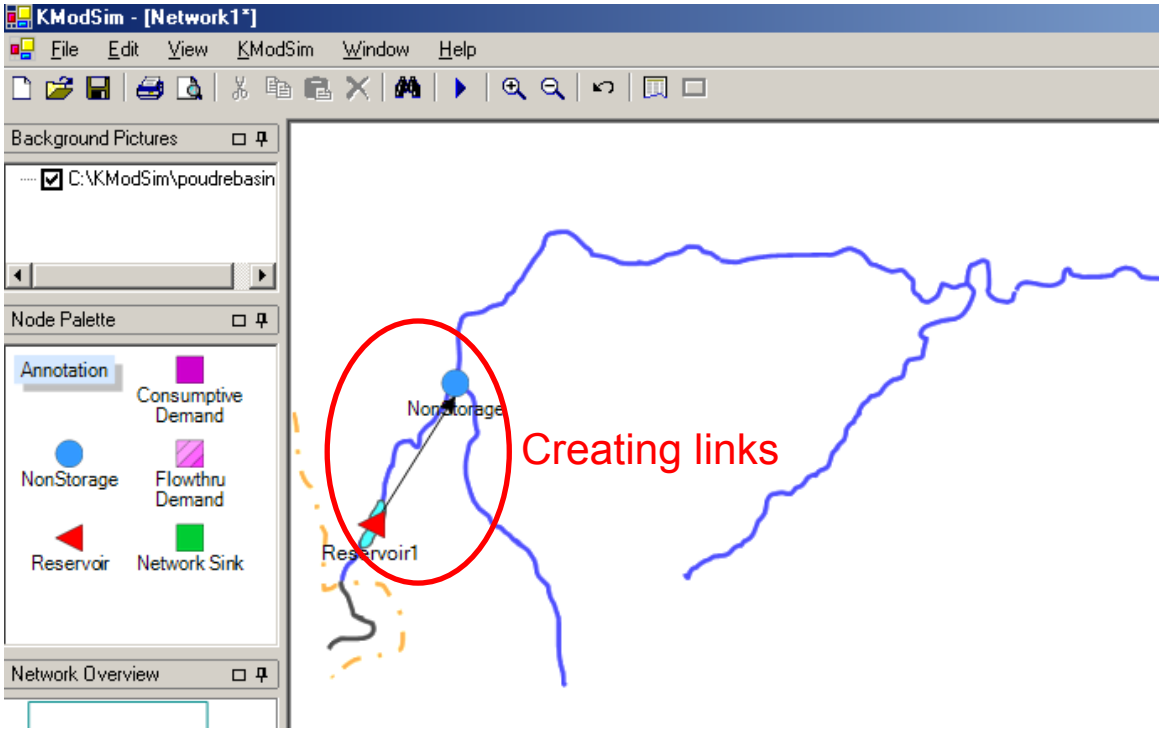
Networks are created by dragging node objects from the **Node Palette** to the **Network Editor Window**. Alternatively, users can click on an icon in the **Node Palette** and then click a desired location in the **Network Editor Window**. Note that in this Tutorial, the term "click" will refer to "left mouse button click."



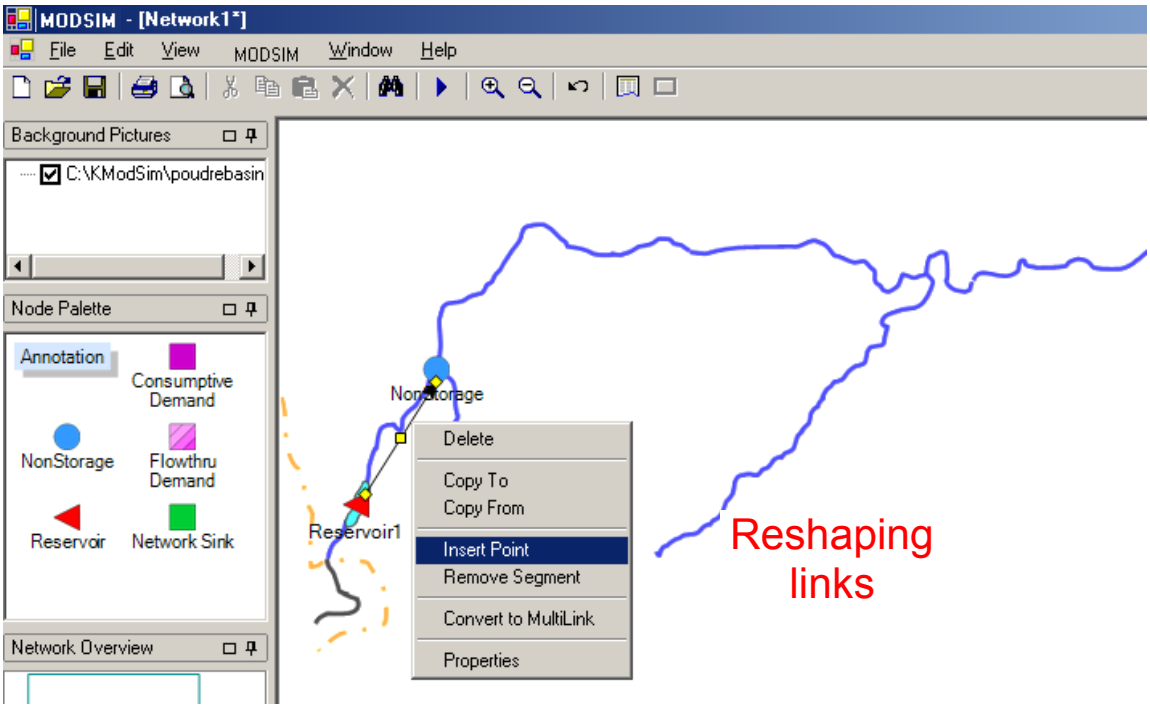
Objects placed within the **Network Editor Window** are easily moved by mouse click, hold and drag operations



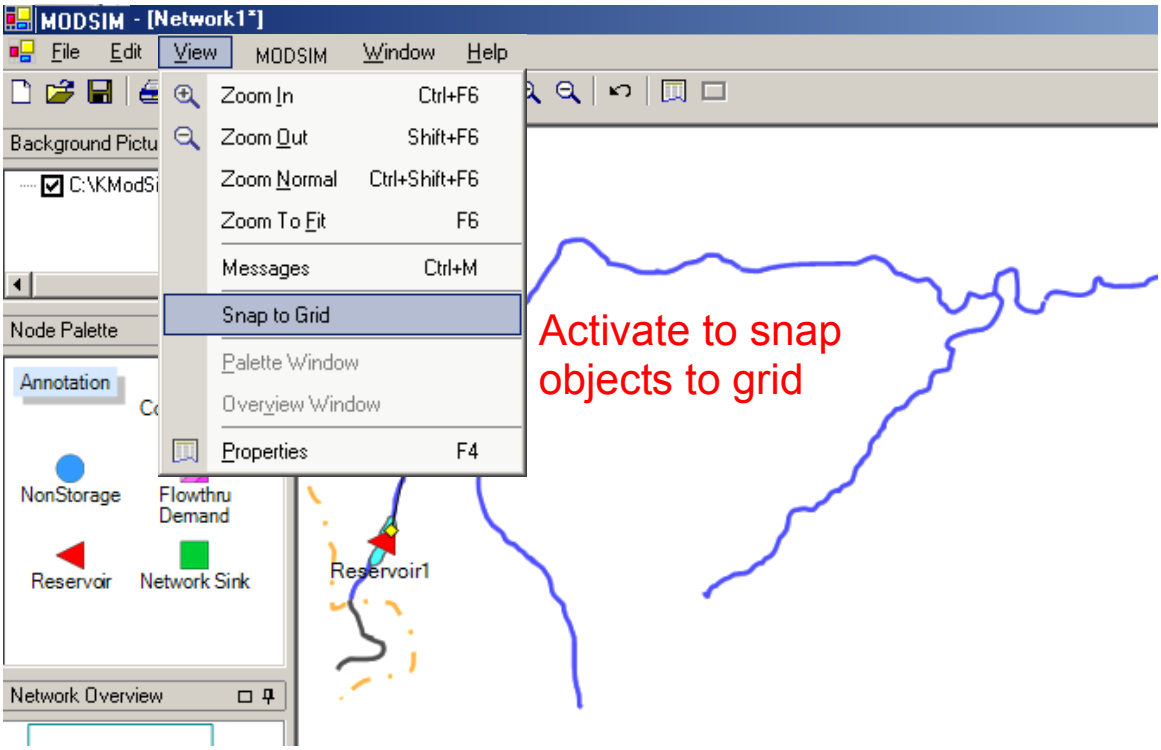
To create links in the **Network Editor Window**, simply hover the cursor over the start node until the pointer icon appears, then click, hold and drag to ending node.



To change the shape of a link, right mouse click at any location along the link and select *Insert Point* from the context menu. As many vertices as desired can be inserted, as represented by yellow boxes. Hovering over a vertex results in a 4 sided arrow icon , allowing the vertex to be moved to any location.

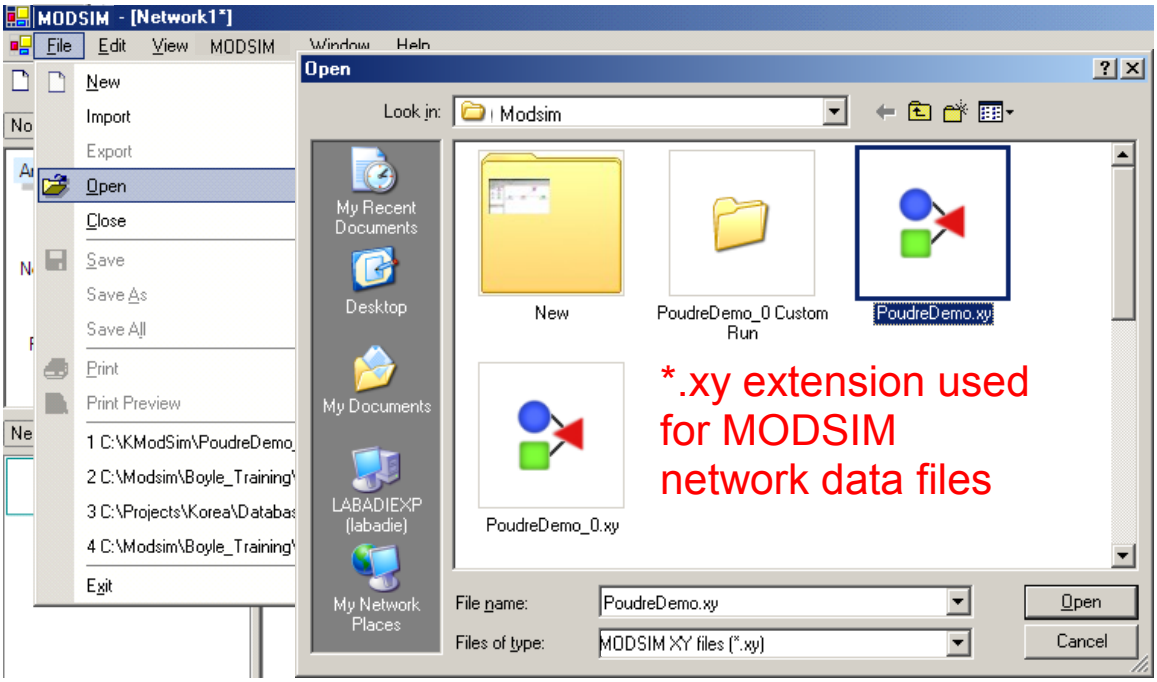


For more consistent placement of objects, the **View > Snap to Grid** item can be activated, which snaps objects to a background grid in the window.

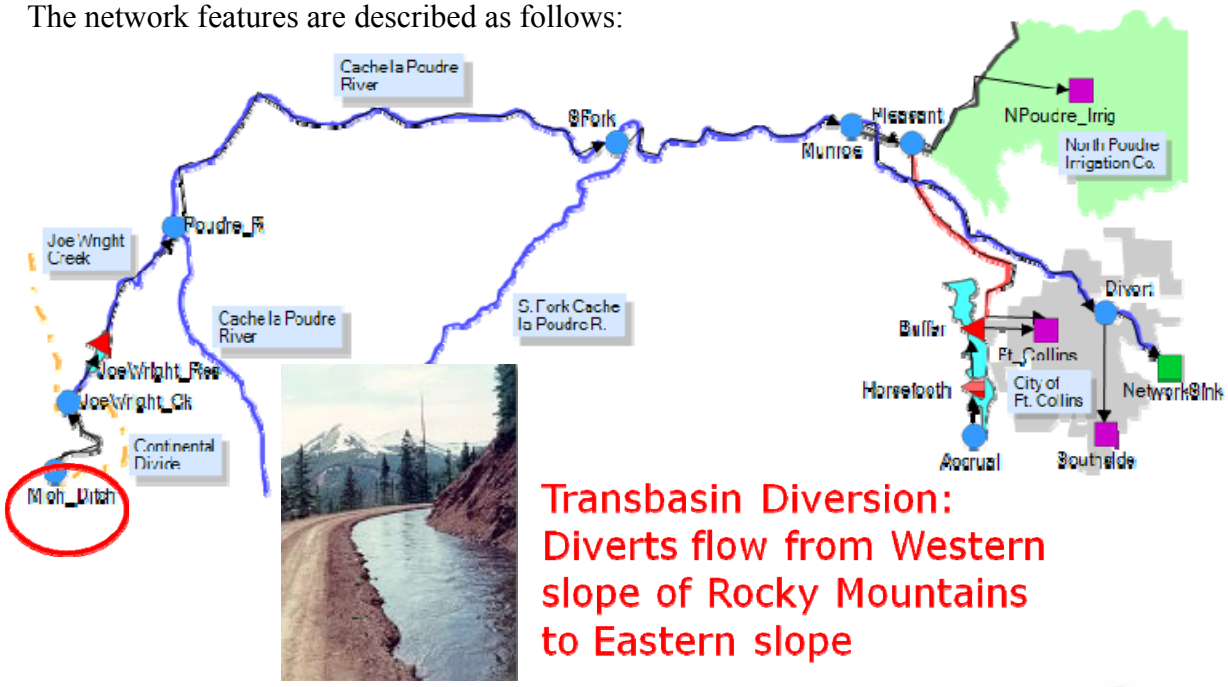


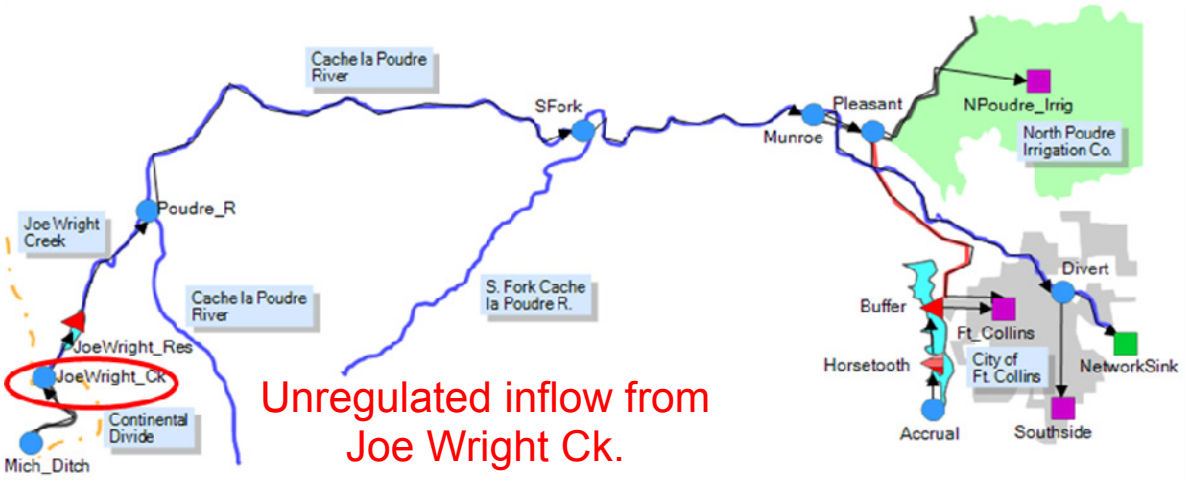
A.3 Base Network Development

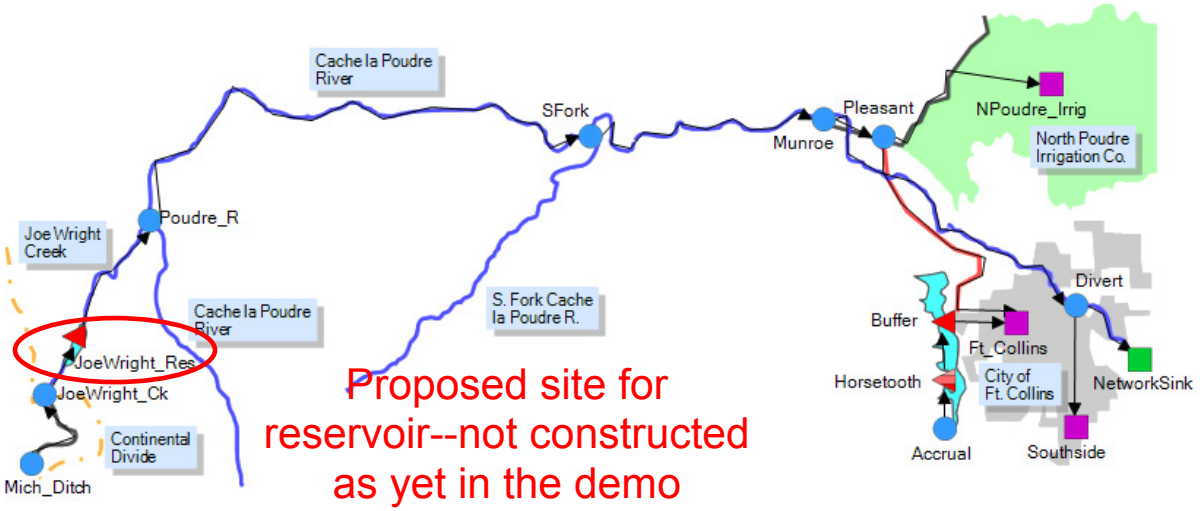
The base network **PoudreDemo.xy** can be loaded into the interface by **File > Open** and browsing to the location of the file.



The network features are described as follows:

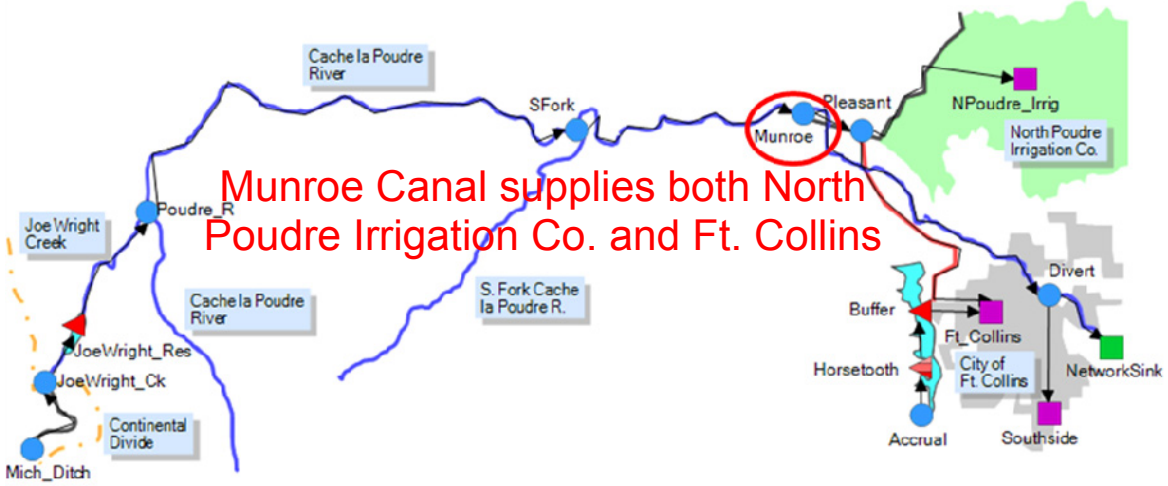




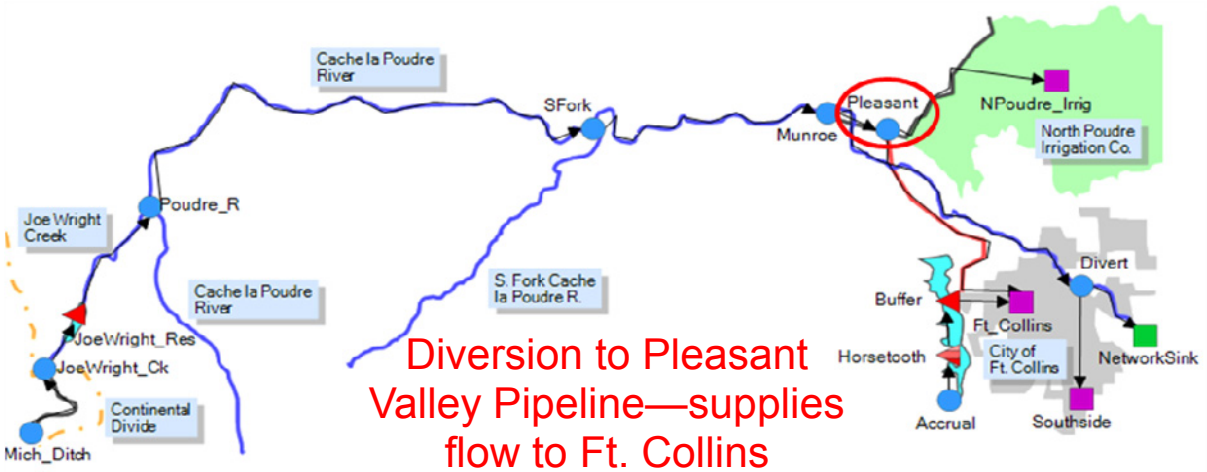


Inflow from Upper Segment of the Cache la Poudre River

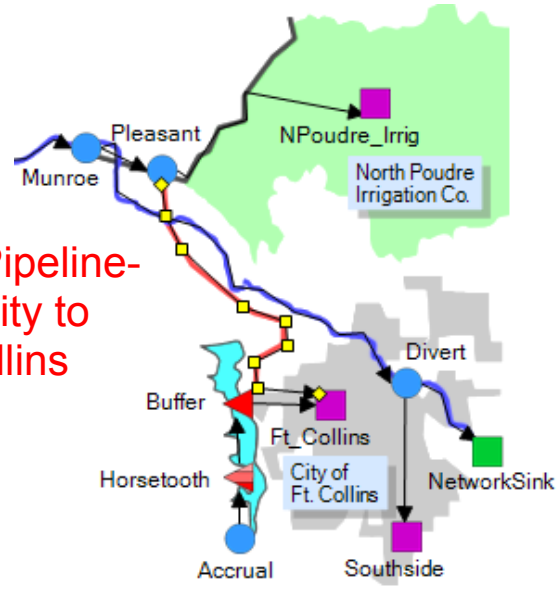
Inflow from the South Fork of the Poudre River; portions of the South Fork are protected by designation as “Wild and Scenic”



Munroe Canal supplies both North Poudre Irrigation Co. and Ft. Collins

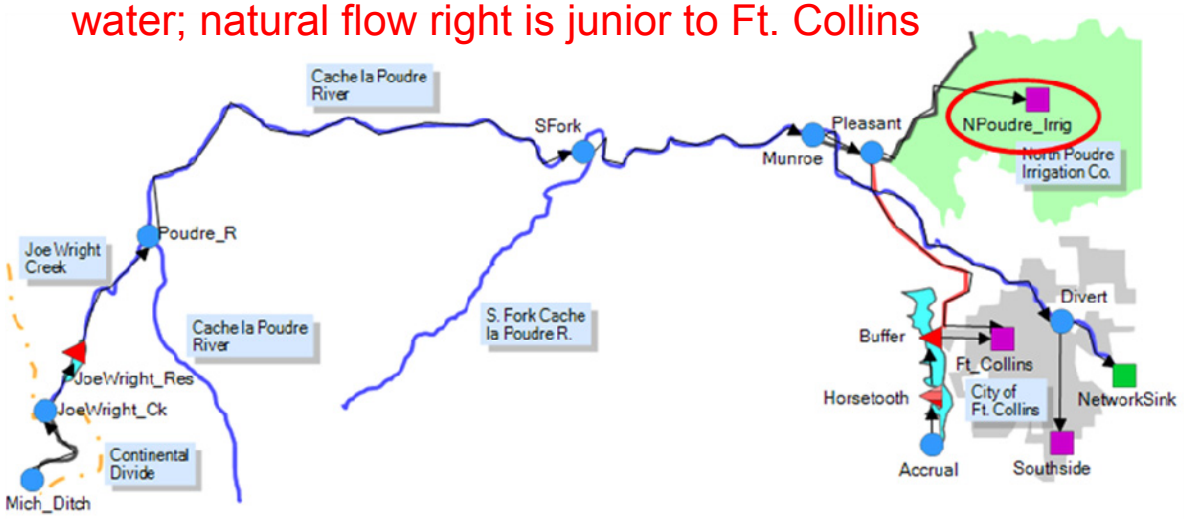


Diversion to Pleasant Valley Pipeline—supplies flow to Ft. Collins



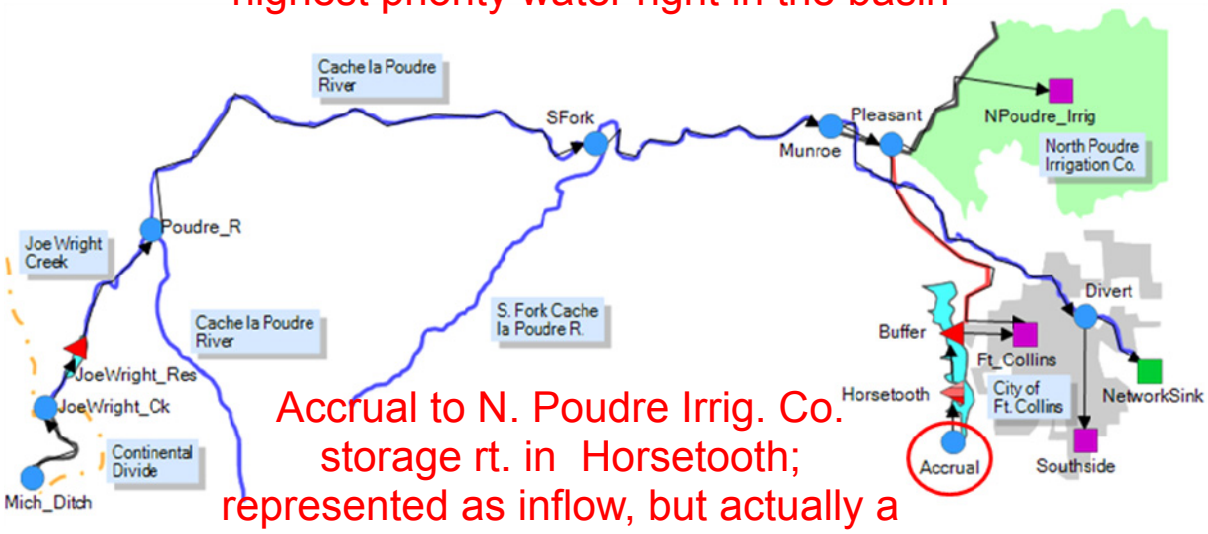
Pleasant Valley Pipeline--limited capacity to supply Ft. Collins

North Poudre Irrigation Co. Demand--owns shares in Horsetooth, but unable to physically receive the water; natural flow right is junior to Ft. Collins



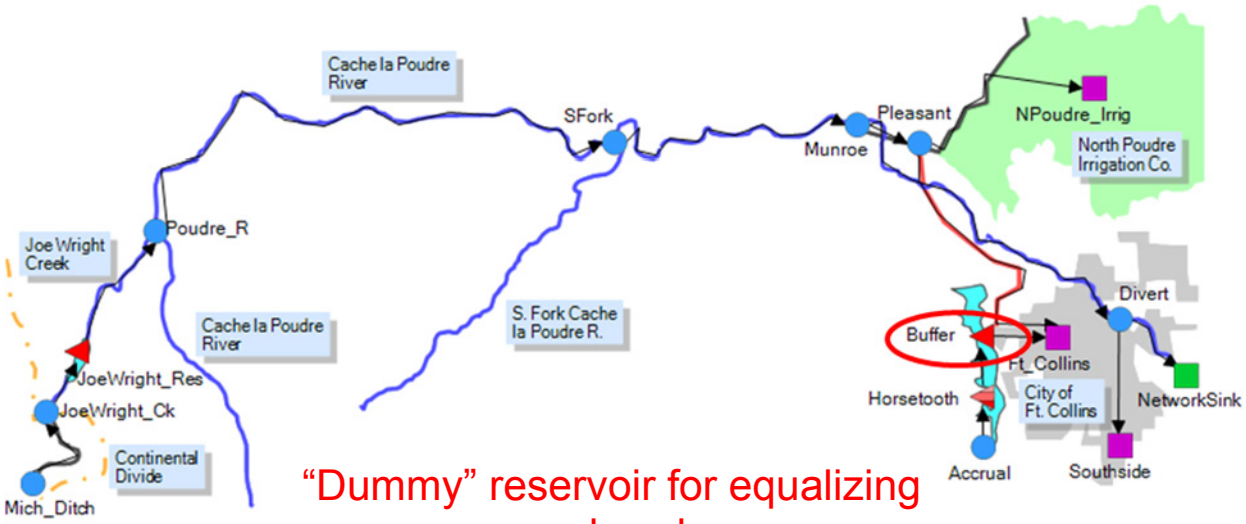
City of Ft. Collins Demand; can only obtain water from Horsetooth through an Exchange

Southside Irrigation Ditch Demand; oldest and highest priority water right in the basin



Accrual to N. Poudre Irrig. Co. storage rt. in Horsetooth; represented as inflow, but actually a storage right

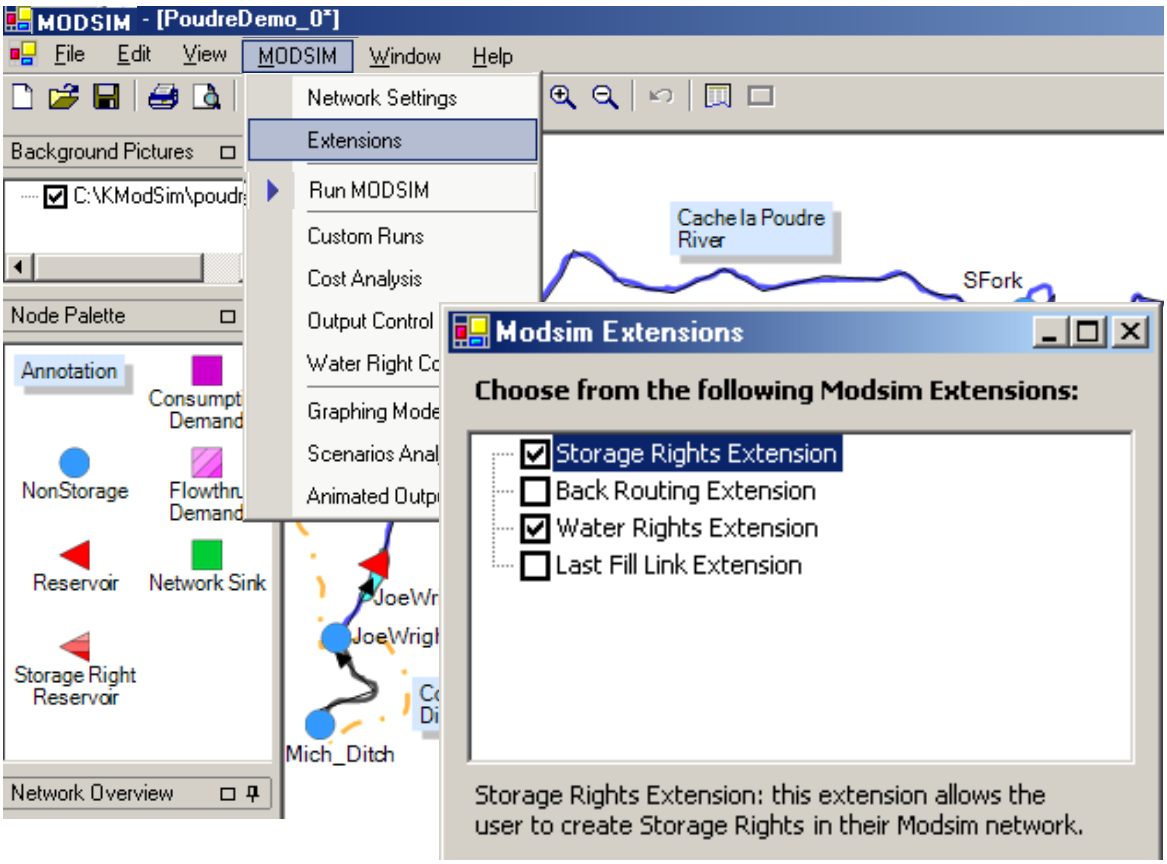
All inflow to Horsetooth Reservoir comes from transbasin diversion—negligible local inflow



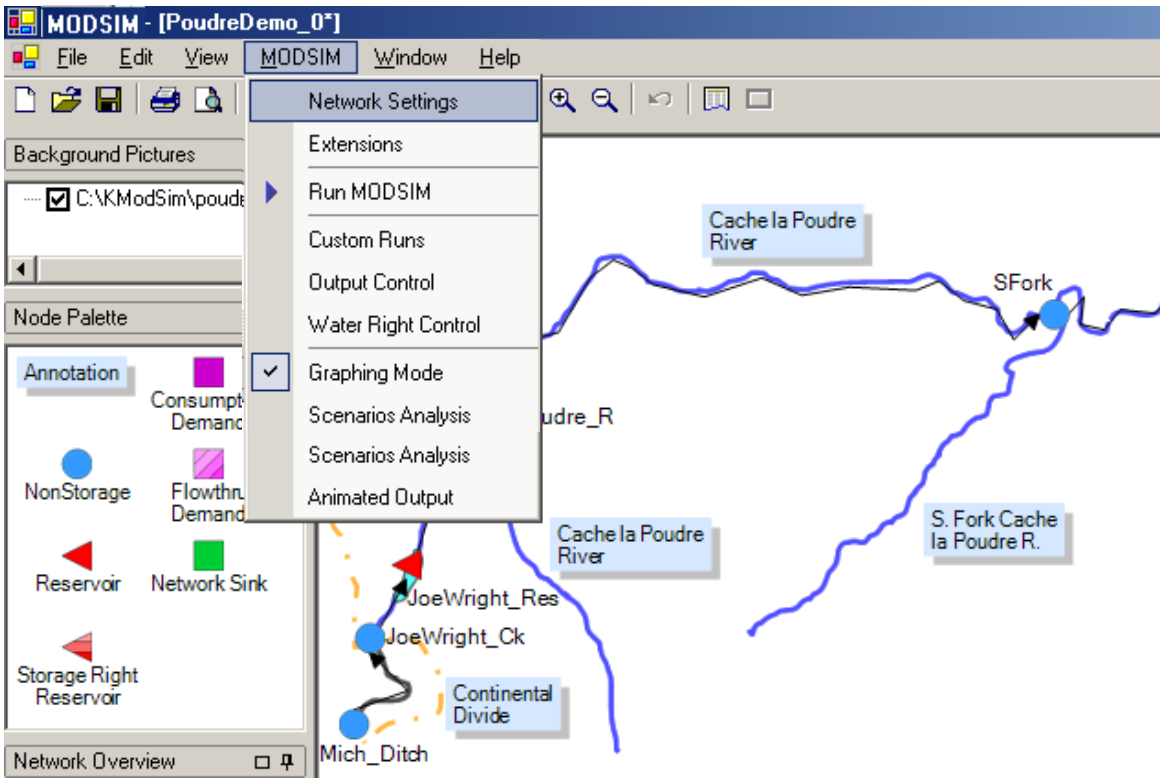
"Dummy" reservoir for equalizing seasonal exchange

A.4 Network Settings and Data Import

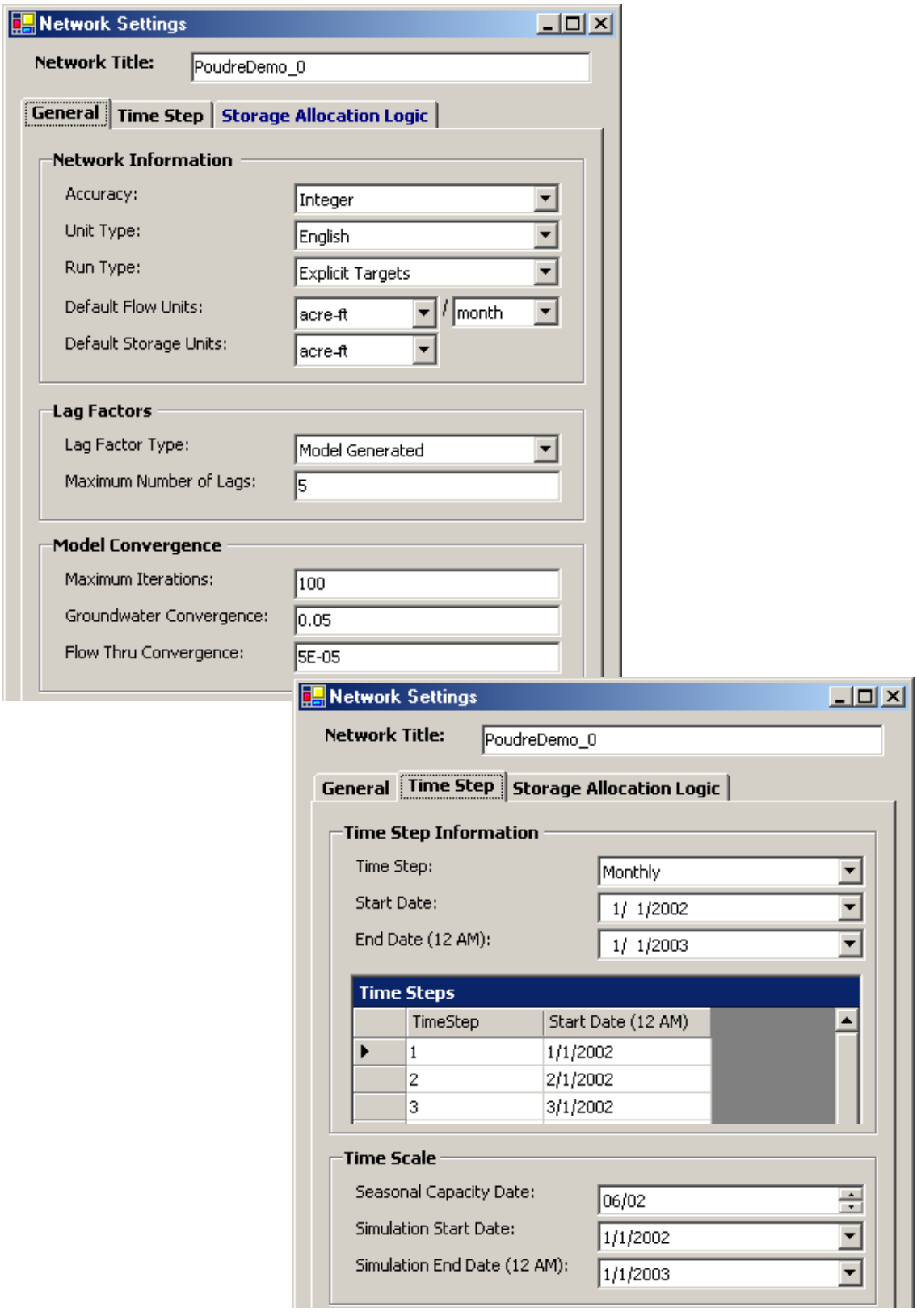
For this Tutorial, the *Storage Rights Extension* and the *Water Rights Extension* must be selected by clicking the checkboxes in the form opened by selecting **MODSIM > Extensions.**



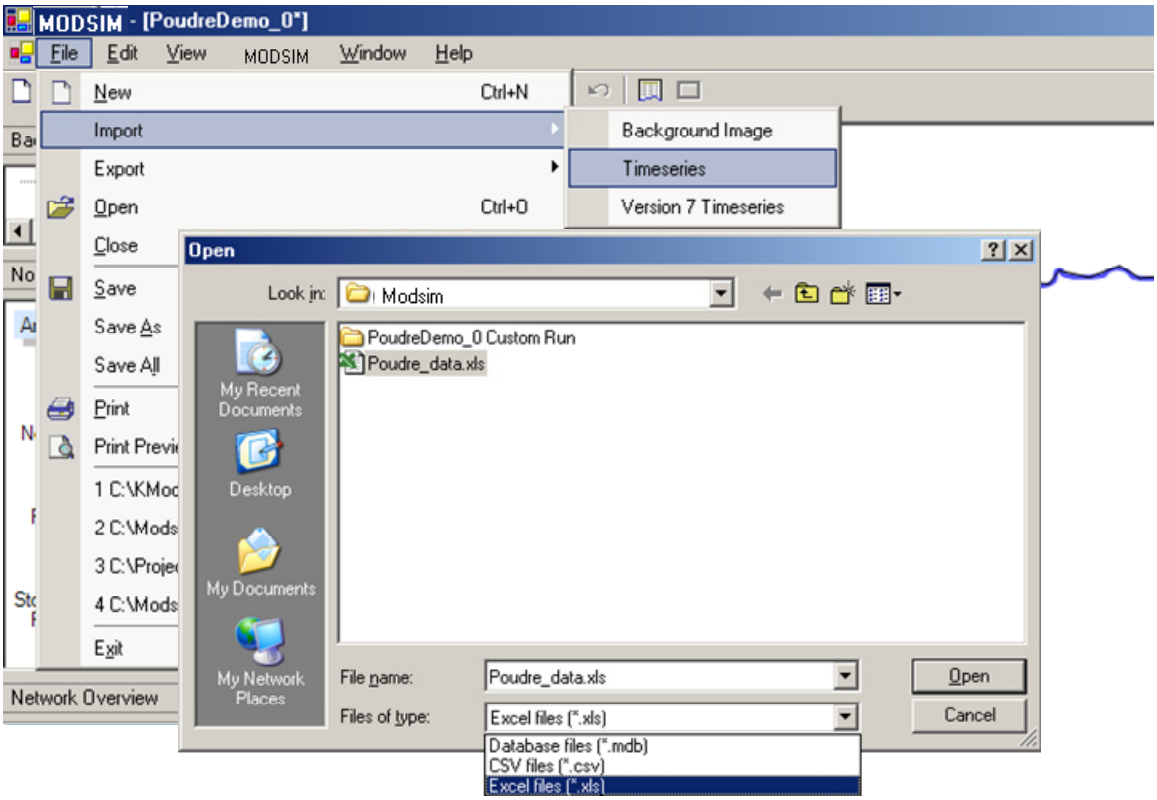
The Network Settings are entered by selecting **MODSIM > Network Settings**:



The following selections should be made under the **General** and **Time Step** tabs of the **Network Settings** form:



Included with this Tutorial is the data base *Poudre_data.xls* containing time series data for inflows, demands, net evaporation rates, and variable link capacities for this demonstration network. These data can be automatically loaded into the network using the **File > Import > Timeseries** tool:

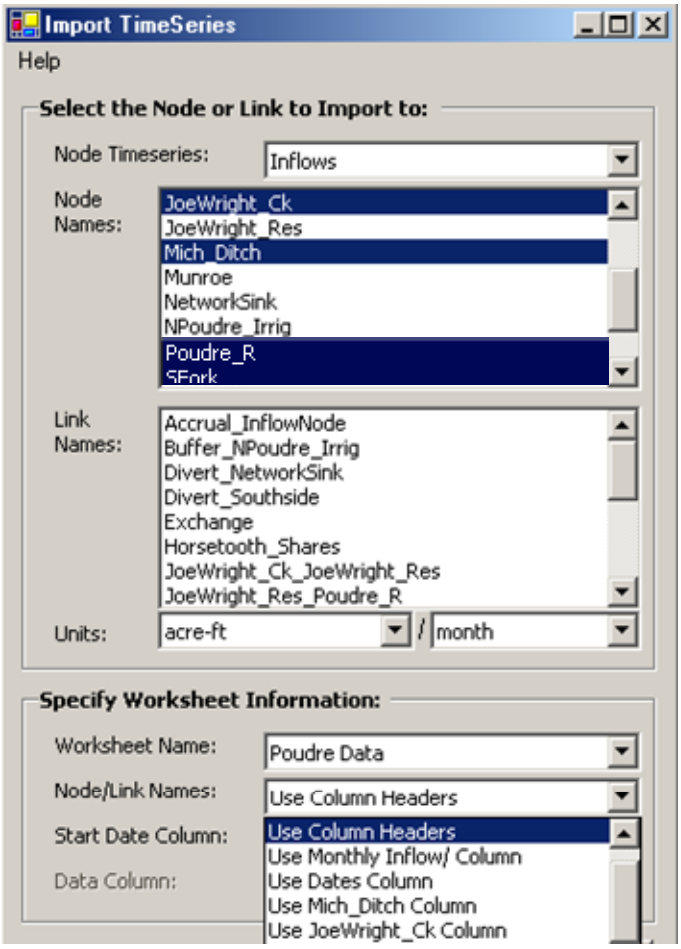


The time series data can be stored as MS Access *.mdb database files, MS Excel *.xls files, or *.csv text files. The Excel database in Poudre_data.xls is listed below. This is the desired format for the database, where a Dates column is followed by columns with the exact names (case sensitive) of nodes and links in the network for which time series data are being loaded.

Dates	Mich_Ditch	JoeWright_Ck	Poudre_R	SFork	Ft_Collins	NPoudre_Irrig	Net Evap	JoeWright_Res_Poudre_R *
1/1/2002	0	0	450	150	500	0	-0.05	500
2/1/2002	0	0	500	250	500	0	-0.02	500
3/1/2002	0	100	1100	600	500	0	0.01	500
4/1/2002	0	1000	2000	1500	600	500	0.04	500
5/1/2002	700	1800	3200	1800	1000	1000	0.14	99999
6/1/2002	700	950	1200	600	1500	2000	0.22	99999
7/1/2002	700	300	300	100	2000	4000	0.27	99999
8/1/2002	700	100	75	25	2000	2000	0.35	99999
9/1/2002	700	75	50	50	1500	500	0.28	99999
10/1/2002	0	50	200	150	600	0	0.17	500
11/1/2002	0	30	300	200	500	0	0.06	500
12/1/2002	0	0	350	150	500	0	0.01	500

* foreign water

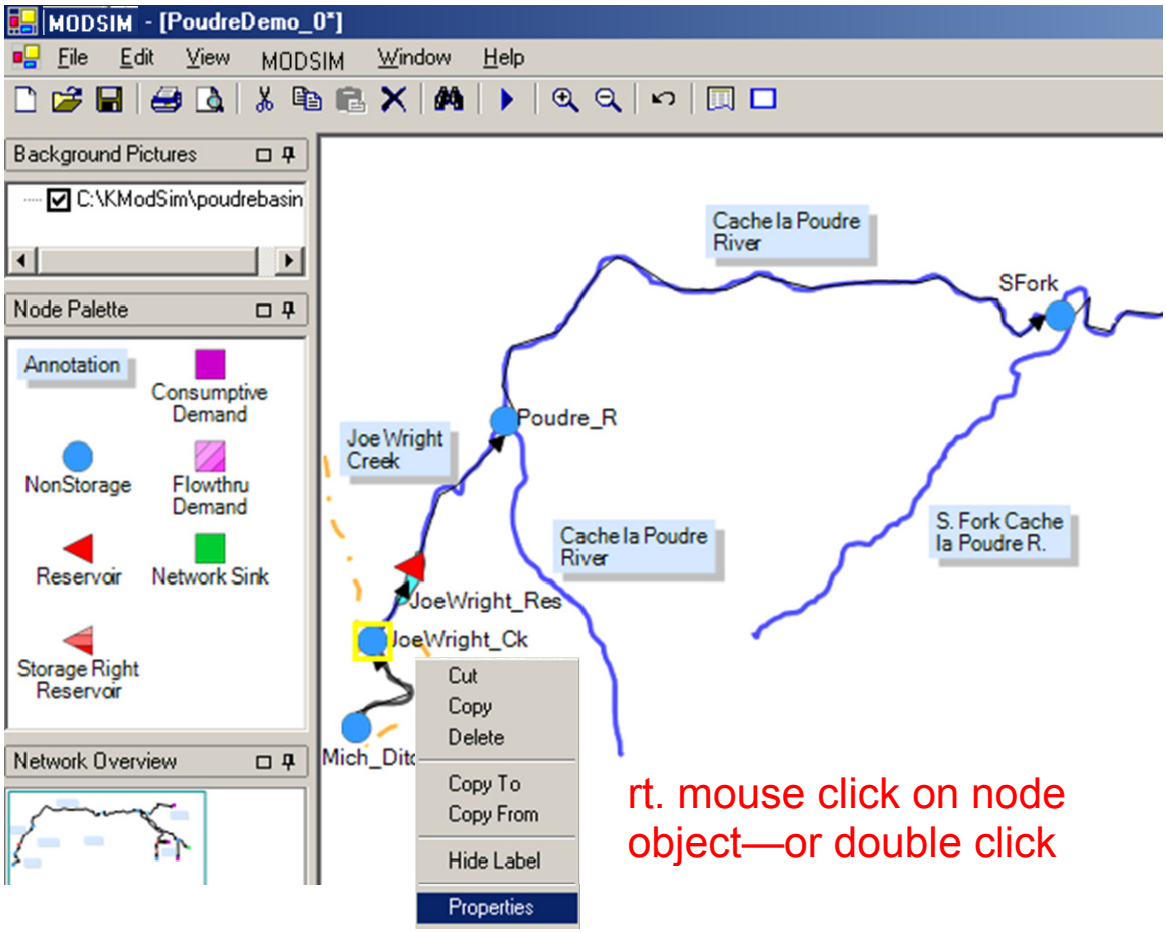
* link capacity restriction due to canal icing



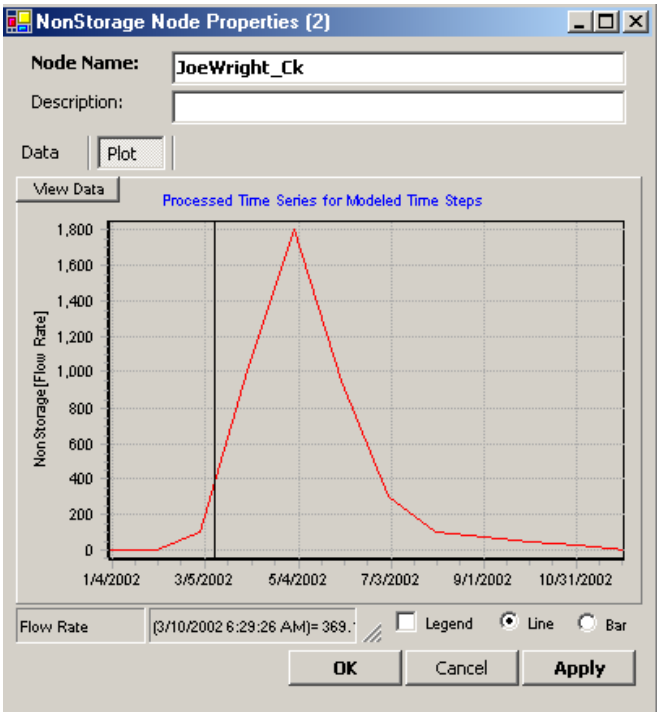
The **Import TimeSeries** form allows time series data for all relevant nodes and links within a particular data category to be loaded automatically into the network.

The **Node Timeseries** dropdown list is activated by first selecting any relevant node for time series data import in the **Node Names** dropdown list (e.g., *JoeWright_Ck*). The Inflows category is then selected from the **Node Timeseries** dropdown list, and then other nodes within the Inflows time series data category can then be selected in the **Node Names** dropdown list. Any node is easily "deselected" by just clicking on that item in the list. Again, names of nodes in the Header row of the database must *exactly* correspond to the names of the nodes in order for the data import to be successful. Selected here are the following inflow nodes (note: **do not** include spaces in any node names): *JoeWright_Ck*, *Mich_Ditch*, *Poudre_R*, and *SFork*. The **Worksheet Name** "Poudre Data" should be selected, with "Column Headers" selected in the **Node/Link Names** dropdown list and "Use Dates Column" selected in the **Start Date Column** dropdown list.

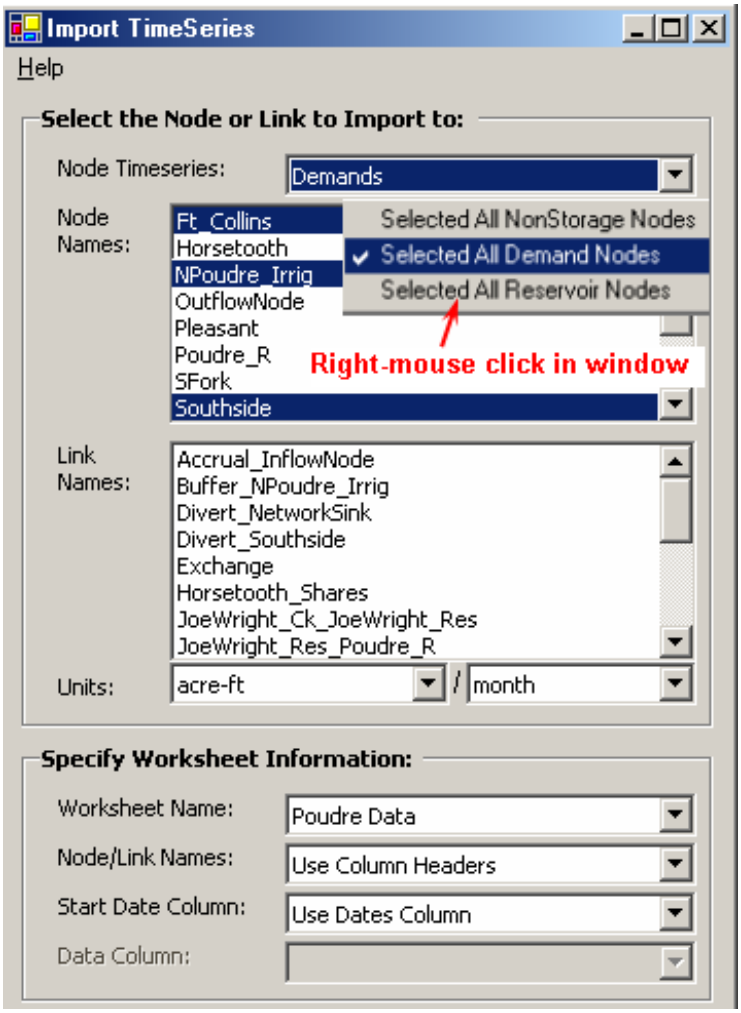
To check the time series data base import, right-mouse click on the *JoeWright_Ck* node and select Properties in the context menu. Alternatively, the **Node Properties** form can be directly displayed by double-clicking on the node. In the **NonStorage Node Properties** form for *JoeWright_Ck*, the user can select **Data** to view a table of the inflow time series data, or select **Plot** to view a graph of the time series data. Viewing the graph is useful for detecting errors or outliers in the data set. The other nonstorage nodes for which inflow time series data were imported can also be similarly viewed either in tabular or graphical form.



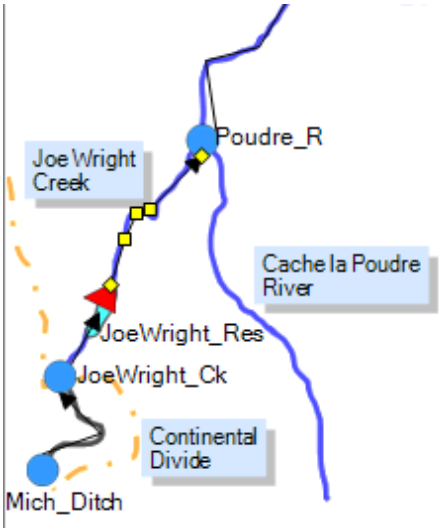
rt. mouse click on node object—or double click

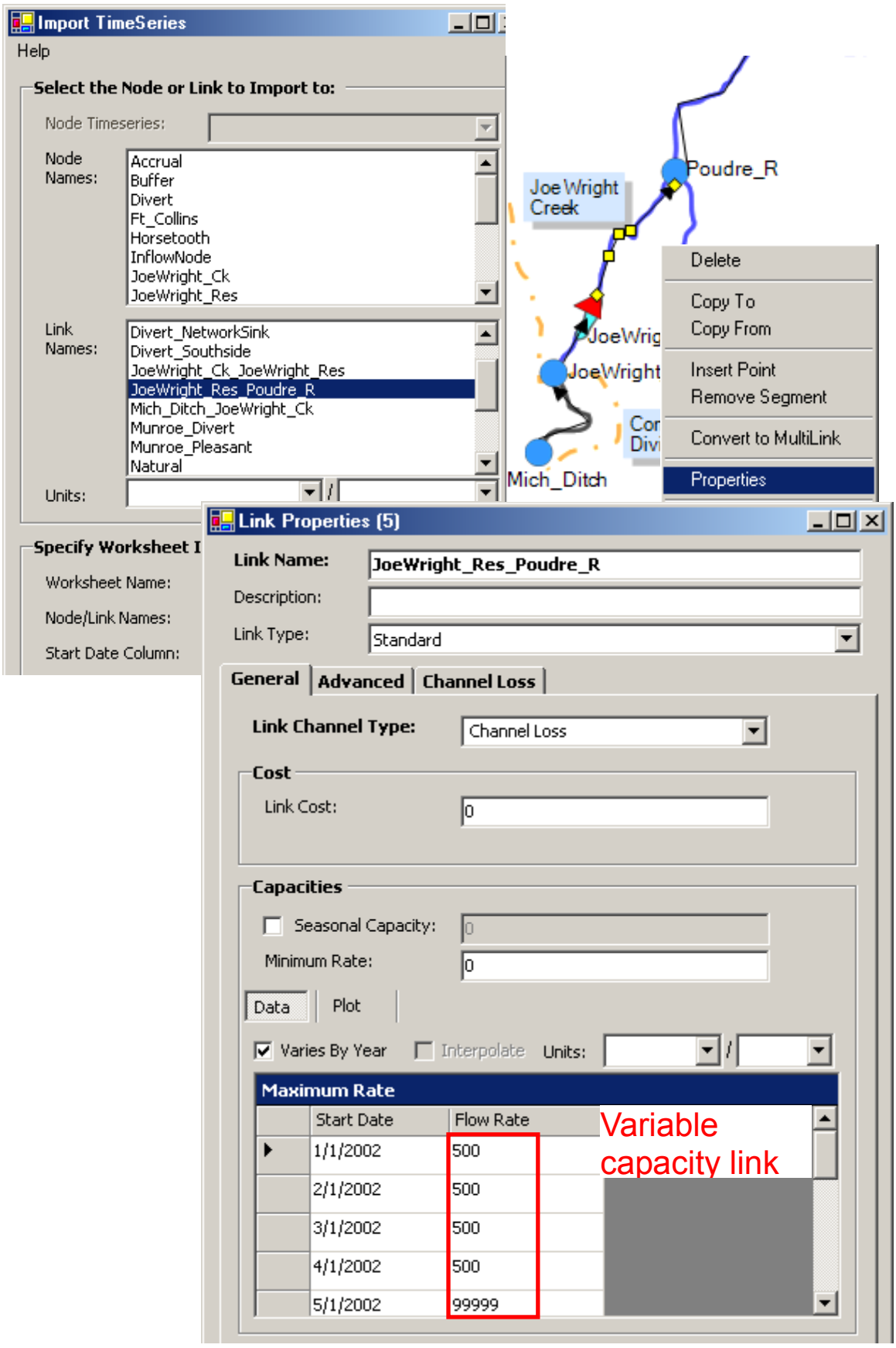


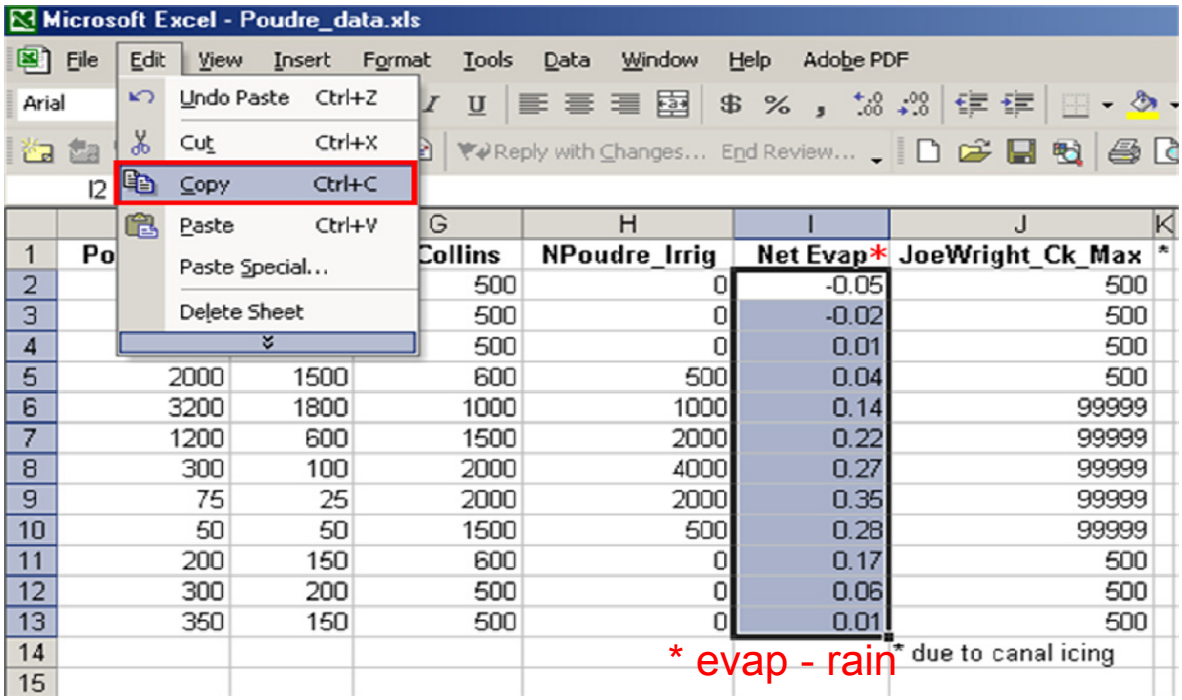
Now, demand time series data can be imported into the network. Again opening the **Import Timeseries** form, selecting any demand node in the **Node Names** dropdown list activates the **Node Timeseries** dropdown list and allows selection of the **Demands** data type. As an alternative to manually selecting each demand node in the **Node Names** dropdown list, right-mouse click within this window opens a context menu which allows selection of the item "Selected All Demand Nodes." Clicking this item results in all demand nodes automatically being selected in the **Node Names** list. However, the *Southside* demand needs to be 'deselected' since it is not included in the Excel database, and must be manually entered in the Node Properties form for the *Southside* demand node, as shown subsequently. Make the same selections as before in the **Specify Worksheet Information** portion of the **Import Timeseries** form as well as the same units, and click OK.



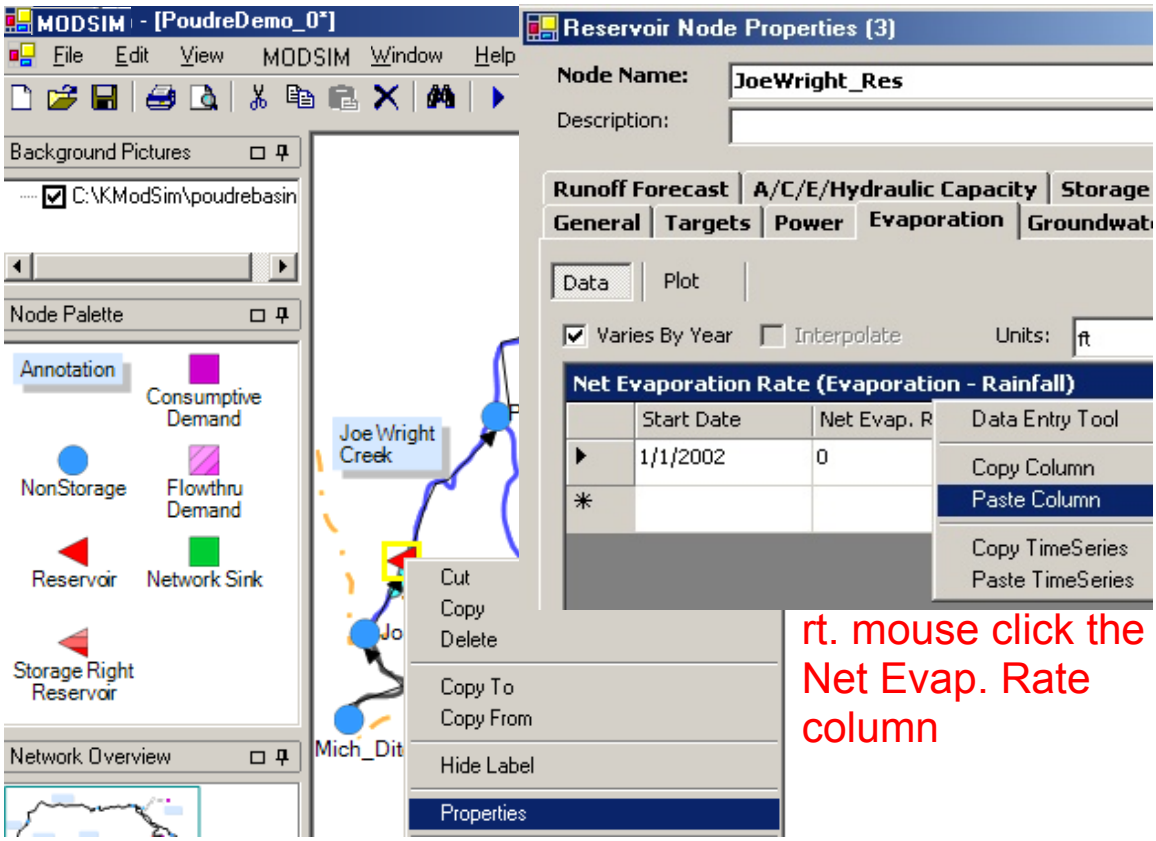
Remaining time series data to be imported include variable capacity data for the link downstream of the proposed location of Joe Wright Reservoir. The reason for specification of this as a variable capacity link is that significant ice buildup occurs during the winter months in this reach since it is located high in the mountains. The capacity of the link is effectively reduced during these months. These time series data are also included in the data base. Again opening the **Import Timeseries** form, the variable capacity link *JoeWright_Res_PoudreR* is selected in the **Link Names** dropdown list for automatic import of the variable capacity time series data.



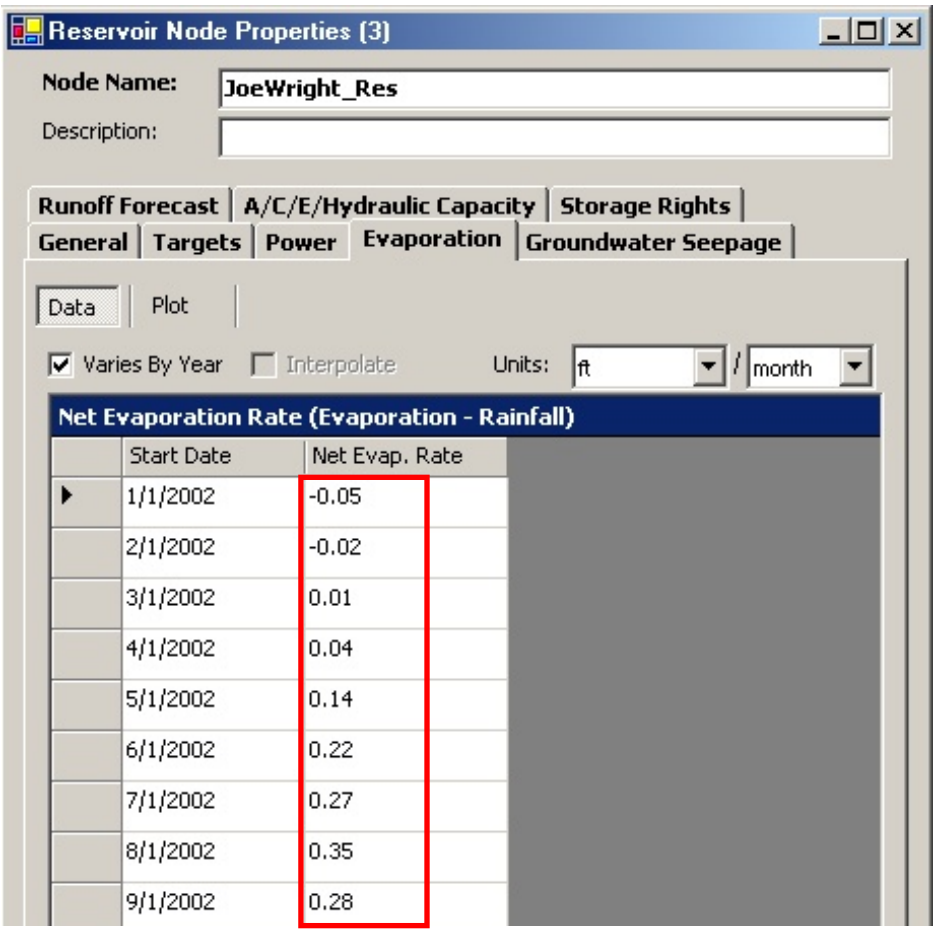

Import TimeSeries
Help
Select the Node or Link to Import to:
Node Timeseries:
Node Names:
Accrual
Buffer
Divert
Ft_Collins
Horsetooth
InflowNode
JoeWright_Ck
JoeWright_Res
Link Names:
Divert_NetworkSink
Divert_Southside
JoeWright_Ck_JoeWright_Res
JoeWright_Res_Poudre_R
Mich_Ditch_JoeWright_Ck
Munroe_Divert
Munroe_Pleasant
Natural
Units:
Specify Worksheet I
Worksheet Name:
Node/Link Names:
Start Date Column:
Joe Wright Creek
Poudre_R
JoeWrig
JoeWright
Mich_Ditch
Delete
Copy To
Copy From
Insert Point
Remove Segment
Convert to MultiLink
Properties
Link Properties (5)
Link Name:
JoeWright_Res_Poudre_R
Description:
Link Type:
Standard
General
Advanced
Channel Loss
Link Channel Type:
Channel Loss
Cost
Link Cost:
0
Capacities
Seasonal Capacity:
Minimum Rate:
0
Data
Plot
Varies By Year
Interpolate
Units:
Maximum Rate
Start Date
Flow Rate
1/1/2002
500
2/1/2002
500
3/1/2002
500
4/1/2002
500
5/1/2002
99999
Variable capacity link



Another way of entering time series data is through copy and paste operations. Net evaporation rate time series data (i.e., evaporation rates minus rainfall rates) are selected under the Net_Evap heading in *Poudre_data.xls* and copied to the Windows Desktop.



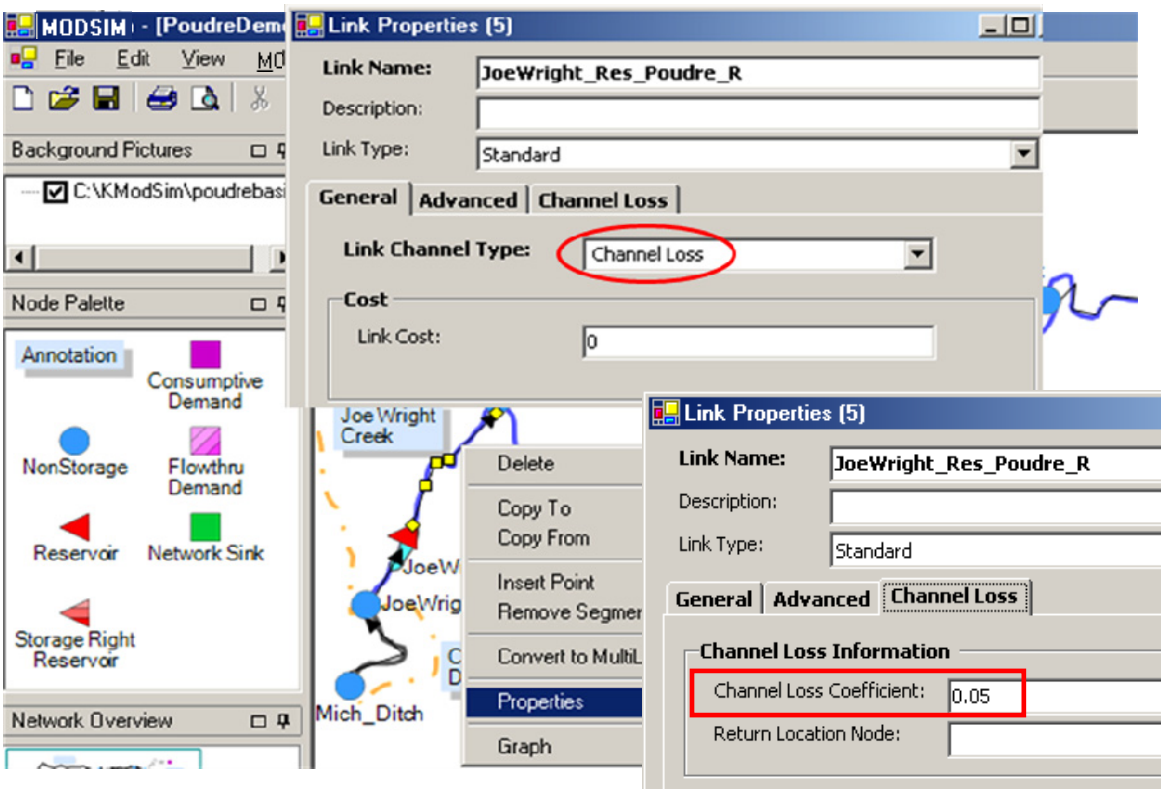
Clicking **Paste Column** results in the net evaporation rate data populating the **Reservoir Node Properties** form.



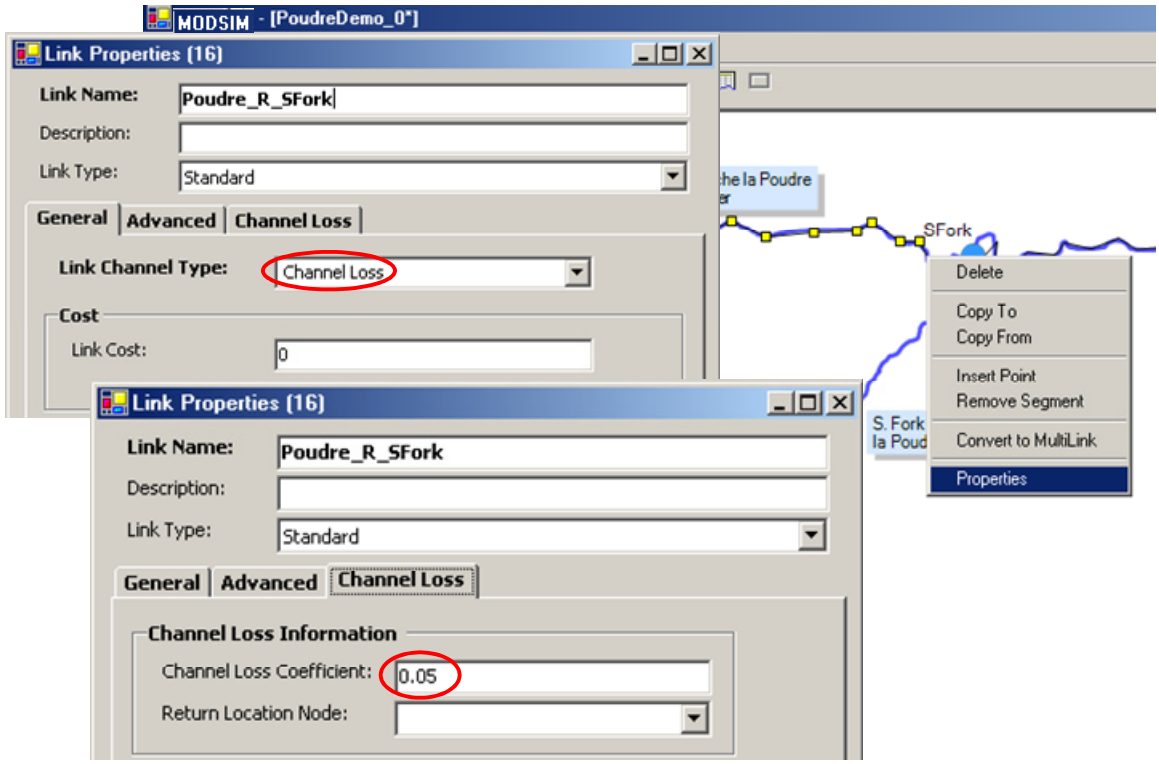
The following network data are also included in *Poudre_data.xls*:

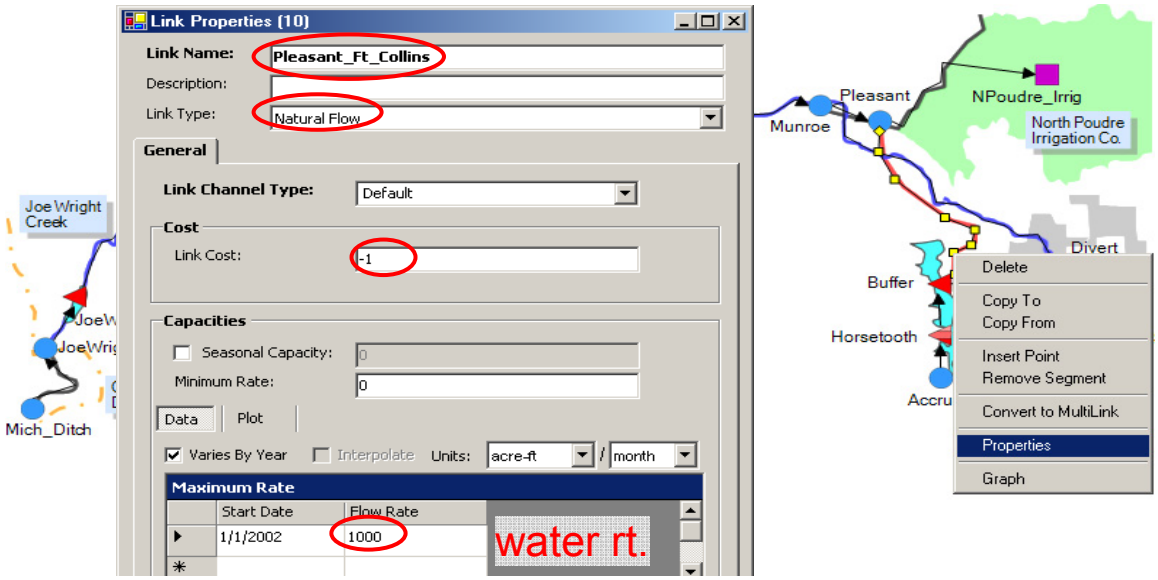
Channel loss*	Max. Cap Pleasant Valley Pipeline	JoeWright_Res. Area	JoeWright_Res. Capacity	North Poudre Irrigation Co. Horsetooth shares	North Poudre Irrigation Co. Water Rights	Ft_Collins Water Rights
0.05	1000	0	0	5000*	1000 AF/mo	4000 AF/mo*
* [downstream of Joe Wright Res to Munroe Canal]		100 200 300	2000 4000 6000	* specified as accrual to Horsetooth		* only if avail.

Opening the **Link Properties** form for the link *JoeWright_Res_PoudreR* allows specification as a *Channel Loss* link, which creates a new **Channel Loss** tab where the **Channel Loss Coefficient** (i.e., the fraction of flow entering the top of the link that is lost due to seepage, evaporation, etc.) can be entered.

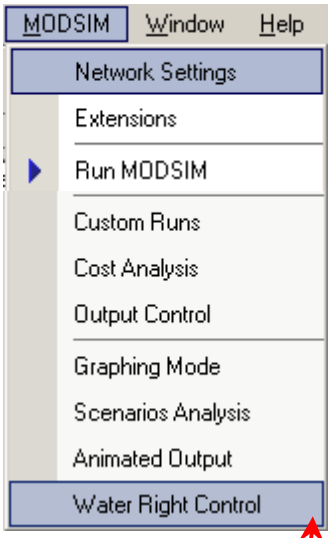


Similarly, the link *PoudreR_SFork* is designated as a *Channel Loss* link, with the same *Channel Loss Coefficient* entered.

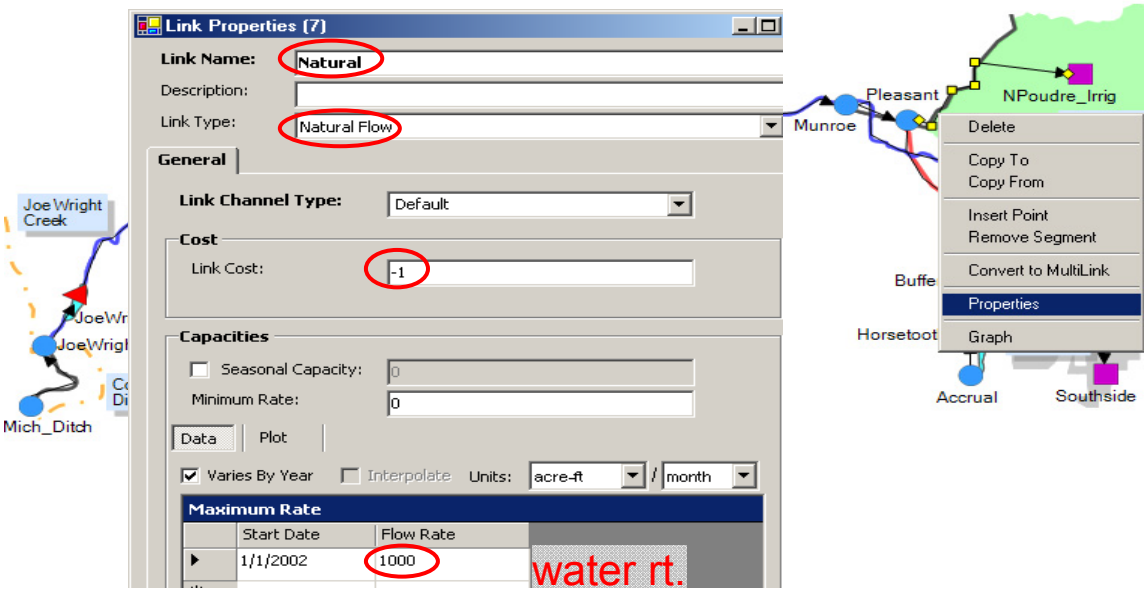




The natural flow rights for the City of Ft. Collins are specified by assigning these rights as the **Maximum Rate** for the link assigned the name *Pleasant_Ft_Collins*. Default link names can usually be used, but the link is renamed in this case. Once the **Link Type** is specified as *Natural Flow*, the interface then automatically assigns a **Link Cost** of -1. Negative costs assigned to links conveying flows to demand nodes identifies them as **Natural Flow** links in MODSIM. In this case, assigning a small negative cost does not impede with the water rights priority ranking designated for the demand node. This approach is preferred if there is only one natural flow right, as in this Tutorial, whereas multiple natural flow rights generally require use of the **Water Right Control** tool which is used for assigning negative costs to several water right links corresponding to the water right decree dates. For multiple natural flow links, the demand node is assigned a neutral priority, with the negative costs on the natural flow links performing the function of correctly prioritizing flow deliveries

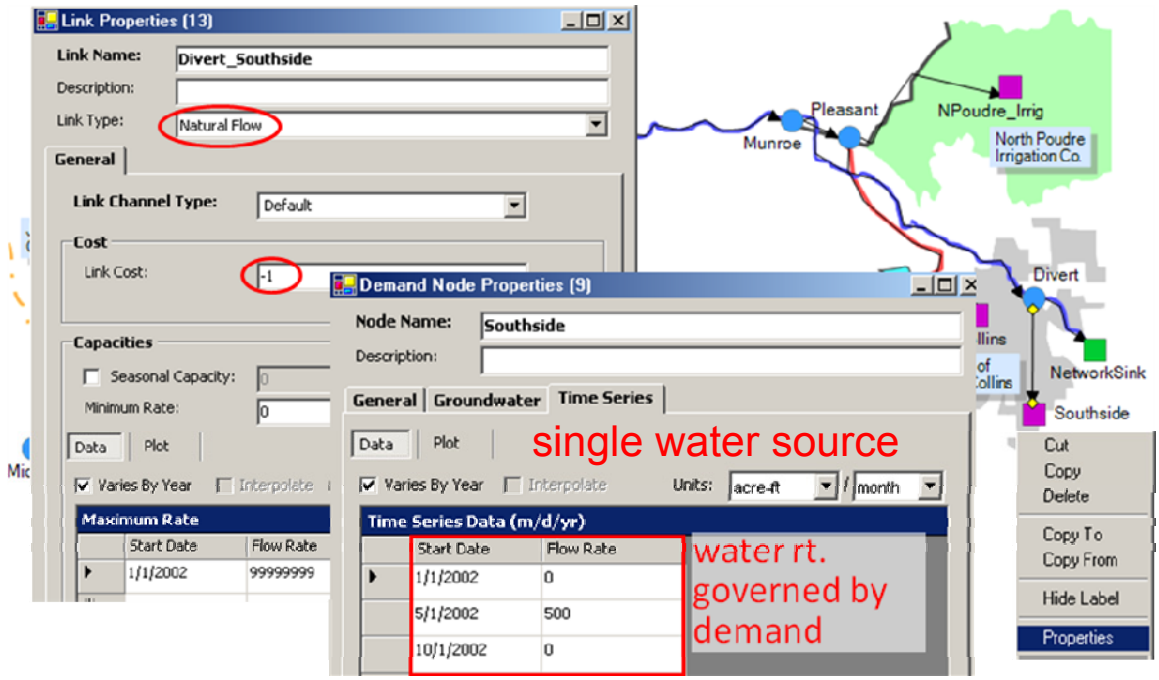


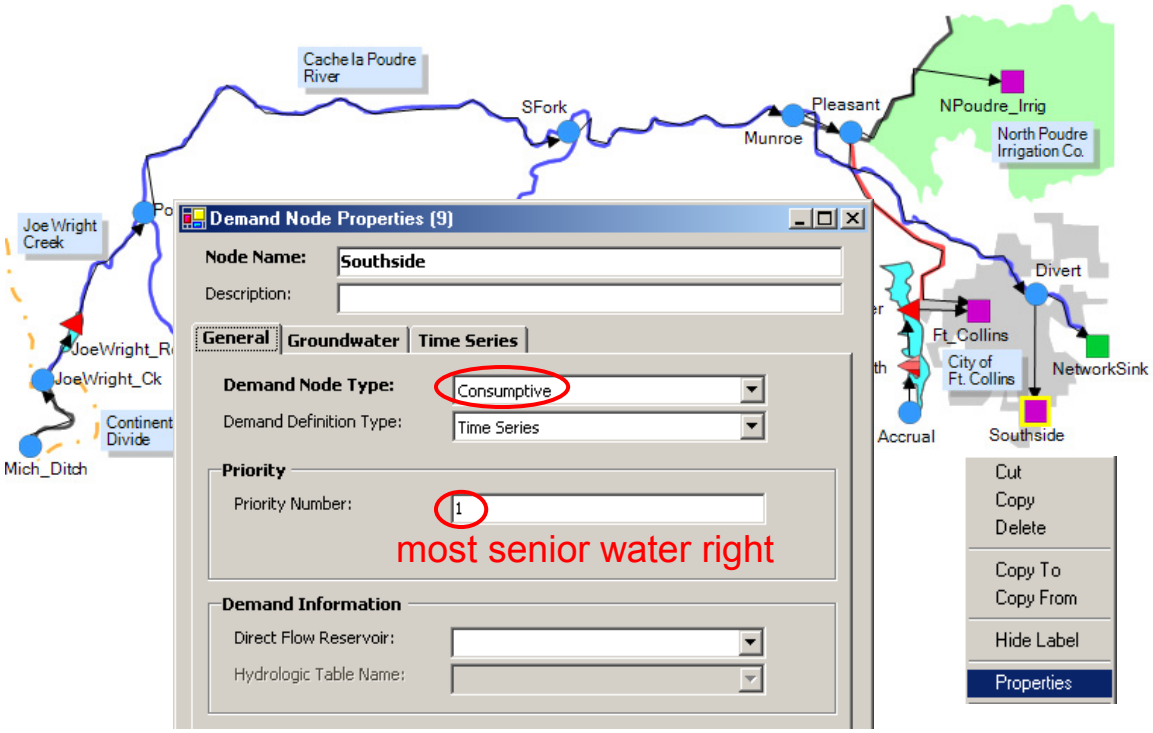
Similarly, natural flow rights for the North Poudre Irrigation District are specified by assigning these rights as the **Maximum Rate** for the *Natural* link conveying flow to the demand. Again, in most cases, the default link names assigned by MODSIM (based on the names of the starting and ending nodes for the link) can be used, whereas in this case, the name *Natural* is assigned by the user to this link to distinguish it from storage ownership links conveying flow to the demand, as discussed subsequently. As before, once the **Link Type** is specified as *Natural Flow*, the interface then automatically assigns a **Link Cost** of -1.



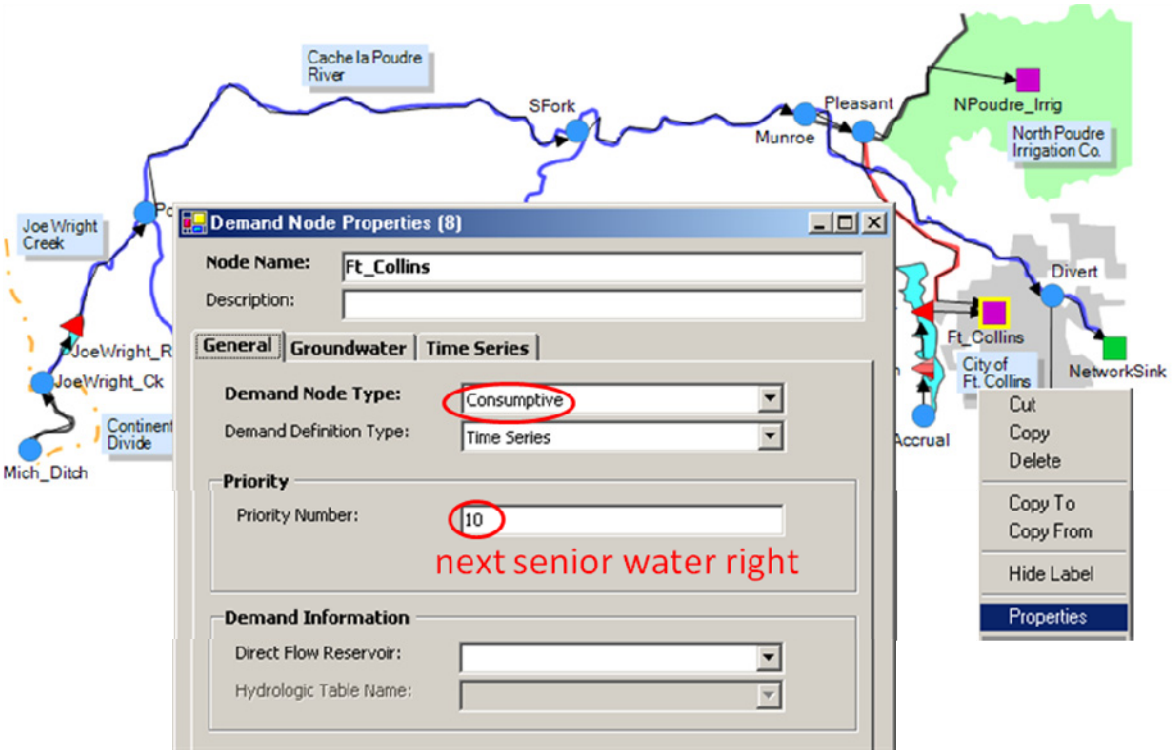
A different approach is taken for assigning water rights to the Southside Ditches. In this case, the natural flow rights are assigned directly to the demand rather than to the link conveying flow to the demand. This is an acceptable approach if the demand has only one water supply source. The link *Divert_Southside* is still specified as a *Natural Flow* link, but with the default **Maximum Rate** for flow in that link retained.

Directly assigning a **Priority Number** of 1 to the *Southside Ditches* gives this demand the most senior priority in the basin. In addition, the *Southside* demands are manually entered, from information in Poudre_data.xls, by entering 500 acre-ft/month for 5/1/2002, and then 0 for 10/1/2002, which MODSIM interprets as 500 acre-ft/month demand for the 5 month period May – September.

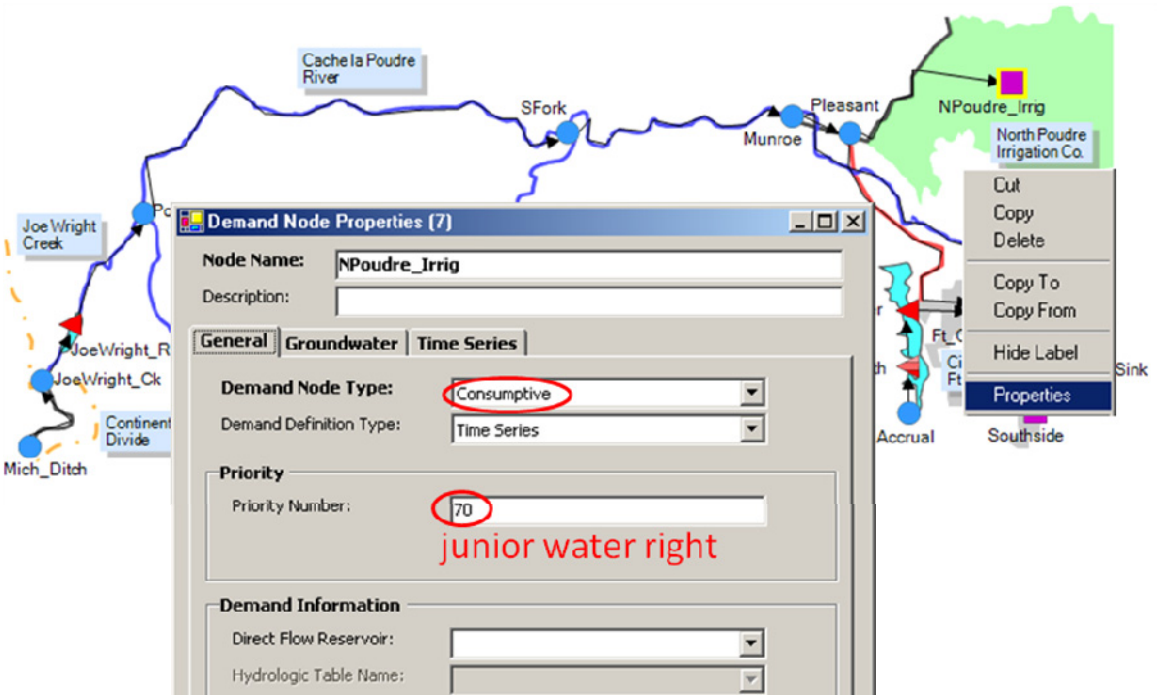




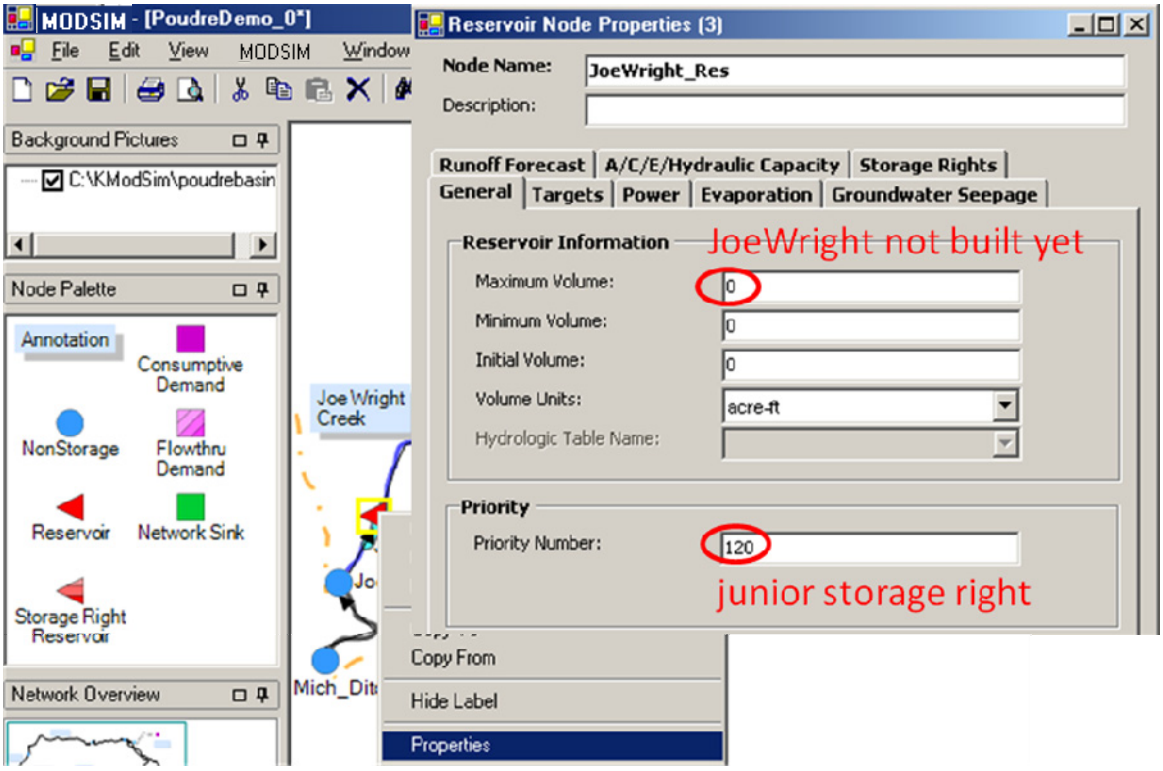
Since the City of Ft. Collins has the second highest priority natural flow right in the basin, a **Priority Number** of 10 is assigned (again, the absolute value of the number selected is inconsequential; rather, it is the relative value compared with other water rights in the basin).



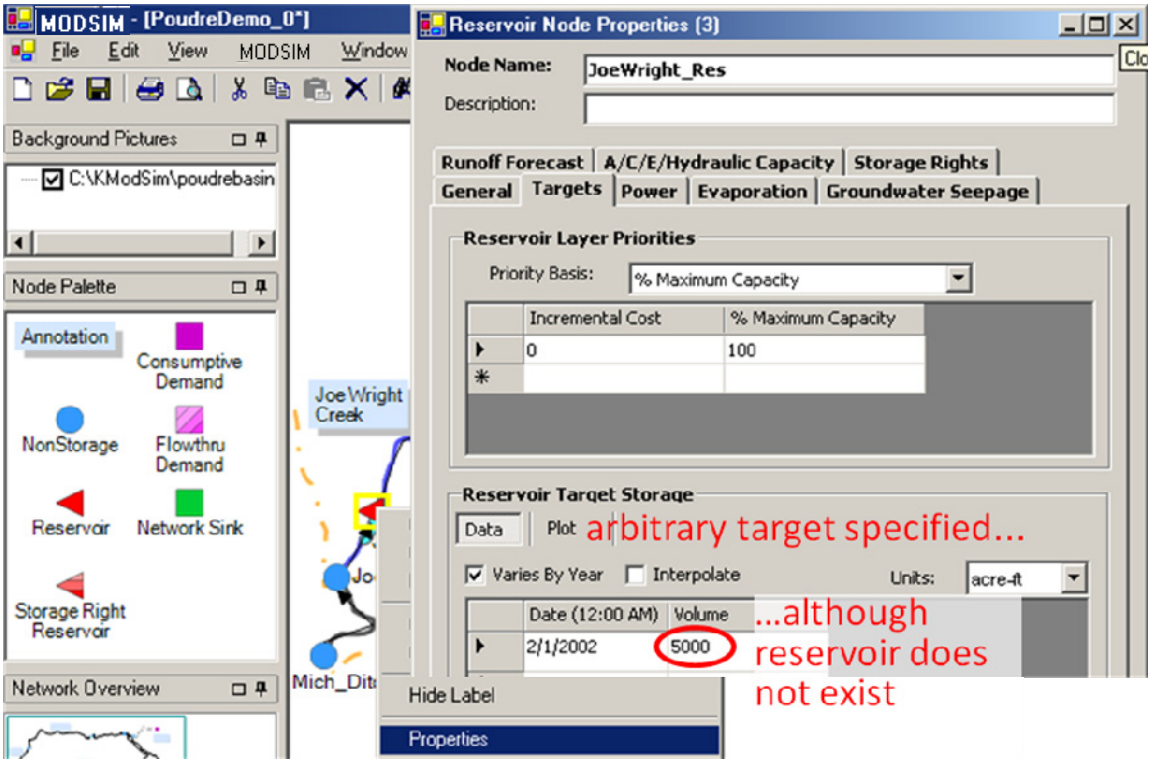
The North Poudre Irrigation Company is assigned a **Priority Number** of 70, reflecting its designation as the most junior water right in the basin.

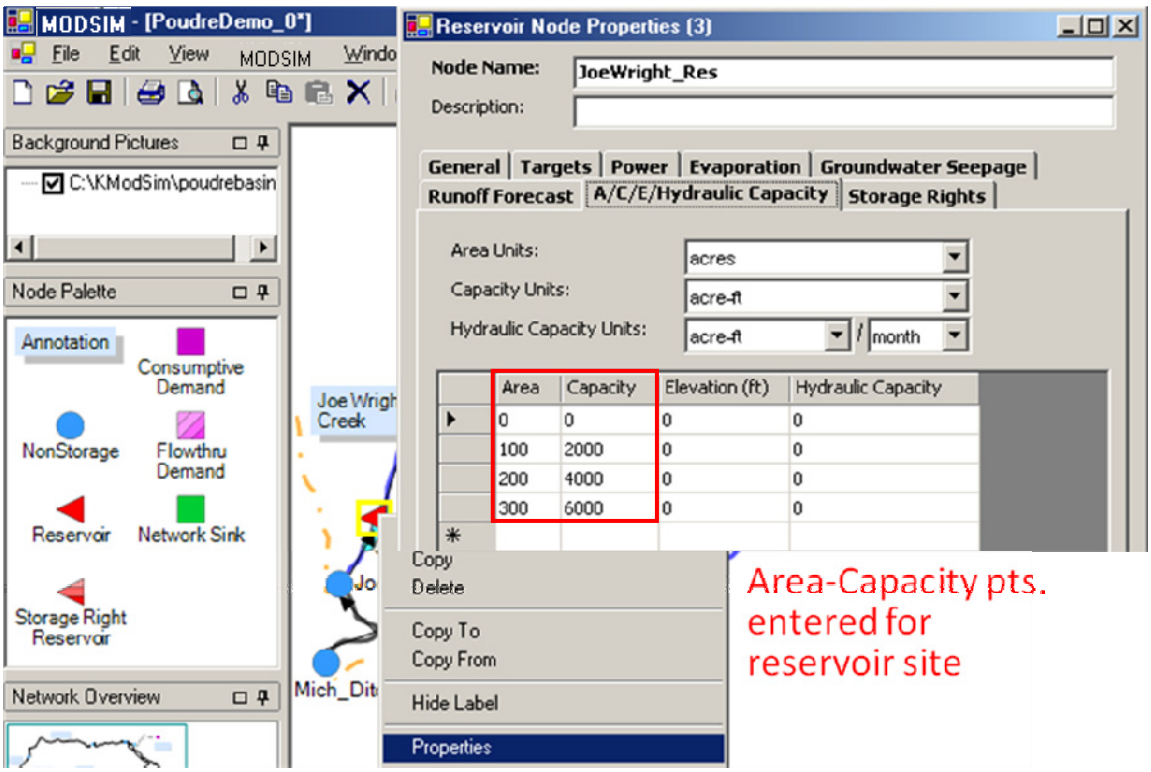


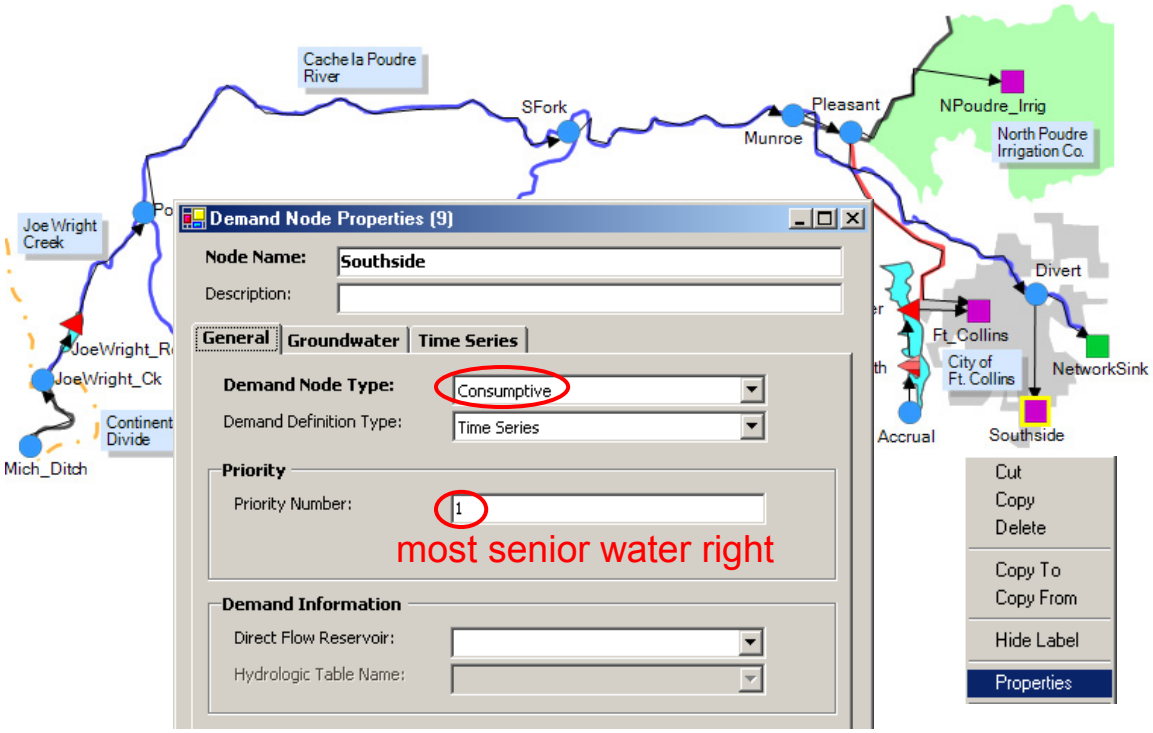
Although it assumed for this Tutorial that Joe Wright Reservoir has not been constructed as yet, reservoir data can still be entered. However, specification of *Maximum Volume* as 0 means that the reservoir is effectively ignored in the network solution. Since this is a new reservoir, it will have the most junior right to accrue water to storage in the basin, hence, the specification of a *Priority Number* of 120.



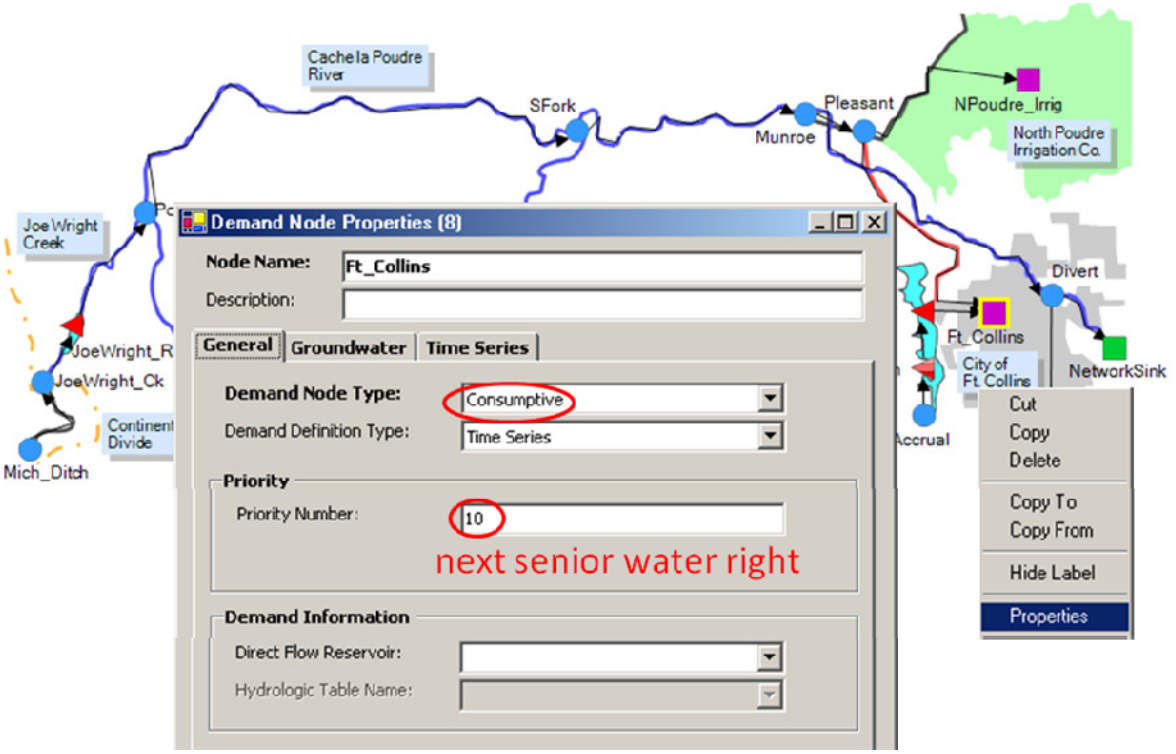
An arbitrary **Reservoir Target Storage** level can be specified, as well as **Area-Capacity** points, even though the reservoir does not yet exist, since these targets and capacity points are automatically truncated to the **Maximum Volume** for the reservoir if the latter is lower.



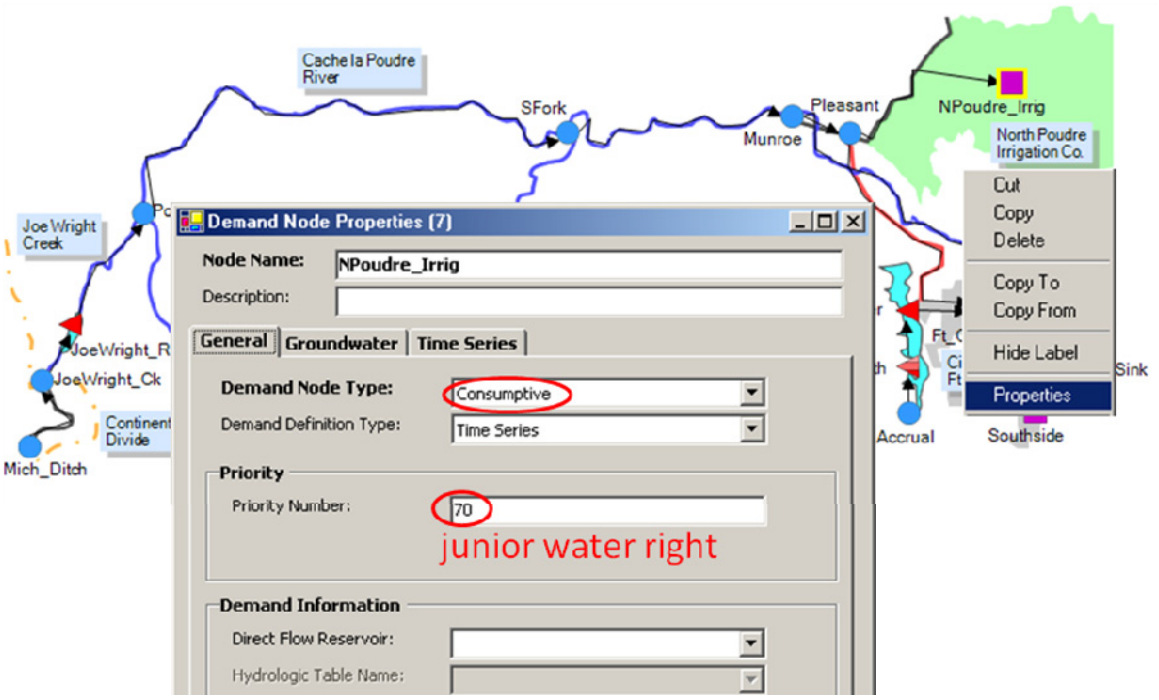




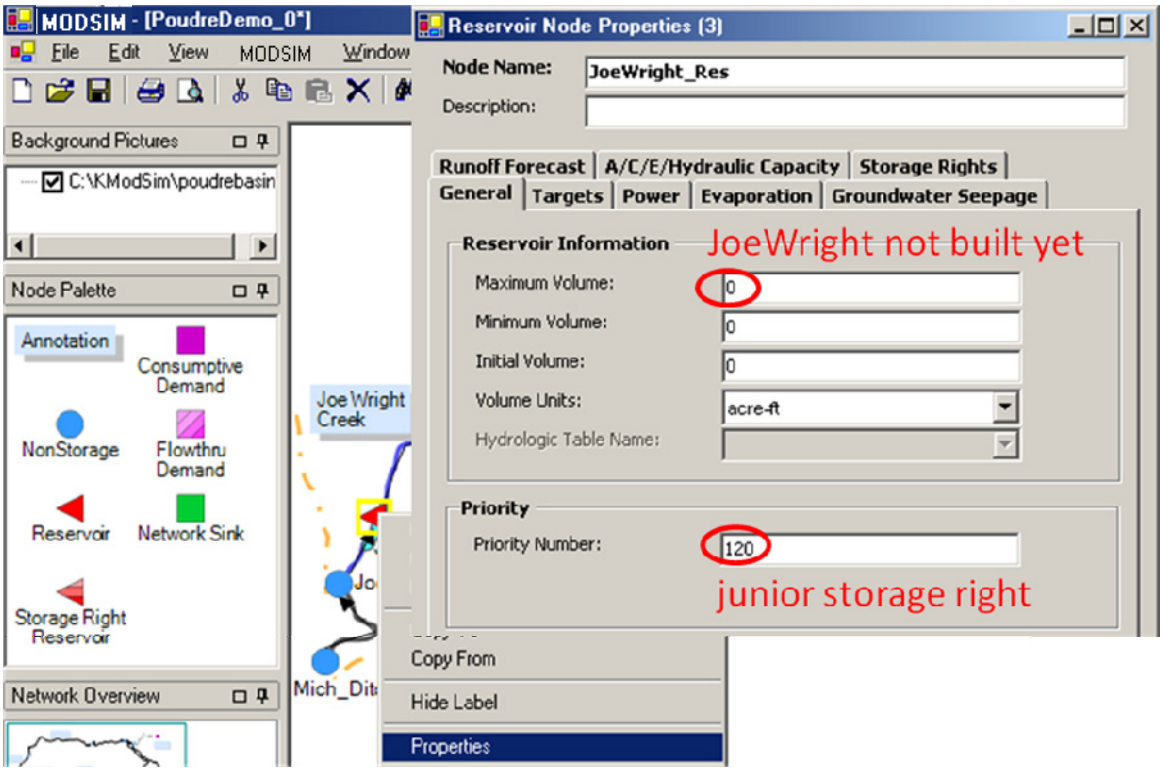
Since the City of Ft. Collins has the second highest priority natural flow right in the basin, a **Priority Number** of 10 is assigned (again, the absolute value of the number selected is inconsequential; rather, it is the relative value compared with other water rights in the basin).



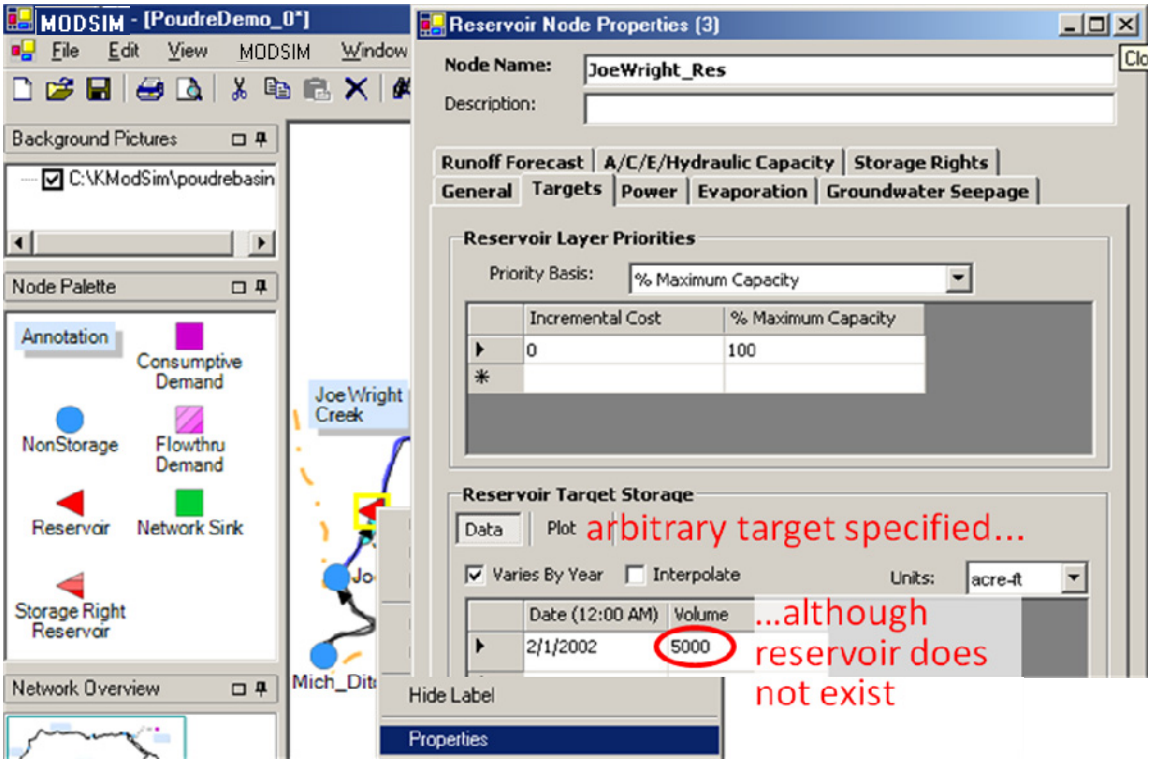
The North Poudre Irrigation Company is assigned a **Priority Number** of 70, reflecting its designation as the most junior water right in the basin.

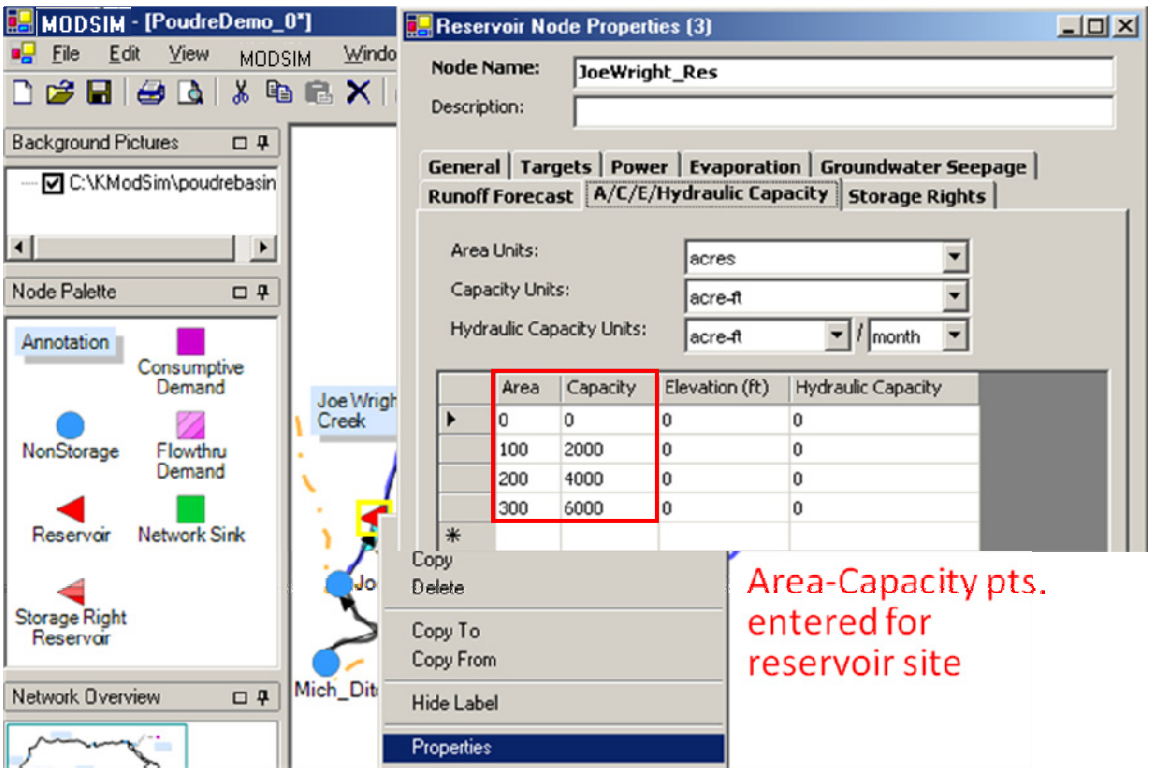


Although it assumed for this Tutorial that Joe Wright Reservoir has not been constructed as yet, reservoir data can still be entered. However, specification of *Maximum Volume* as 0 means that the reservoir is effectively ignored in the network solution. Since this is a new reservoir, it will have the most junior right to accrue water to storage in the basin, hence, the specification of a *Priority Number* of 120.



An arbitrary **Reservoir Target Storage** level can be specified, as well as **Area-Capacity** points, even though the reservoir does not yet exist, since these targets and capacity points are automatically truncated to the **Maximum Volume** for the reservoir if the latter is lower.

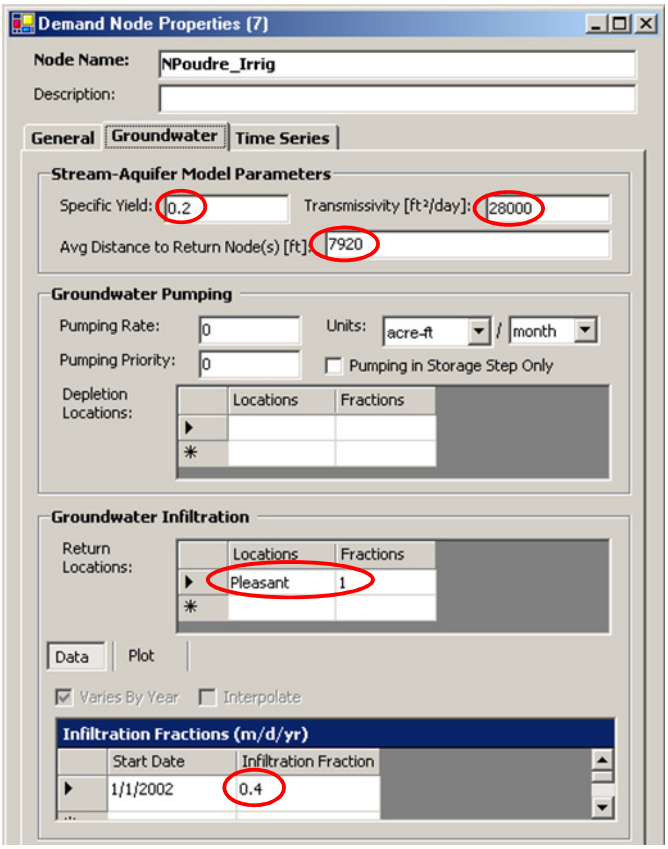


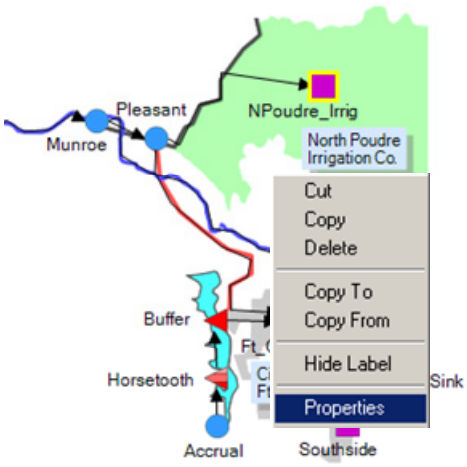


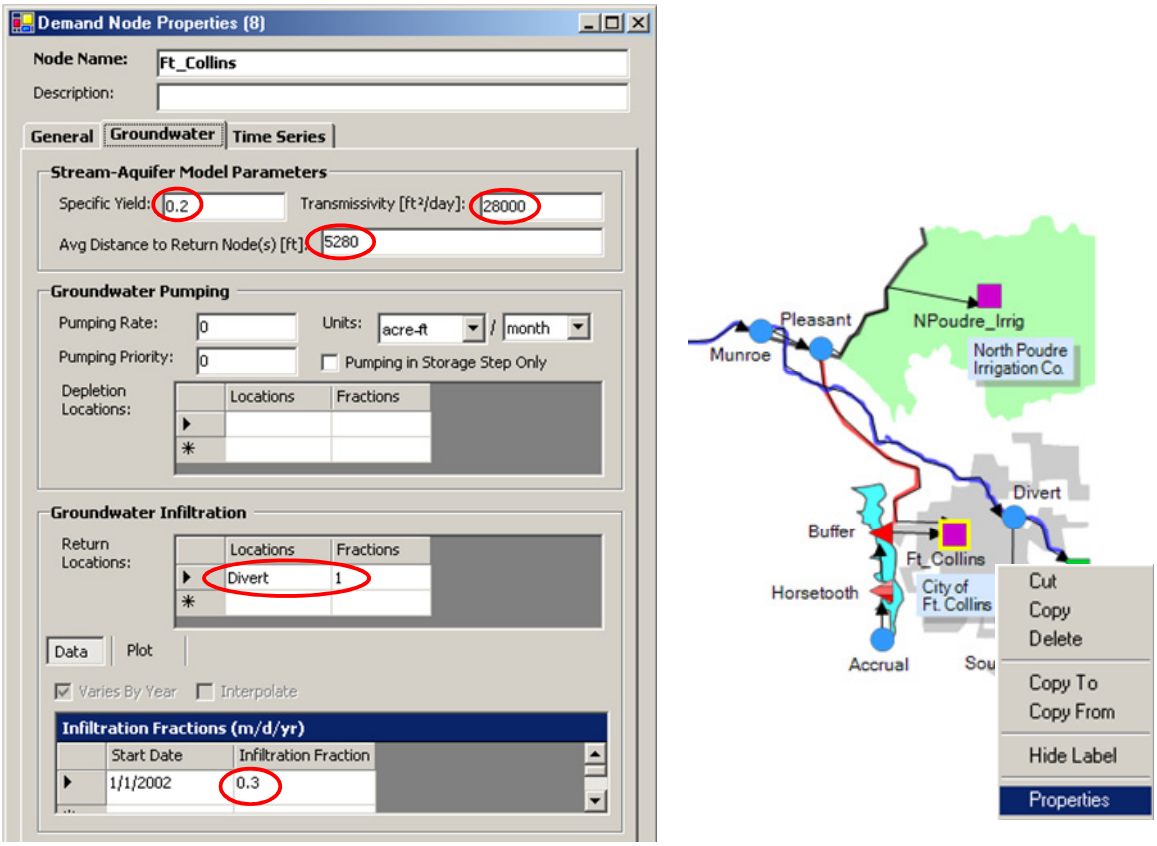
The following data are specified for the stream-aquifer system of the Poudre basin:

* Irrigation return flows from North Poudre Irrig Co. return to node **Pleasant**
  Glover model: Specific Yield: 0.2
  Transmissivity: 28000 ft^2/day
  Avg. distance to Return Node: 7920 ft.
  Infiltration fraction: 0.4
* Depletions from groundwater pumping also occur at node **Pleasant**--same groundwater parameters are used for calculating depletions
* Return flows from Ft. Collins appear at node **Divert**
  Glover model: Specific Yield: 0.2
  Transmissivity: 28000 ft^2/day
  Avg. distance to Return Node: 5280 ft.
  Infiltration fraction: 0.3

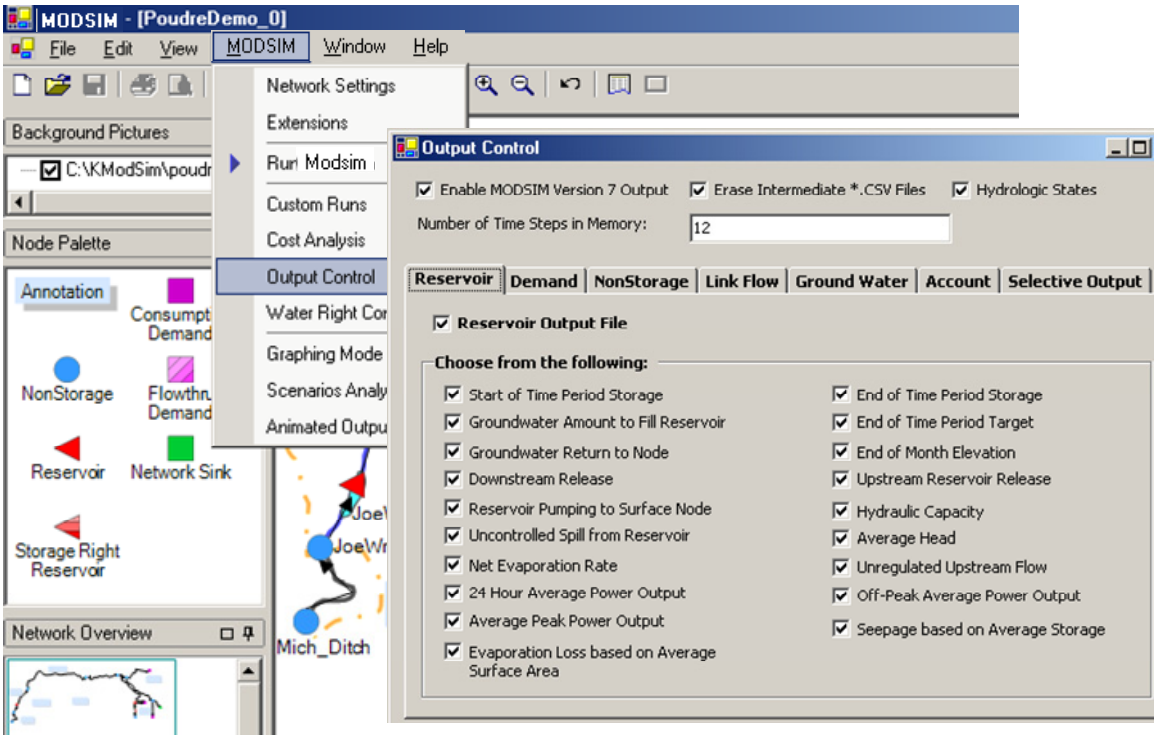
These data are entered into the **Groundwater** tab of the **Demand Node Properties** forms for NPoudre_Irrig and Ft_Collins for calculation of return flows based on the Glover equation.



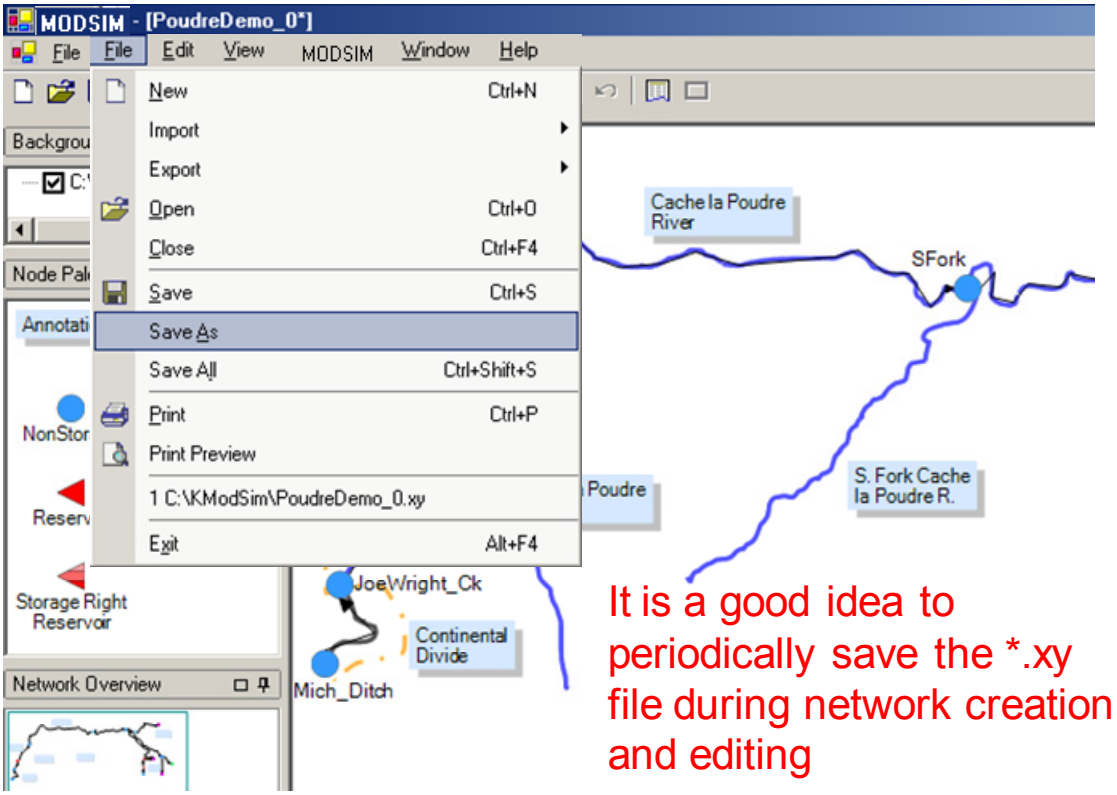




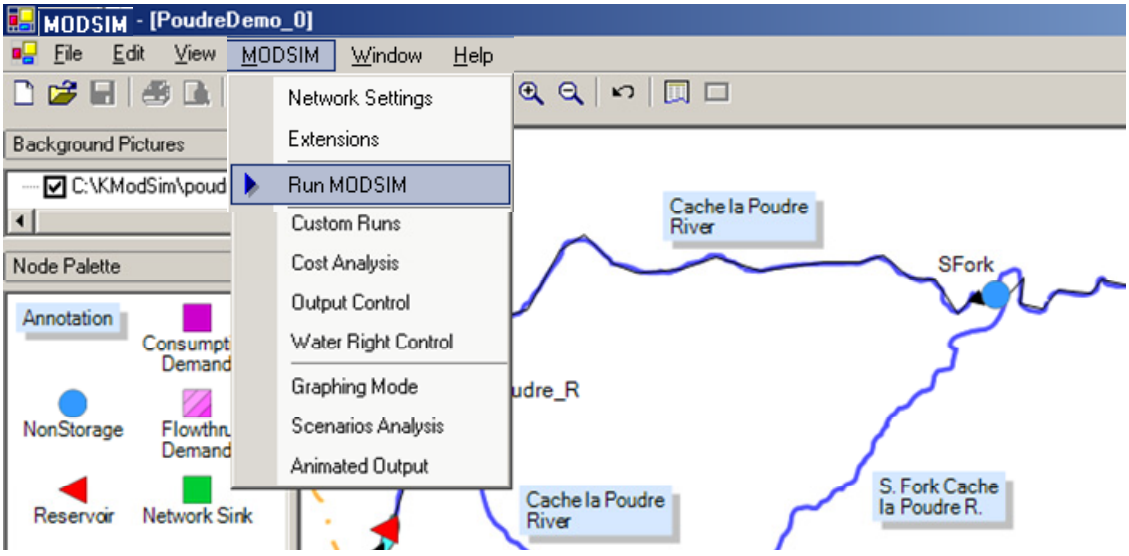
For this base run, **Output Control** can be accessed to select nodes and links to be included in the simulation results, as well as selection of output categories. By default, all nodes and links are selected and all output categories.



This base run file for the Tutorial should be saved as **PoudreDemo_0.xy**. During network creation, it is generally a good idea to periodically save the current network under development to avoid losing data if the program crashes due to incorrect data entry.

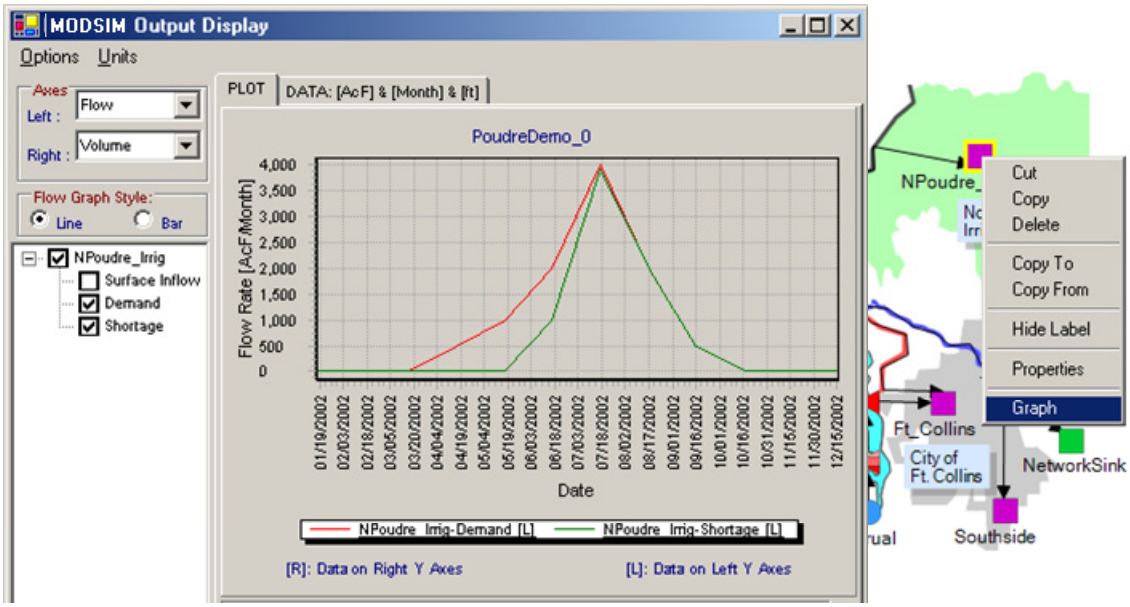


The base run network can now be executed from the main menu or by clicking the "Run" icon  .

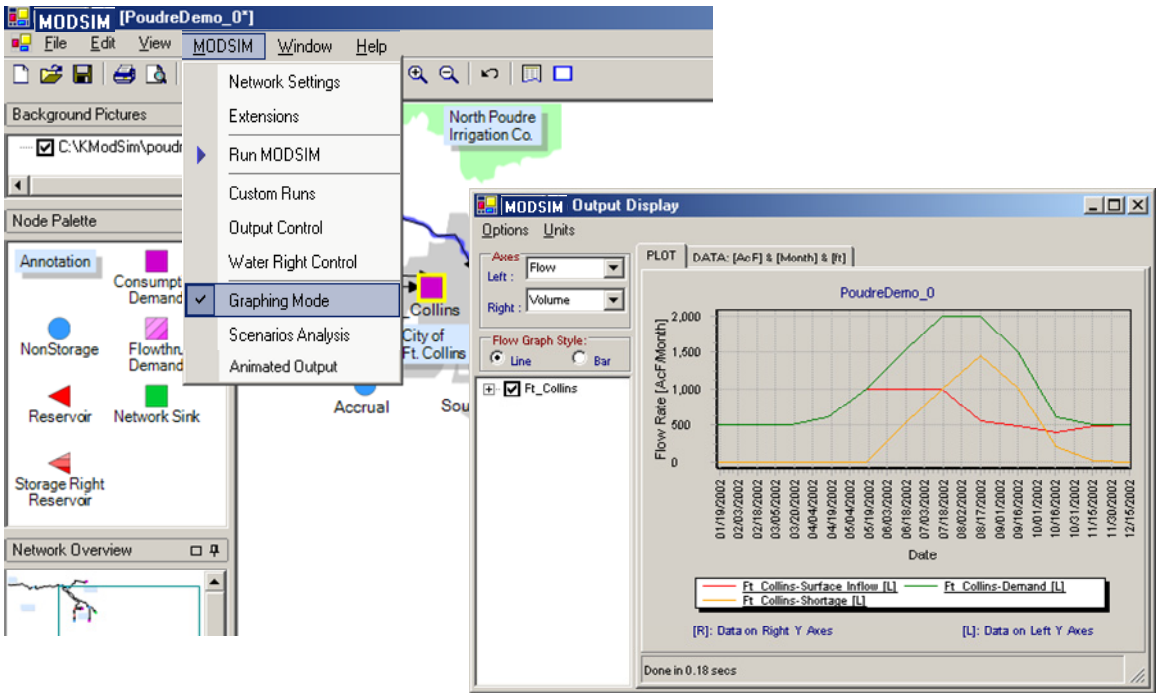


…or click the Run button…

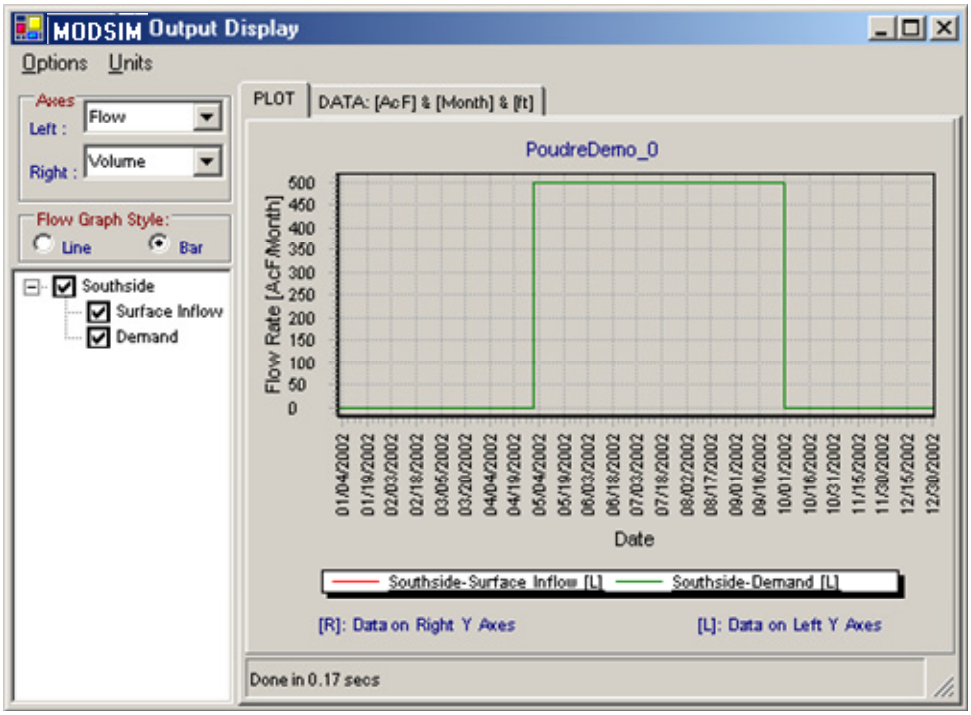
After successful run, a new **Graph** item added to the context menu for any network object.



Alternatively, **MODSIM > Graphing Mode** can be selected which produces graphical plots immediately with left mouse click on any network object.

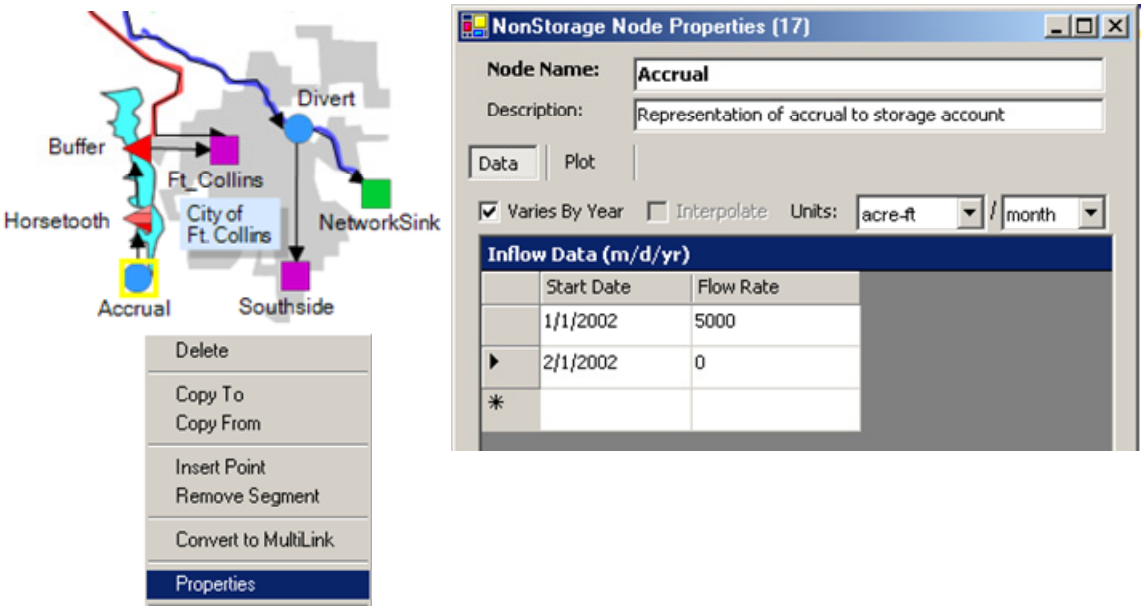


Results of the base run reveal significant shortages to the City of Ft. Collins and the North Poudre Irrigation Co. However, as expected, the senior Southside Ditch demands are fully satisfied in the base run.

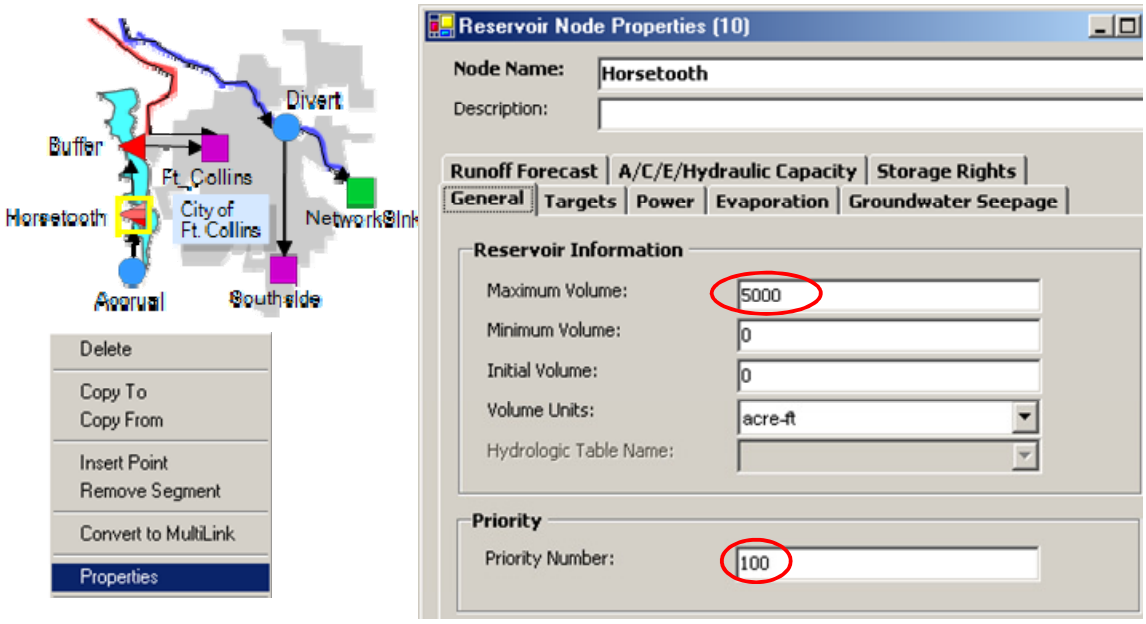


A.5 Storage Accounts and Exchanges

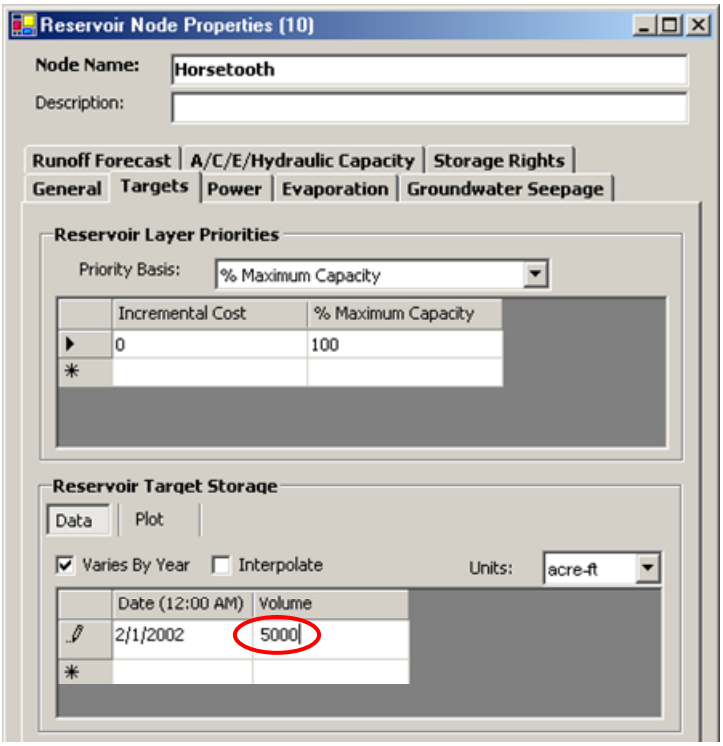
In order to simulate the storage account in Horsetooth Reservoir, the node *Accrual* is created with an inflow corresponding with the amount of the storage account owned by the North Poudre Irrigation Co. In fact, this is not an actual inflow, but rather represents a credit to the storage account



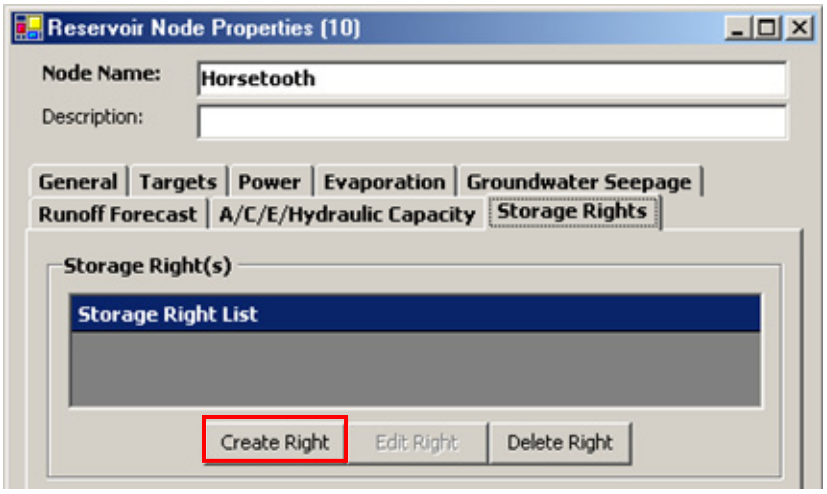
The volume available in the storage account is specified in the **Reservoir Node Properties** form for *Horsetooth*. This does not represent actual physical storage in the reservoir, but an account volume that can be withdrawn by the North Poudre Irrigation Co. much like a bank savings account. In this case, the **Priority Number** assigned relates to the priority of the right to accrue flow to the storage account. In other cases, the right to accrue water to storage is based on a negative cost assigned directly to the accrual link to the reservoir rather than assigning a **Priority Number** to the reservoir. This is required when there are several storage ownerships accruing water to the same reservoir.



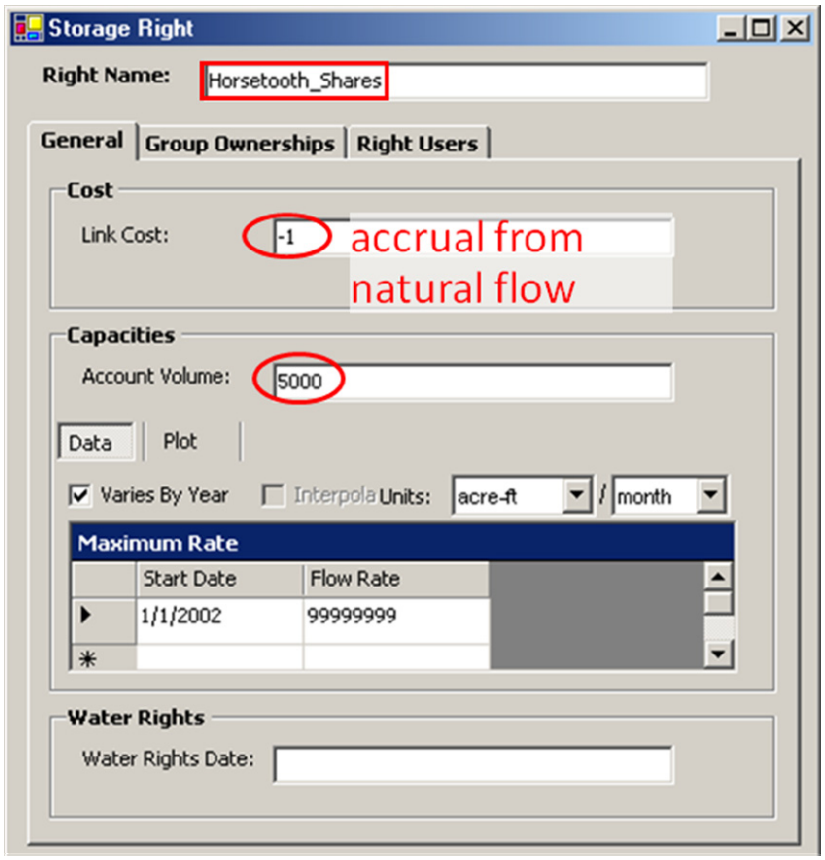
The **Reservoir Target Storage** can be set to the maximum volume in the account in this case.



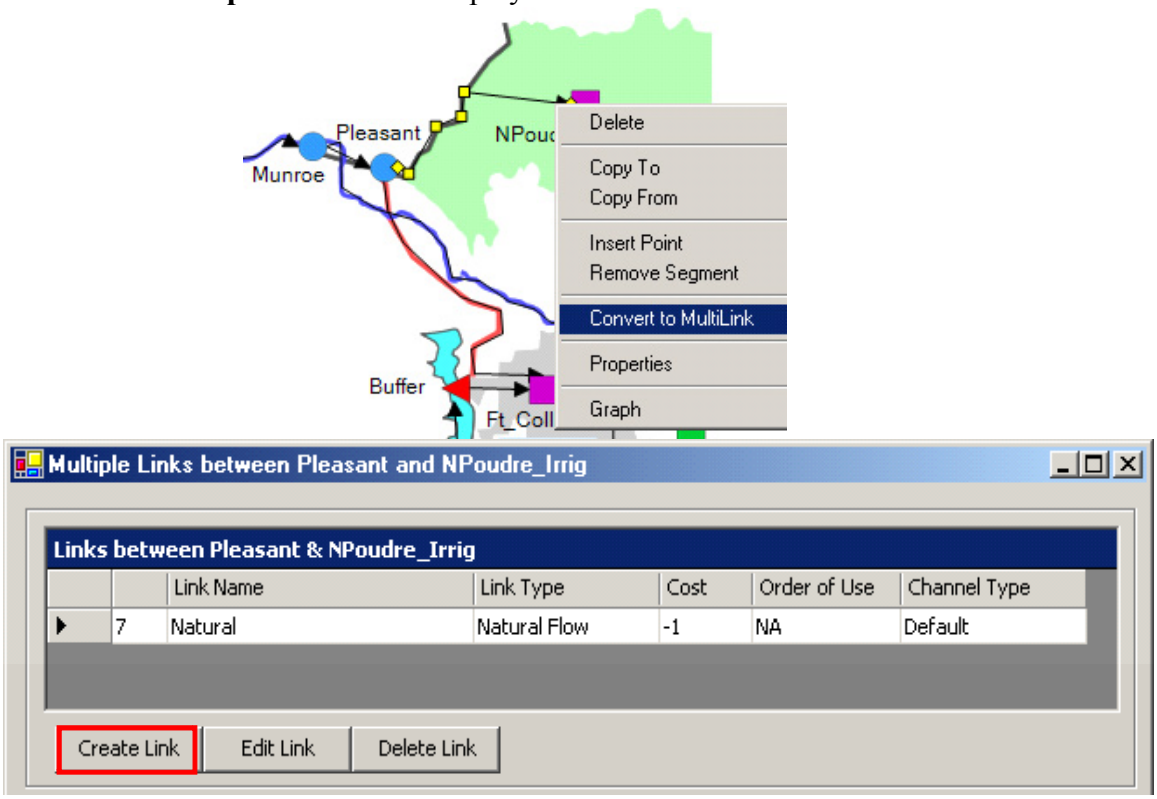
Clicking on the **Storage Rights** tab for Horsetooth allows creation of the storage right, or several storage rights if desired.



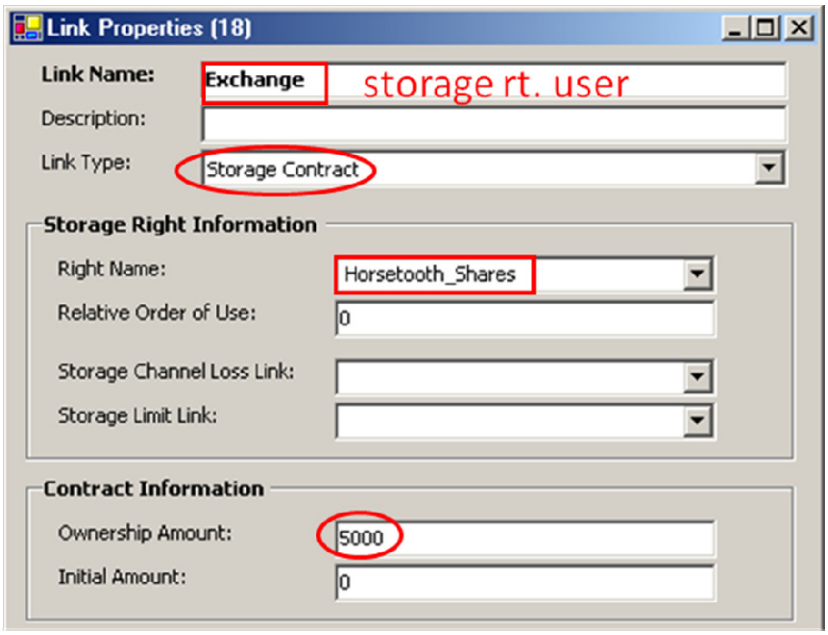
Specify the storage right name as *Horsetooth_Shares*, enter -1 as the **Link Cost** to designate this as a natural flow right to accrue water, and enter the **Account Volume** as 5000.



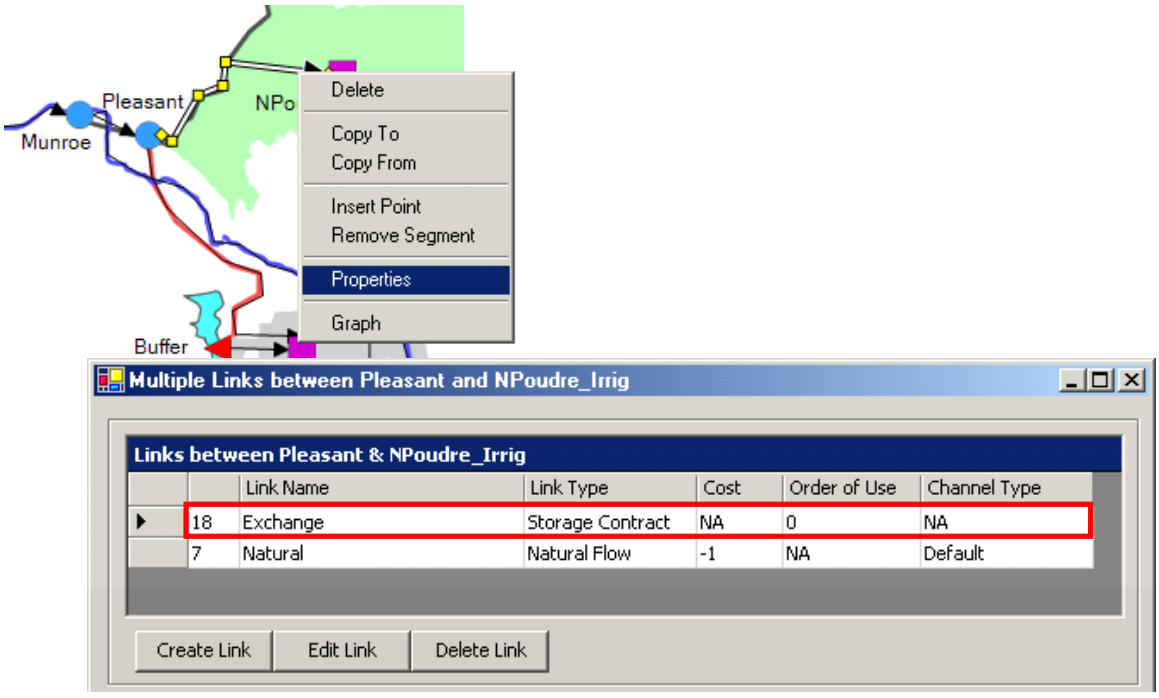
Once the storage right accounts have been specified, the ownership of those accounts must be assigned. A separate storage ownership link must be specified to represent releases delivered to *NPoudre_Irrig*, even though the North Poudre Irrigation Co. is unable to directly receive these releases. This accomplished by right-mouse click on the *Natural* link, clicking **Convert to Multilink** in the context menu, and clicking Create Link in the **Multiple Links** form displayed.



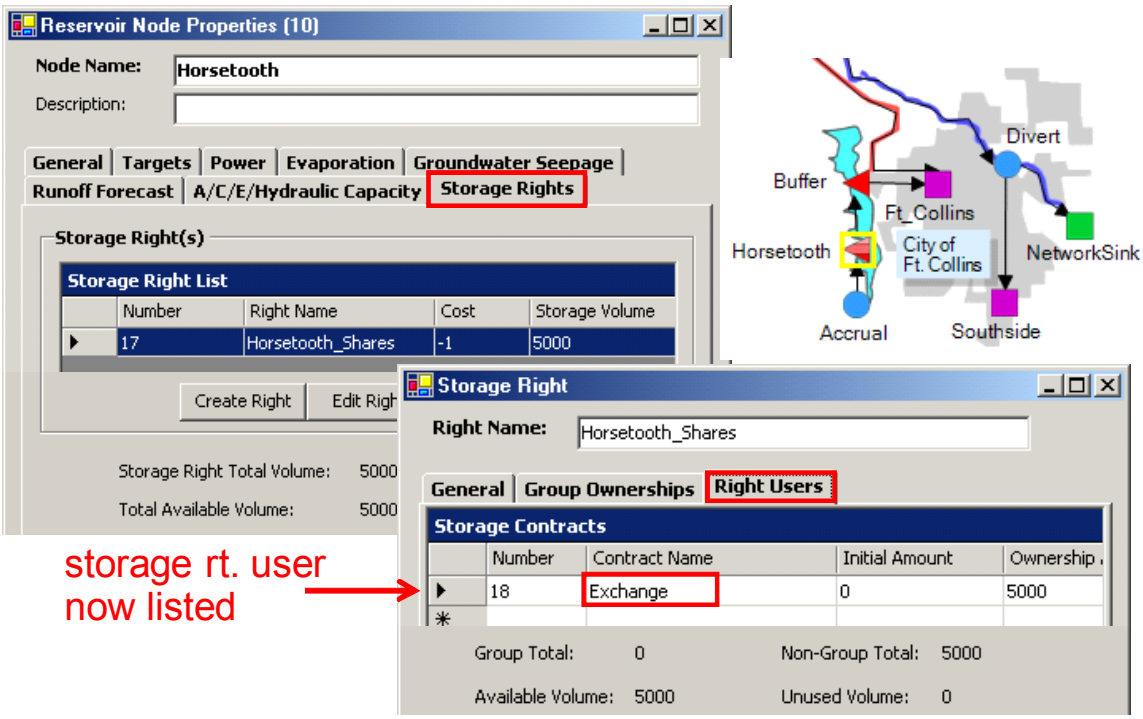
The storage ownership link is assigned the name *Exchange* with the **Link Type** as *Storage Contract*. The Right Name *Horsetooth_Shares* is selected from the dropdown



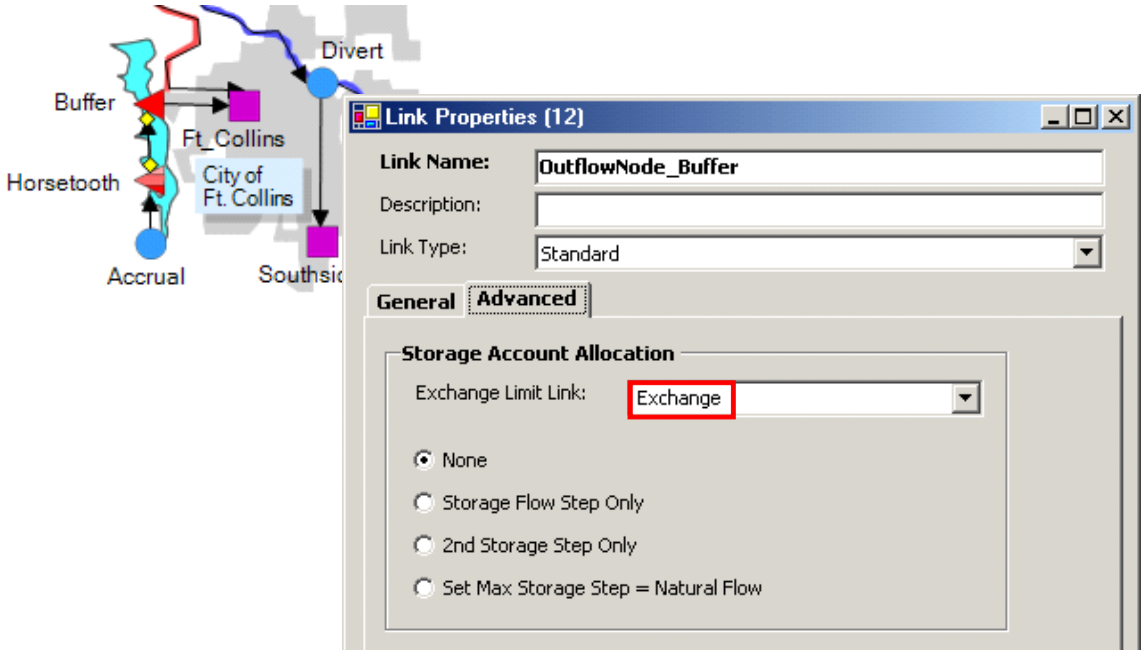
list in the **Storage Right Information** area of the **Link Properties** form, which is storage account that was created in *Horsetooth*. The same ownership amount of 5000 is specified for the ownership link and the new storage ownership or storage account link is now created.



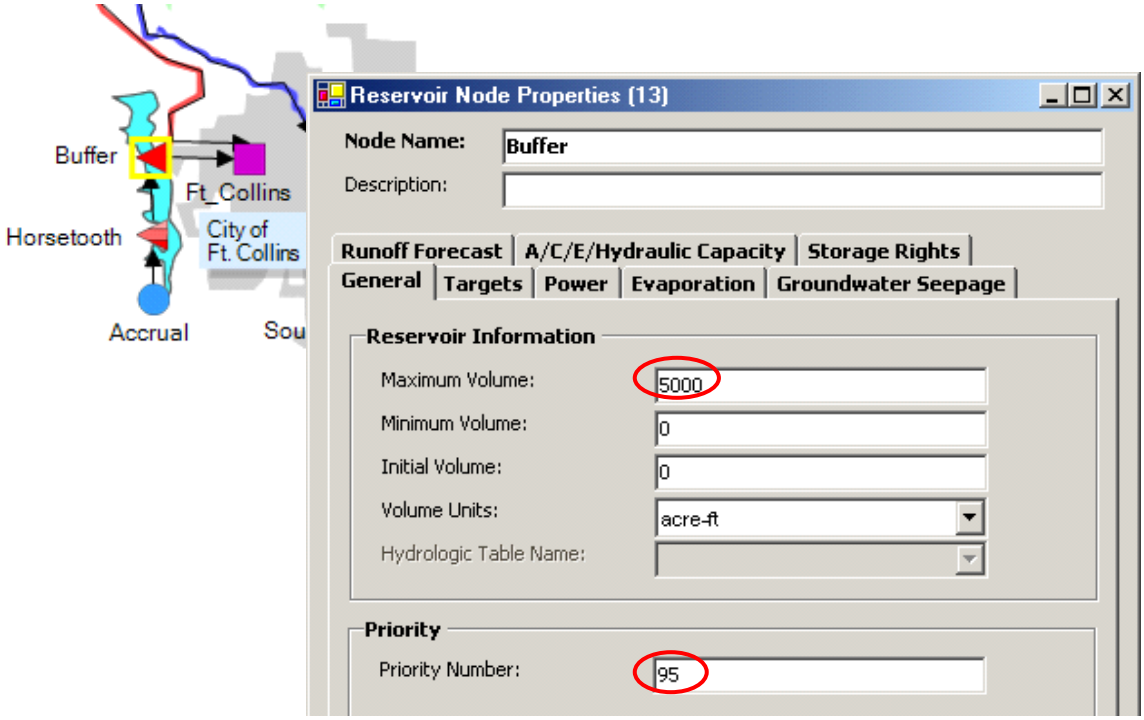
Notice that once again opening the **Reservoir Node Properties** form for *Horsetooth* and clicking the **Storage Rights** tab reveals that the **Contract Name** *Exchange* as the storage ownership account for the North Poudre Irrigation Co. is now listed under the **Right Users** tab in the **Storage Right** form that is displayed.



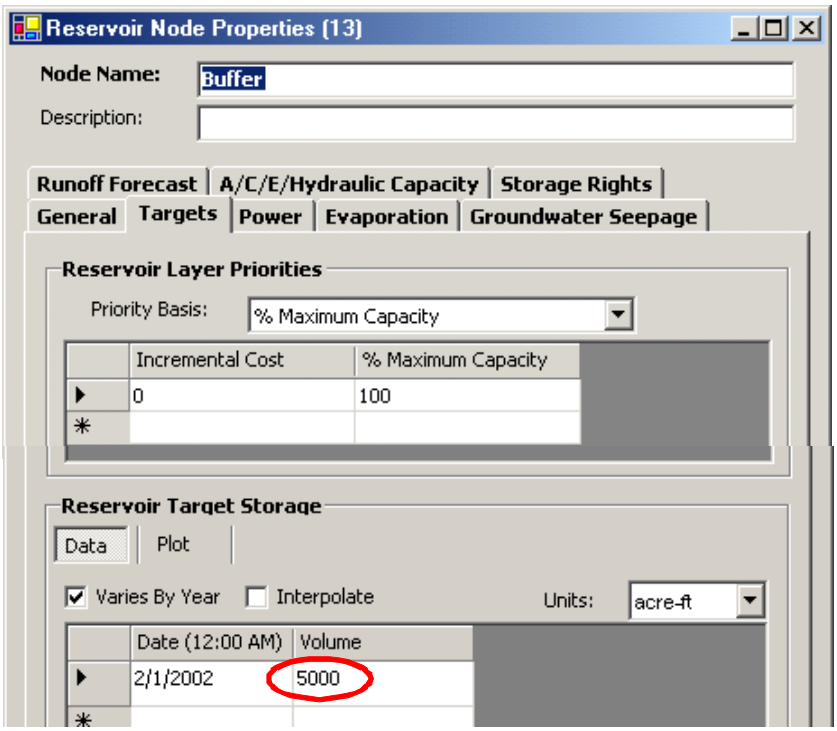
As example of Watch Logic in MODSIM, flow in the Exchange link to *NPoudre_Irrig* is assured to be the same as flow as conveyed in the link *OutflowNode_Buffer* by clicking the **Advanced** tab and specifying the **Exchange Limit Link** as the ownership link *Exchange*. This is what makes the exchange work!



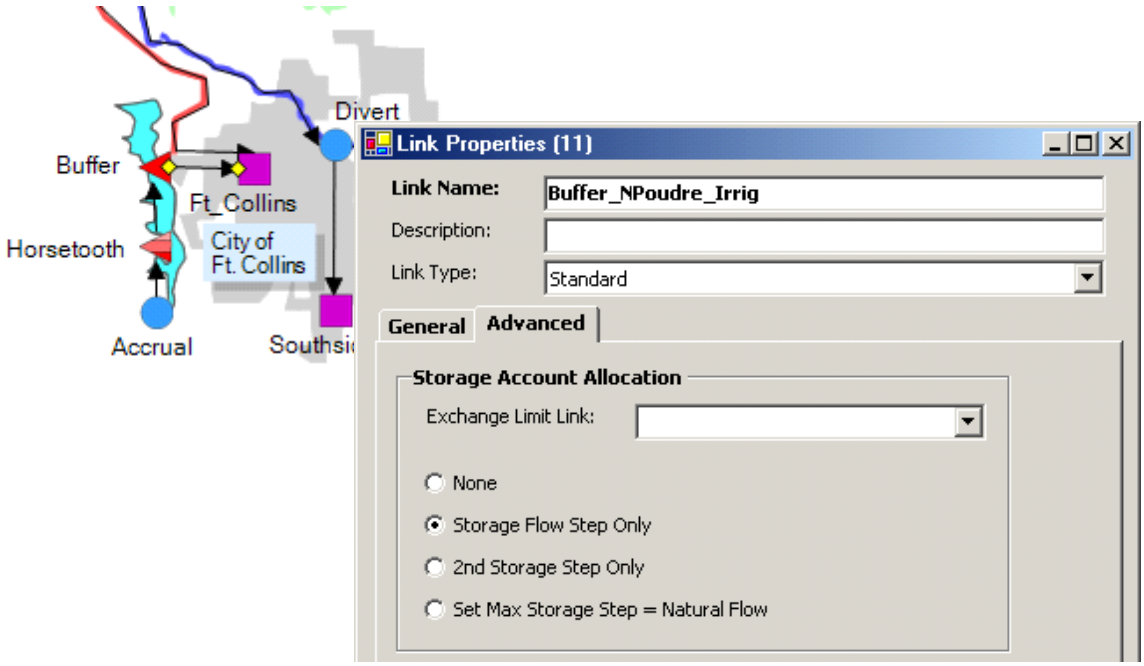
Buffer is not an actual reservoir, but is a mechanism for allowing Ft. Collins to take the Exchange flows when needed, as long as the total seasonal amount of the exchange is the same. This solves the timing issue related to the fact the North Poudre Irrigation Co. only needs water during the irrigation season, where the municipal Ft. Collins demand must be



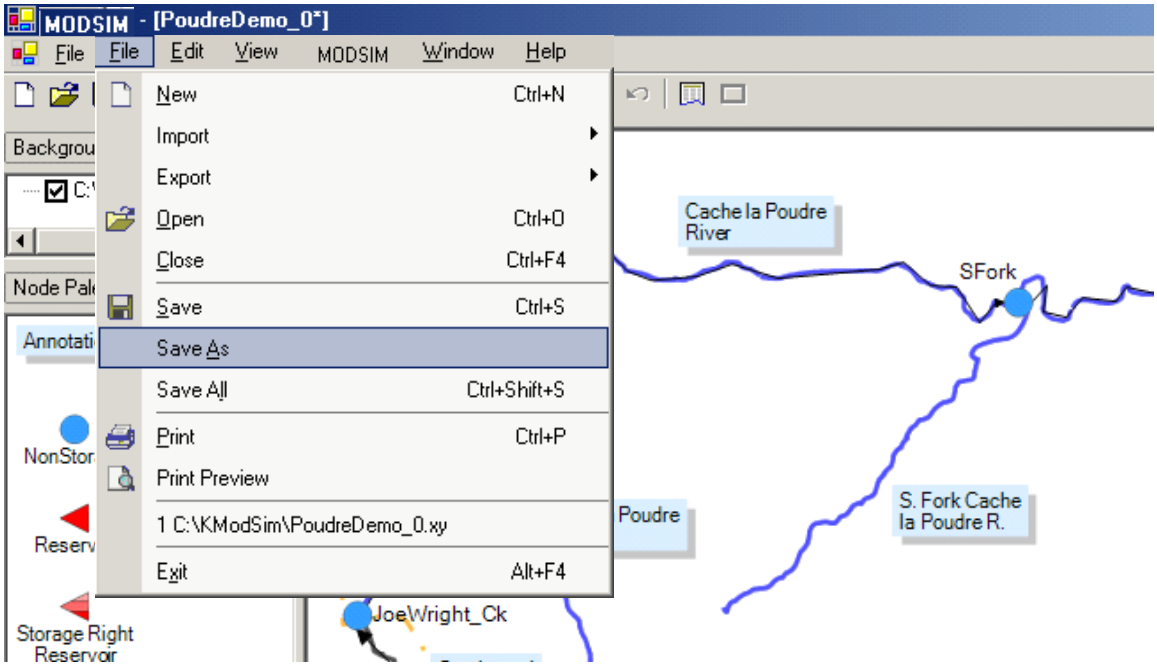
satisfied year-round. The **Maximum Volume** for *Buffer* is set to the same capacity as *Horsetooth*, but with a slightly higher priority that acts to pull water out of Horsetooth Reservoir for eventual delivery to Ft. Collins. The Reservoir Target Storage is also set to the maximum volume.



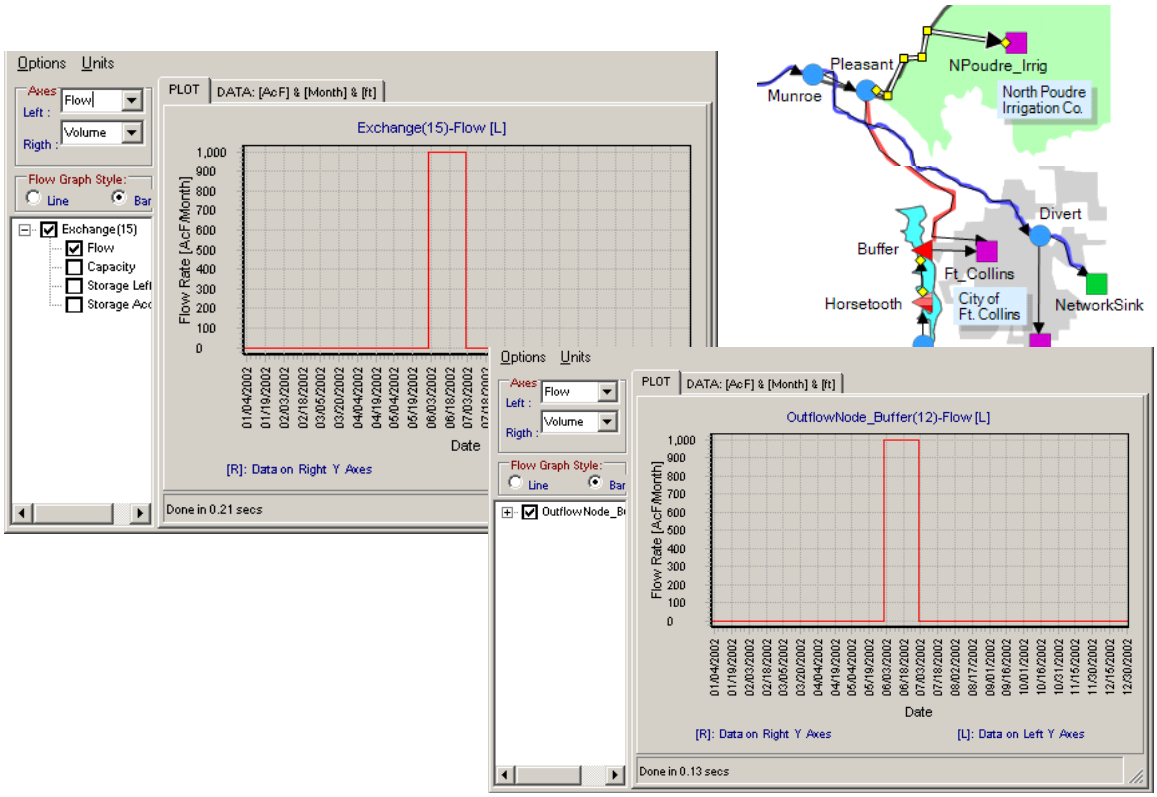
To make sure that Ft. Collins fully utilizes its natural flow rights **before** taking the Exchange water from Horsetooth Reservoir, the **Advanced** tab of the **Link Properties** form for link *Buffer_NPoudre_Irrig* allows selection of *Storage Flow Step only.* This



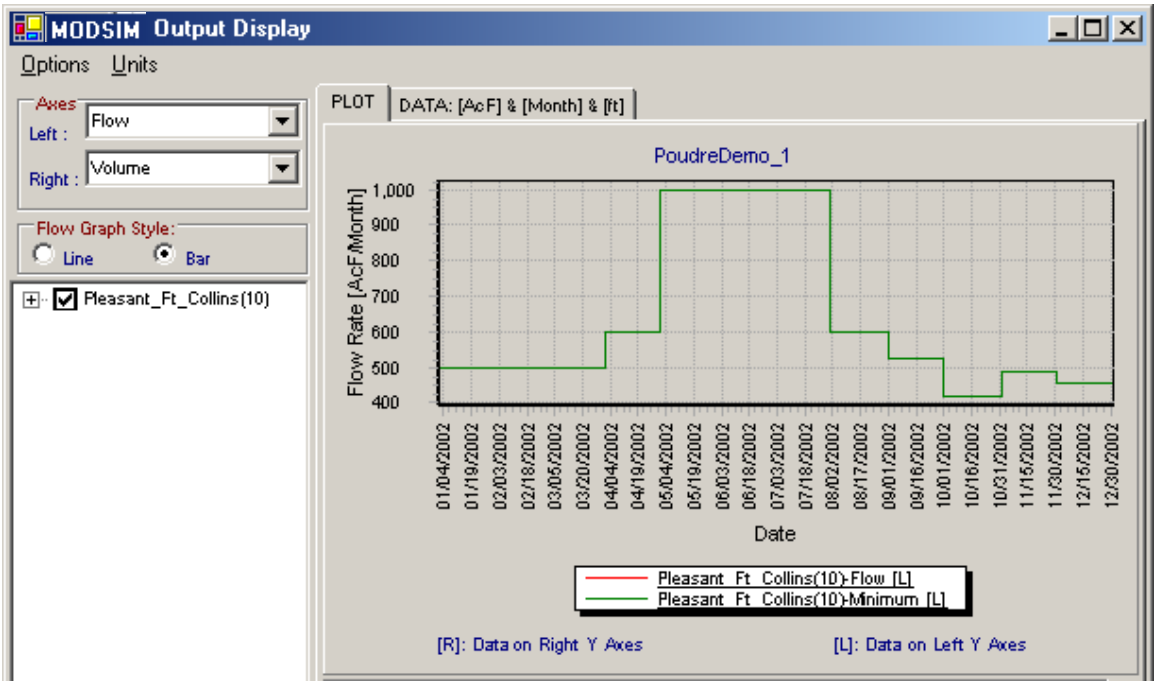
guarantees that flow in this link only occurs during the Storage Step, which is executed after the Natural Flow step. Save this network as **PoudreDemo_1.xy** and execute using the Run icon  .



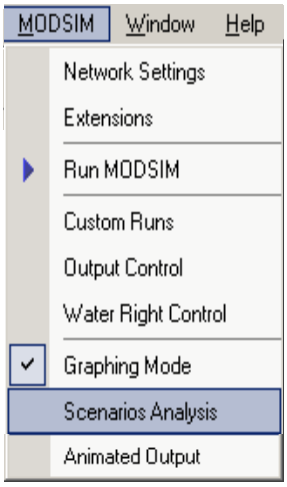
Plotting the flows in the links *Exchange* and *OutflowNode_Buffer* reveals that exactly the same discharges are occurring, indicating that the exchange is performing correctly.

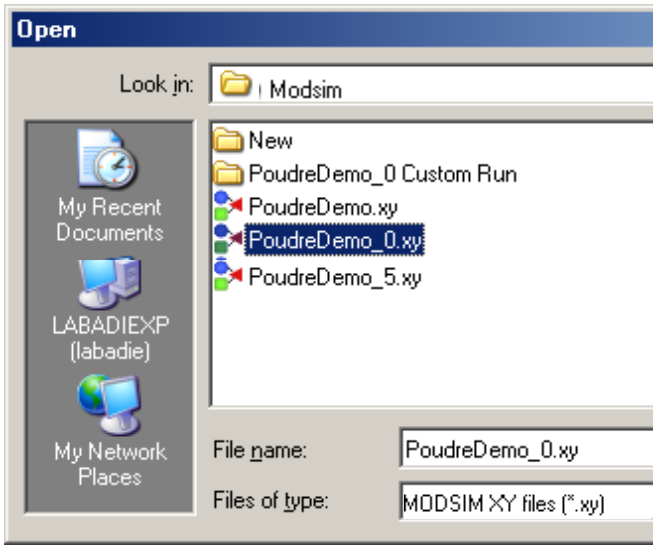


Plotting flow in the natural flow link to Ft. Collins *Pleasant_Ft_Collins* shows that the natural flow rights for Ft. Collins are fully utilized before any exchange flows are delivered.

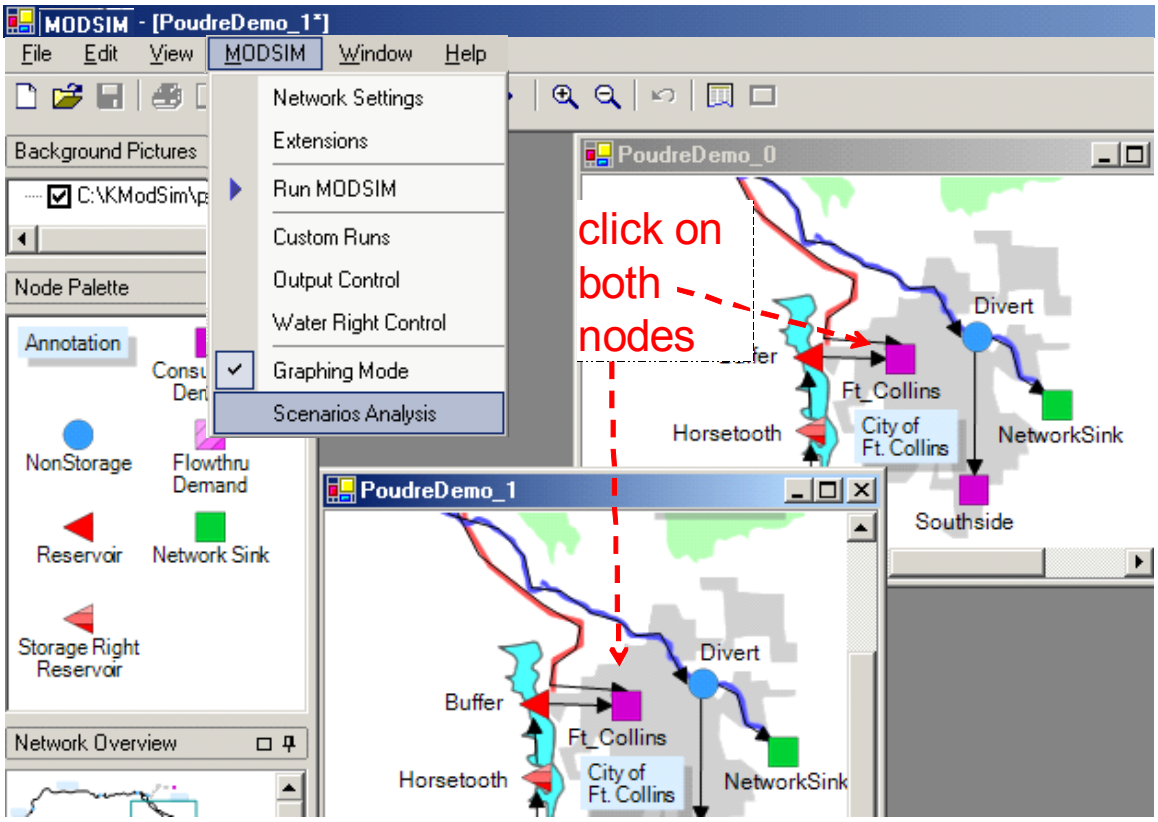


Selecting **MODSIM > Scenarios Analysis** allows results at several nodes or links to be simultaneously displayed in the same graphical plot. With the network **PoudreDemo_1.xy** displayed, the file **PoudreDemo_0.xy** can be opened simultaneously.

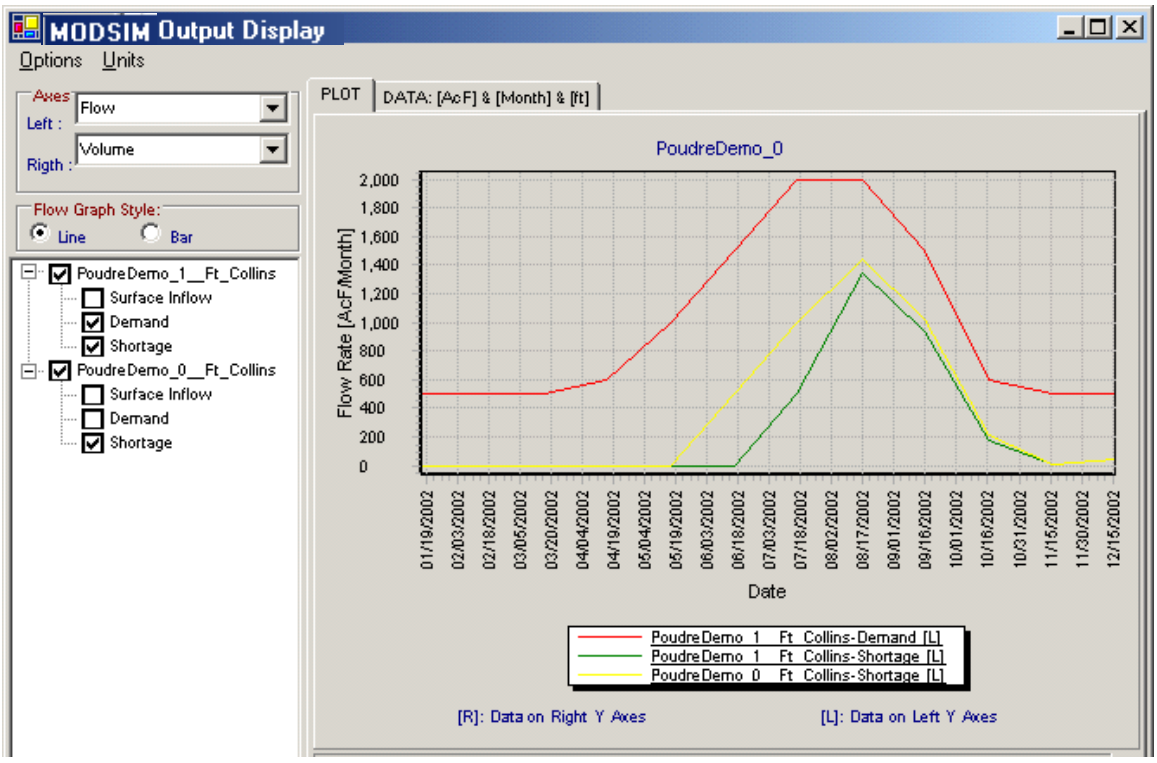




Clicking on both nodes in each network produces a graphical plot simultaneously displaying the results for each node selected.

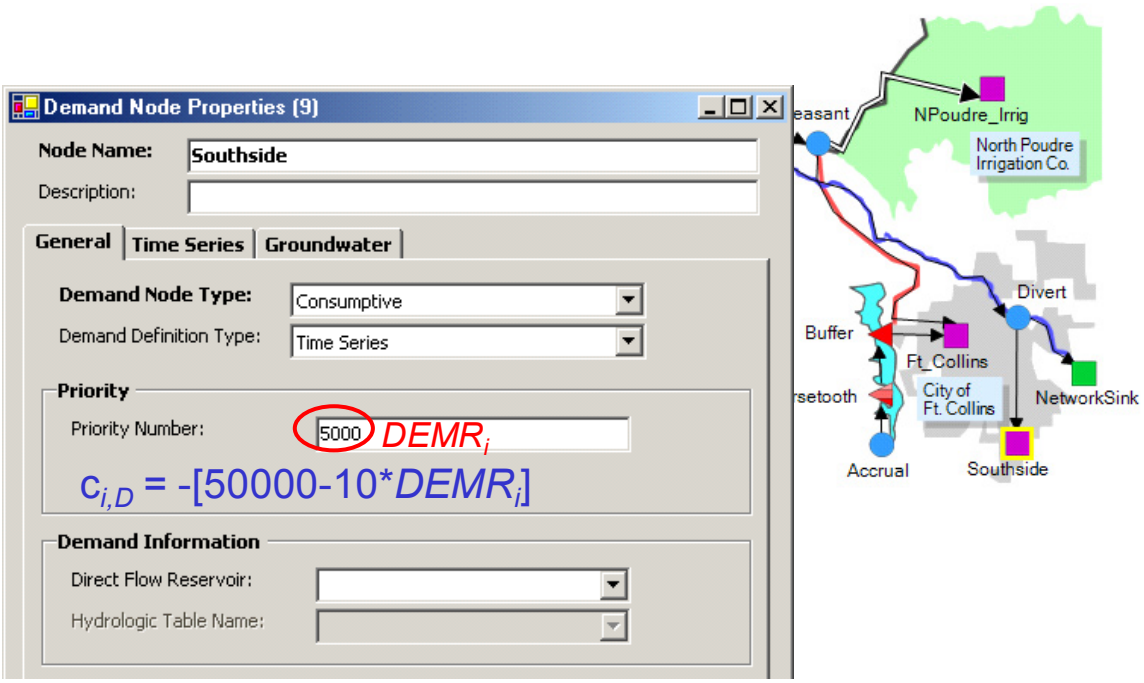


Results show reduction in shortages to Ft. Collins as a result of the exchange with the North Poudre Irrigation Co.

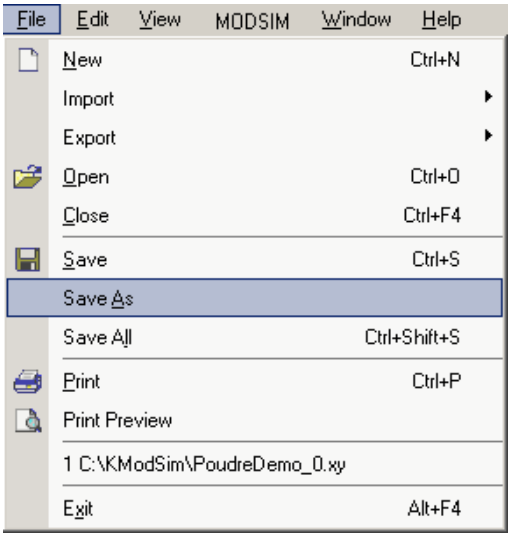


A.6 Purchase Southside Ditch Water Rights

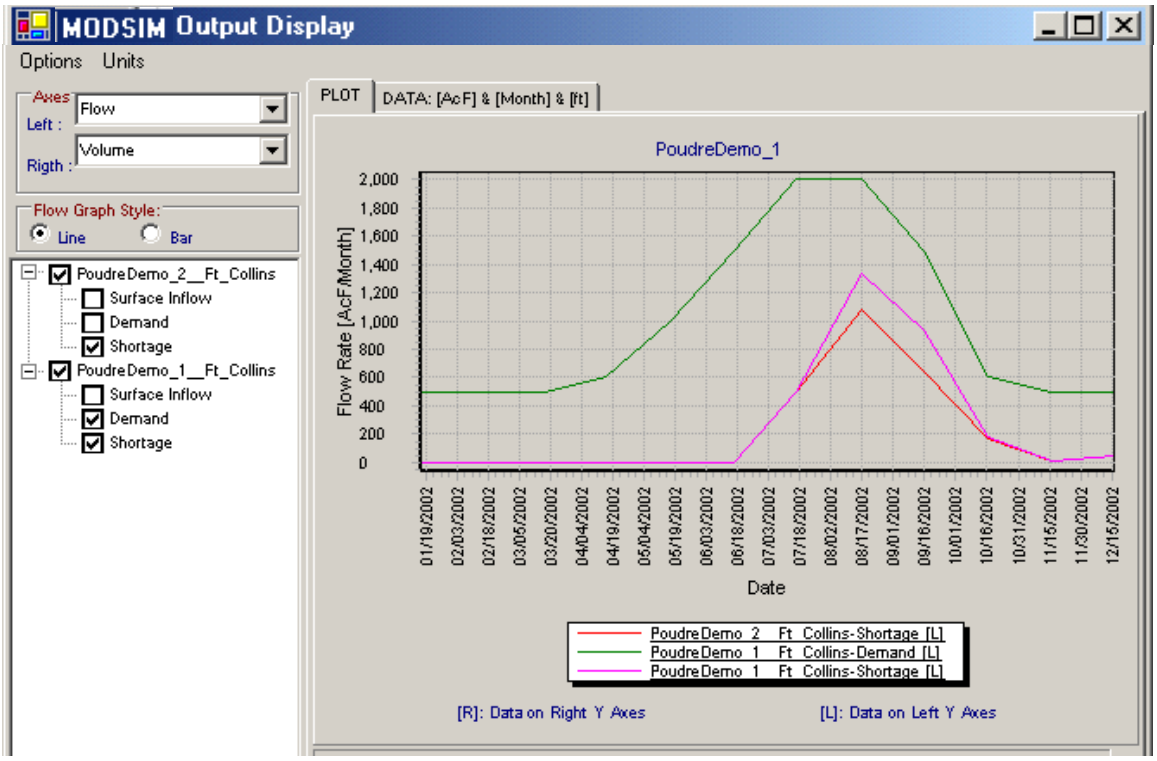
Purchase of Southside Ditch water rights by Ft. Collins is simulated by changing the priority for the Southside demand to 5000. Based on the formula used by MODSIM for calculating the artificial demand link cost: $c_{i,D} = -[50000-10 \cdot DEMR_i]$ setting the priority $DEMR_i$ to 5000 results in a 0 cost, thereby giving the Southside Ditches the lowest priority in the basin.



Save this network as **PoudreDemo_2.xy**.

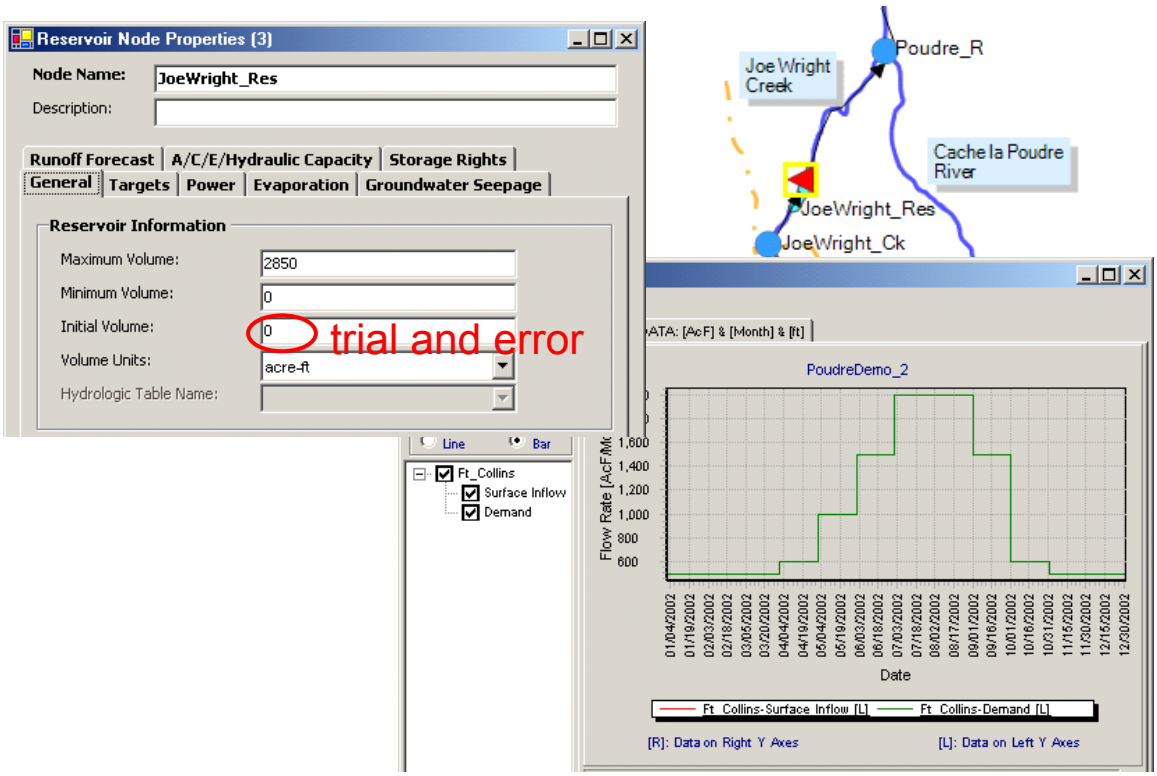


Again performing **Scenarios Analysis** to compare the solutions for **PoudreDemo_2.xy** and **PoudreDemo_1.xy**, shortages to Ft. Collins are further reduced as a result of purchase of the Southside Ditch water rights.

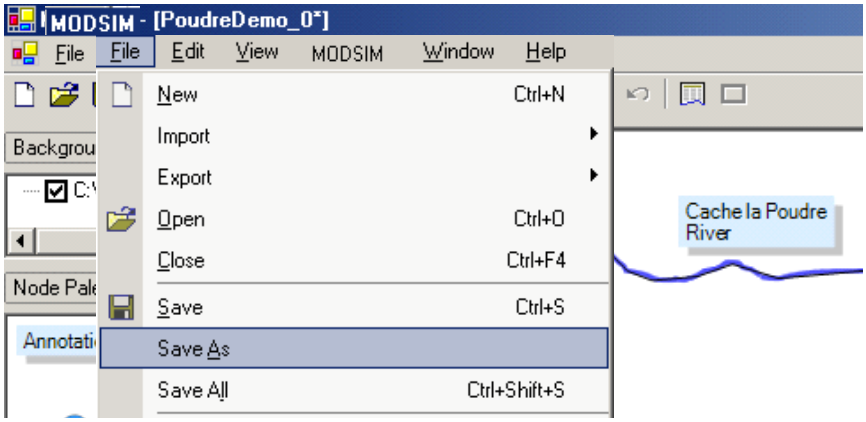


A.7 Construction of Joe Wright Reservoir

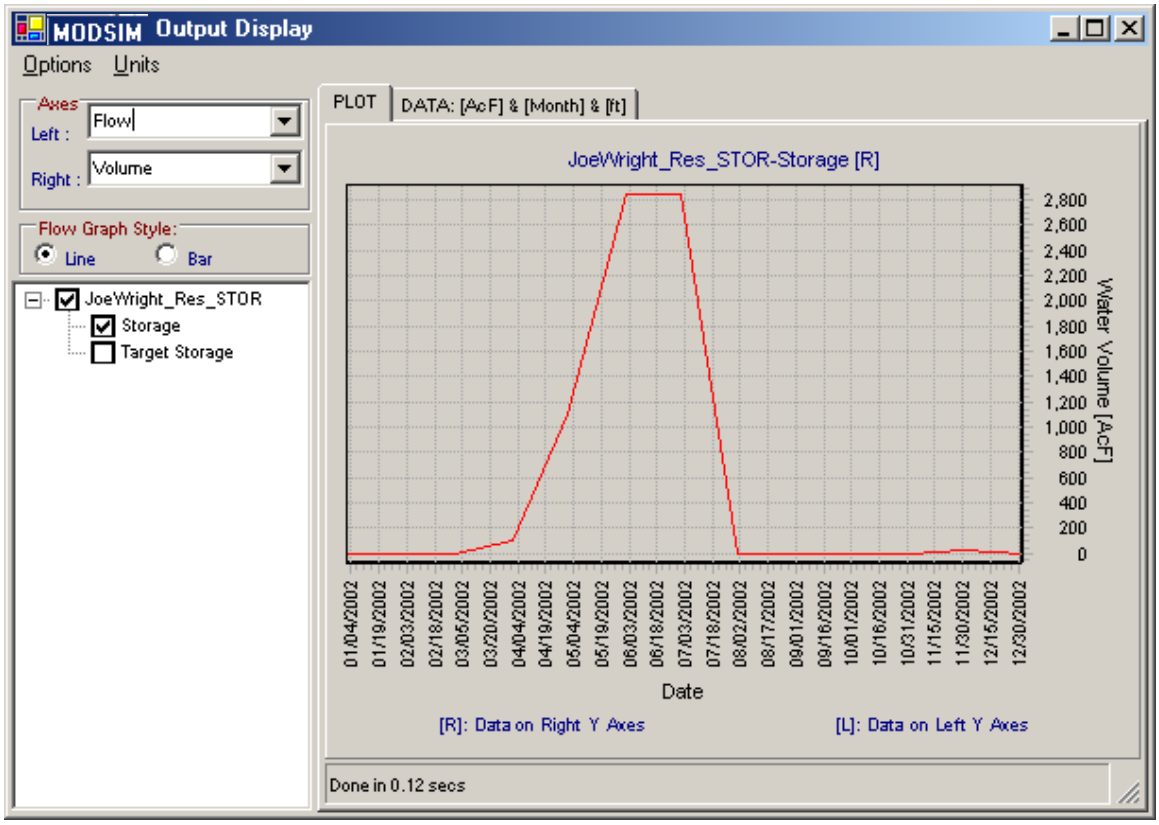
The next highest cost alternative is to construct Joe Wright Reservoir, but only at the minimum storage capacity required to remove all water shortages at Ft. Collins. The **Maximum Volume** for Joe Wright Reservoir can be adjusted by trial-and-error, since all other necessary data for the proposed reservoir have been entered into the **Reservoir Node Properties** form. This process results in a minimum storage capacity of 2850 ac-ft such that a capacity slightly below this volume results in small shortages. In using MODSIM for planning the development of new facilities, it is recommended that the facilities be included in the network as it is created, but with capacities set to a value of zero. This makes easy to later examine various combinations of development scenarios for the river basin using MODSIM.



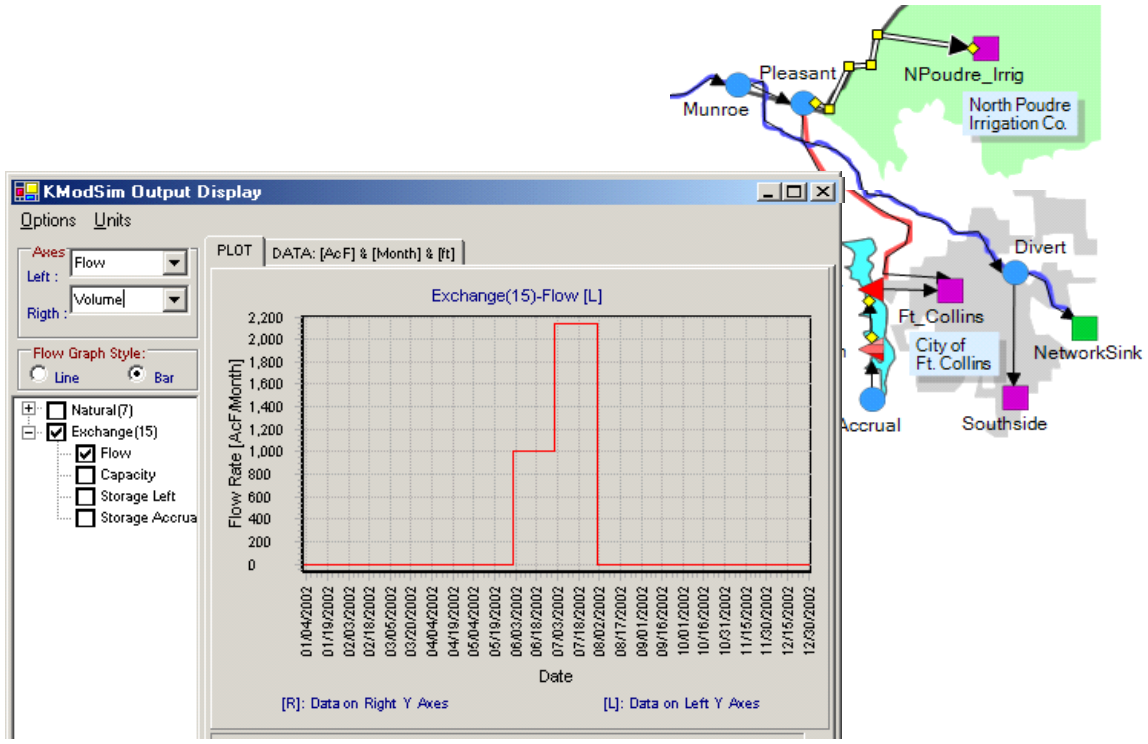
Save this network that includes construction of Joe Wright Reservoir as **PoudreDemo_3.xy**.



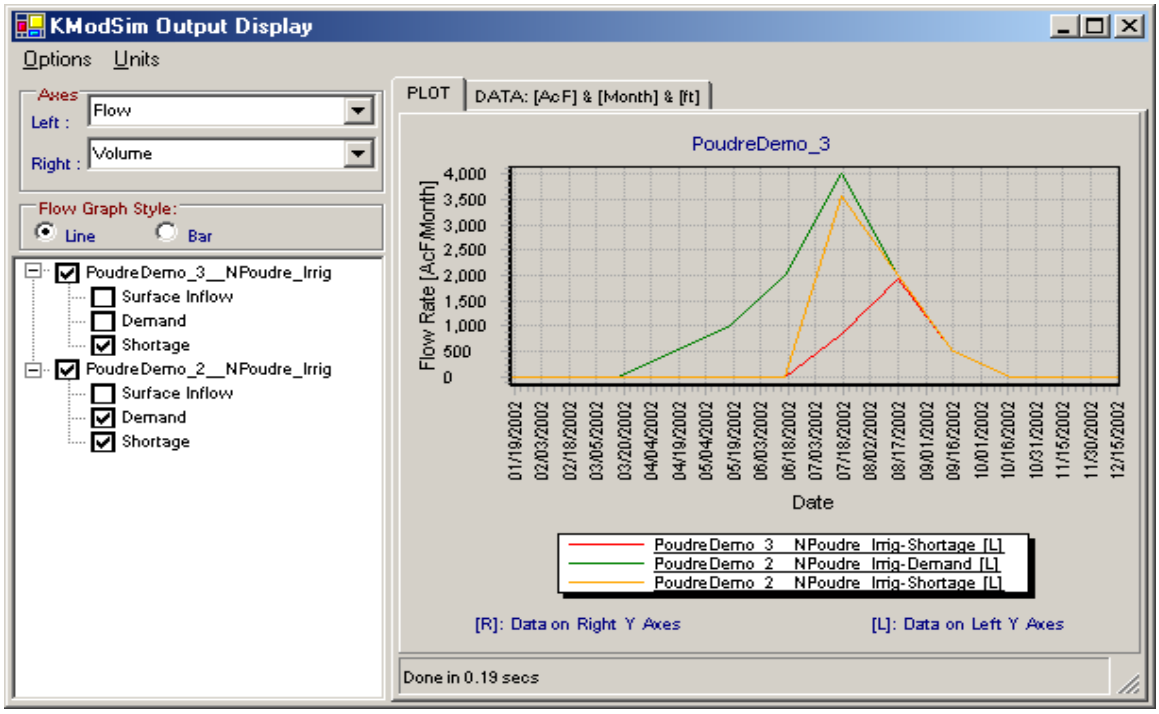
Plotting storage in Joe Wright Reservoir for the optimum capacity determined by trial-and-error indicates that the available storage capacity is fully utilized. This would not be the case if the graphical plot showed remaining storage in the reservoir at the end of the season.



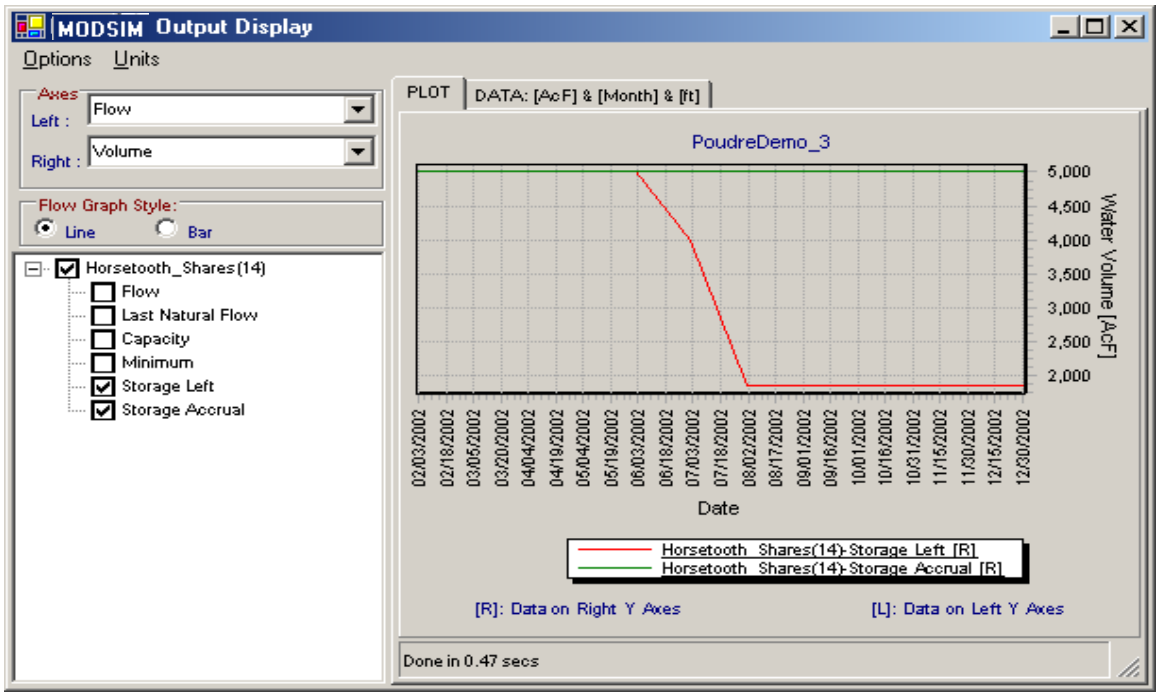
A direct benefit of construction of Joe Wright Reservoir is that it enhances the Exchange Agreement between Ft. Collins and the North Poudre Irrigation District, as evidenced by the graphical plot of the *Exchange* link to *NPoudre_Irrig* showing increased flow.



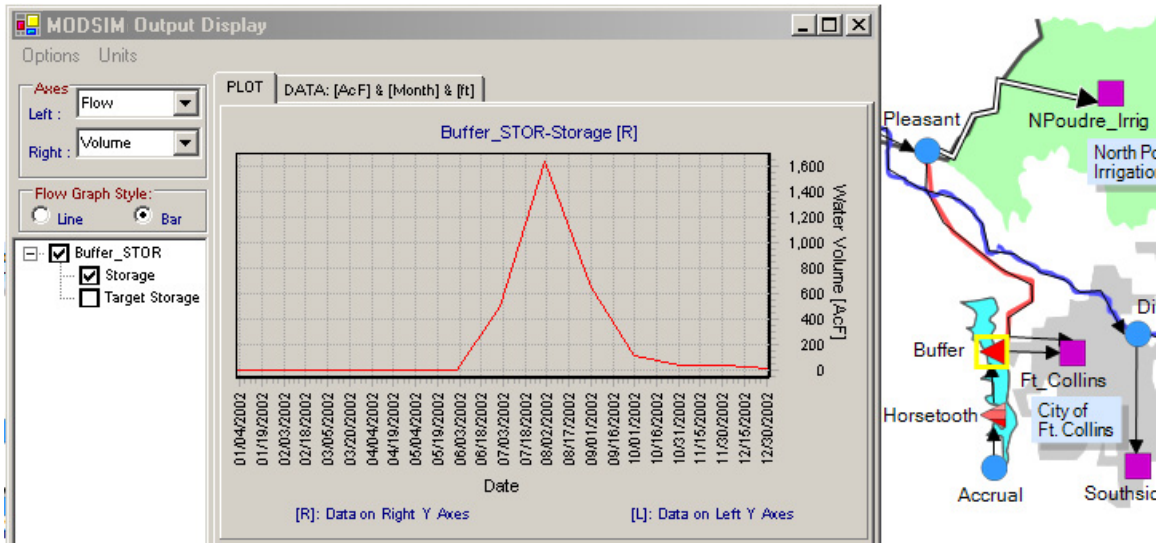
Performing Scenarios Analysis comparing the **PoudreDemo_3.xy** solution with Joe Wright Reservoir included with **PoudreDemo_2.xy** without the reservoir shows that the North Poudre Irrigation Co. indirectly benefits from construction of Joe Wright by enhancing the exchange with Ft. Collins.



Even with the enhanced exchange, the graphical plot of the storage account in Horsetooth Reservoir still has a remaining volume.

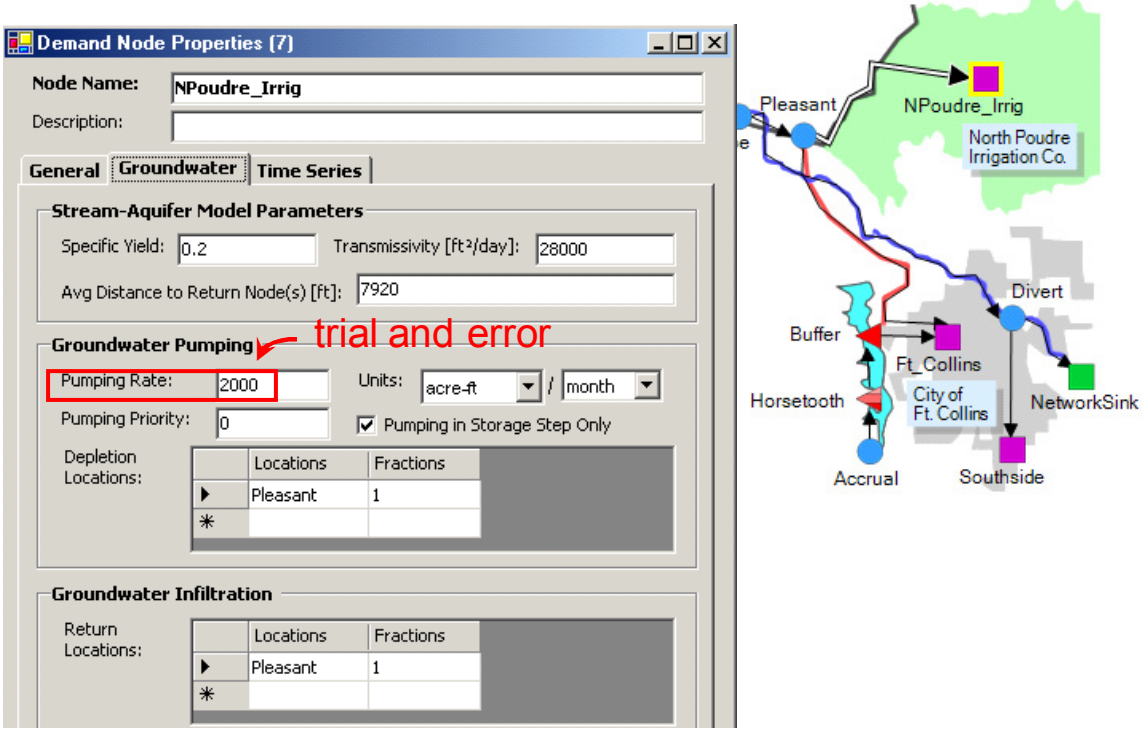


Plotting the storage volume in *Buffer* dummy reservoir reveals that the total seasonal exchange between Ft. Collins and the North Poudre Irrigation Co. is balanced since the storage in Buffer reduces to zero at the end of the season. This indicates that the total seasonal Poudre River flow taken by the North Poudre Irrigation Co. that actually belongs to Ft. Collins is equal to the total exchange flow taken by Ft. Collins from the storage account in Horsetooth Reservoir owned by the North Poudre Irrigation Co.

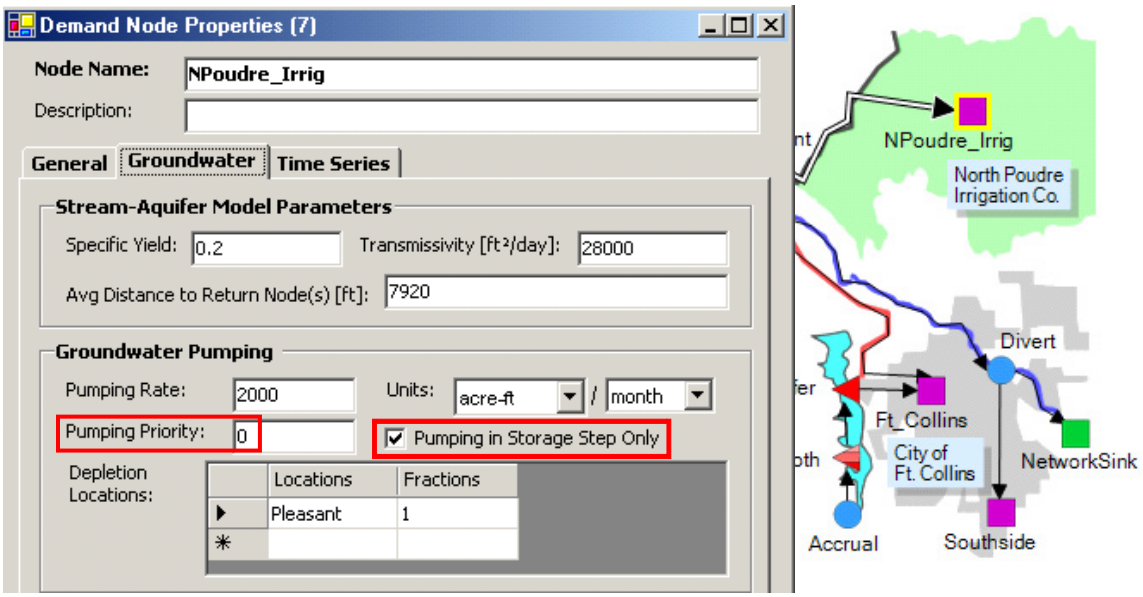


A.8 Groundwater Development for North Poudre Irrigation Co.

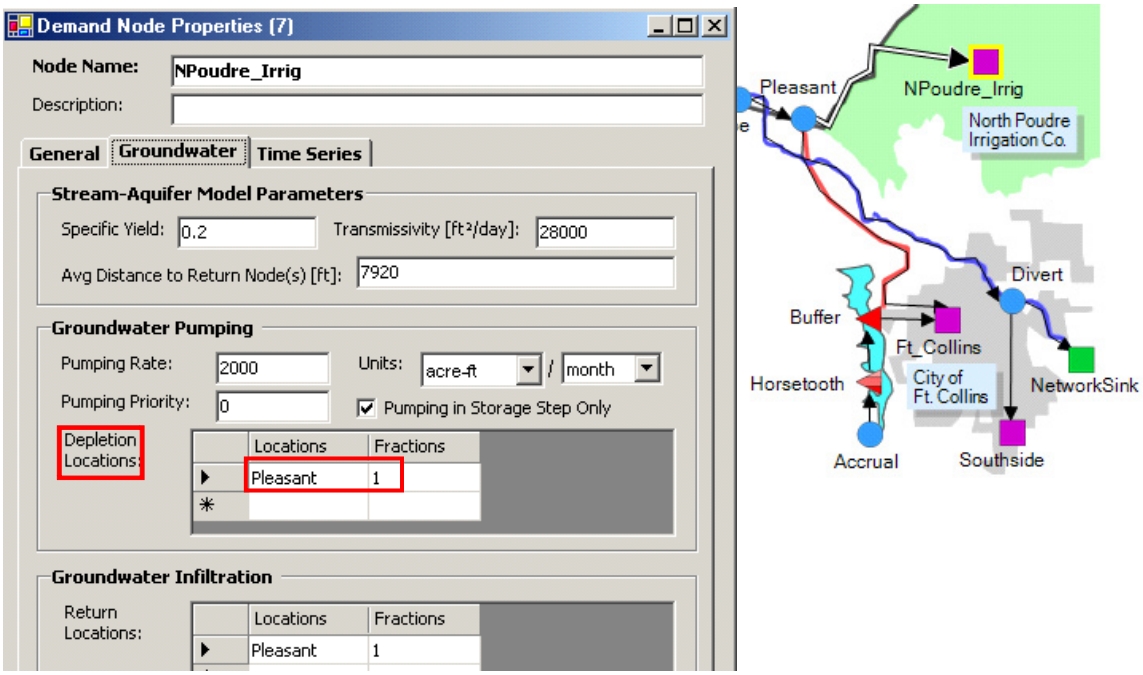
Makes ground-water supply available to node *NPoudre_Irrig* simply requires specification of a maximum pumping capacity for the well field, assuming that it does not exceed safe yield requirements. This pumping rate can be adjusted by trial-and-error



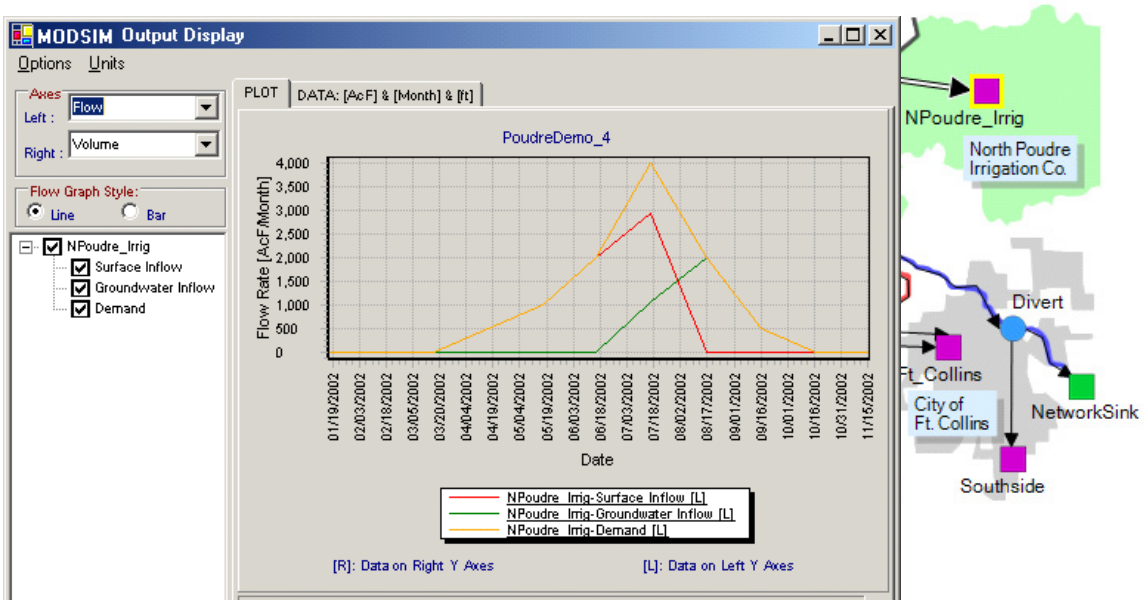
until shortages at *NPoudre_Irrig* reduce to zero. It is important that groundwater pumping is only used supplementally, where all surface water supplies from natural flow rights and exchanges are fully utilized by the North Poudre Irrigation Co. before any groundwater is pumped. This is guaranteed by clicking the check box next to **Pumping in Storage Step Only** under the **Groundwater** tab in the **Demand Node Properties** form of *NPoudre_Irrig*. This means that groundwater pumping will occur during the storage step iteration subsequent to the distribution of natural flow. Setting the **Pumping Priority** to 0 guarantees that exchange flows are delivered first to *NPoudre_Irrig* since they have a high negative cost that supersedes the value of the pumping.



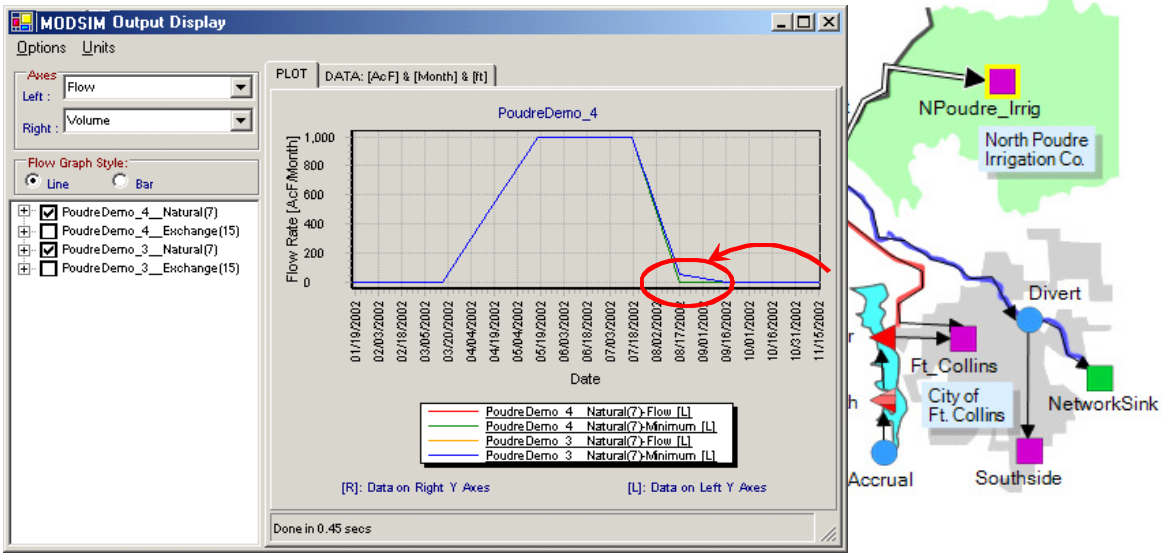
To consider the impacts of groundwater pumping on the stream-aquifer system, the Pleasant node is also specified as a location where flow depletion occurs due to pumping.



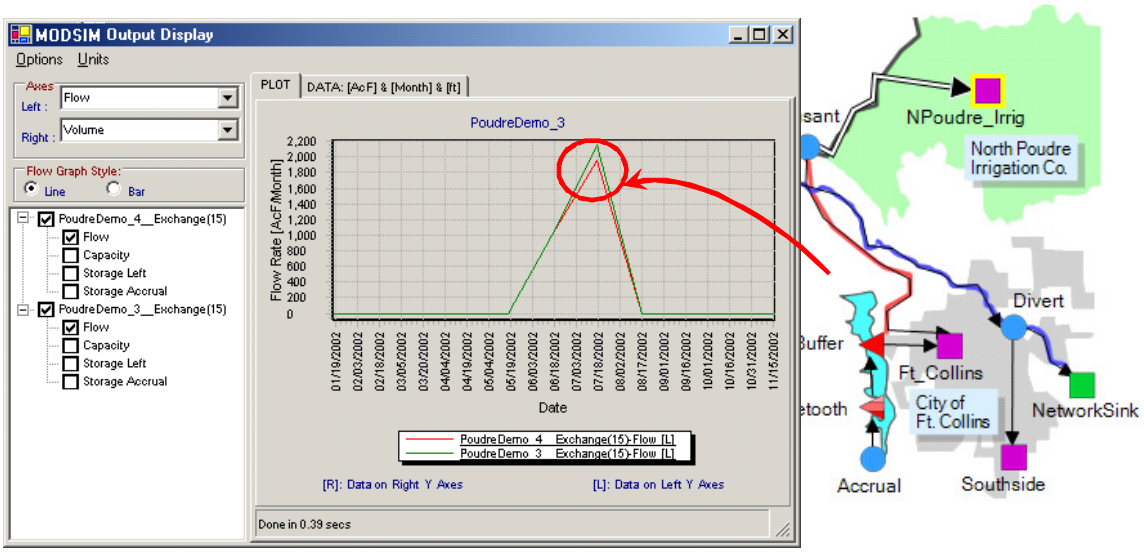
Prior to running the network with groundwater pumping included, the file should be saved as **PoudreDemo_4.xy**. Execution of MODSIM for this scenario and display of results for the *NPoudre_Irrig* demand node indicate that shortages are reduced to zero when a maximum Pumping Rate of 2000 is selected. This plot is a representation of the beneficial conjunctive use of surface water and groundwater supplies in the Poudre River basin.



To provide assurance that use of groundwater has not changed the natural flow deliveries to North Poudre Irrigation Co., the plot below under Scenarios Analysis shows essentially natural flow delivery as obtained prior to groundwater use, with a slight difference due to streamflow depletion caused by the groundwater pumping.

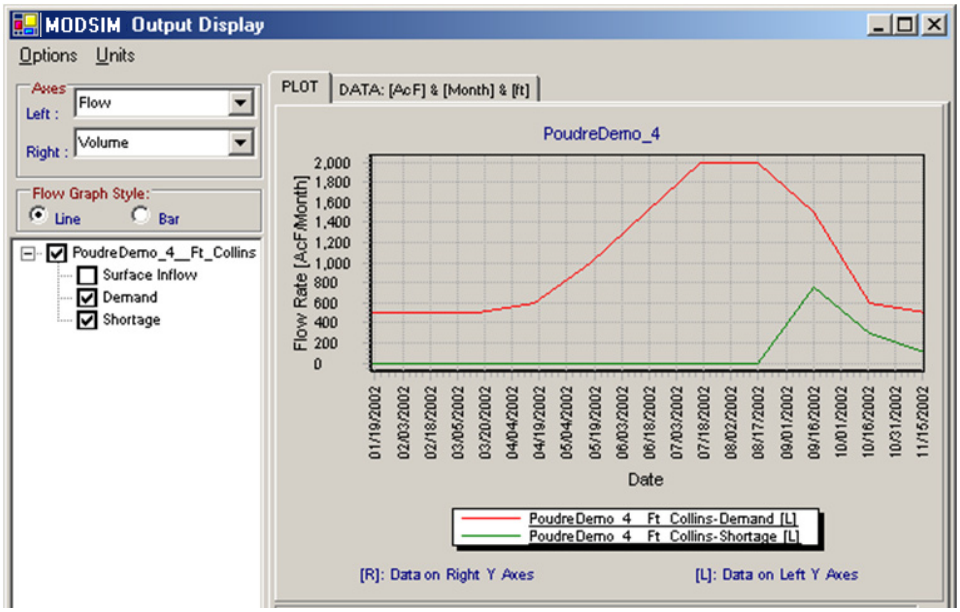


It can also been seen that flow in the Exchange link to NPoudre_Irrig is essentially unchanged, with a slight difference again due to streamflow depletion.



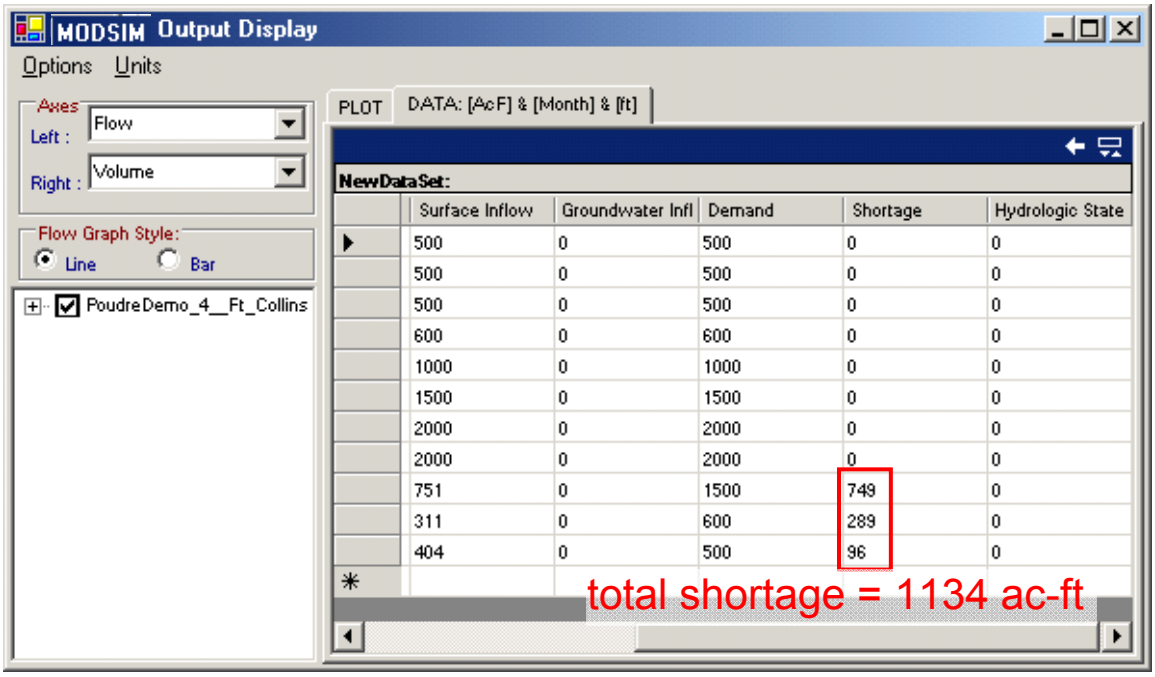
A.9 Plan for Augmentation

Generation of a graphical plot of the results for the *Ft_Collins* demand node indicates that a shortage is now occurring due to stream depletion caused by groundwater pumping by the North Poudre Irrigation Co.

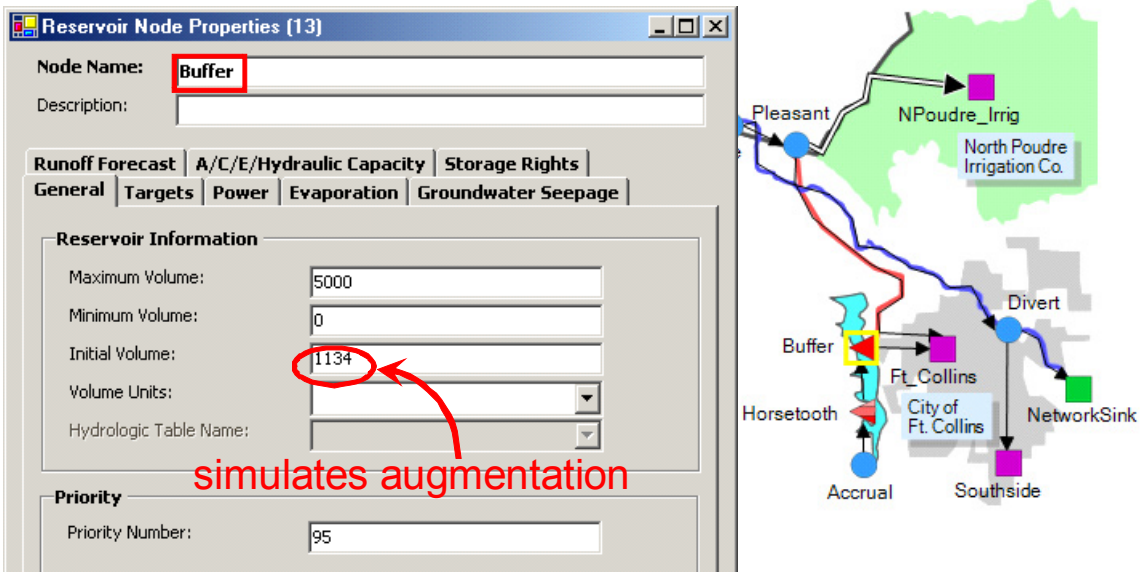


This depletion would normally give rise to a "call on the river" whereby the North Poudre Irrigation Co. is required by the Office of the Colorado State Engineer to cease groundwater pumping operations since they are damaging the senior surface water rights owned by the City of Ft. Collins. In this case, a *Plan for Augmentation* is put into place,

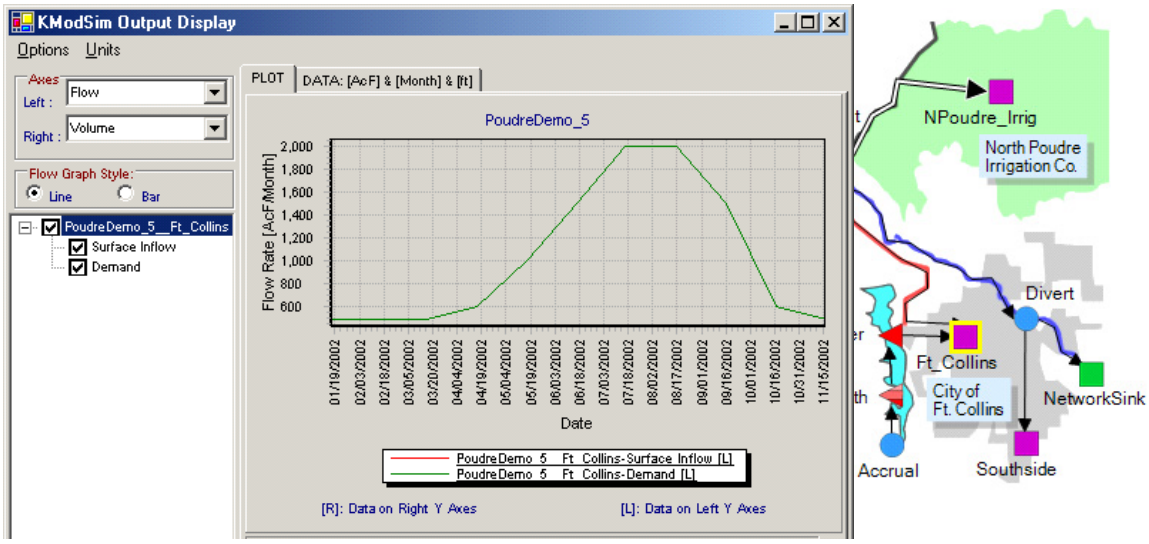
whereby the North Poudre Irrigation Co. augments the water supply to Ft. Collins in order to continue to pump groundwater. Since as shown previously, the North Poudre Irrigation Co. still has remaining volume in their Horsetooth storage account after the exchange agreement is fully executed, the *Plan for Augmentation* entails determining additional delivery requirements from the storage account to satisfy the Ft. Collins demands. Displaying the tabular results for the Ft_Collins demand node reveals the total shortage incurred due to pumping operations by North Poudre Irrigation Co.



Since there is additional volume remaining in North Poudre Irrigation Co. storage account, the Plan for Augmentation can be simulated by entering the total shortage to Ft. Collins as Initial Volume in the Buffer Reservoir for satisfying the Ft. Collins demand. This network can now be saved as **PoudreDemo_5.xy**.



Execution of the network **PoudreDemo_5.xy** gives results showing that the Ft. Collins demand is now fully satisfied.



TUTORIAL B
Storage Accounts and Group Ownerships

B.1 Prineville/Crooked River Project

The main body of the Bureau of Reclamation's Crooked River Project lies north and west of Prineville, Oregon, with the water resources of Ochoco Creek and Crooked River used to furnish irrigation water for approximately 20,000 acres. Project features include Arthur R. Bowman Dam impounding Prineville Reservoir (148,600 ac-ft. active capacity) on the Crooked River, Ochoco Reservoir (39,600 ac-ft. active capacity) on Ochoco Creek, a diversion canal and headworks on the Crooked River; Lytle Creek Diversion Dam and Wasteway; two major pumping plants, nine small pumping plants, and Ochoco Main and distribution canals. The MODSIM network developed for the Crooked River project demonstrates the powerful capabilities of storage ownership and group ownership accounting in MODSIM. In addition, the use of storage limit links and exchange limit links is discussed.

Fig. E.1. Location map for Crooked River Project, Oregon

A schematic diagram of the network displayed in the MODSIM GUI for file **Prineville8_1.xy** is shown in Fig. E-1 (not to scale).

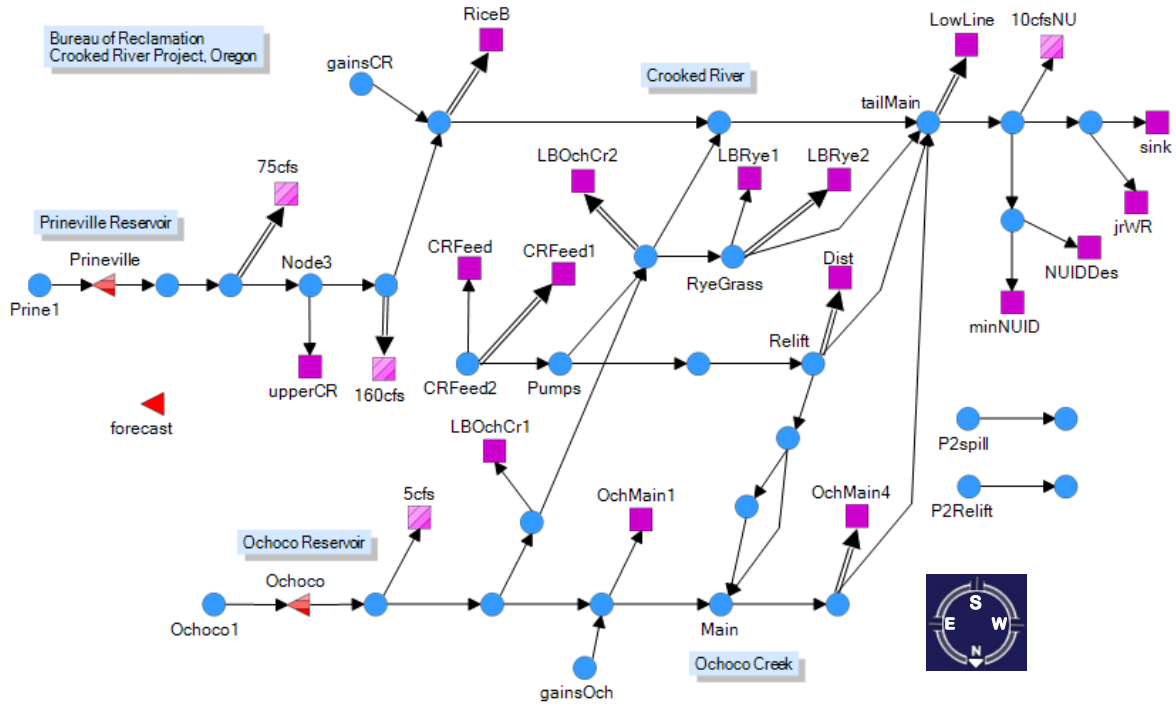


Figure E-2. Prineville/Crooked River MODSIM network.

Many river basin systems require distribution of *stored* water by water right. Storage water rights can be interpreted and administered in several ways. For the Crooked River Project, storage water rights are modeled as contractual agreements between the spaceholders in the reservoirs and the Bureau of Reclamation. Reclamation holds the natural flow right to accrue water to Prineville Reservoir in priority, and delivers the stored water to the spaceholders by request. Each spaceholder is limited to a contracted percentage of the reservoir fill, and shares in the losses which occur due to operational releases, flood control and evaporation. The spaceholder accounts behave much like a bank account:

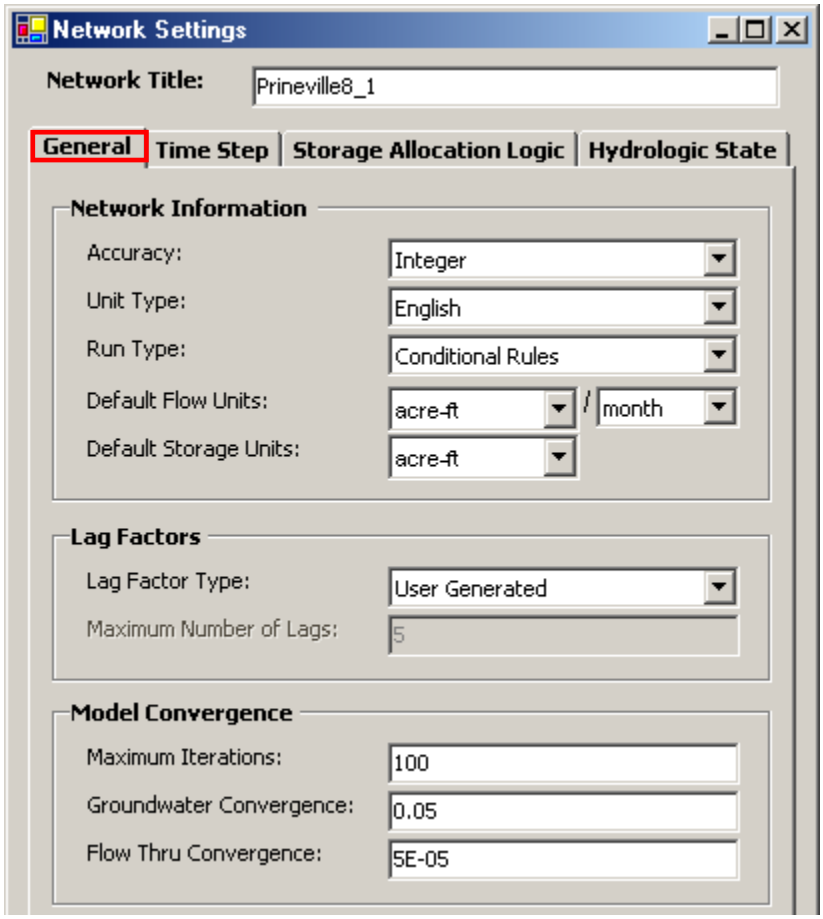
- water accrues each season to the spaceholder account
- the spaceholder withdraws water from the account as needed and the account balance is debited accordingly
- the spaceholder can never withdraw more water than exists in the current balance
- although water can be carried over into the next accrual season, carryover water is subject to flood control and evaporation losses

In MODSIM, storage ownerships are calculated using a two-step iterative solution process where the first step performs an allocation of natural or direct flow water rights either to storage accounts or demands holding those rights. In the first step, releases from storage accounts are not allowed. At the end of this step, all links conveying natural flow accruals based on water rights are constrained to exactly carry those flows, thereby ensuring that these natural flow entitlements will be maintained during the storage step.

In the next step, outflow links from the storage account reservoirs and storage ownership links connected to demands are opened to allow releases to be conveyed to those owning storage accounts in the reservoirs. The amount delivered to these demands is the smaller of the remaining demand after delivery of any natural flow rights and the remaining balance in the storage account. Again, natural flow rights are first allocated prior to any storage account releases since spaceholders frequently have primary natural flow rights and use their storage water as a supplemental supply.

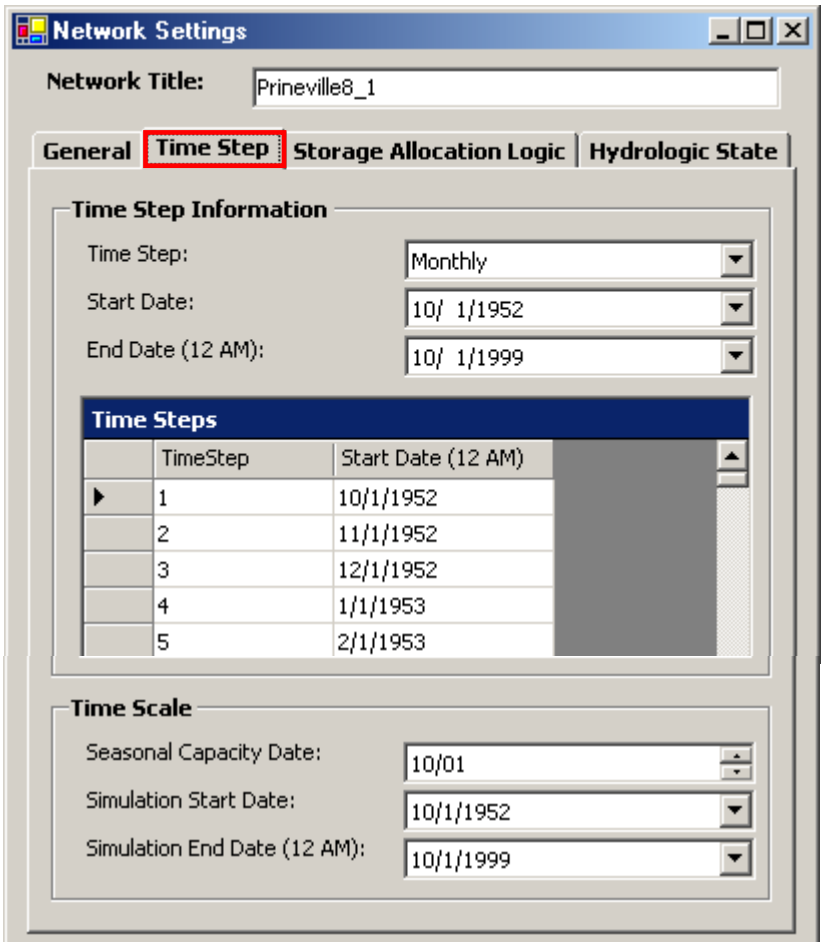
B.2 Network Settings

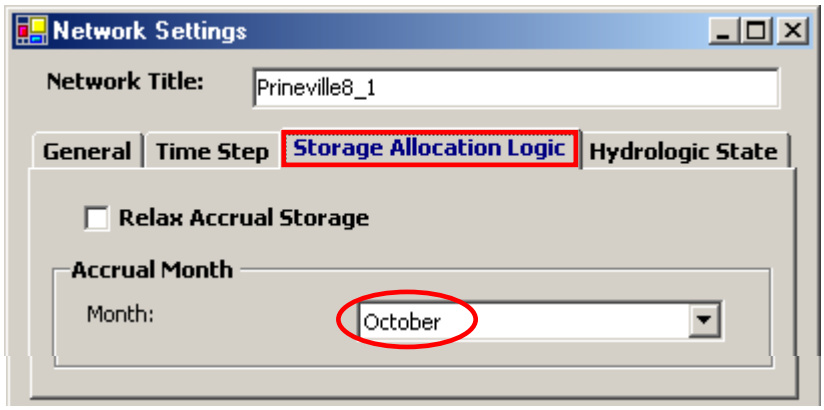
Selecting **MODSIM > Network Settings**, clicking the **General tab** allows specification of **Integer** accuracy for the calculations, **English** units, and *Conditional Rules* as the **Run Type**. The latter selection creates the **Hydrologic State** tab in the **Network Settings** form, allowing entry of hydrologic state tables for developing the conditional rules.



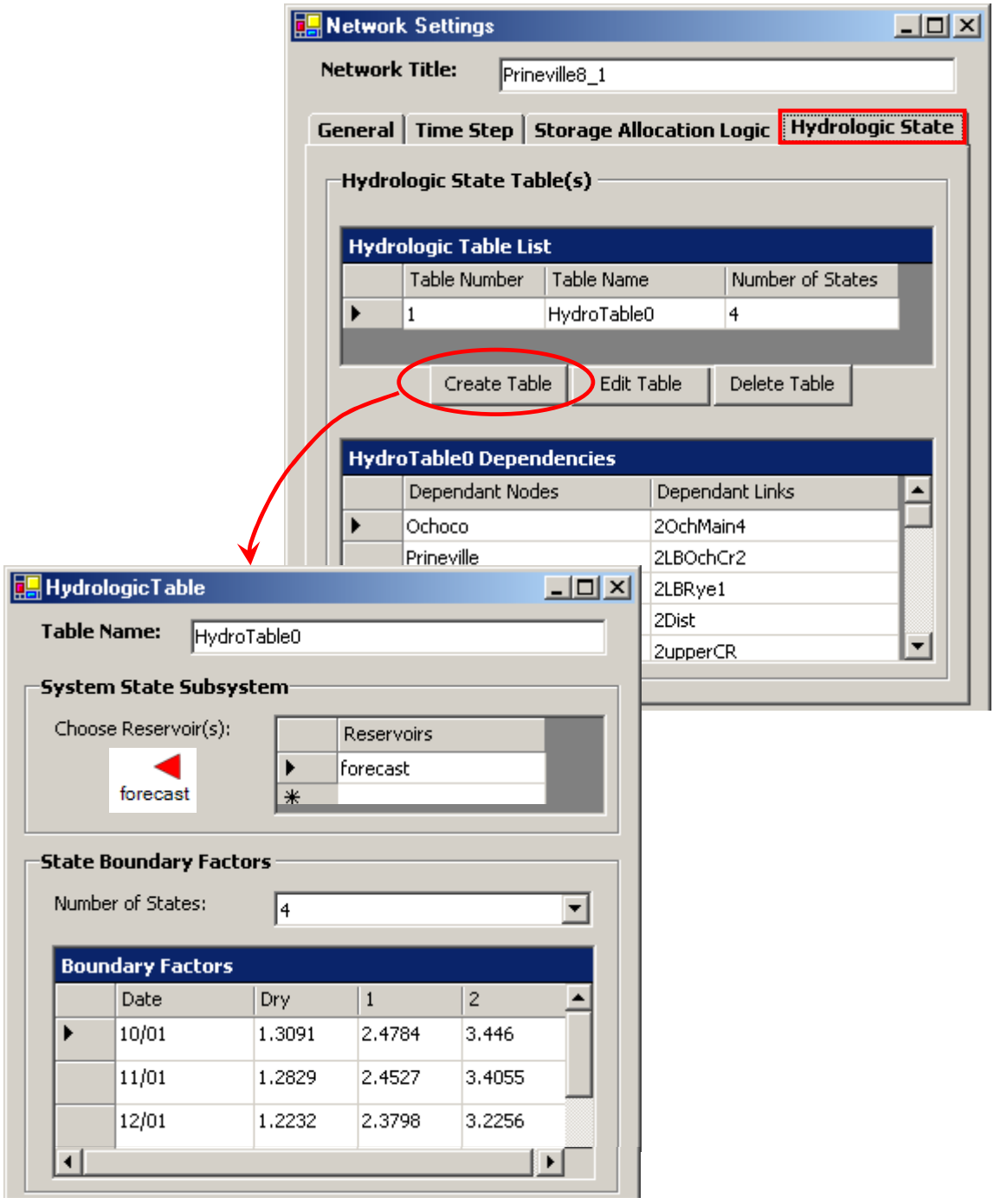
Clicking the **Time Step** tab under the **Network Settings** form allows specification of the simulation **Time Step** as *Monthly*, in this example, as well as the **Start Date** and **End Date** for the time series data entered into the model. Under the **Time Scale** field, the user is allowed to specify any **Simulation Start Date** and **Simulation End Date** within the time series data range for simulating over any portion of the data set. Notice that all dates

for the time series processing are assumed to be 12 AM on that date. The **Seasonal Capacity Date** entered into the **Time Scale** field represents the ending date of the accrual season, which is also defined under the **Storage Allocation Logic** tab. In this example the accrual season ends in October and a new accrual season begins the following month.



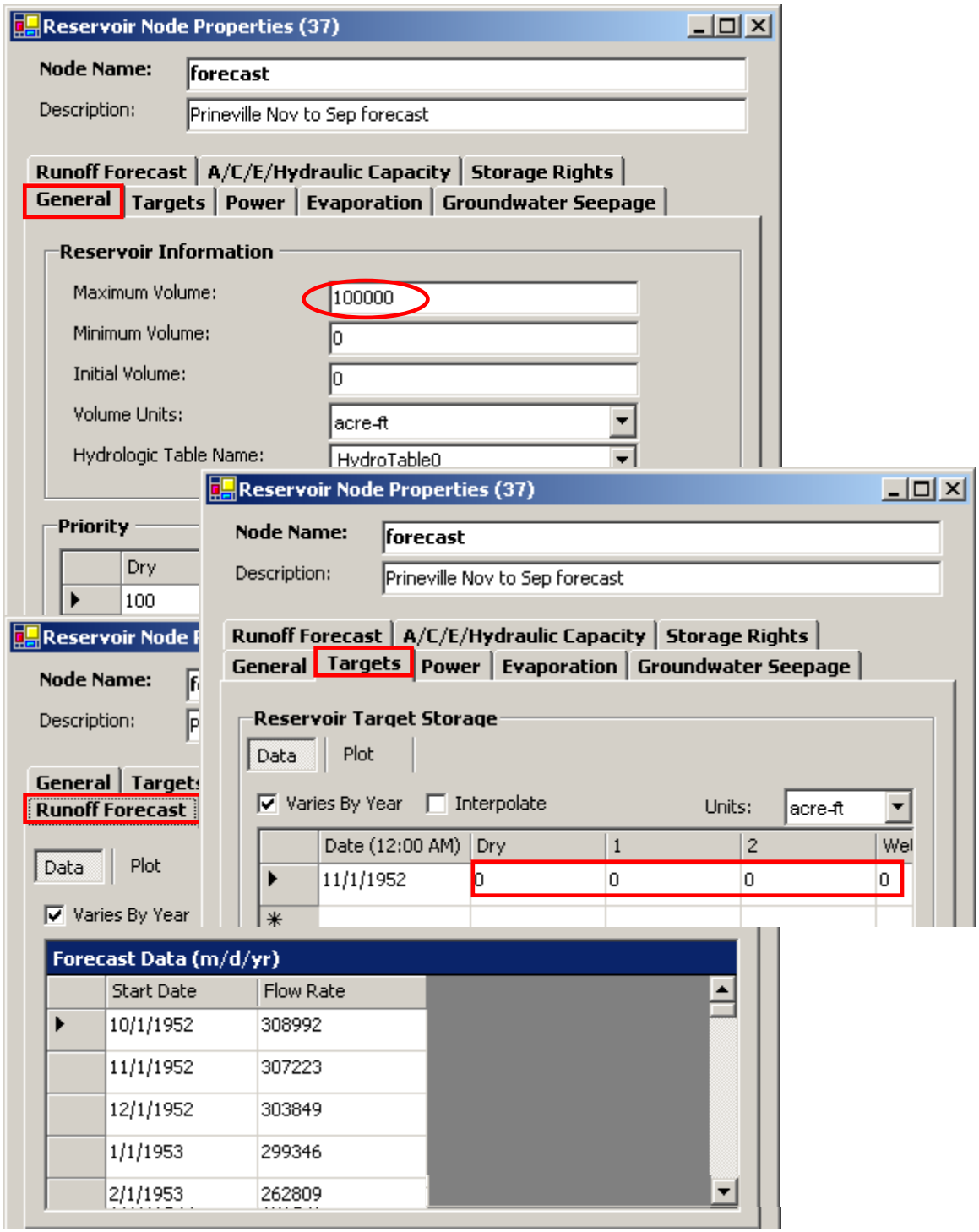


The **Hydrologic State** tab allows entry of hydrologic state information by clicking the **Create Table** button, allowing development of any number of hydrologic state tables as desired by the user. In this example, a single hydrologic state table *HydroTable0* with 4



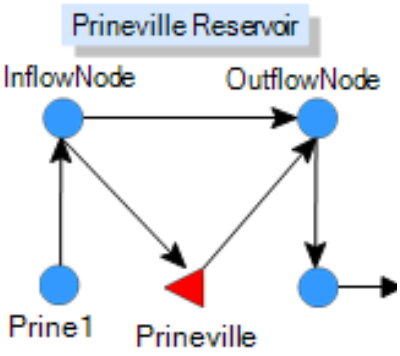
hydrologic states is created. Boundary factors are entered as fractions as determined by sorting the inflow forecasts (which are also the **Runoff Forecast** values entered into the *forecast* reservoir), dividing by 100,000 (the maximum volume of the *forecast* reservoir), and selecting break points at about the 25, 50, and 75 percentiles. The *forecast* reservoir does not represent a real world reservoir, but is only used for defining the hydrologic state. Forecasted inflow values to the basin entered in the **Runoff Forecast** tab of the **Reservoir Node Properties** form are the sum of the current month through September gains to Prineville Reservoir. These perfect forecasts are used by *HydroTable0* to

categorize the current month as entering wet, average, or dry hydrologic periods. No physical water is stored in the *forecast* reservoir since the target storages levels are zero, and the runoff forecast entries are forecasts and not actual inflow. The maximum volume setting of 100,000 ac-ft is arbitrarily selected for the purpose of scaling the values of the *Boundary Factors* defining the hydrologic state.

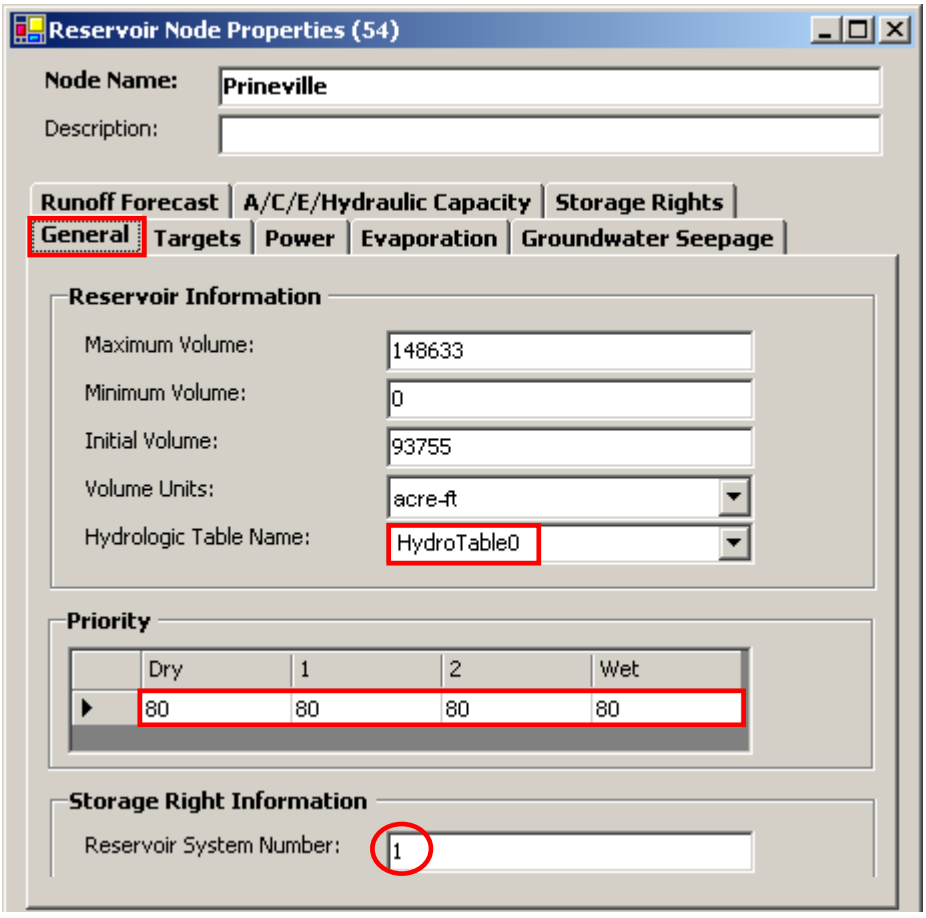


B.3 Reservoirs and Storage Rights

Prineville Reservoir. Prineville Reservoir is created as a storage account reservoir complex of 3 nodes and 3 links, which is revealed by first saving the network with a different name, rt. clicking on the **Prineville** node, and selecting **Break Apart Construct** in the context menu. The inflow node *Prine1* introduces the historic net gains to Prineville Reservoir from unregulated inflows and rainfall on the reservoir surface area, less the losses from evaporation and seepage. The storage right reservoir *Prineville* stores accrued water (not to exceed the target storage); releases water in the current time step to satisfy downstream requests or to meet the target storage; and allows for the carryover of storage into the next time step.



The **Reservoir Node Properties** form allows entry of *Maximum Volume*, *Minimum Volume*, *Initial Volume*, and selection of the *Hydrologic Table Name* upon which to base the conditional operating rules. Priorities for each hydrologic state can vary and are entered in the **Priority** field, although they are assumed constant for this example. The *Reservoir System Number* is entered in the **Storage Right Information** field of the form. Prineville Reservoir is assigned a *Reservoir System Number* of 1, in contrast with the

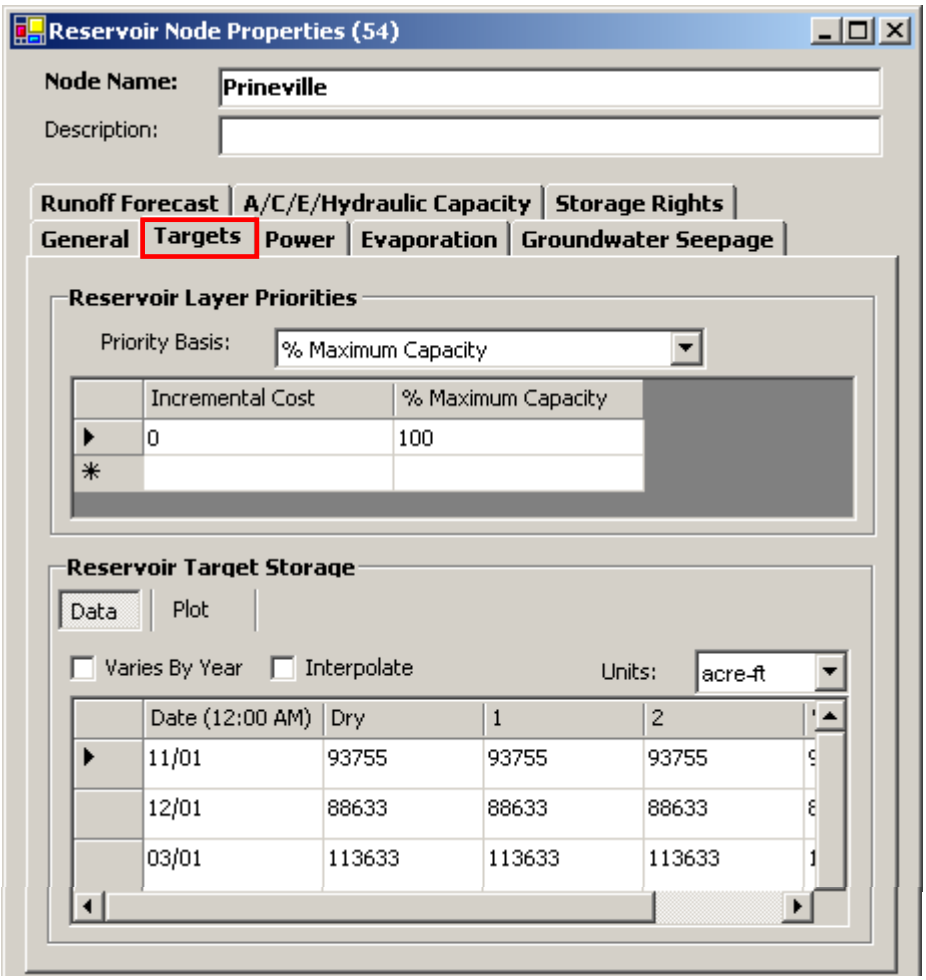


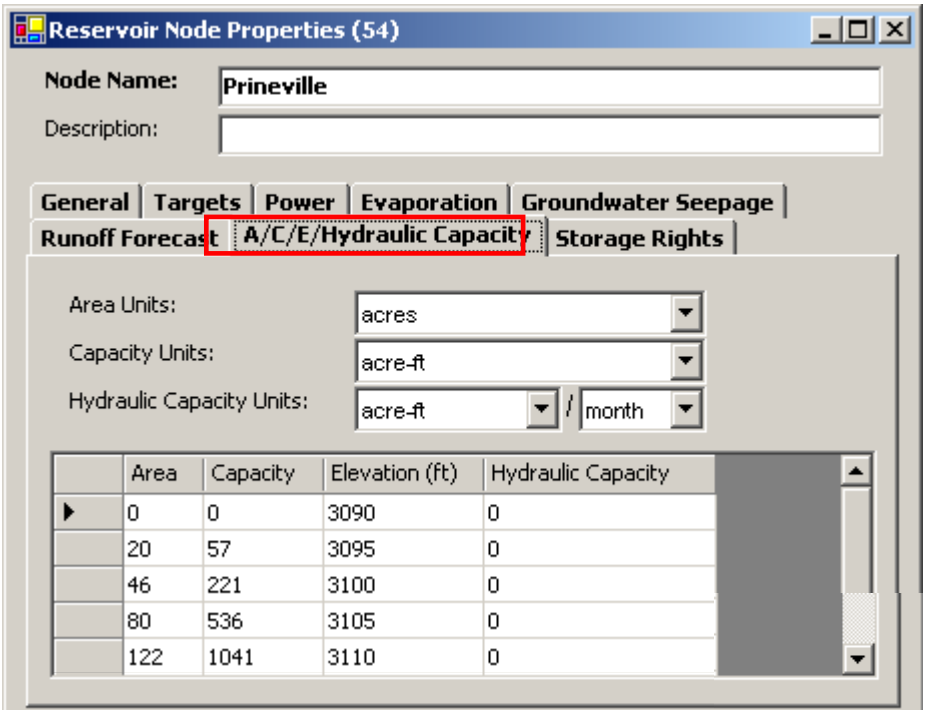
Number 2 assigned to Ochoco Reservoir. Since both numbers are nonzero, this forces Ochoco storage water to compete with Prineville if a demand downstream of both requires a release to be made. The *Reservoir System Number* assigned to Ochoco is selected to be different from the Prineville *Number* 1 for this example. This is important since at selected time steps, storage ownership account balances are adjusted such that the total theoretical or *paper* accounts equal the *physical* water in the reservoir at the end of the time step. Reservoirs with different nonzero system numbers have *paper* accounts balanced against the *physical* water for that particular reservoir. If the same nonzero *Reservoir System Number* is assigned to both reservoirs, then total *paper* accounts for both reservoirs are balanced with total *physical* water in both reservoirs. For this example, it was determined to be most appropriate that the accounts for each reservoir be adjusted to the *physical* contents of each reservoir.

The values entered into the **Priority** field are translated into relative costs on the artificial target storage link during the storage step iteration of the solution process in a given time step, with link upper bound settings based on the target storage levels entered into the **Targets** tab. As with all priorities and link costs in MODSIM, the lower the numeric value of the priority, the higher the benefit for flowing water through the link. This is how MODSIM simulates reservoir storage; since it is actually a flow through the artificial target storage link. If the river system has two (or more) reservoirs with nonzero *Reservoir System Numbers*, demands downstream of both reservoirs tend to draw first on the reservoir with the least benefit for holding water in storage. Here, Ochoco Reservoir is assigned a lower priority number (i.e., higher benefit) of 70 compared to Prineville Reservoir at 80. Therefore, releases are first made from Prineville Reservoir to satisfy demands that could be met from either reservoir. For this example, demand magnitudes and locations are such that is not necessary to define reservoir balance tables that would equalize the amounts of water taken from the reservoirs when both can satisfy demands.

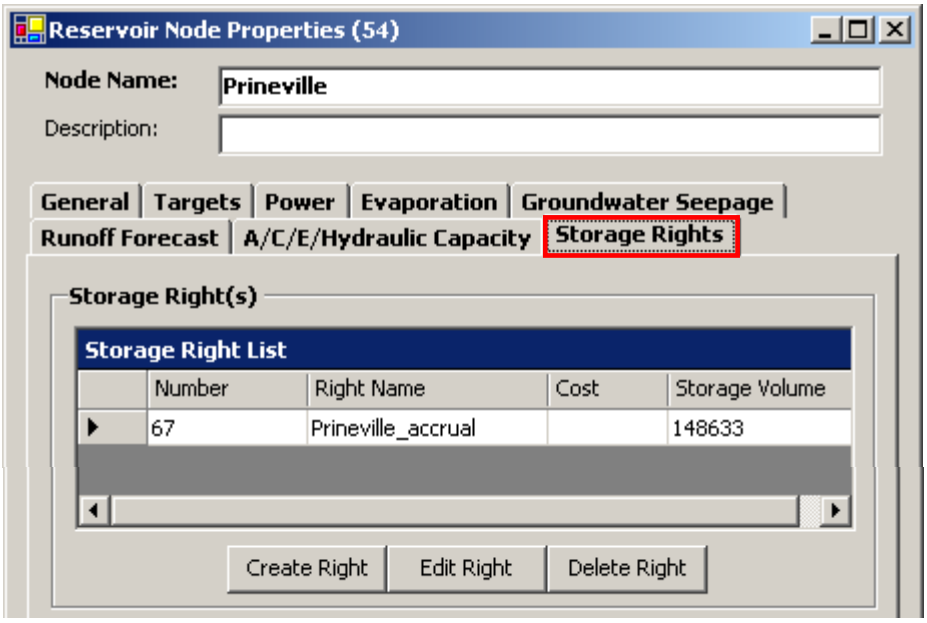
Target storage values are entered for each month under the **Targets** tab for each hydrologic state, and are defined as desired reservoir contents at the end of each time step. Releases are made from Prineville Reservoir throughout the irrigation season to meet a Labor Day (end of August) objective of 104,000 ac-ft. Target storage values were developed for each month to draw down the reservoir linearly in order to meet the end-of-August target. If the flows requested by spaceholders fail to sufficiently draw the reservoir down to the target storage value for the month, then additional water is released to meet the target for that month. Target storage values for winter and spring months were derived from flood control rule curves. Although target storage values conditioned on hydrologic state are usually designed to mimic historical operations, Prineville operators rely on a linear interpolation from the fill date to the end of August, and therefore require that the model always maintain 60,000 ac-ft of flood control space. Therefore, target storage values which do not vary with hydrologic state are appropriate in this case.

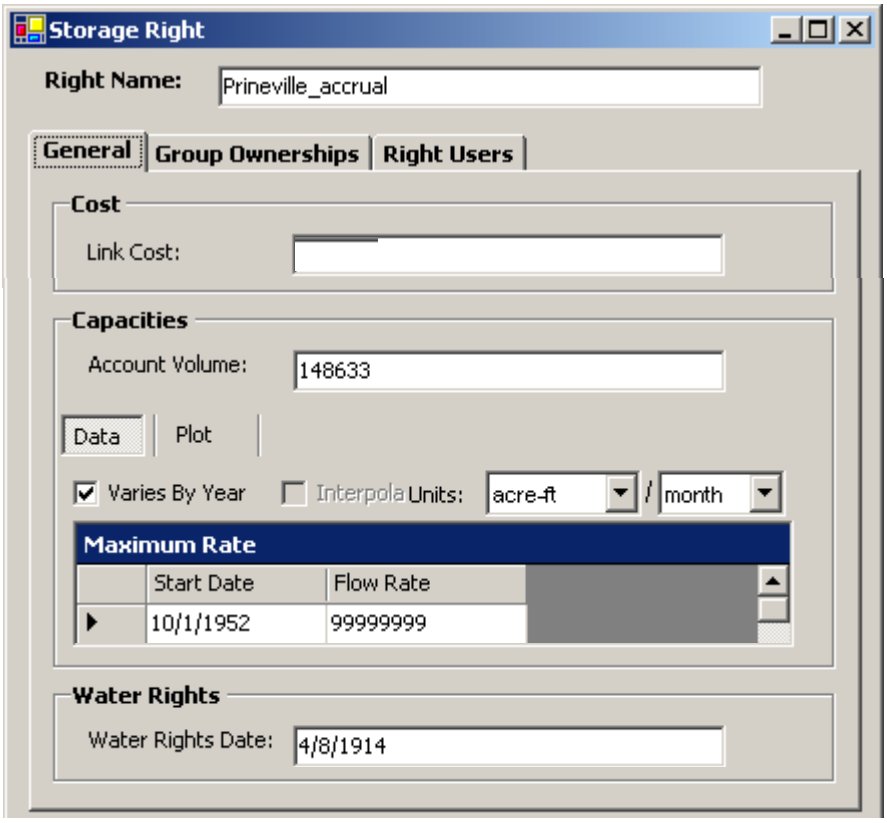
Although Area-Capacity-Elevation tables are entered for each reservoir, elevation values are only needed if the reservoirs include hydropower plants. The hydropower plant at Prineville Reservoir is not included in this example.


Reservoir Node Properties (54)
Node Name:
Prineville
Description:
Runoff Forecast
A/C/E/Hydraulic Capacity
Storage Rights
General
Targets
Power
Evaporation
Groundwater Seepage
Reservoir Layer Priorities
Priority Basis:
% Maximum Capacity
Incremental Cost
% Maximum Capacity
0
100
Reservoir Target Storage
Data
Plot
Varies By Year
Interpolate
Units:
acre-ft
Date (12:00 AM)
Dry
1
2
11/01
93755
93755
93755
12/01
88633
88633
88633
03/01
113633
113633
113633


Reservoir Node Properties (54)
Node Name:
Prineville
Description:
General
Targets
Power
Evaporation
Groundwater Seepage
Runoff Forecast
A/C/E/Hydraulic Capacity
Storage Rights
Area Units:
acres
Capacity Units:
acre-ft
Hydraulic Capacity Units:
acre-ft
/
month
Area
Capacity
Elevation (ft)
Hydraulic Capacity
0
0
3090
0
20
57
3095
0
46
221
3100
0
80
536
3105
0
122
1041
3110
0

As storage account reservoirs, both Prineville and Ochoco can accrue water to storage based on a storage rights that compete with other natural flow rights in the basin. Although reservoirs can have several storage rights with differing decree dates and amounts, in this case each reservoir has a single accrual right with amounts corresponding to the active storage capacity of the reservoir. However, several group ownerships share in the distribution of water accrued to the storage accounts. Storage rights are created by clicking the **Storage Rights** tab on the **Reservoir Node Properties** form, and then clicking the **Create Right** button, which displays the **Storage Right** form

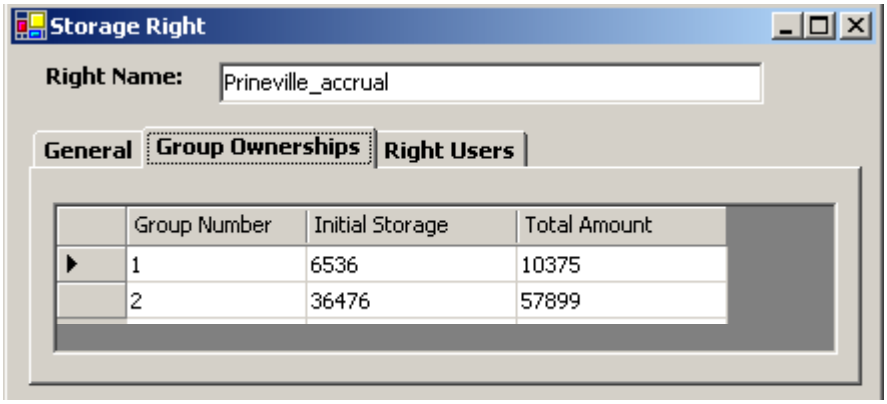




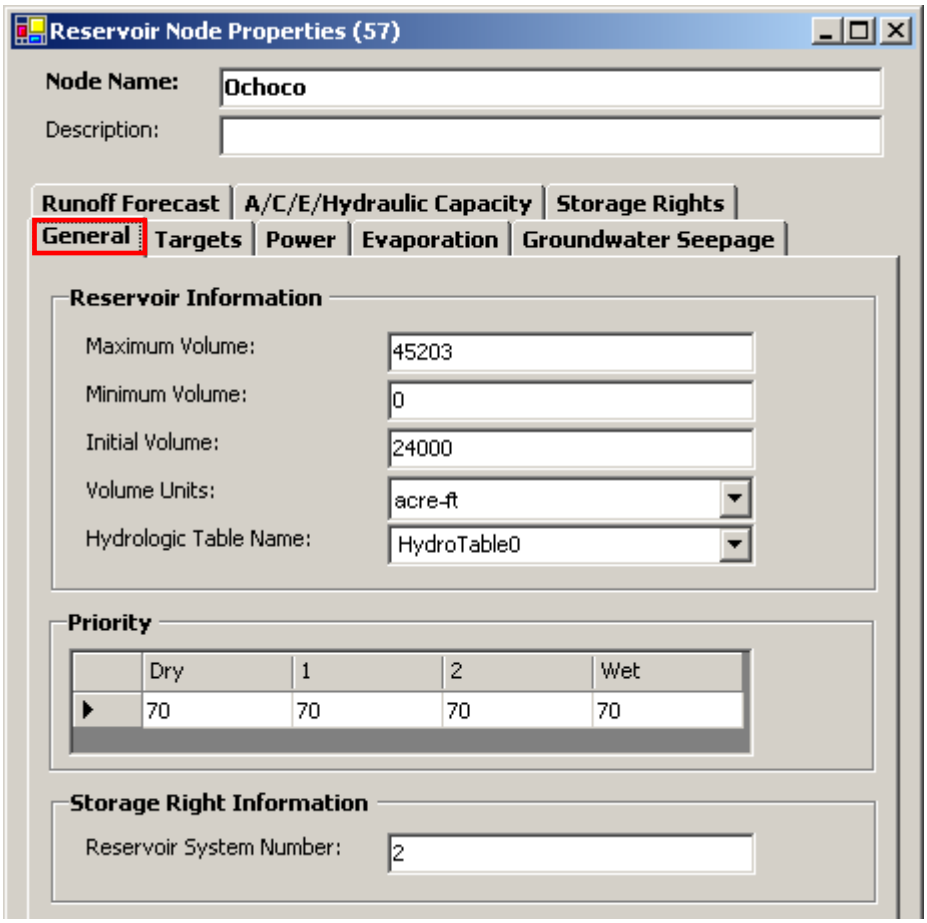
allowing users to enter a unique *Right Name* for that storage right, the *Account Volume* of the account, and the *Water Rights Date*.

After all storage rights and natural flow rights have been created for the network, the **MODSIM > Water Right Control** form is used to create the link costs assigned to the accrual and natural flow links based on the *Water Rights Date* ranking. For example, the storage right *Prineville_accrual*, has a *Cost* field that is not directly entered by the user, but rather is automatically generated by the **Water Right Control** form after all water rights have been created with associated decree dates. Since the *Water Rights Date* of 4/8/1914 for the *Prineville_accrual* water right is junior to several natural flow rights in the Crooked River system, downstream water rights senior to this fill right are not available for storage right accrual, but instead bypass the *Prineville* node for downstream delivery. The *Account Volume* field allows the reservoir to fill through the accrual link to its maximum capacity of 148,633 ac-ft for each accrual season (i.e., if the water is available in right from the inflow node *Prine1*.

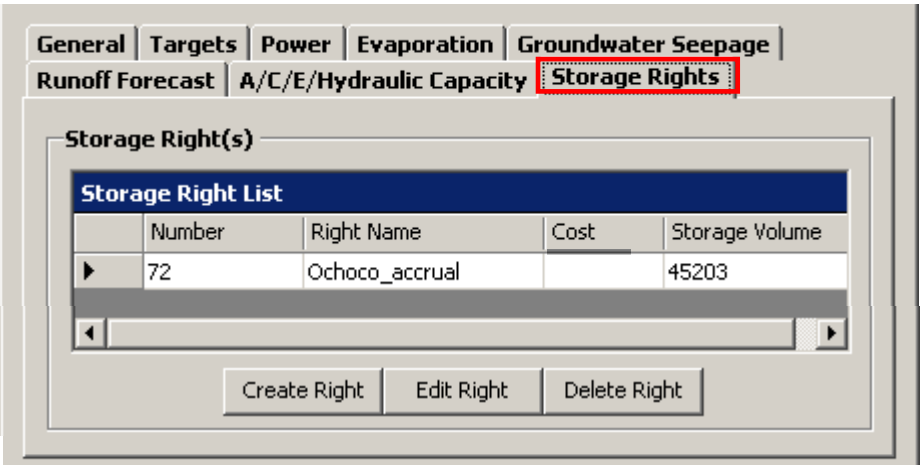
Three spaceholders share in accrual to storage account *Prineville_accrual*: Ochoco Irrigation District (OID), other irrigators (nonOID), and uncontracted space (Reclamation). In the model, accrual is distributed proportionally to these accounts as water is supplied through the accrual link. Reclamation requests water to meet an instream flow demand at the flow-through demand node *75cfs*. Spaceholder OID requests water to supply irrigation demand nodes *CRFeed1*, *LBOchCr2*, *OchMain4*, *Dist*, and *LBRye*, whereas spaceholder nonOID supplies irrigation demand nodes *upperCR*, *RiceB*, *Lowline*, *CRFeed0*, and *LBRye1*. Reclamation's ownership is recognized in the by settings on the link supplying its single demand node *75cfs*. However, since spaceholders OID and nonOID distribute accrued storage to several demand nodes, the space to be shared among the demands must be specified. OID and nonOID are therefore identified as group ownerships by clicking the **Group Ownerships** tab on the **Storage Right** form, allowing users to enter any number of group ownerships. *Group Number* 1, with a total size of 10,375 ac-ft, represents nonOID ownership space, whereas *Group Number* 2, with a total size of 57,899 ac-ft, represents OID ownership space.

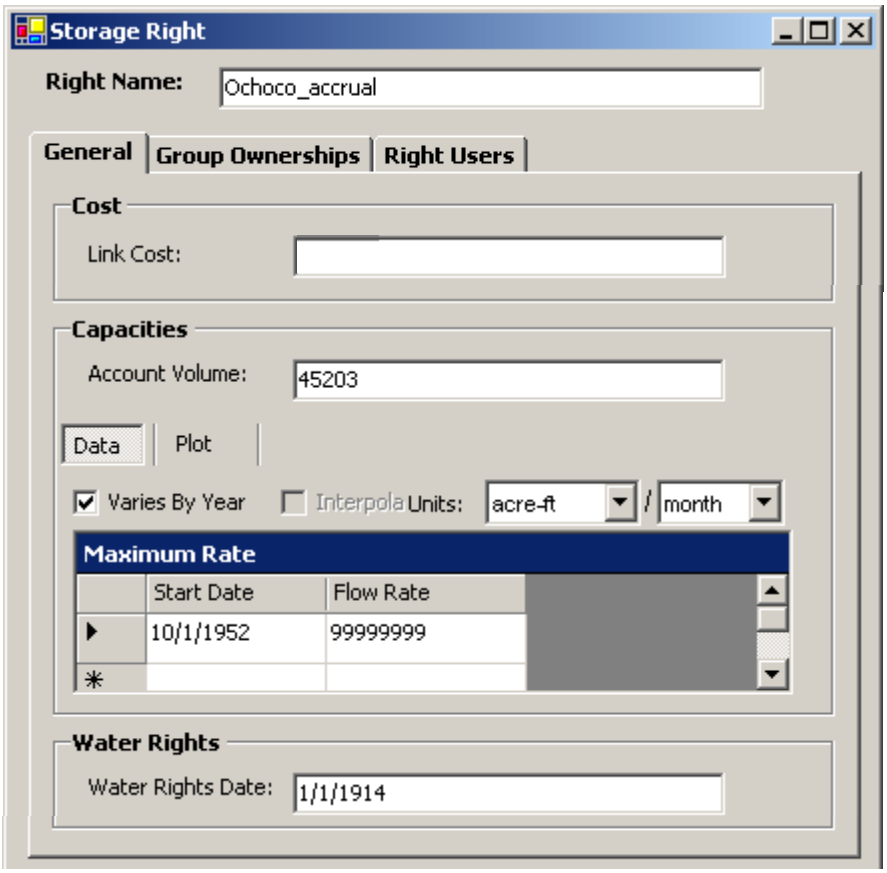


Ochoco Reservoir. Data for Ochoco Reservoir are similarly entered into the **General** tab of the **Reservoir Node Properties** form, including the higher storage *Priority* of 70 (in relation to Prineville Reservoir) and *Reservoir System Number* 2 as distinct from that of Prineville Reservoir. Target storage levels conditioned on hydrologic states defined by *HydroTable0* are entered into fields under the **Targets** tab.

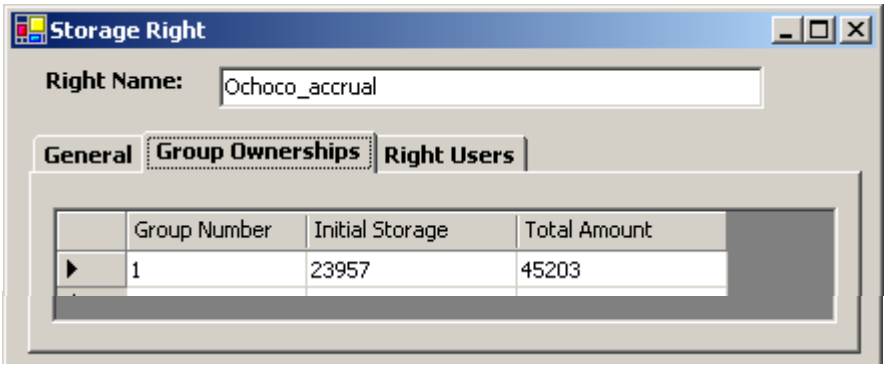


Storage right *Ochoco_accrual* is created under the **Storage Rights** tab and clicking the *Create Right* button for an account volume of 45203 ac-ft and water right date 1/1/1914. Again, *Link Cost* will be generated subsequently using the **Water Right Control** form.



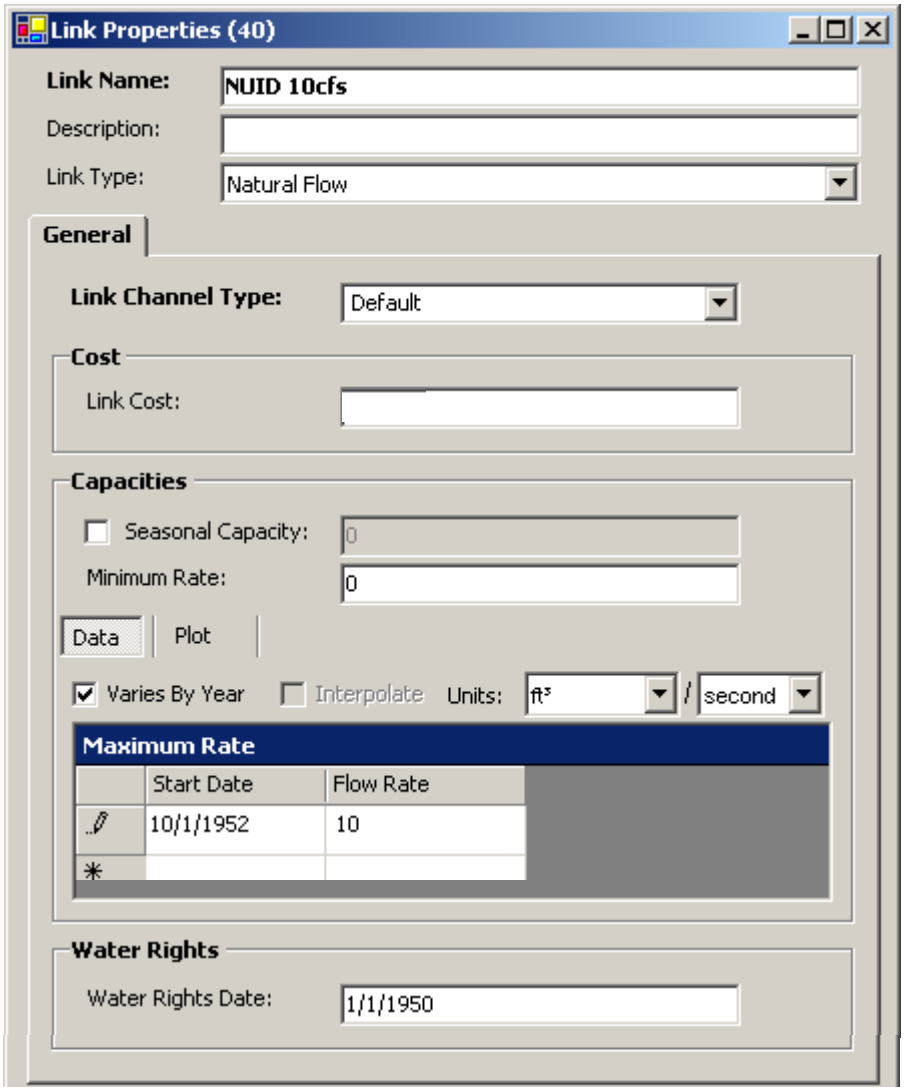


Although only one spaceholder owns storage accrued to Ochoco Reservoir, it is shared by more than one demand node, thereby requiring a group ownership. The **Group Ownership** tab under the **Storage Right** form shows one entry in the ownerships table with a *Total Amount* of 45203 ac-ft, corresponding to the capacity of the reservoir.

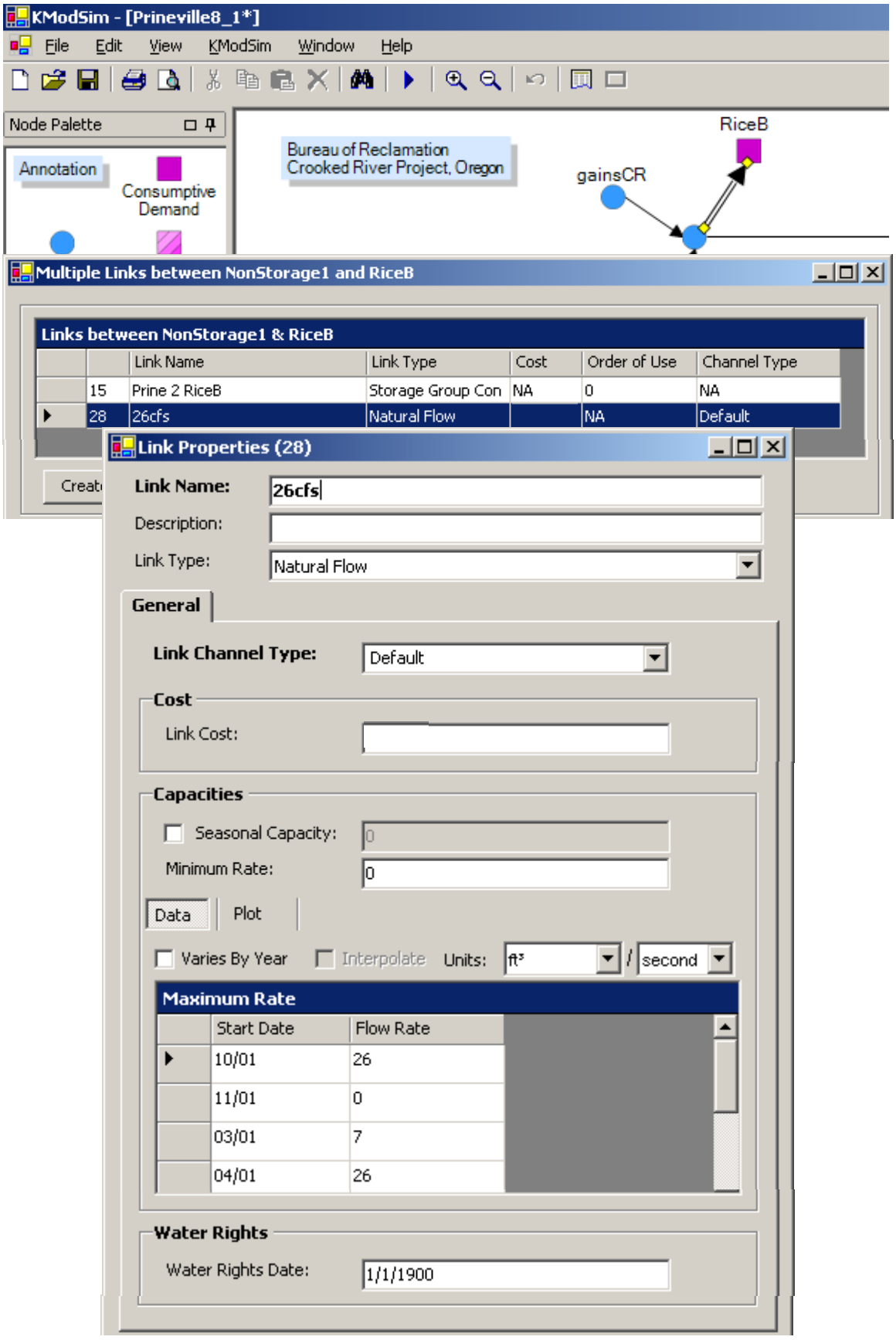


B.4 Natural Flow Rights

Demand nodes *10cfsNU*, *minNUID*, and *NUIDDes* are all served by natural flow rights only, whereas demands *75cfs*, *160cfs*, *RiceB*, and *LowLine* have both natural flow rights as well as shared ownerships in the storage accounts. For example, link *NUID 10cfs* conveying natural flow to the demand node *10cfsNU* has a **Water Rights Date** of 1/1/1950 and is designated with a **Link Type** as *Natural Flow*.



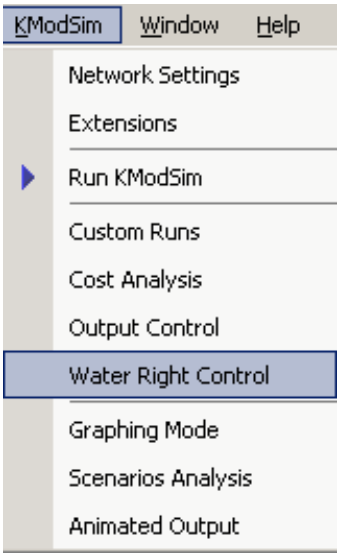
Selecting the multilink diverting flow to the demand *RiceB* displays the **Multiple Links** form showing the *Natural Flow* link along with a *Storage Group Ownership* link. Although the link is already created in this example network, the user would click the **Create Link** button, which opens the **Link Properties** form, allowing specification of the **Link Name** as *26cfs*, the **Link Type** as *Natural Flow*, the decreed maximum flow rate of 26 ft^3/sec represented as a variable capacity link, and the *Water Rights Date* of 1/1/1990. Again, the **Link Cost** will be automatically generated using **Water Right Control** based on the water rights dates after all the natural flow links have been created.

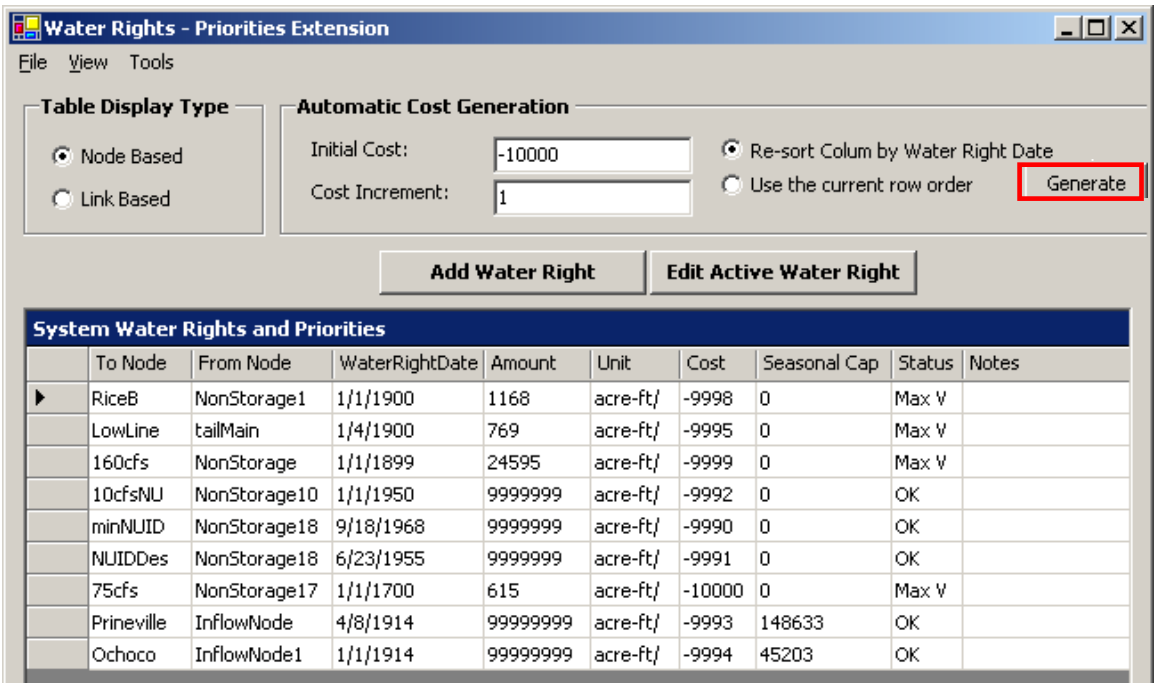

KModSim - [Prineville8_1*]
File Edit View KModSim Window Help
Node Palette
Annotation
Consumptive Demand
Bureau of Reclamation
Crooked River Project, Oregon
RiceB
gainsCR
Multiple Links between NonStorage1 and RiceB
Links between NonStorage1 & RiceB
Link Name
Link Type
Cost
Order of Use
Channel Type
15 Prine 2 RiceB Storage Group Con NA 0 NA
28 26cfs Natural Flow NA Default
Link Properties (28)
Link Name: 26cfs
Description:
Link Type: Natural Flow
General
Link Channel Type: Default
Cost
Link Cost:
Capacities
Seasonal Capacity: 0
Minimum Rate: 0
Data
Plot
Varies By Year
Interpolate
Units: ft³ / second
Maximum Rate
Start Date
Flow Rate
10/01 26
11/01 0
03/01 7
04/01 26
Water Rights
Water Rights Date: 1/1/1900

The link serving the demand *jrWR* at the downstream end of the network is also a natural flow link. It does not represent a specific water right, but instead is used to force unclaimed natural flows out of the system for delivery downstream during the natural flow step. If these flows were not claimed during the natural flow step, they would be available during the storage step and could be distributed through storage ownership links. Although this may be appropriate in some systems, the modeler needs to be aware that if the downstream *jrWR* demand is not included, not only is excess natural flow not passed downstream to users beyond the scope of the model, but excess natural flow distributed through an ownership link in the storage step is debited from that storage account. In this case, a small negative **Link Cost** of -1 is entered manually, although a very junior water right date could also have been applied.

The link serving the demand *5cfs* is a natural flow link which does not represent a water right. Ochoco Dam leaks at least 5 cfs consistently, so this demand forces 5 cfs through the dam in every time step. A very low unit cost of -90,000 ensures that this link receives water in the natural flow step. If the demand is still not satisfied, the link competes with the artificial target storage link on Ochoco Reservoir (which has a cost on the order of -70,000) so that the link can pull water out of storage (without an ownership).

From menu option **MODSIM > Water Right Control**, the **Generate** button is used to automatically produce the costs on natural flow links with entry of a *Water Rights Date*, including the storage accrual links. For this example, unit costs for these links are arbitrarily selected to be within an order of magnitude of -10,000, with a cost increment of 1 used to rank the link costs. Natural flow links with decree amounts entered as variable capacities in the **Maximum Rate** field are labeled as *Max V* in the **Status** column. Alternatively, decree amounts can be entered in the **Seasonal Capacity** field, or if there is only one link and it is a natural flow link, then demands entered into the **Demand Node Properties** form serve to represent the water right decree amounts.

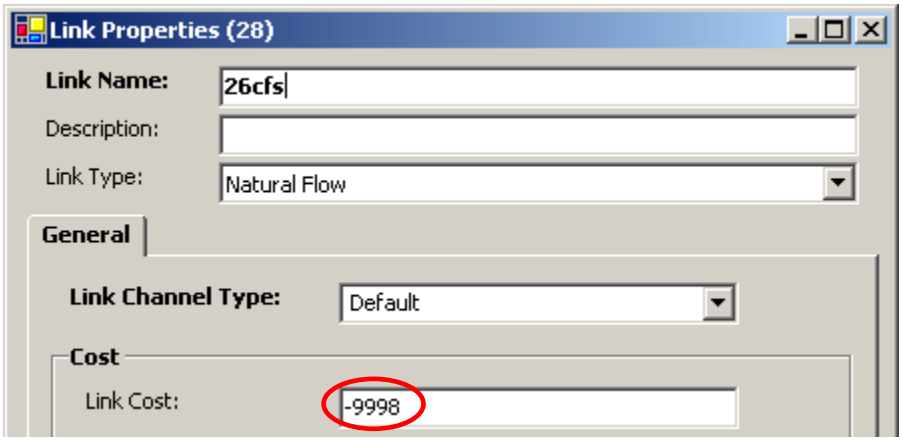




	To Node	From Node	WaterRightDate	Amount	Unit	Cost	Seasonal Cap	Status	Notes
▶	RiceB	NonStorage1	1/1/1900	1168	acre-ft/	-9998	0	Max V	
	LowLine	tailMain	1/4/1900	769	acre-ft/	-9995	0	Max V	
	160cfs	NonStorage	1/1/1899	24595	acre-ft/	-9999	0	Max V	
	10cfsNU	NonStorage10	1/1/1950	9999999	acre-ft/	-9992	0	OK	
	minNUID	NonStorage18	9/18/1968	9999999	acre-ft/	-9990	0	OK	
	NUIDDes	NonStorage18	6/23/1955	9999999	acre-ft/	-9991	0	OK	
	75cfs	NonStorage17	1/1/1700	615	acre-ft/	-10000	0	Max V	
	Prineville	InflowNode	4/8/1914	99999999	acre-ft/	-9993	148633	OK	
	Ochoco	InflowNode1	1/1/1914	99999999	acre-ft/	-9994	45203	OK	

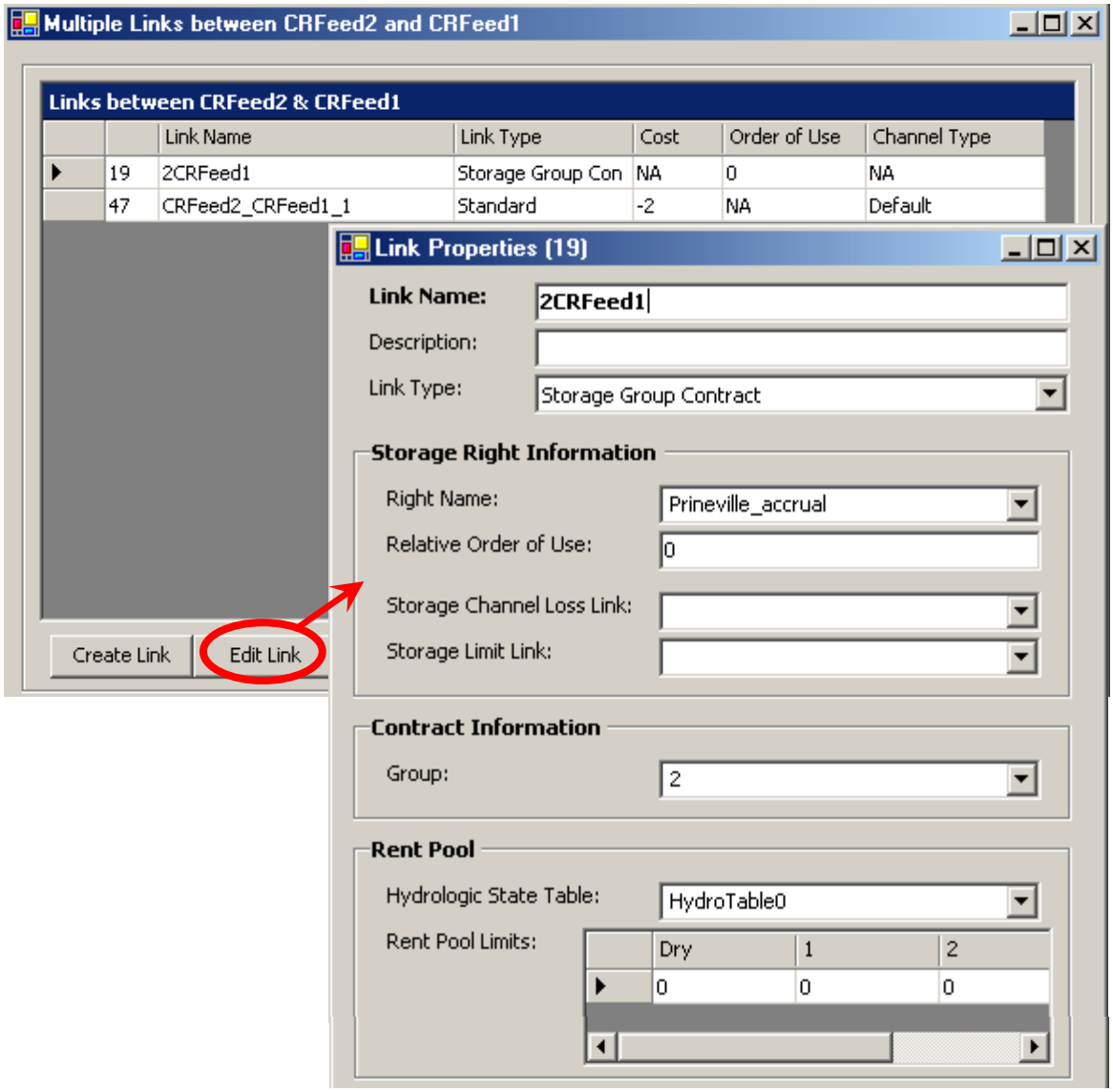
For this example, all demands retain the default **Priority** value of 100 since the water right priorities are established based on the assignment of link costs. Rather than specifying a water rights date, the natural flow links serving demands *CRFeed1*, *Dist* and *OchMain4* are assigned unit costs of -2, -3, and -4, respectively, which are small enough to avoid competition with the other natural flow links having costs on the order of -10,000. However, these links are able to receive water during the natural flow step since the flow-through demand *160cfs* for instream flow uses has a senior priority that "pulls" flow to where *CrFeed1*, *Dist* and *OchMain* can divert water during the natural flow step.

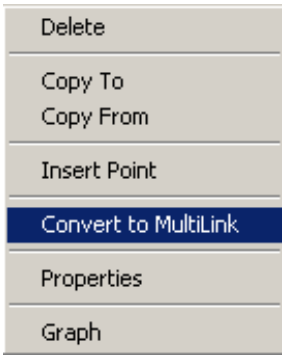
Following generation of link costs using **Water Right Control,** the *Link Cost* is now included in the **Link Properties** form, as shown here for the natural flow link *26cfs* connected to demand *RiceB*.



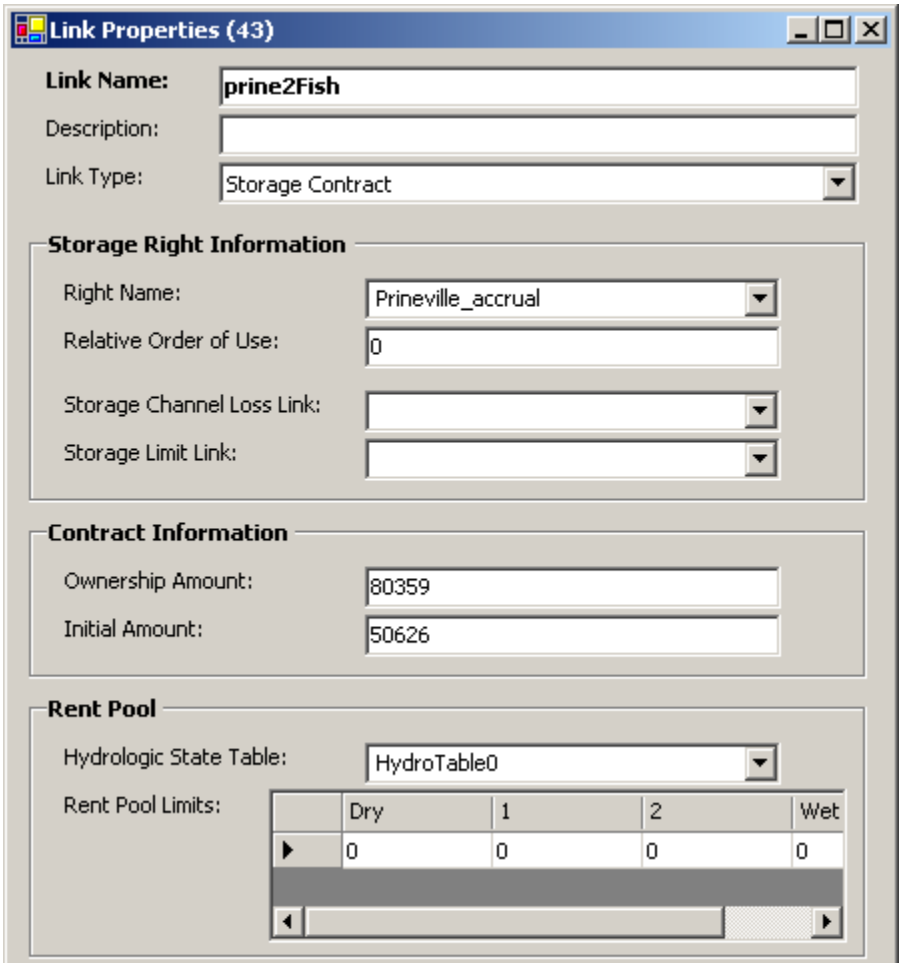
B.5 Storage Contracts and Group Ownerships

In the Prineville/Crooked River model, most of the demand nodes which are supplied by natural flow links also have storage ownership links representing their storage contracts. If the demands cannot be satisfied by natural flows alone, the ownership links supplying the demands are opened and water is released from the reservoirs to meet that need. For example, consider the multilink structure connecting nodes *CRFeed2* and *CRFeed1* as seen in Fig. E.2. Opening the **Link Properties** form for the multilink displays rows representing each link connecting the two nodes. The link *2CRFeed1* is a S*torage Group Contract* link which when edited allows entry of the **Link Name** of the ownership link, **Link Type** as *Storage Group Contract*, the **Right Name** as *Prineville_accrual*, and the *Group* number associated with the storage right ownership.

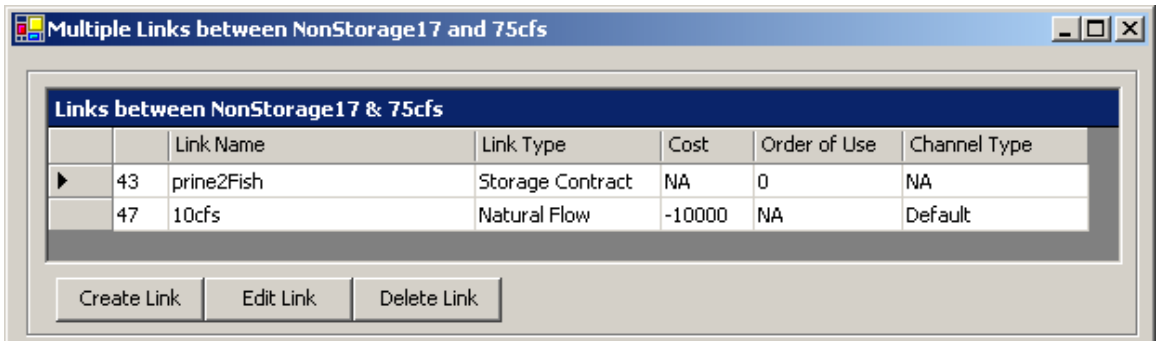




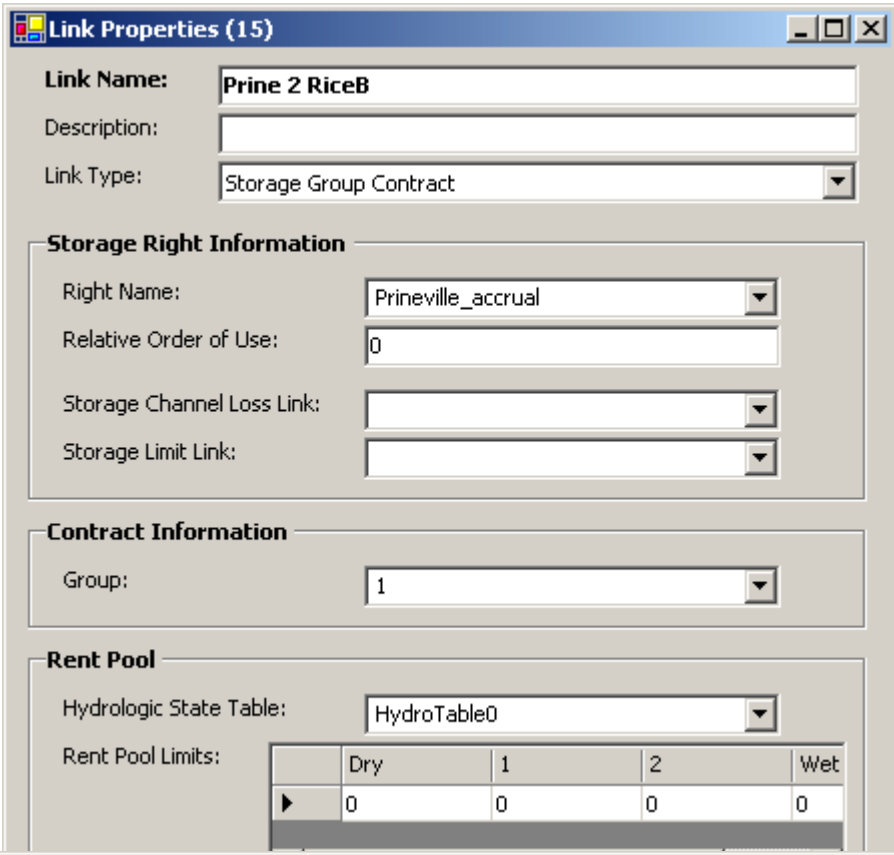
The Bureau of Reclamation owns the uncontracted space in Prineville Reservoir, which is used to supply a 75 cfs minimum instream flow downstream of the dam (i.e., flow through demand node *75cfs*). The storage ownership link supplying this demand is created by rt. mouse click on the natural flow link *10cfs*, clicking **Convert to MultiLink** on the context menu, clicking the **Create Link** button on the **Multiple Links** form, and then entering the required data in the **Link Properties** form. The **Link Name** is *prine2Fish* and the **Link Type** is *Storage Contract*. The **Ownership Amount** field shows how much space is owned, 80,359 ac-ft., with an **Initial Amount** of 50626 carried over from the previous season. Although *prine2Fish* is a storage ownership link, it is not a group ownership link since the spaceholder only requests water to serve a single demand node.

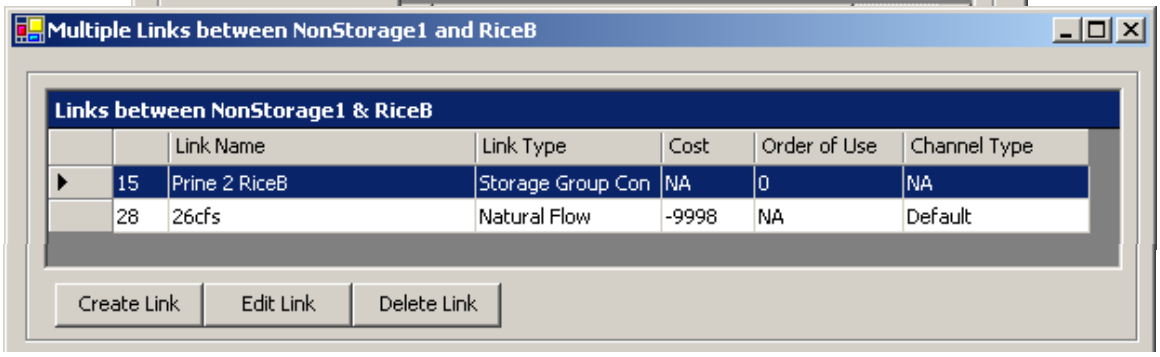


Double mouse click on the multilink now created for delivery to the flow-through demand *75cfs* reveals that the connected is represented by two links: the **Storage Contract Link** *prine2Fish* and the **Natural Flow** link *10cfs*.



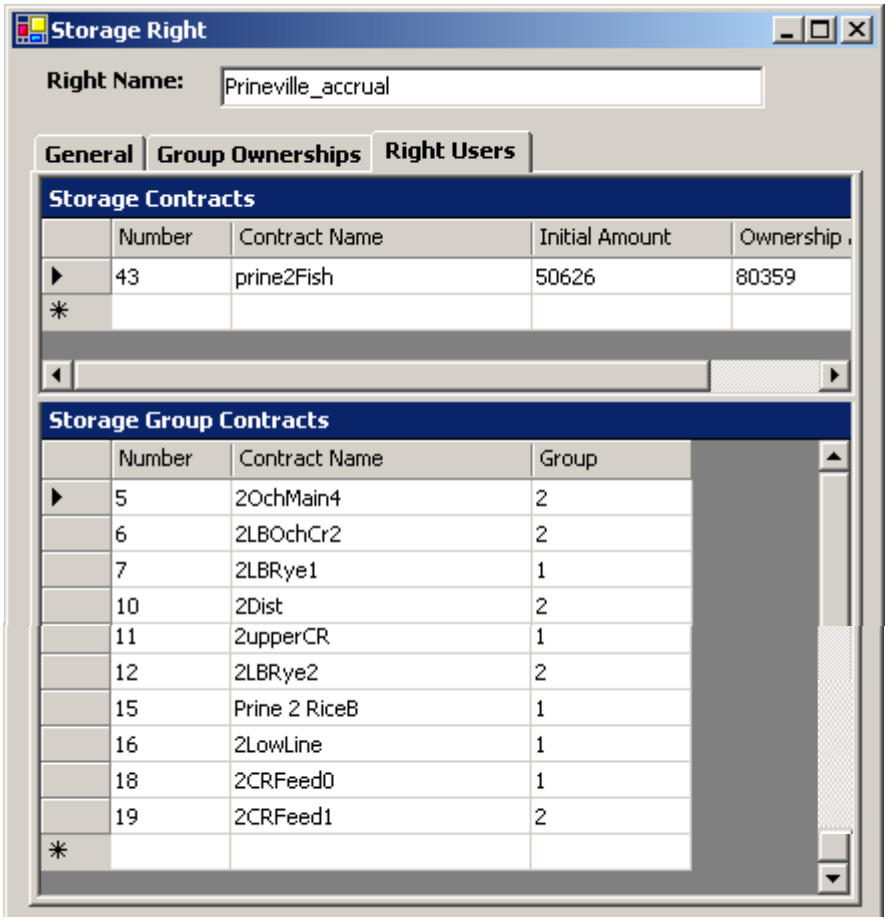
Demand nodes *LowLine*, *RiceB*, *LBRye1*, *upperCR*, *CRFeed0* share 10,375 ac-ft of space in Prineville Reservoir. For example, converting the link conveying flow to *RiceB* to a multilink, opening the **Multiple Links** form, and clicking **Create Link** displays the **Link Properties** form for creation of link *Prine 2 Rice* of **Link Type** *Storage Group Contract* belonging to *Group* 1 of the *Prineville_accrual* storage right.





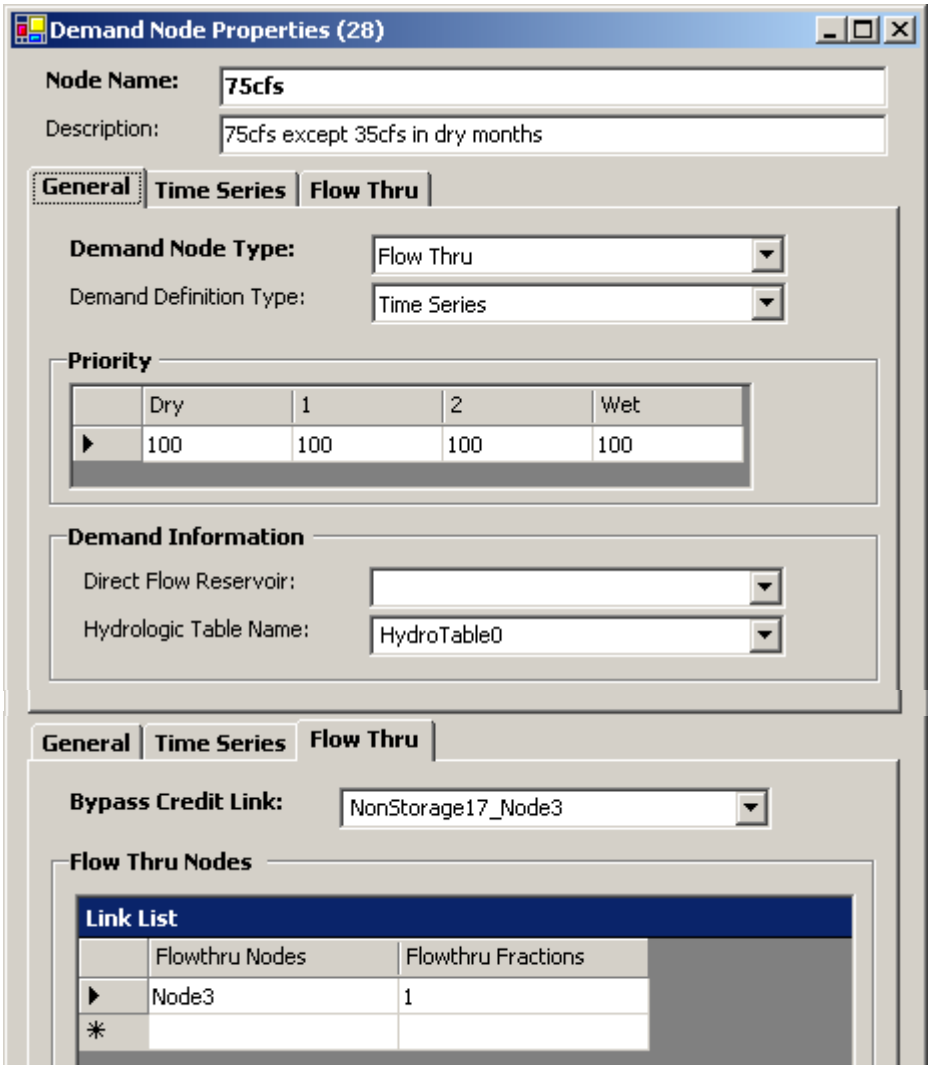
Demand nodes *OchMain4*, *OchCr2*, *LBDist1*, *LBRye2*, *CRFeed1* share 57,899 ac-ft of space in Prineville Reservoir. They are group ownerships belonging to *Group* 2 of the *Prineville_accrual* storage right. Following the same procedure, Storage Contract links to these demands are similarly created. Demand nodes *OchMain4*, *LBRye2*, *LBOchCr1*, *LBOchCr2*, *OchMain1* share the total space in Ochoco Reservoir, 45,203 ac-ft. This space is identified in the *Ochoco_accrual* storage right as *Group* 1, and Storage Contract links are also created for conveying the shared storage ownership to these demands.

After creation of all Storage Contract links and all Storage Group Contract links, clicking the **Right Users** tab under the **Storage Right** tab of the **Reservoir Node Properties** form for the *Prineville* storage node lists the Storage Contract and all Storage Group Contracts associated with *Prineville_accrual*. The list of all Storage Group Contracts for *Ochoco_accrual* can be similarly accessed.

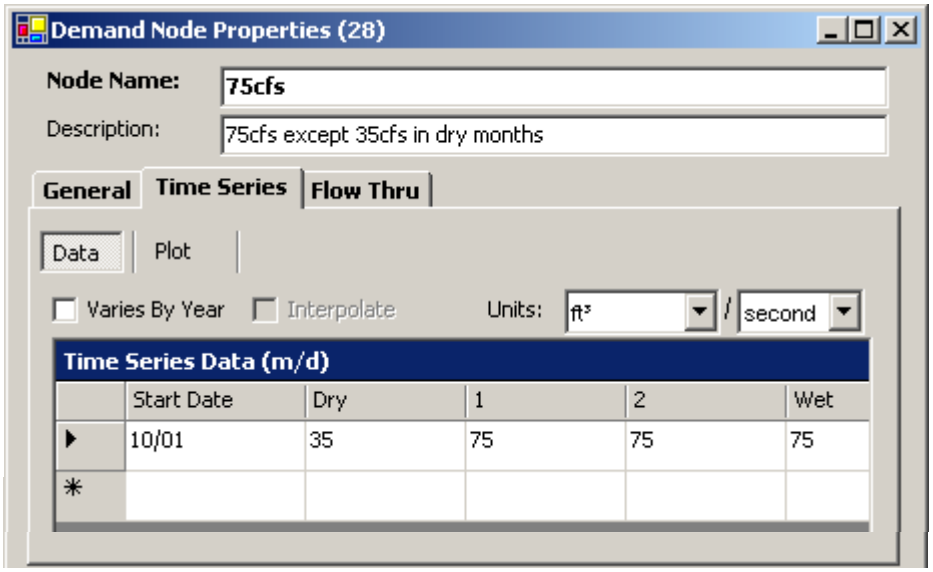


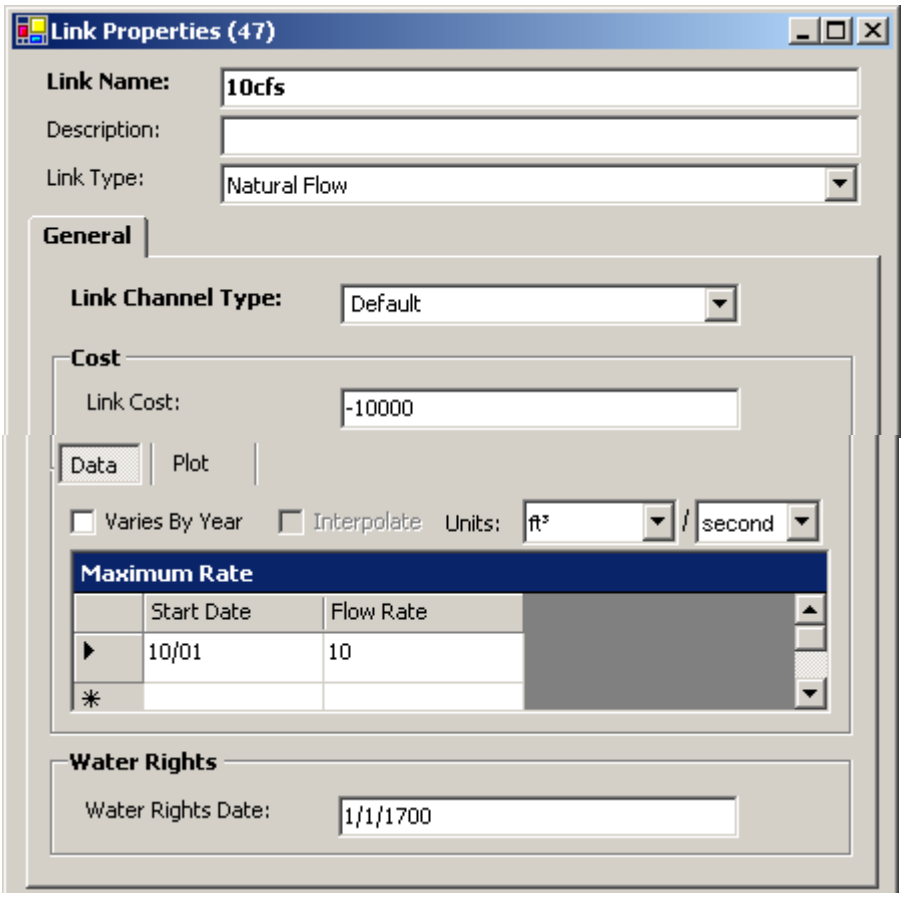
B.6 Flow-Through Demands and Pumped Flow Distribution

The flow-through demand node *75cfs* represents a 75 cfs minimum instream flow request below Prineville Reservoir. Opening the **Demand Node Properties** form for *75cfs* allows specification of the **Demand Node Type** as *Flow Thru*. Water flowing in the reach below Prineville Reservoir for purposes other than the 75 cfs demand passes through the **Bypass Credit Link**, identified by selecting link *Nonstorage17_Node3* from the dropdown list under the **Flow Thru** tab. *Node3* is identified as the node to which the flow-through demand is accruing. Although the flow-through demand is represented as a diversion, the total flow is returned downstream to represent a nonconsumptive or instream flow use. The flow-through demand for the remainder of the 75 cfs (i.e., after flows in the bypass credit link are subtracted) are satisfied using a 10 cfs natural flow right and a *prine2Fish* storage ownership in Prineville Reservoir. In the natural flow step, water flows in priority through the natural flow link *10cfs* up to its capacity. In the storage step, the remaining request is satisfied through the storage ownership link *prine2Fish*, if water is available in the account balance.

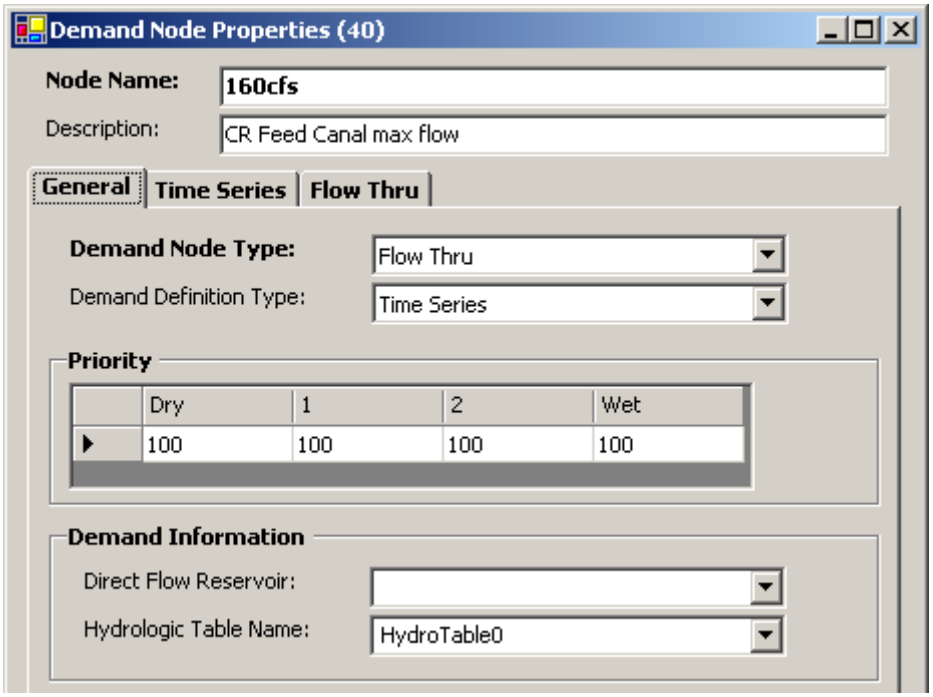


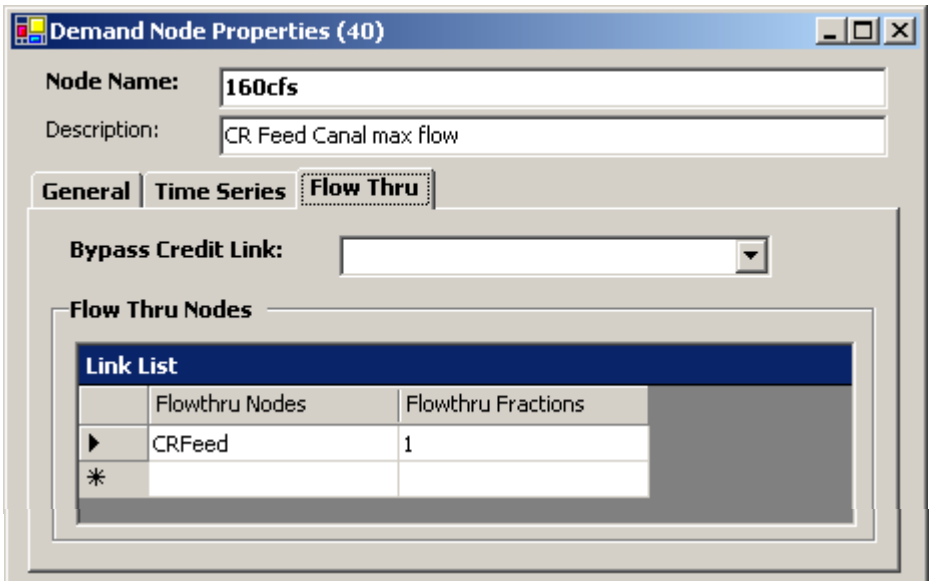
The demand time series fields in *75cfs* allow demands to be entered in units of ft^3/sec, with MODSIM automatically converting the units to acre-feet per month for purposes of the network simulation. The demand is 75 cfs in most months, but with 35 cfs for dry months. Again, the priority on the demand node *75cfs*, as with all the other demand nodes in this example, is 100, which is considered a neutral priority. Neutral priorities are preferable in this example since the distribution of water is driven by the costs on the Canal at all times.

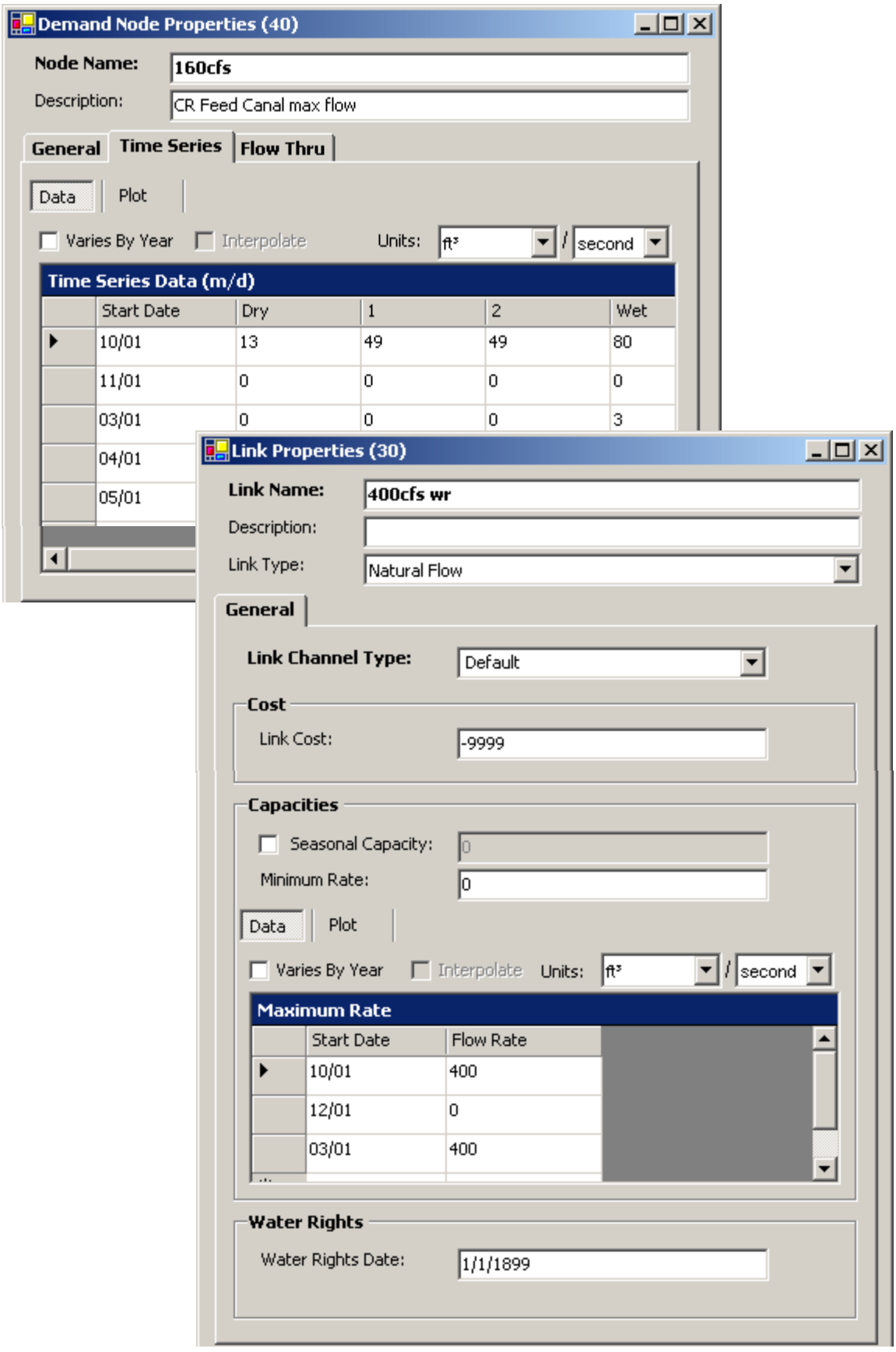




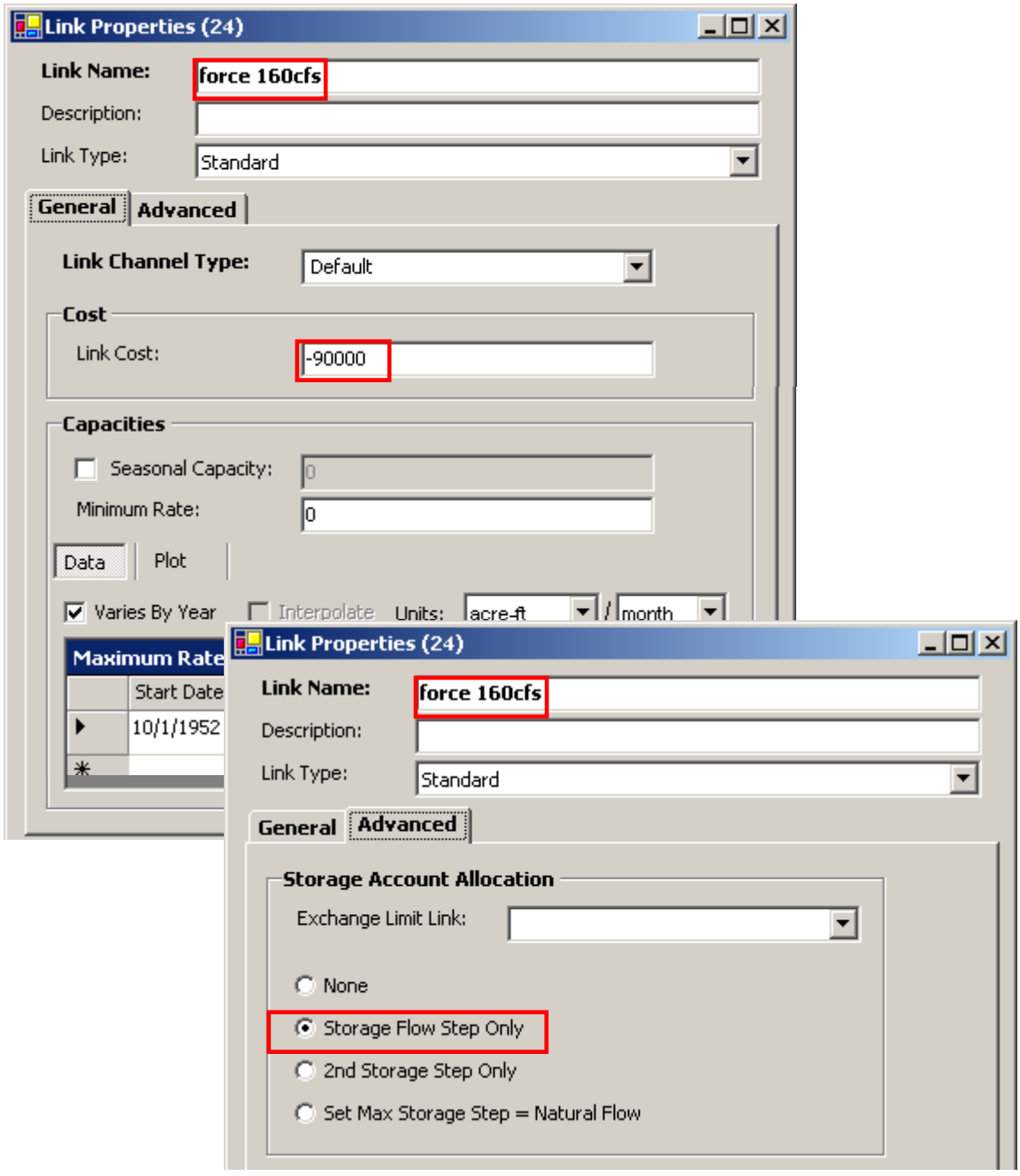
The *160cfs* demand node draws water from the Crooked River with a very senior natural flow right, represented by the link *400cfs wr*. This flow-through demand node behaves similarly to the *75cfs* flow-through demand node, but represents a different type of request. Water is pumped is pumped from the Crooked River to serve lands represented by demand nodes *CRFeed0*, *CRFeed1*, *Dist*, *OchMain4*, *LBRye1*, and *LBRye2*. Once this water flows through to node *CRFeed* at the head of the feeder canal, both natural flow and storage ownerships are distributed lands. Demands *CRFeed1*, *Dist* and *OchMain4* are served by natural flow links which are arbitrarily assigned units costs -2, -3 and -4, respectively. Notice that these natural flow links do not compete directly for natural flow in the Crooked River, but rather compete for available natural flows that have been pumped to node *CRFeed*. In order to maintain operation of the pumps at maximum efficiency, 160 cfs is required through the Crooked River Feed





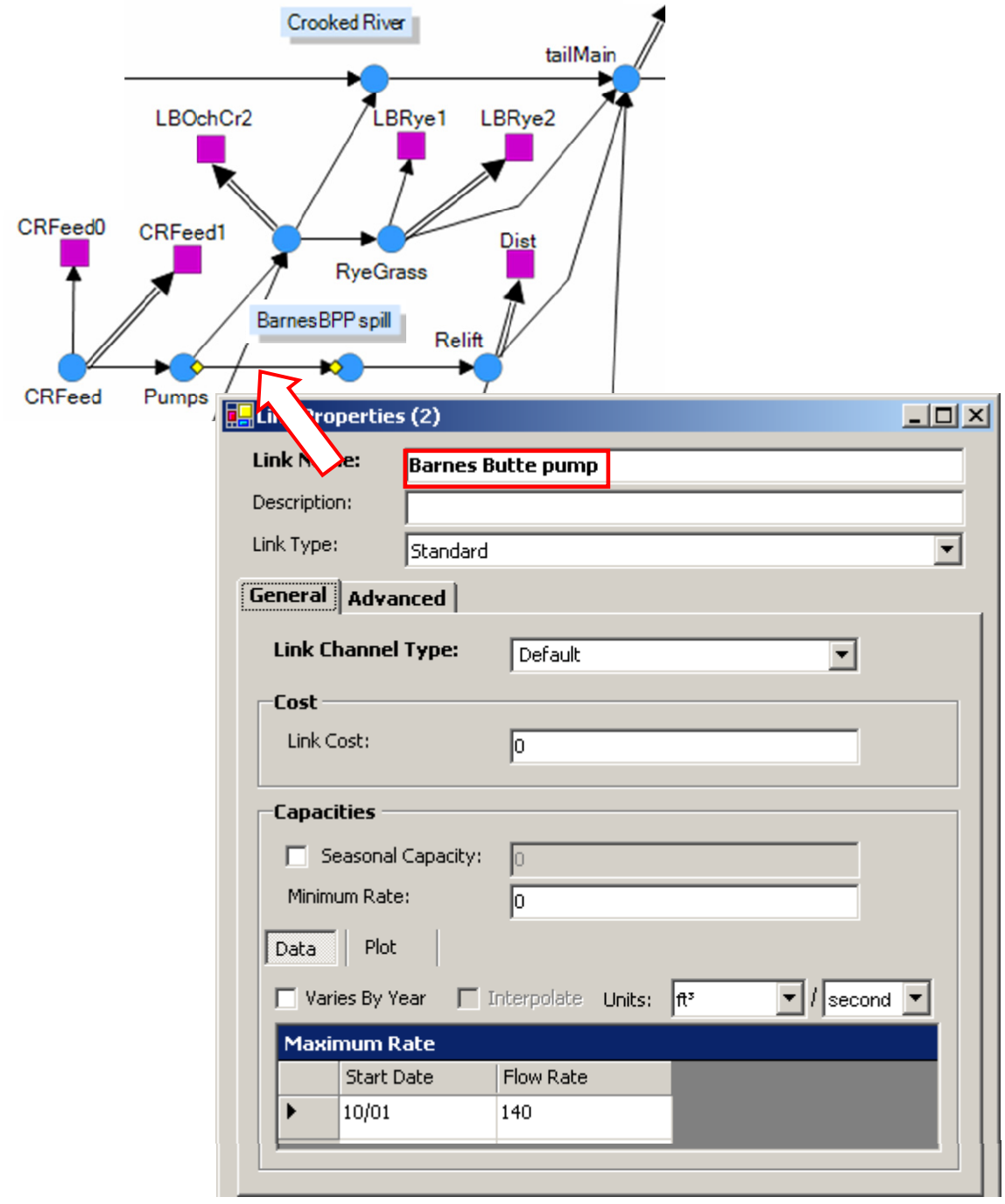

Demand Node Properties (40)
Node Name:
160cfs
Description:
CR Feed Canal max flow
General
Time Series
Flow Thru
Data
Plot
Varies By Year
Interpolate
Units:
ft³
/
second
Time Series Data (m/d)
Start Date
Dry
1
2
Wet
10/01
13
49
49
80
11/01
0
0
0
0
03/01
0
0
0
3
04/01
05/01
Link Properties (30)
Link Name:
400cfs wr
Description:
Link Type:
Natural Flow
General
Link Channel Type:
Default
Cost
Link Cost:
-9999
Capacities
Seasonal Capacity:
0
Minimum Rate:
0
Data
Plot
Varies By Year
Interpolate
Units:
ft³
/
second
Maximum Rate
Start Date
Flow Rate
10/01
400
12/01
0
03/01
400
Water Rights
Water Rights Date:
1/1/1899

If the natural flow right is insufficient to meet the 160 cfs request, link *force160cfs* draws water out of the reservoir with a *Link Cost* of -90,000 during the Storage Step if the radio button next to *Storage Flow Step Only* under the **Advanced** tab of the **Link Properties** form is checked. The link *force160cfs* competes successfully for reservoir storage since its cost is numerically less than that of the artificial target storage link cost for Prineville, which is on the order of -70,000. Since the *force160cfs* link is not a storage ownership link, flow in this link is not debited from a storage account. Once it flows through to node *CRFeed* at the head of the feed canal, it is distributed through the storage ownership links serving the demand nodes mentioned above, and then is debited from the appropriate accounts.



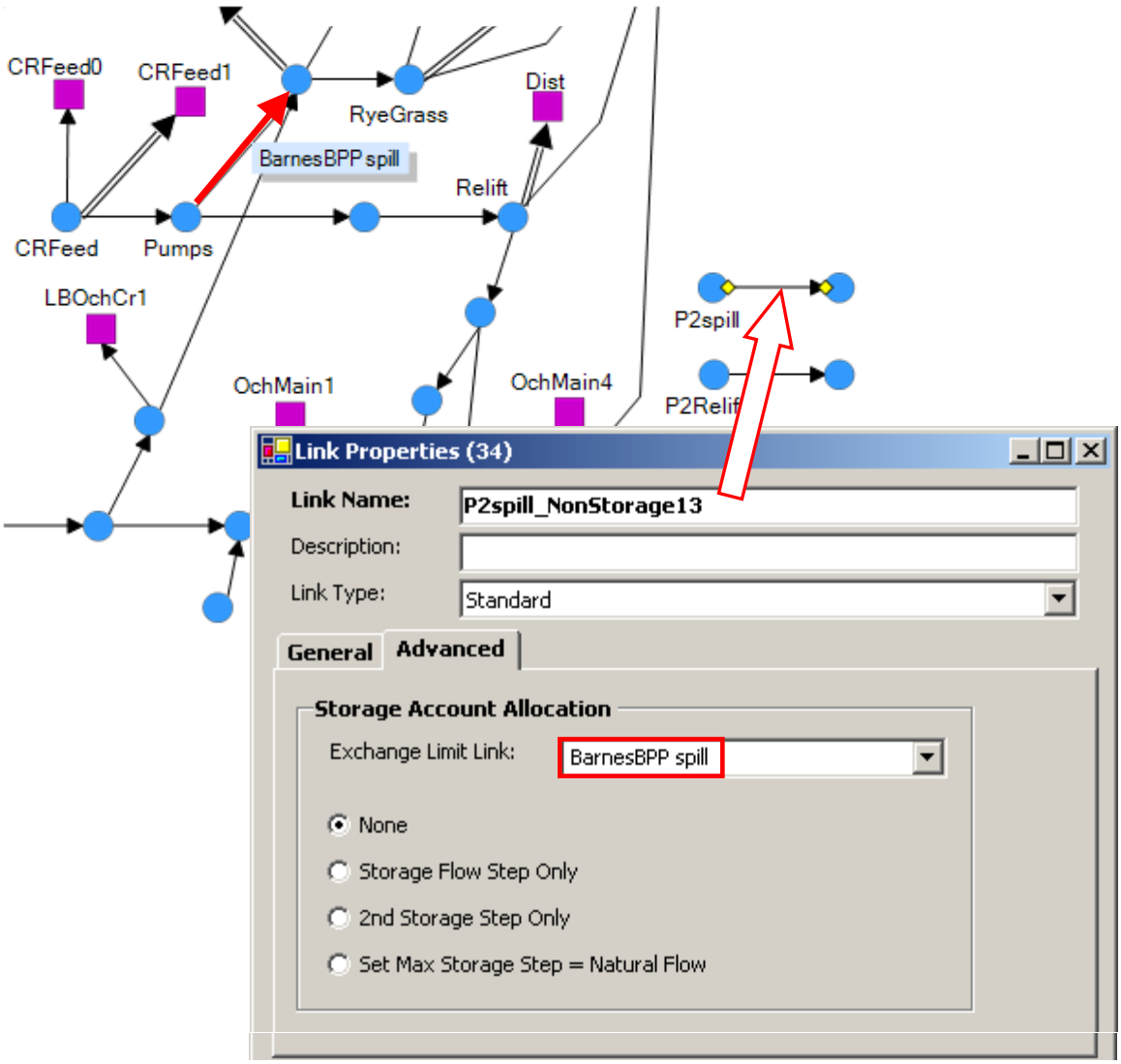
B.7 Special Constructs, Exchange Limit Links, and Storage Limit Links

Although the Crooked River Feed Canal requests 160 cfs to keep the pumps operating efficiently, the capacity of the pumps is actually only 140 cfs (and the requests for water in this portion of the system might be even less), so water spills back to the Crooked River from the *Pumps* node via the *BarnesBPP spill* link. This water, although spilled, is still considered *Prineville* storage water. The 140 cfs restriction is set in the Variable Capacity fields of the link *Barnes Butte pump*.

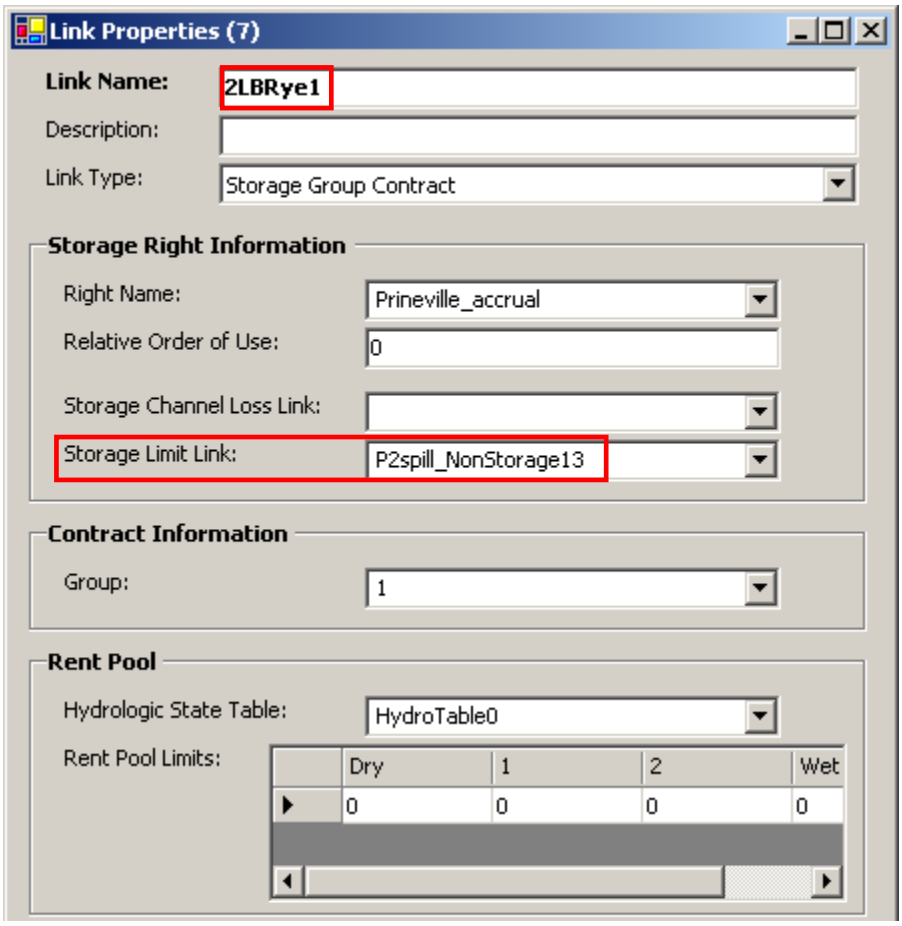


Demand nodes *LBOchCr2*, *LBRye1*, and *LBRye2* receive *Prineville* water only by utilizing this spill. But when a storage ownership link opens up during the storage step, it does not know which reservoir it should draw the water from, and simply pulls water from the reservoirs according to their priorities. It is possible, therefore, for the *Prineville* ownership links which serve these demands to draw water from *Ochoco* and that water would be debited to *Prineville*.

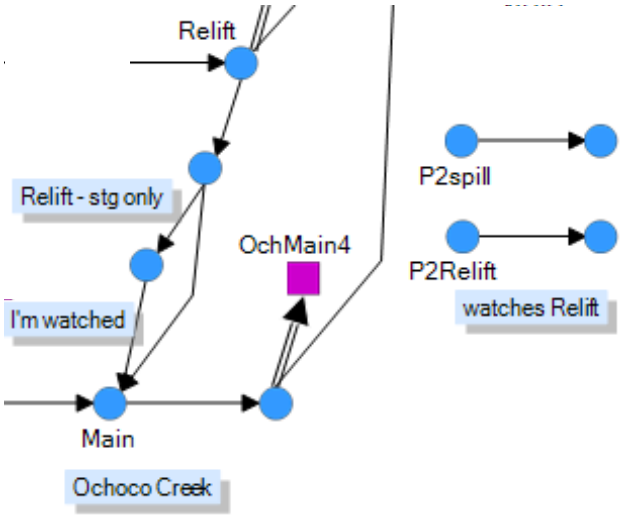
To prevent this situation, the *Prineville* storage ownership links conveying flow to demand nodes *LBOchCr2*, *LBRye1*, and *LBRye2* should be limited by the amount of spill. This is accomplished by constructing the separate *P2spill* "dummy" shown on the right hand side of the main network. As an application of "Watch Logic" in MODSIM, the link in the *P2spill* network, link *P2spill_NonStorage13*, "watches" the flow spilled through the *BarnesBPP spill* link by selecting that link as the *Exchange Limit Link* in the **Storage Account Allocation** field under the **Advanced** tab of the **Link Properties** form. The *Exchange Limit Link* is a mechanism that sets the capacity of link *P2spill_NonStorage13* to the flow in *BarnesBPP spill* link in the previous iteration. Then, the *Prineville* storage ownership links to the demand nodes *LBOchCr2*, *LBRye1*, and

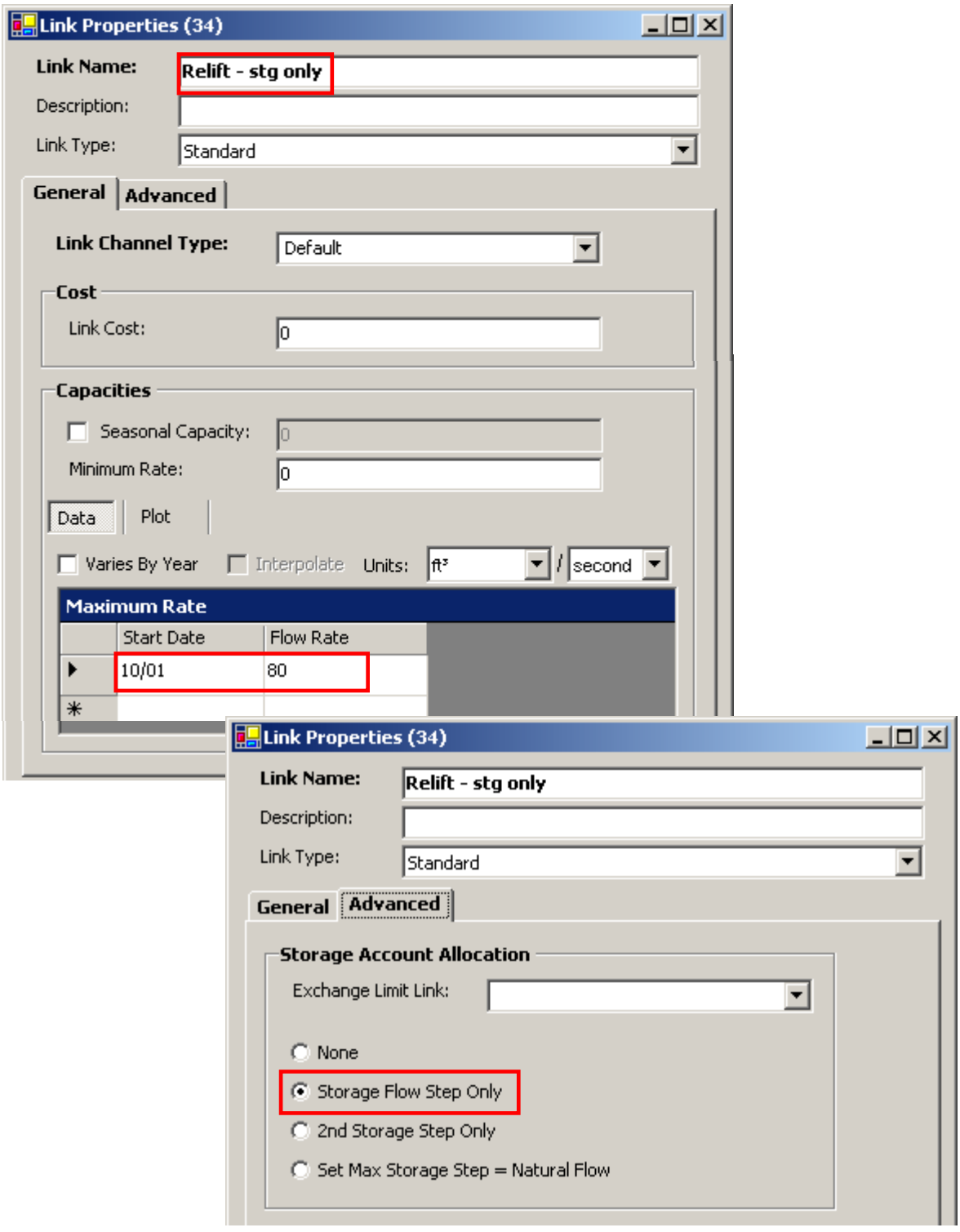


LBRye2 have their **Storage Limit Link** fields set to link *P2spill_NonStorage13*. This mechanism constrains the total capacities of all links sharing the same Storage Limit Link to the capacity of that Storage Limit Link. The net result is that the total of the flows through the Prineville storage ownerships serving the demand nodes *LBOchCr2*, *LBRye1*, and *LBRye2* cannot exceed the spill from the pumping plant.

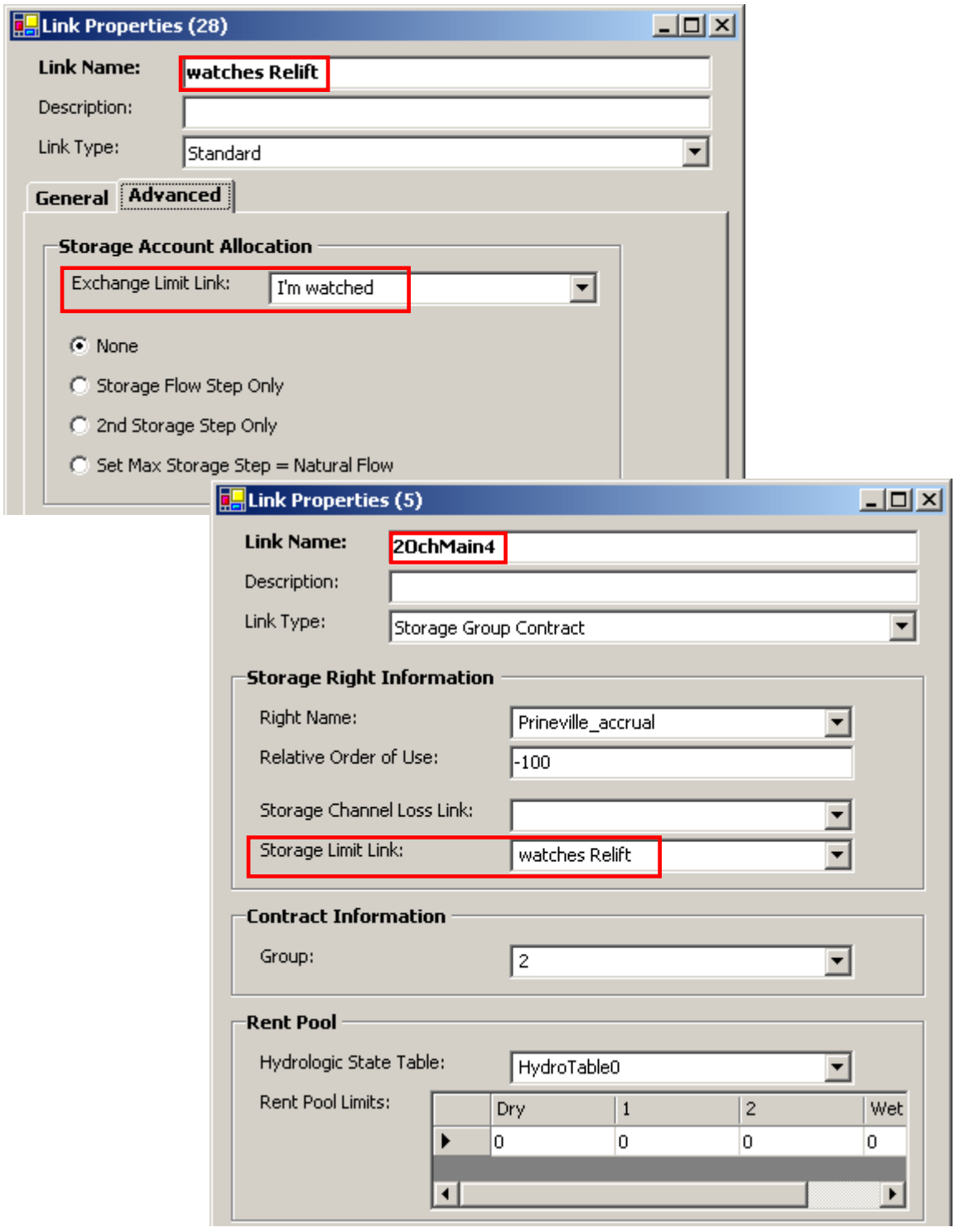


A similar situation exists for water moved by the pump at the *Relift* node to node *Main*. The demand node *OchMain4* can be served by several sources, including natural flow, storage ownership in Prineville Reservoir, and storage ownership in Ochoco Reservoir. Prineville storage water is delivered through the *Relift - stg only* link by checking the *Storage Flow Step Only* radio button. This link limits flow to the pumping capacity of 80 cfs by specification of the *Maximum Rate*.




Link Properties (34)
Link Name:
Relift - stg only
Description:
Link Type:
Standard
General
Advanced
Link Channel Type:
Default
Cost
Link Cost:
0
Capacities
Seasonal Capacity:
0
Minimum Rate:
0
Data
Plot
Varies By Year
Interpolate
Units:
ft³
/
second
Maximum Rate
Start Date
Flow Rate
10/01
80
Link Properties (34)
Link Name:
Relift - stg only
Description:
Link Type:
Standard
General
Advanced
Storage Account Allocation
Exchange Limit Link:
None
Storage Flow Step Only
2nd Storage Step Only
Set Max Storage Step = Natural Flow

This water also flows through the link *I'm watched*, which, as its name implies, is watched by the link *watches Relift* in the *P2Relift* "dummy" network by setting its **Exchange Limit Link** to *I'm watched*. The Prineville storage ownership link *2OchMain4* serving the demand *OchMain4* has its **Storage Limit Link** set to *watches Relift*, thereby limiting flow through this link to what is pumped through the *Relift* node.



TUTORIAL C
Customization of MODSIM

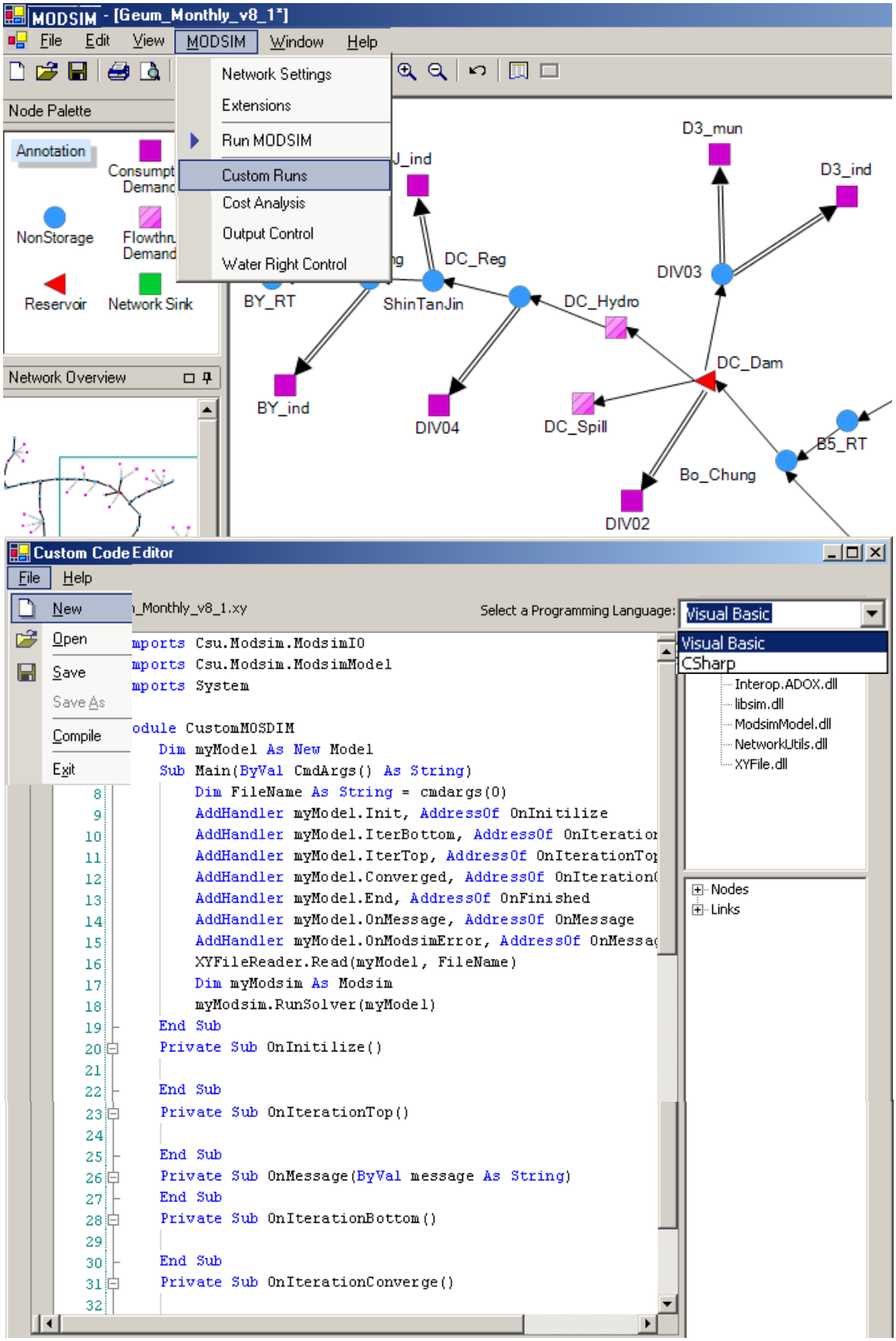
C.1 Introduction

Extensive capabilities are available for users to customize MODSIM for any specialized river basin conditions, rules, agreements, and data input and output structures. External modules may be dynamically linked to MODSIM through custom code, including user selected hydrologic models, water quality models, models predicting environmental and ecological impacts, and groundwater models. For example, de Azevedo et al. (2000) and Dai et al. (2001) linked MODSIM to the EPA QUAL2E river and stream water quality model; Fredericks et al. (1998), joined MODSIM with the USGS MODFLOW 3-dimensional finite-difference groundwater flow model through generation of stream-aquifer response functions; Campbell et al. (2001) connected MODSIM to the HEC-5Q water quality model and an Aquatic Habitat Component; and Marques et al. (2006) coupled MODSIM with a water pricing and economic evaluation module developed as custom script.

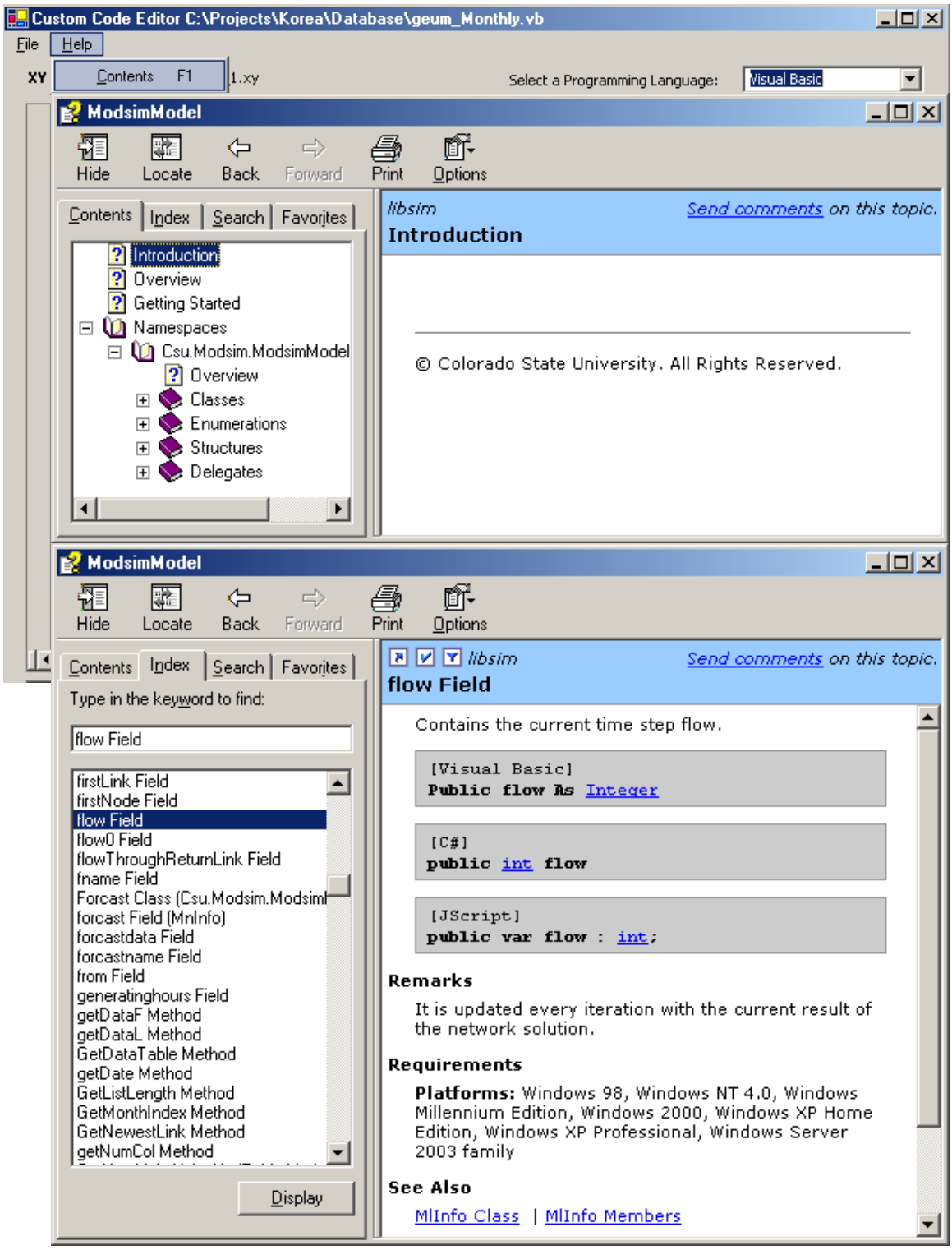
In addition to executing external modules, custom code can be developed in the several Microsoft .NET languages freely provided with the .NET Framework to perform any user desired calculations beyond the scope of MODSIM. A convenient template is provided in the Custom Code Editor for guiding users in custom code preparation. Numerous help features are included, with point and click access to any MODSIM node or link object methods. The custom code can interface with MODSIM at any desired strategic location in the computational process, including data input, execution at the start of any time step, processing at intermediate iterations, and model output. Users are provided direct access to all *public* variables, parameters, and object classes in MODSIM for specifying knowledge-based operating rules, demand forecasts, water costs and benefits, linkages with on-line database management systems, customized user interfaces (e.g., for integration with GIS), and customized output reports. The customization capabilities and flexible simulation time step structures in MODSIM allow it be utilized as a decision support system for tasks ranging from long-term water policy analysis and infrastructure planning to daily real-time river basin operations.

C.2 Tools for MODSIM Customization

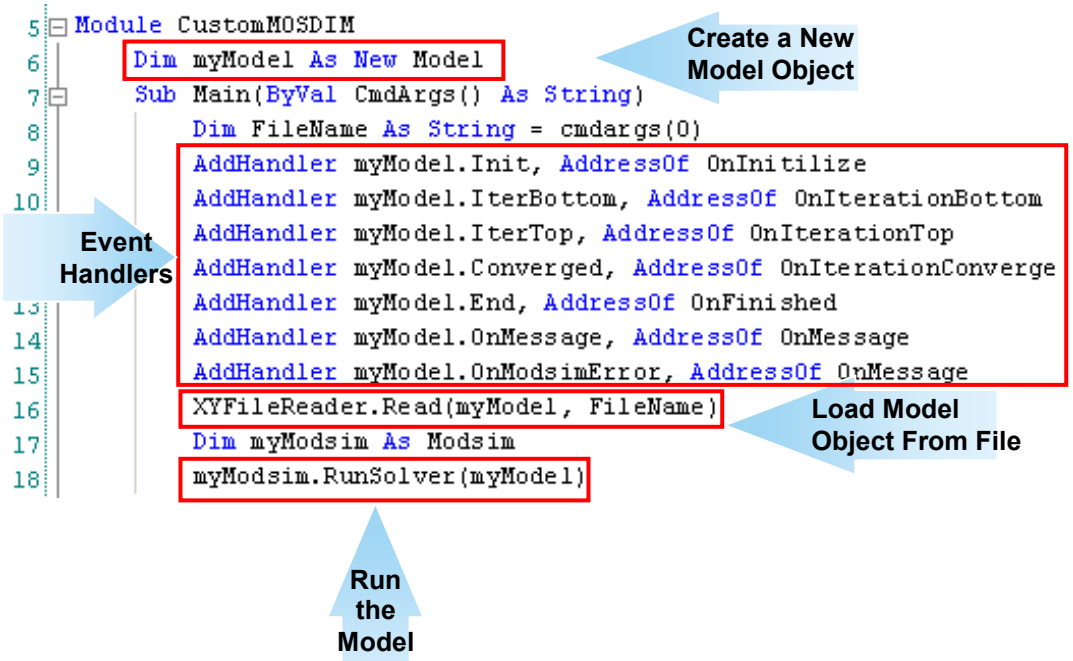
MODSIM includes extensive tools incorporated within the graphical user interface for aiding users in the development of custom code to be run concurrently with MODSIM. These tools are accessed from the MODSIM GUI by selecting **MODSIM > Custom Runs**. A **New** code template can be opened in the **Custom Code Editor**, or existing code can be loaded. Custom code can be developed in the Visual Basic .NET or C# **.**Net languages.


MODSIM - [Geum_Monthly_v8_1*]
File Edit View MODSIM Window Help
Network Settings
Extensions
Run MODSIM
Custom Runs
Cost Analysis
Output Control
Water Right Control
Node Palette
Annotation
Consumpt Demand
NonStorage
Flowthru Demand
Reservoir
Network Sink
Network Overview
D3_mun
D3_ind
DIV03
DC_Reg
DC_Hydro
DC_Dam
ShinTanJin
BY_RT
BY_ind
DIV04
DC_Spill
B5_RT
Bo_Chung
DIV02
Custom Code Editor
File Help
New
Open
Save
Save As
Compile
Exit
Select a Programming Language:
Visual Basic
CSharp
Interop.ADOX.dll
libsim.dll
ModsimModel.dll
NetworkUtils.dll
XYFile.dll
Nodes
Links
Imports Csu.Modsim.ModsimIO
Imports Csu.Modsim.ModsimModel
Imports System
Module CustomMODSIM
Dim myModel As New Model
Sub Main(ByVal CmdArgs() As String)
Dim FileName As String = cmdargs(0)
AddHandler myModel.Init, AddressOf OnInitilize
AddHandler myModel.IterBottom, AddressOf OnIteratio
AddHandler myModel.IterTop, AddressOf OnIterationTo
AddHandler myModel.Converged, AddressOf OnIteration
AddHandler myModel.End, AddressOf OnFinished
AddHandler myModel.OnMessage, AddressOf OnMessage
AddHandler myModel.OnModsimError, AddressOf OnMessa
XYFileReader.Read(myModel, FileName)
Dim myModsim As Modsim
myModsim.RunSolver(myModel)
End Sub
Private Sub OnInitilize()
End Sub
Private Sub OnIterationTop()
End Sub
Private Sub OnMessage(ByVal message As String)
End Sub
Private Sub OnIterationBottom()
End Sub
Private Sub OnIterationConverge()

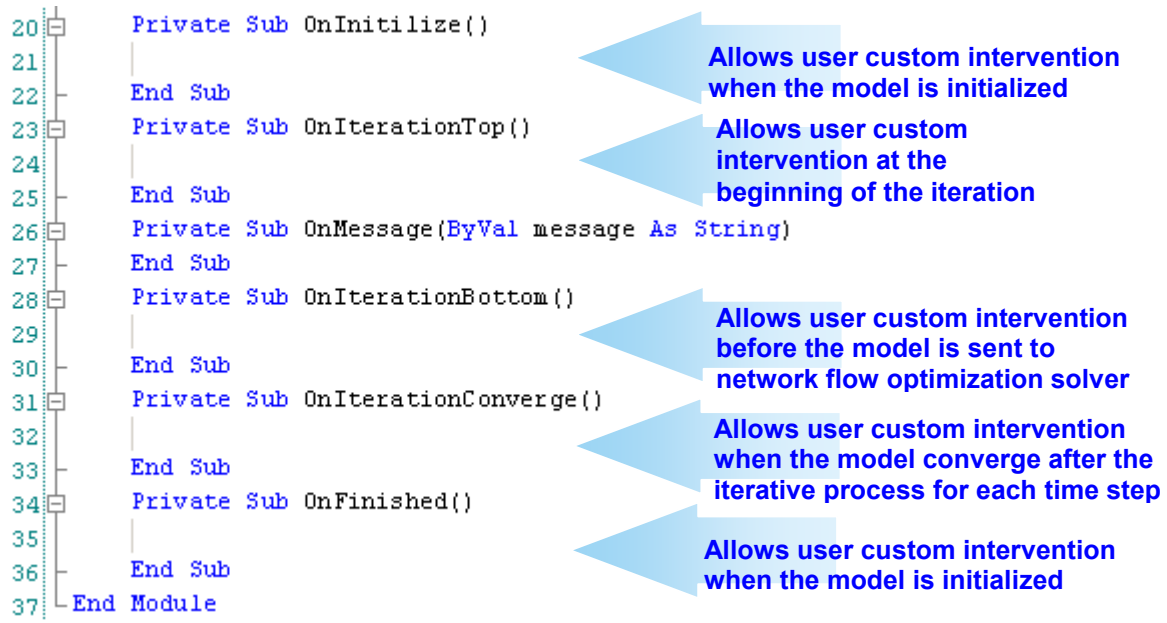
Selecting **Help > Contents** in the Custom Code Editor opens Help information for creating custom code in MODSIM, including an Overview of the Contents, an Index with all object classes, methods and fields , as well as a search engine for finding MODSIM objects needed for Custom Code development.



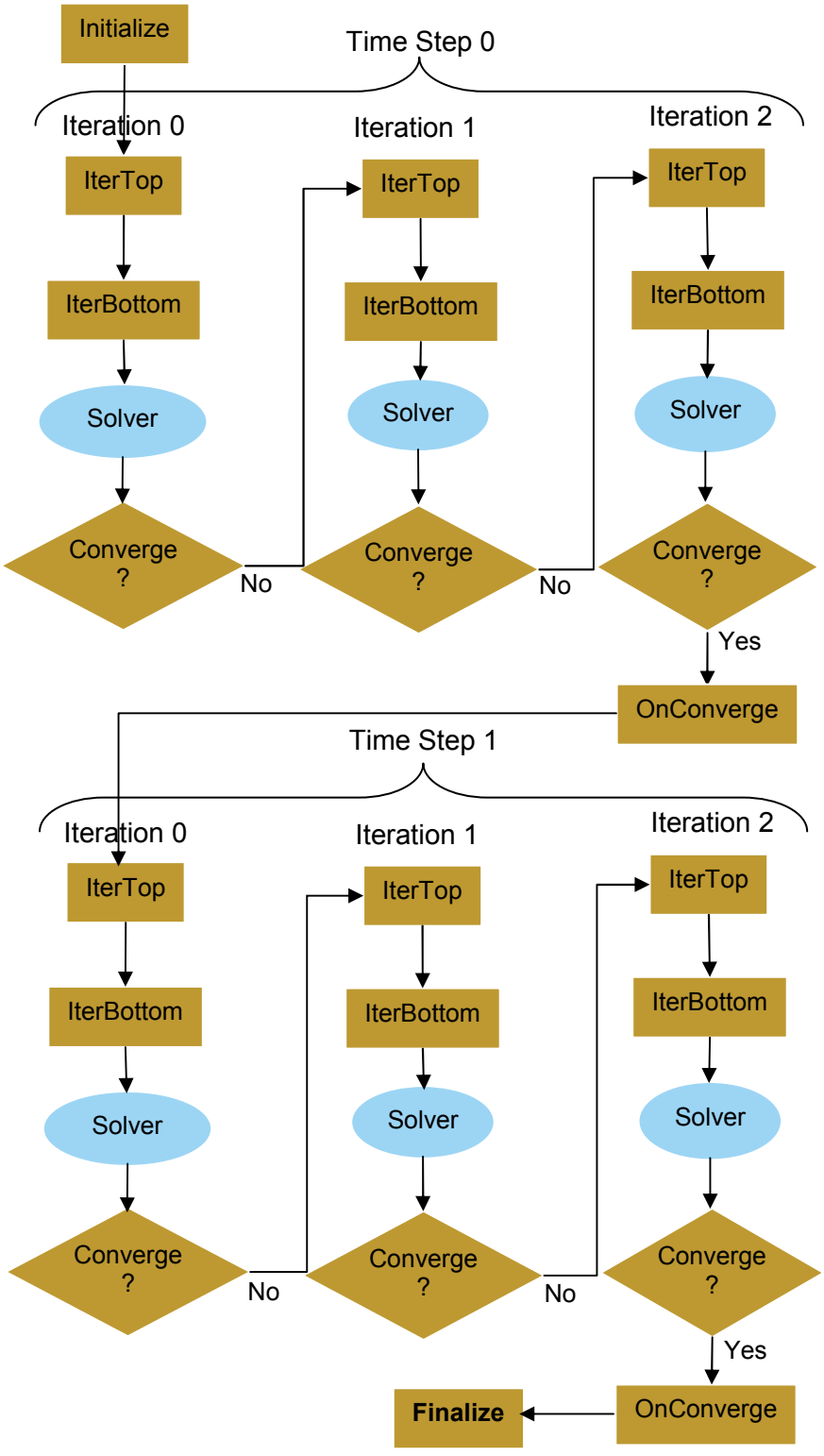
The **Custom Code Editor** provides a template within which custom code can be inserted. The `Main` Subroutine provides the statements for creating a new model object, providing handlers for controlling events triggered by inserting code in the `Subroutines` `OnInitilize`, `OnIterationBottom`, `OnIterationTop`, `OnIterationConverge`, `OnFinished`, and `OnMessage`; loading the Model object from a file, and executing the model.



MODSIM allows customization by writing code that is triggered at the model events specified by the Event Handlers. Access is allowed to all model variables for pre/post processing, including setting and changing model variables at run time to perform custom simulations.

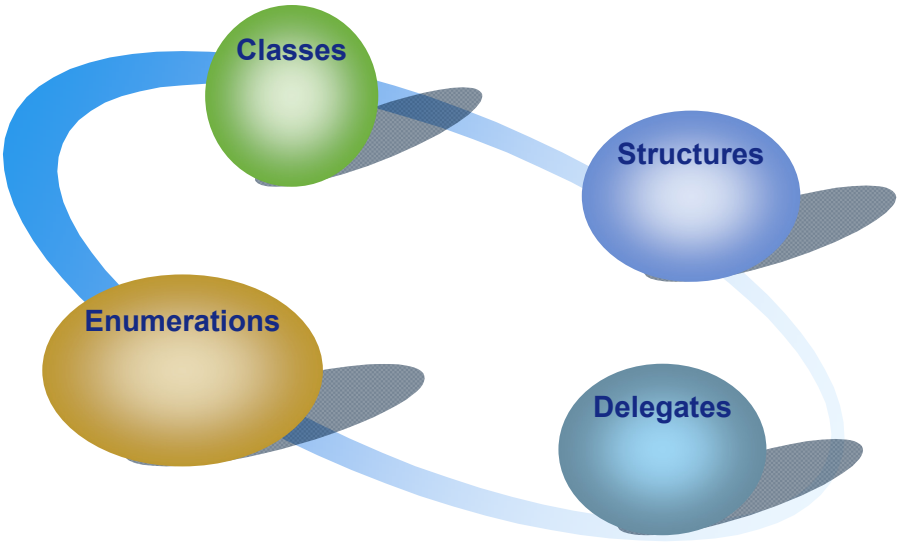


The iterations in MODSIM represent successive approximation calculations of channel loss, groundwater return flows, streamflow depletions due to groundwater pumping and flow-through demands. The iteration counter starts at 0, with activation of the **Storage Rights Extension** resulting in an additional iteration at odd iteration numbers for performing the *Storage Step* (p. 87, User Manual).



C.3 ModsimModel Namespace

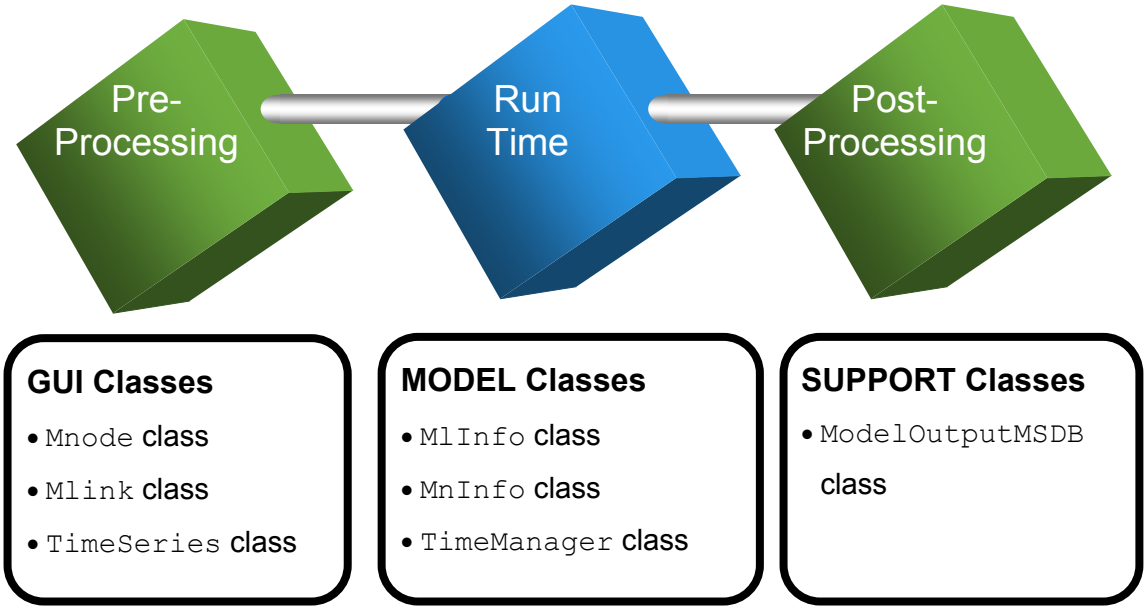
The Namespace `Csu.Modsim.ModsimModel` contains all the available classes, enumerations, structures, delegates and routines for MODSIM modeling.



Class	Description
`DateList`	Contains a list of dates.
`Forcast`	Specifies a time series that contains inflow forecast values to be used in the hydrologic states definition.
`HydrologicStateTable`	Contains the tables used to determine the current hydrologic state in a network.
`LagInfo`	Holds data for depletion/accretion lag factors or response coefficients due to pumping and return flows.
`Link`	Contains all the generic information for a link in the MODSIM network.
`LinkList`	Provides the means to loop through a group of links.
`MinfoStr`	Contains model related information.
`MlInfo`	Contains link variables used by the model solver.
`Mlink`	Contains all the information for a link that will be used in the model and can be edited in the GUI.
`MnInfo`	Contains all node variables used only by the model.
`Mnode`	Contains all the information for a node that will be used in the model and can be edited in the GUI.

`Model`	Contains all the variables and functions to perform a MODSIM network simulation; includes all data used by MODSIM and holds all information pertaining to all network nodes and links.
`Node`	Contains all network node elements for MODSIM network simulation.
`ResBalance`	Holds reservoir balance Information for storage nodes.
`TimeManager`	Provides all handling for model simulation time steps.
`TimeSeries`	Provides handling for the model time series data.

The object classes in MODSIM are organized into three families: (i) graphical user interface (GUI) classes primarily for pre-processing of data, (ii) the MODSIM Model classes, and (iii) support classes providing database management tools for post-processing of MODSIM output results.



C.4 The Model Class

The **`Model` class** is the primary object class in MODSIM modeling. This class contains all of the variables and functions required to perform river basin network flow simulation in MODSIM. The `Model` class holds all link and node information, and includes all required for MODSIM. Access to all elements of the network is provided to the user through the `Model` class. Note that the highlighted rows represent the most commonly used variables in constructing custom code.

Public Properties	
LinkCount	Gets the number of real links in the network model.
NodeCount	Gets the number of real nodes in the network model.
Public Methods	
AddNewNode	Adds a new artificial/real node to the network.
AddRealLink	Adds a real link to the network.
FindLink	Variable that searches for a link in the network (overloaded).
FindNode	Variable that searches for nodes in the network (overloaded).
GetTsIndex	Returns the index for a time series for the specified date.
IsAccrualLink	Determines if a specified link falls into the accrual link category.
Remove	Variable that removes elements from the network model (overloaded).
Public Fields	
accBalanceDates	List of account balance dates.
firstLink	Points to the first link created in the MODSIM network
firstNode	Points to the first node in the network model.
fname	Contains the full path of the *.xy file containing the network model.
HydStateTables	Contains an array of available Hydrologic State Tables in the current model.
maxit	Maximum number of iterations.
mInfo	Holds the class for the model information.
name	Name given to the current network.
TimeStepManager	Handles the simulation model/time series data time steps.

C.5 The Link Class

The **`Link` class** contains all of the generic information for a link in the MODSIM network.

Public Fields	
`description`	Text placed in the Link Properties form in the GUI providing a description of the link.
`from`	The origin or starting node for a link.
`m`	Contains link data for the model that is edited/available in the graphical user interface.
`mlInfo`	Groups all the link variables used in the MODSIM solver.
`name`	Unique name given to a link.
`next`	Points to the next link in the network link sequence.
`number`	Indicates the unique number assigned to the link in the network.
`prev`	Points to the previous link in the network link list (sequence).
`Tag`	User defined object to store with network links any additional data or processes that are outside the standard MODSIM model.
`to`	Node where a link ends or terminates.

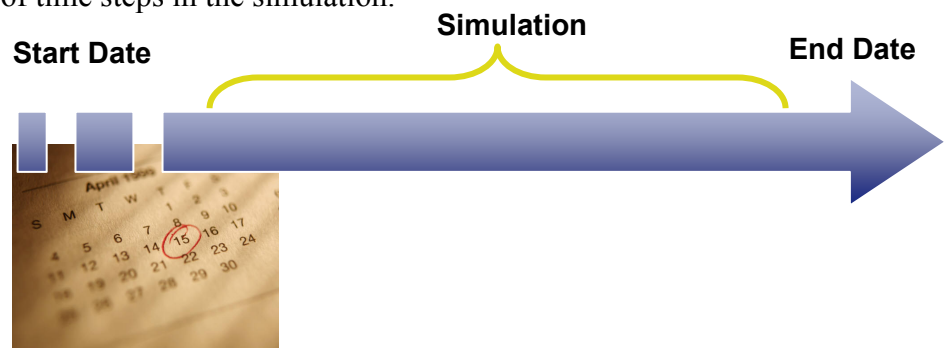
C.6 The Node Class

The **`Node` class** holds information on all the network node elements for MODSIM.

Public Fields	
`backRRegionID`	Contains the Backrouting region that a node belongs to.
`description`	Provides the user description for a node.
`in`	Link list of the nodes that connect TO this node.
`Mnode`	Contains all the information for a node that will be used in the model and can be edited in the GUI.
`mnInfo`	Contains all node variables used exclusively by the model.
`name`	Contains the user name given to a node.
`next`	Contains a pointer to the next node in the network sequence.
`number`	Contains the unique number assigned internally to the node.
`out`	Link list containing the links that go out FROM this node.
`prev`	Points to the previous node in the network sequence.
`NodeType`	Returns/sets the node type (e.g., demand, reservoir, etc.)

C.7 The TimeManager Class

The **`TimeManager` class** provides handling of the model simulation time steps in MODSIM. Access is provided to all information for data as well as simulation time steps, including start date-end date time intervals, simulation and data time steps, and the total number of time steps in the simulation.



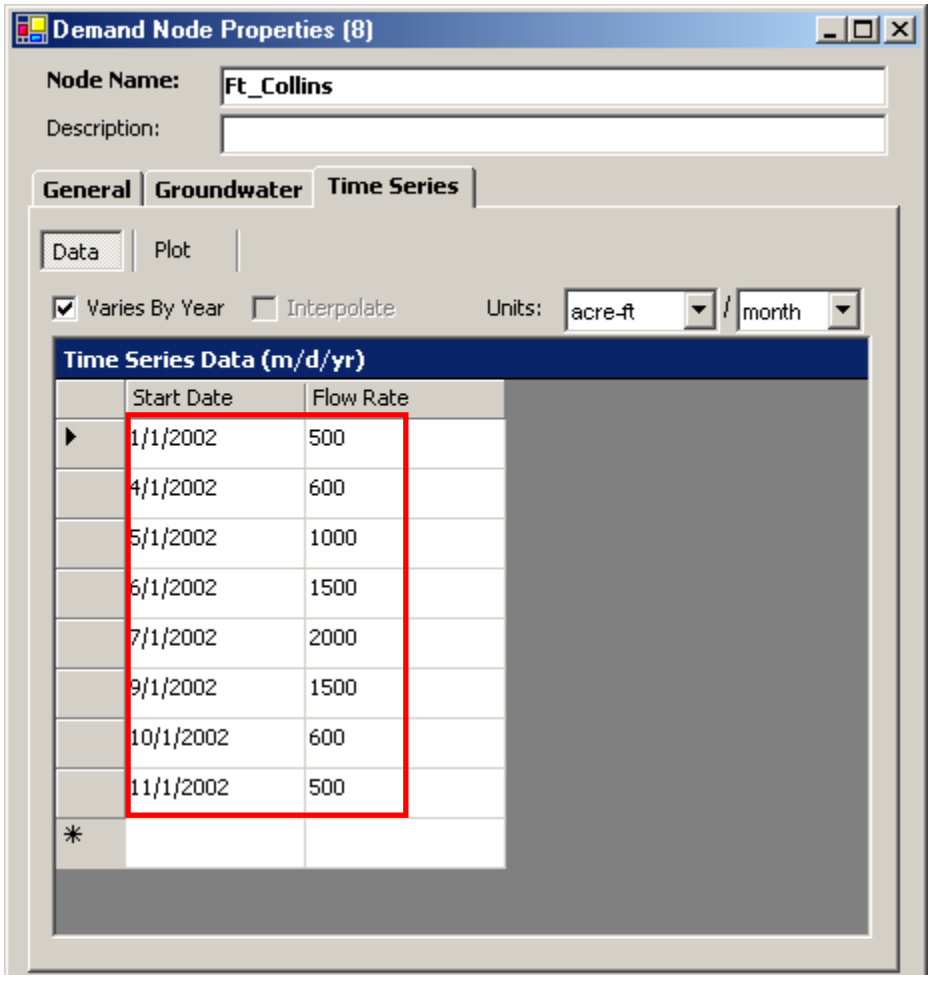
Public Fields	
dataEndDate	Date when the time series data ends.
dataStartDate	Date when the time series data begins.
endingDate	Date when the simulation run ends.
startingDate	Date when the simulation run starts.
timeStepsList	Table containing a listing of all time steps
Public Properties	
noBackRAdditionalTSteps	Contains the number of additional time steps created for backrouting calculations.
noDataTimeSteps	Number of time steps for the time series data.
noModelTimeSteps	Number of time steps for the simulation run.
Public Methods	
Date2Index	Finds the corresponding index for a given start date of the time interval
EndDate2Index	Finds the corresponding index for a given end date of the time interval.
GetMonthIndex	Variable calculating MODSIM vs. 7 month index based on given date of time step index (overloaded).
GetYearIndex	Variable calculates MODSIM vs. 7 year index based on given date of time step index (overloaded).
Index2Date	Finds the start date of the interval for a given time step index.
Index2EndDate	Finds the end date of the interval for a given time step index.

C.8 Graphical User Interface Variables

Variables that are edited in the graphical user interface (GUI) are useful if custom code is constructed that relies on previously entered values. Examples would be values representing percentages of previously defined variables, or reduction or increase in current values, such as planned capacity expansion of a reservoir.

Time Series Class

This class provides handling for the model time series data and is populated in the GUI. Custom Code is used to programmatically import data into the model.



Public Constructors	
`TimeSeries Constructor`	Variable requiring specification of FLOAT/LONG INT type (overloaded).
Public Fields – Flags* **These flags must be assigned when creating a new time series*.	
`isFloatType`	Flag indicating that time series contains float type values
`MultiColumn`	Flag indicating time series table has multiple columns
`VariesByYear`	Flag indicating that time series data varies by year
Public Methods	
`Clone`	Copies to new time series and returns the new time series.
`getDataF`	Gets the float table value for row = time index , first hydrologic state (overloaded).
`getDataL`	Gets the long INT table value for row = index , first hydrologic state (overloaded).
`GetDataTable`	Returns the table portion of the TimeSeries instance.
`getDate`	Finds the associated date for a given table index.
`getNumCol`	Gets the number of columns in the table of the time series instance.
`getSize`	Gets the number of rows in the table.
`GetTable`	Gets a table without read-only columns.
`GetTsIndex`	Returns the active index in a table for a given date.
`increment`	Increments all columns (except the date) of indexed row by a value (overloaded).
`IncrementDate`	Adds the corresponding number of days to a given date according to the model time step.
`setDataF`	Sets the data and table of a FLOAT type time series (overloaded).

setDataL	Sets the data and table of a LONG INT type time series (overloaded).
setDate	Sets the date for a specified row in the table.
SetTable	Assigns the given table to the time series table.
SetupTable	Creates a LONG INT or FLOAT type table and sets the time series flag to the default values.

Mnode Class

Contains all the information associated with a node that will be used in the model and can be edited in the GUI.

Public Fields	
adaDemandsM	Demand time series as entered in the interface.
adaEvaporationsM	Evaporation time series as entered in the interface
adaForcastsM	Forecasted inflow time series data as entered in the interface
adaGeneratingHrsM	Hydropower generating hours time series data as entered in the interface.
adaInfiltrationsM	Infiltration rate time series data as entered in the interface.
adaInflowsM	Inflow time series data as entered in the interface.
adaTargetsM	Reservoir target storage time series data as entered in the interface.
apoints	Area-Capacity-Elevation-Hydraulic Capacity points: Area
cpoints	Area-Capacity-Elevation-Hydraulic Capacity points: Capacity
epoints	Area-Capacity-Elevation-Hydraulic Capacity points: Elevation
hpoints	Area-Capacity-Elevation-Hydraulic Capacity points: Hydraulic Capacity

idstrmfraction	Fractions of flow distributed by flow-through demand.
infil	Contains infiltration rates for demand nodes.
infLagi	List of groundwater infiltration lag factor information
max_volume	Maximum volume for reservoir node.
min_volume	Minimum volume for reservoir node.
next	Points to the next node in the sequence.
pcap	Contains groundwater pumping capacity.
pcost	Cost per unit groundwater pumping rate.
peakGeneration	Peak power generation-only checkbox entry.
powmax	Maximum power plant output capacity.

Mlink Class

This class contains all the information for a link that will be employed in the model and can be edited in the GUI.

Public Fields	
cost	Unit cost associated with transport of water through a link.
lagfactors	Array of lag factors used in channel routing or channel loss.
loss_coef	Array of channel loss coefficients for the current and next time steps; also used to flag a routing link when the first coefficient equals 1.
maxConstant	Maximum capacity in a link only when constant for all time steps.
maxVariable	Maximum capacities in a link that vary with time.
min	Sets the minimum flow in a link.
selected	Selects a link to be included in the output.
waterRightsDate	Indicates the date when a link has priority for water accrual.

C.9 Network Solver Parameters

MlInfo Class

This class contains the link parameters that are directly utilized by the Lagrangian relaxation network solver. These link parameters can be populated and processed based on variables entered by the user at each time step via the custom code. Modification of these parameters affects the model solution directly. These variables are calculated between iteration top and iteration bottom based on the user preferences, although it is recommended that they be modified at iteration bottom.

Public Fields	
`accrualLink`	Indicates if a link is an accrual link.
`artificial`	Indicates if a link is real or artificial.
`cost`	Contains the cost associated with the ability to transport water through a link.
`flow`	Contains the current time step link flow.
`flow0`	Contains the immediately previous iteration flow.
`hi`	Contains the upper bound for flow in a link; maximum flow in the current time step.
`hiVariable`	Array containing the upper bounds on flow in a link for all the model time steps.
`lo`	Contains the lower bound for flow in a link; minimum flow in the current time step.
`ownerLink`	Indicates if this a storage ownership link owning storage rights in a reservoir.

MnInfo Class

Contains all node parameters used exclusively by the Lagrangian relaxation network flow solver, and contains information on all nodes including both real and artificial.

Public Fields – General	
chanLossLink	Holds the artificial channel loss link; *applies to:* FROM node of a link with channel loss.
clossreturn	Returning channel loss to a node in the current time step.
flowThroughReturnLink	Holds the link conveying return flow from a flow-through demand node; *applies to:* Flow-Thru return nodes.
gwoutLink	Holds the groundwater link conveying infiltration /seepage flows from a node.
gwrtnLink	Holds the groundwater return link to a node conveying groundwater return flows.
hydStateIndex	Contains the hydrologic state index calculated for a node.
hiVariable	Array containing variable capacity upper bounds for all model time steps.
lo	Contains the lower bounds of the link; i.e., minimum flow through a link in the current time step.
ownerLink	Indicates if a link owns water in a reservoir.
routingLink	Holds the routing link from the node; *applies to:* (downstream) nodes of routing links.
Public Fields – Reservoir Node	
evapLink	Holds the artificial evaporation link from a node.
evpt	Contains computed reservoir evaporation for the current time step.
excessStoLink	Holds the artificial excess storage link from a node.
forcast	Model array for node forecasts.
generatinghours	Model array for reservoir power generation hours.
head	Contains computed reservoir head for hydropower calculations for current time step; (average reservoir elevation-power-plant elevation or tailwater elevation)

hydCap	Contains the computed reservoir hydraulic capacity for the current time step.
hydraulicCapLink	Holds the reservoir hydraulic capacity link.
hydraulicCapNode	Holds the node located upstream of the hydraulic capacity link.
spillLink	Holds the spill link of the reservoir node.
start	Reservoir contents at beginning of the time step.
stend	Contains the computed end of time step reservoir content for the current time step.
targetcontent	Model array for node target content values.
targetExists	Specifies if the node has target contents defined.
targetLink	Holds the artificial target link from the node.
Public Fields – Demand Node	
demLink	Holds the artificial demand link for the node.
infiltrationrate	Model array for node infiltration rates.
nodedemand	Model array for node demand values.
Public Fields – Nonstorage Node	
infLink	Holds the inflow link to the node.
inflow	Model array for node inflow values.

MInfo Class

This class provides access to variables used by the MODSIM solver during simulation. KEY Class for custom code development. It holds artificial nodes and links (Artificial Network), and during simulation holds the current time step, date, and iteration.

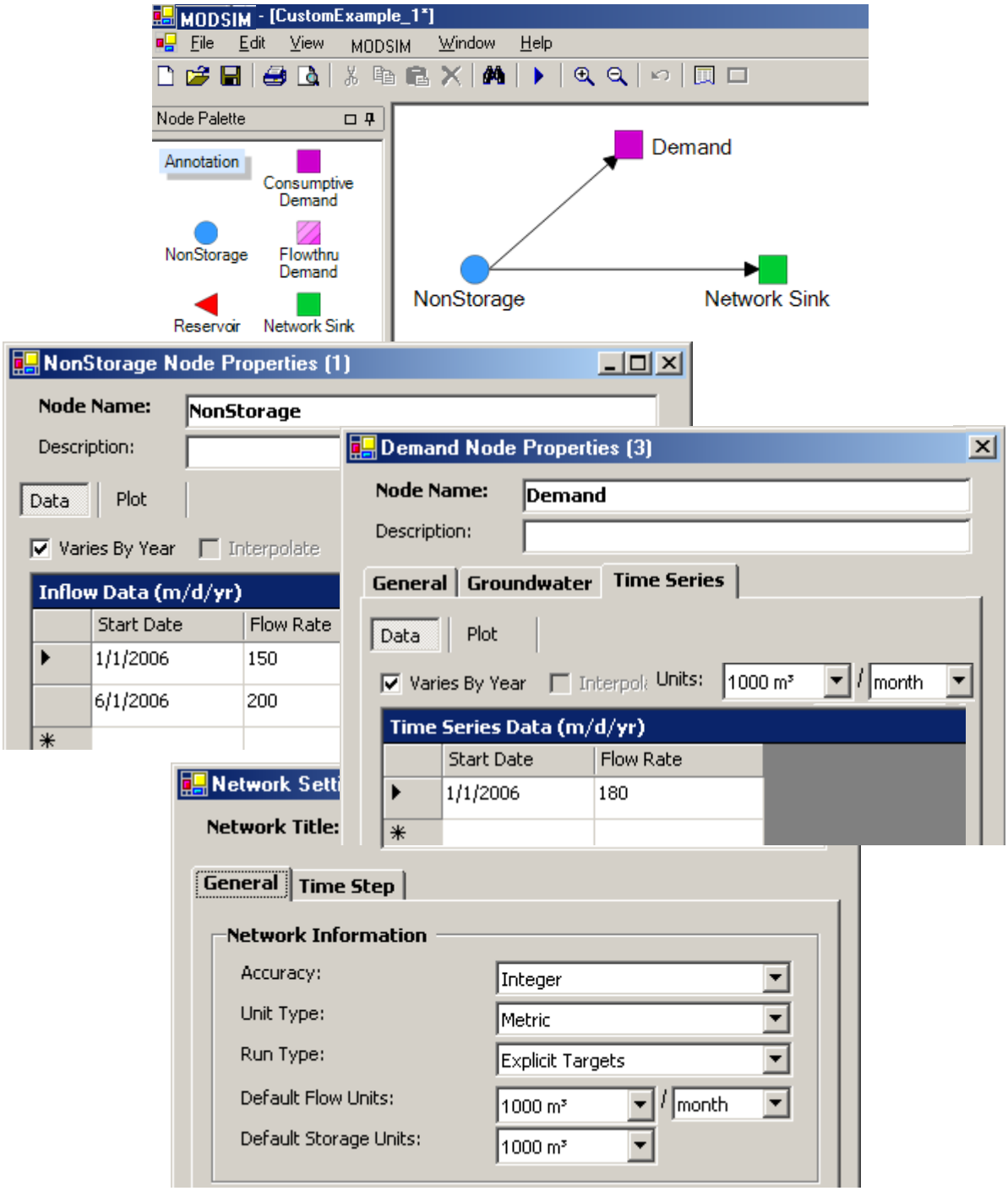
artDemandN	Artificial demand node.
artGroundWatN	Artificial groundwater node.
artInflowN	Artificial inflow node.
artMassN	Artificial mass balance node.
artSpillN	Artificial spill node.
artStorageN	Artificial storage node.
CurrentModelTimeStepIndex	Current time step index during simulation.
demList	List of demand nodes in the network.
demListLen	Number of demand nodes in the network.
demToMassBal	Artificial link from artificial demand node to artificial mass balance node.
gwaterToInf	Artificial link from artificial groundwater node to artificial inflow node.
inflowNodes	List of nonstorage inflow nodes.
inflowNodesLen	Number of nodes that have inflow.
infToGwater	Artificial link from artificial inflow node to artificial groundwater node.
Iteration	Contains current iteration during simulation.
lList	List of all links in the network.
lListLen	Number of links in the network.
massBalToInf	Artificial link from artificial mass balance node to artificial inflow node.
MonthIndex	Contains the month index during simulation.
nList	List of nodes in the network.
nListLen	Number of nodes in the network.

outputLinkList	List of links flagged for inclusion in output.
outputLinkListLen	Number of links flagged for output.
outputNodeList	List of nodes flagged for output.
outputNodeListLen	Number of nodes flagged for output.
ownerChanlossList	List of ownership links with channel loss debit.
ownerChanlossListLen	Number of ownership links with channel loss charged.
ownerList	Link list with ownership in a reservoir account.
ownerListLen	Number of links with ownership in reservoir account
realLinkList	List of real links in the network.
realLinkListLen	Number of real links in the network.
realNodesList	List of real nodes in the network.
realNodesListLen	Number of real nodes in the network.
resList	List of reservoir nodes in the network
resListLen	Number of reservoir nodes in the network.
spillToMassBal	Artificial reservoir spill link to artificial mass balance node.
stoToMassBal	Artificial link from artificial storage node to artificial mass balance node.
variableCapLinkList	List of links with variable capacity defined.
variableCapLinkListLen	Number of links with variable capacity defined.
YearCounter	Year index since start of the simulation.

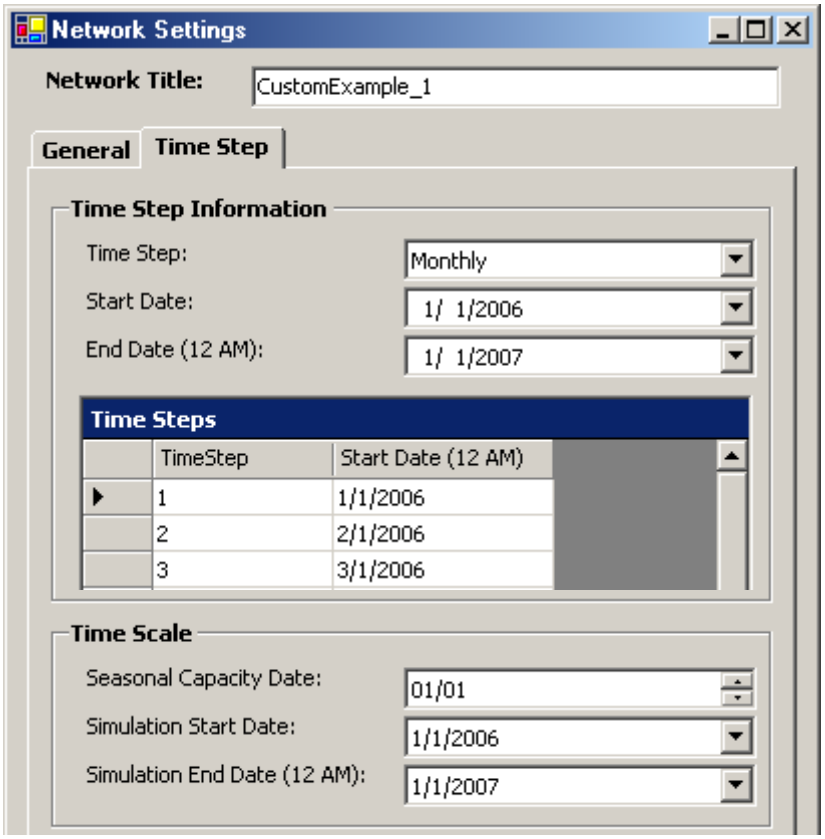
C.10 Basic Customization Tutorials

Example #1: Adjusting Demands through Custom Code

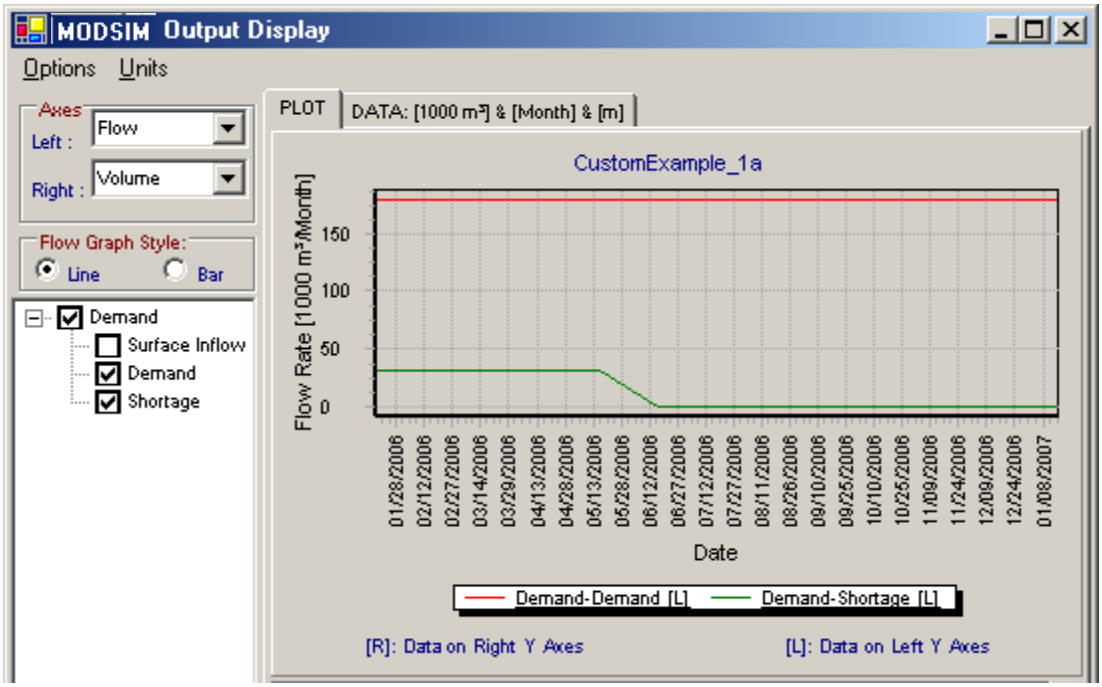
This example provides an introduction to custom code development by illustrating how demands can be adjusted based on development scenarios, demand forecasts, or conservation strategies. Create the following simple network and enter inflow and demand data as shown. **Network Settings** under the **General** tab should be set as shown



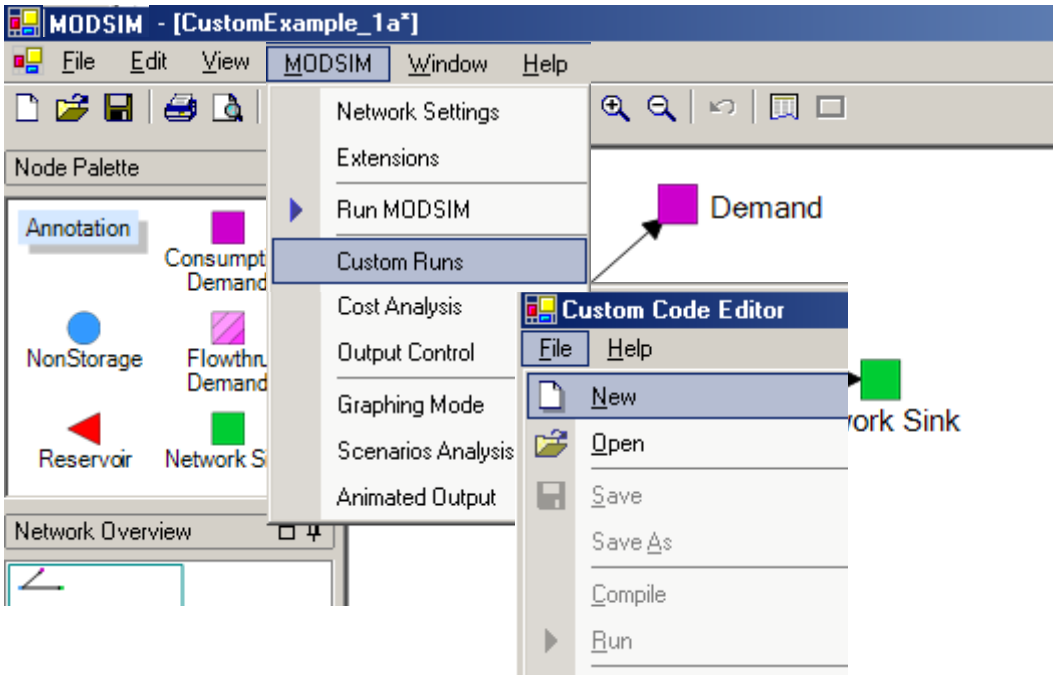
A monthly time step is selected covering a 1 year period and the network is saved as **CustomExample_1a.xy.**



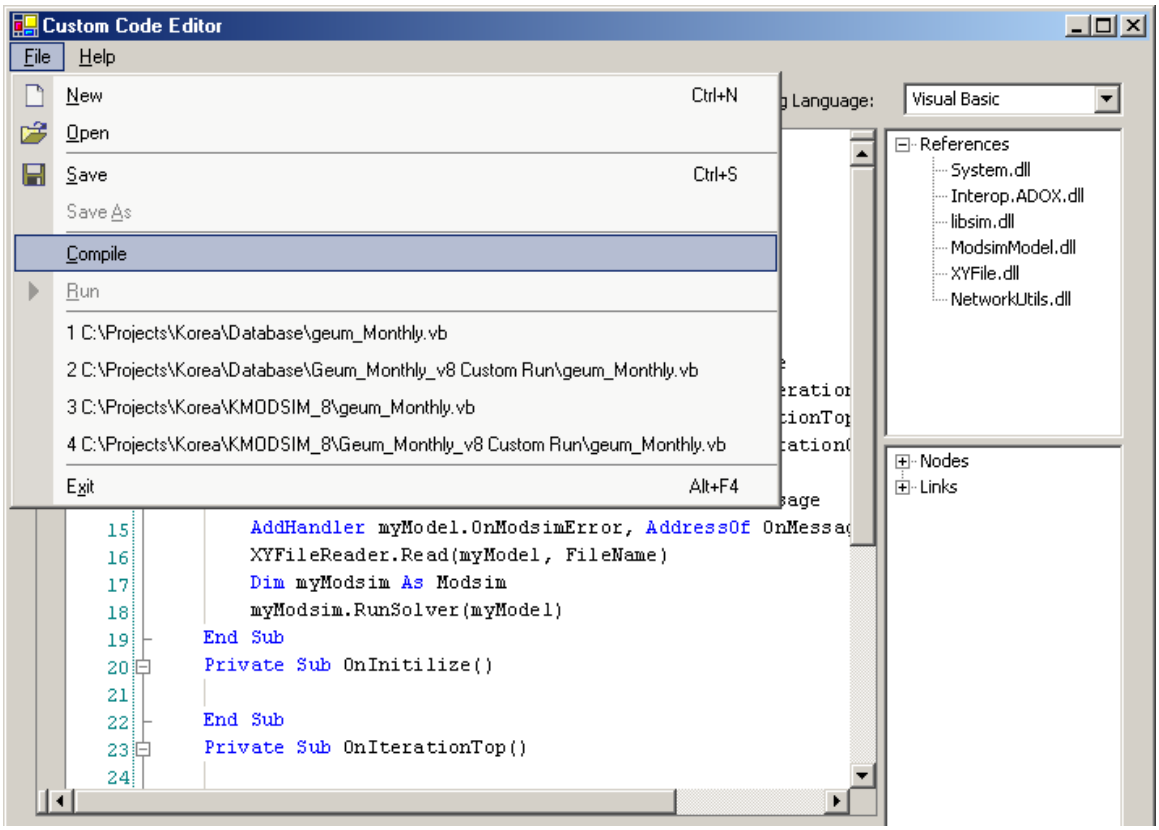
Executing this simple network gives the following results for the node *Demand*:



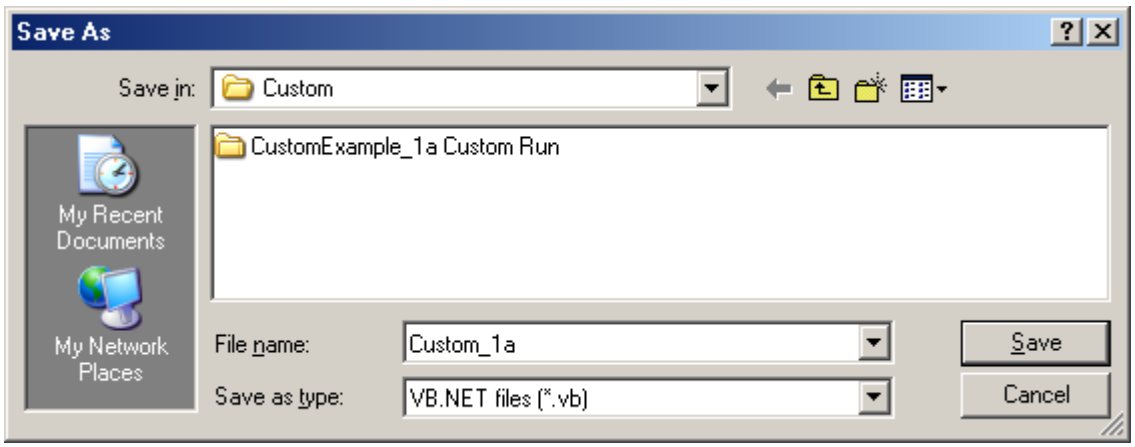
Now we will run the same network from the Custom Code Editor. Select **MODSIM > Custom Runs** and click **New** in the **Custom Code Editor.**



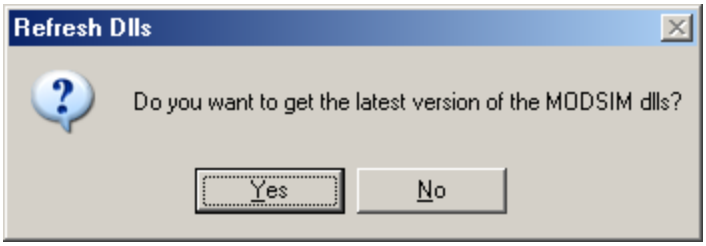
Even though what appears is only the custom code template with no additional code included, the network can still be compiled and run from the Custom Code Editor



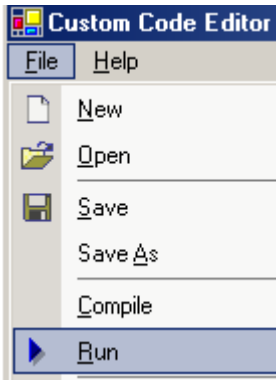
Clicking **Compile** opens the **Save As** window in which users can specify where the custom code (VB.NET code in this case) is to be stored. In this case, browse to any desired directory used for these tutorials and specify the file name **Custom_1a**. The extension ***.vb** will automatically be added to the file when it is saved. Again, this file only contains an empty template with no custom code written as yet.



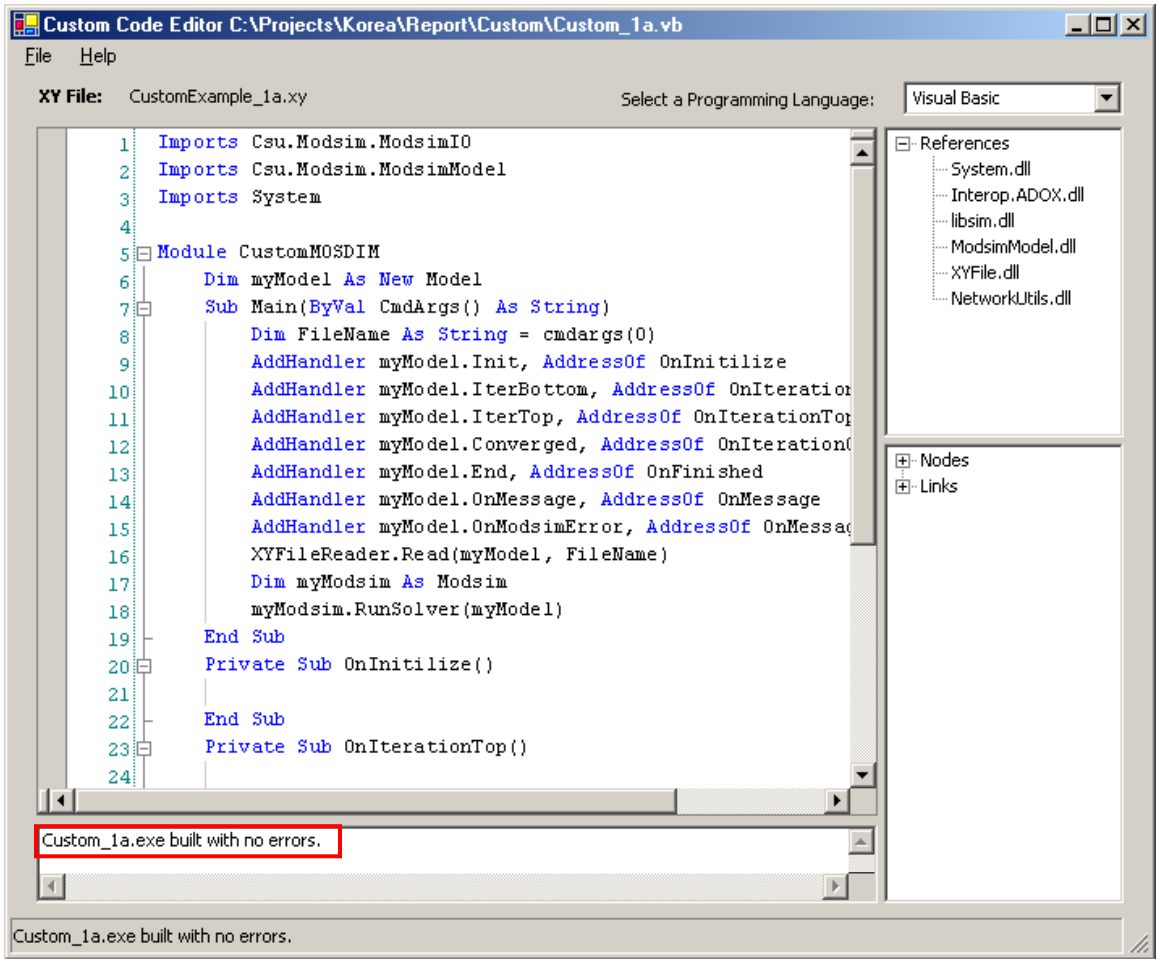
Notice that the **Custom Code Editor** has automatically created a subfolder for this exercise called **CustomExample_1a Custom Run** which is derived from the name of the example network. It is recommended that **Custom_1a.vb** be stored in that new subfolder. After clicking on the subfolder, clicking the **Save** button is followed by display of a box asking: "Do you want to get the latest version of the MODSIM dlls?" Normally, the **Yes** button should be clicked in this case.



Messages as to whether the compilation was successful or errors occurred appear in the window in the lower portion of the **Custom Code Editor**. In this case, the message is "Custom_a1.exe built with no errors" appears. Use Windows Explorer to browse to the subfolder **CustomExample_1a Custom Run** and notice that the executable code **Custom_1a.exe** has been saved there, along with other files needed for the custom run. MODSIM networks compiled in this way must be executed from the **Custom Code Editor** rather than **MODSIM > Run MODSIM** or the **Run** icon in the interface. Alternatively, Custom_1a.exe can be run from the DOS Command Prompt by entering, for this example, the following line:




```
c:\MODSIM\Custom\CustomExample_1a Custom Run>custom_1a.exe
"c:\MODSIM\Custom\CustomExample_1a.xy"
```



where the argument specifying the path and name of the ***.xy** network data file is enclosed in parentheses. After a successful custom run, all of the graphical output features of the interface can be used for displaying results.

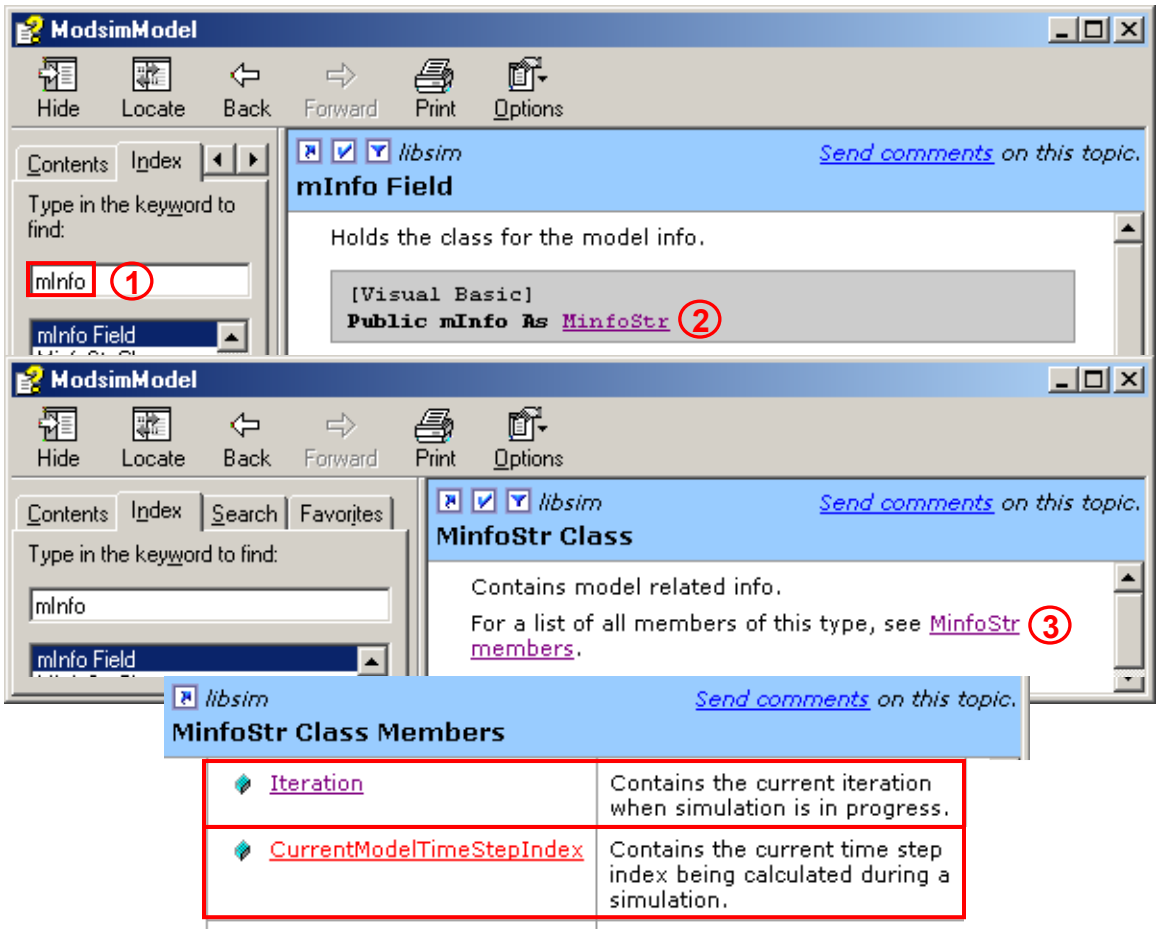
Custom code will now be developed for modifying the demand entered into the MODSIM graphical user interface. The adjustment in the demand will occur at the beginning of the first iteration, so the code will be inserted into the subroutine `OnIterationTop()`.

For this example, the objective is to divide the demand in half. The following Classes, Public Fields and Public Methods are used in the custom code development:

- The model name `myModel` is already declared as a global variable in the `Model` Class.
- `mInfo` is a Public Field in the `Model` class that holds model information.
- `Iteration` is a Public Field in the `mInfo` class that contains the current iteration while simulation is in progress.
- `CurrentModelTimeStepIndex` contains the current simulation time step.
- `FindNode` is a Public Method in the `Model` Class that searches for nodes in the network by name (as a string) or by node number—it is referred to as an *overloaded* variable since it accepts arguments of different data types.

- The `MnInfo` class contains all node variables used exclusively by the model.
- `nodedemand` is a Public Field in the `MnInfo` class that contains the model array for node demand values over all time steps and ?

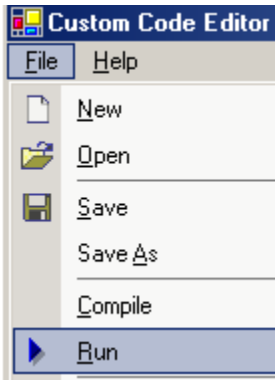
Shown here is an illustration of how to use the Help features to find information.



The VB.NET code entered in `Private Sub OnIterationTop()` is listed as follows:

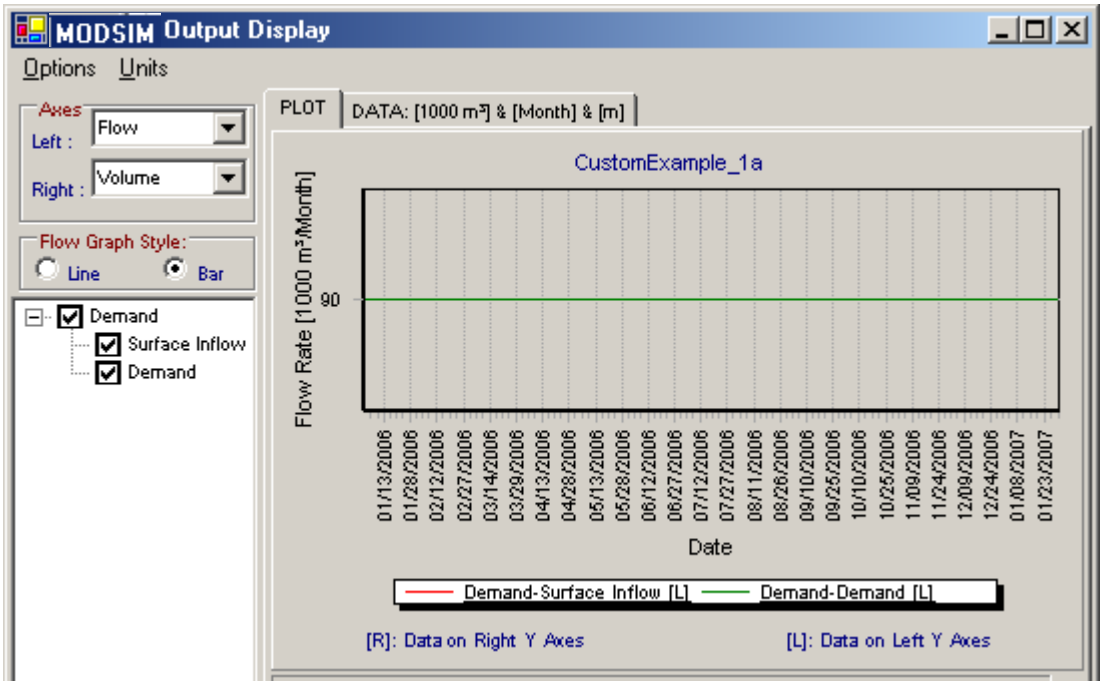
```
Private Sub OnIterationTop()
    'The demand should be set at the beginning iteration
    If myModel.mInfo.Iteration = 0 Then
    'declare the variable that will hold the demand node
      Dim m_DemandNode As Node
    'declare the variable that will hold the current time step
      Dim currentIndex As Integer = myModel.mInfo.CurrentModelTimestepIndex
    'find the particular demand node with the demand to be changed
      m_demandNode = myModel.FindNode("Demand")
    'get the GUI entered demand and assign the new demand
      m_demandNode.mnInfo.nodedemand(currentIndex, 0) /=2
    End If
End Sub
```

In the last line of the code, the notation `/=2` is shorthand for `[variable] = [variable]/2`.



Also, the second element in the array:
`mnInfo.nodedemand(currentIndex, 0)` is a placeholder in case the demand is also a function of the Hydrologic State.

Saving this code as **Custom_1b.vb**, compiling and running from the Custom Code interface gives the following solution for node *Demand*, showing the demand at one-half the amount entered into the Demand Node Properties form in the GUI.

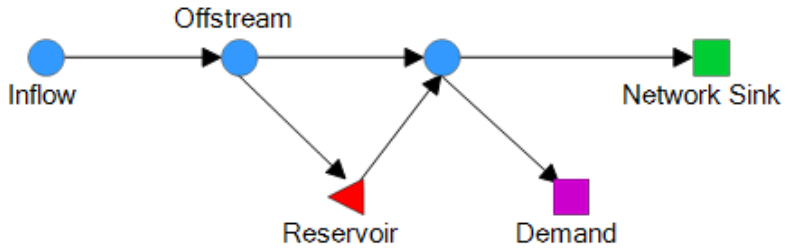


An alternative approach is to change the demand in `Private Sub OnInitilize()`, which is called at the beginning of the simulation run. In this approach, the demand is changed by looping over all time steps. The code below can be inserted into `Private Sub OnInitilize()`, saved as the file **Custom_1c.vb**, compiled, and run from the Custom Code interface. Plotting the results for *Demand* node gives the same solution.

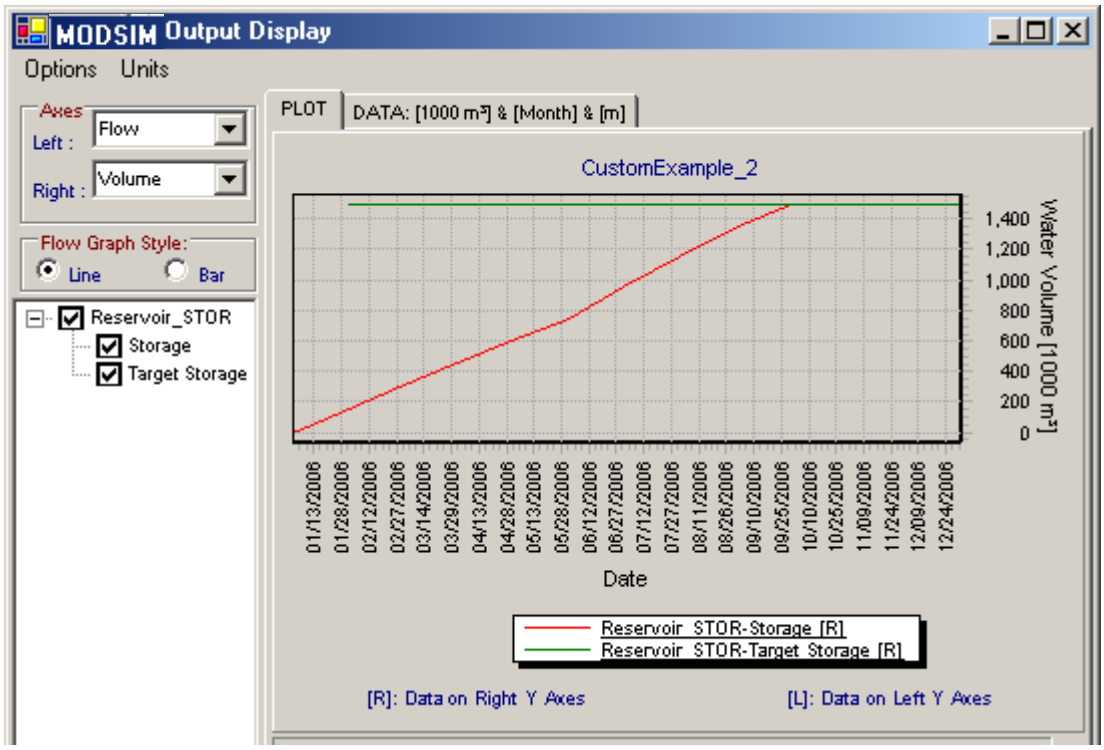
```
Private Sub OnInitilize()
    'The demand array is declared as an instance of the Node class
    Dim m_DemandNode As Node
    'declare time step variable
    Dim m_Index As Integer
    'find the particular demand node with the demand to be changed
     m_demandNode = myModel.FindNode("Demand")
    'assign the new demand for each time step
    For m_Index = 0 To myModel.TimeStepManager.noModelTimeSteps - 1
     m_demandNode.mnInfo.nodedemand(m_Index, 0) /=2
    Next
End Sub
```

Example #2: Custom Reservoir Operating Rules

The objective of this exercise is to implement a custom reservoir operating rule that is conditioned on current inflow. Create the network shown below using the same **Network Properties** and name the network "Custom Reservoir Operating Rule".



Set the *Reservoir* node priority to 50 with a capacity of 1500 and a target storage level set to full. The *Demand* node is given the junior default priority of 100 with a constant demand of 150, and the *Inflow* node has an inflow of 150 from January to June and 200 for the remainder of the year. Save the network as **CustomExample_2.xy** and run the network to make sure that all data have been entered correctly. The solution for the *Reservoir* node should appear as shown below:



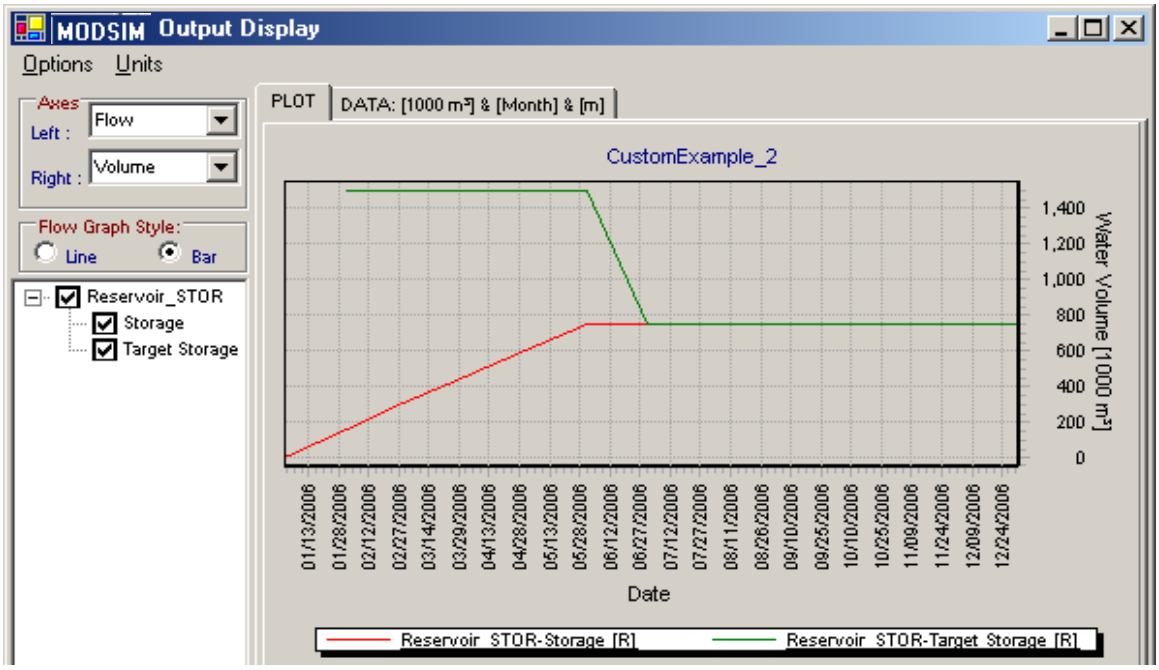
The following reservoir operating rule is implemented using Custom Code: *"When the inflow exceeds 180, the reservoir storage target should be reduced to one-half of the reservoir capacity"*.

Open the **Custom Code Editor**, create a **New** template, and insert the following code as shown:

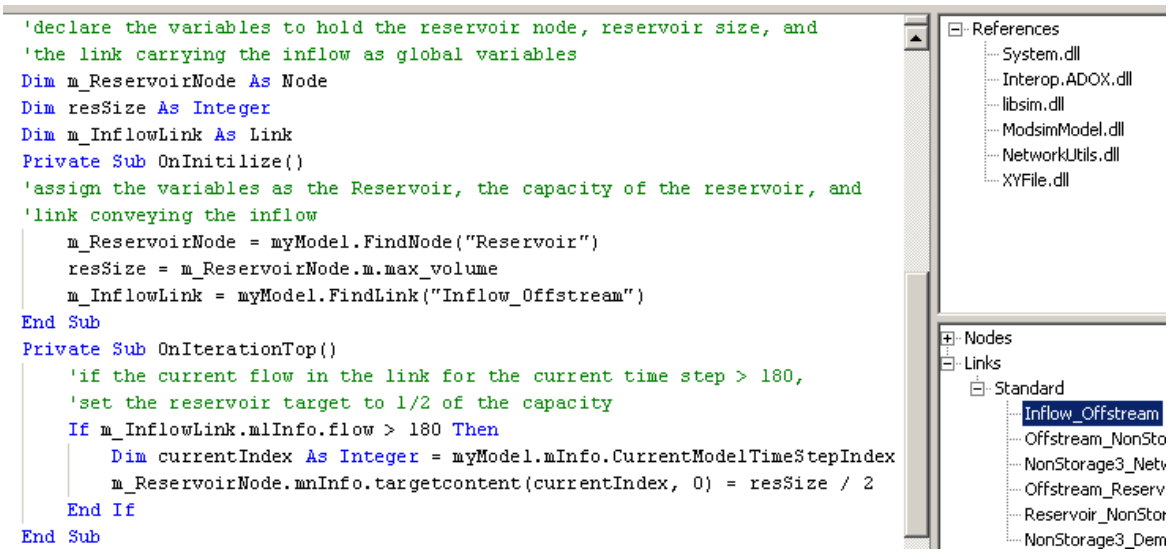
```
'declare the variables to hold the Reservoir node and Inflow node
'as global variables
Dim m_ReservoirNode, m_InflowNode As Node
Private Sub OnInitilize()
'assign the variables as the Reservoir and Inflow nodes
    m_ReservoirNode = myModel.FindNode("Reservoir")
    m_InflowNode = myModel.FindNode("Inflow")
End Sub
Private Sub OnIterationTop()
    'implement the operating rule at the beginning iteration
    If myModel.mInfo.Iteration = 0 Then
    'declare the current time step as a variable
        Dim currentIndex As Integer = myModel.mInfo.CurrentModelTimeStepIndex
    'if the inflow for the current time step > 180, declare a variable for
    'reservoir capacity and set the reservoir target to 1/2 of the capacity
        If m_InflowNode.mnInfo.inflow(currentIndex, 0) > 180 Then
            Dim resSize As Integer = m_ReservoirNode.m.max_volume
            m_ReservoirNode.mnInfo.targetcontent(currentIndex, 0) = resSize / 2
        End If
    End If
End Sub
```

Notice the variables `m_ReservoirNode` and `m_InflowNode` are assigned as global variables since they are declared outside of the modules. Also, although `m_ReservoirNode.m.max_volume` uses the `m` class to access the maximum reservoir capacity in the MODSIM graphical user interface (GUI), the variable `m_ReservoirNode.mnInfo.targetcontent(currentIndex, 0)` uses the `mnInfo` class since all time series data read into the GUI are transferred to the `mnInfo` class within the model.

Saving this file as **Custom_2a.vb**, compiling, and running from the **Custom Code Editor** produces the results shown here for the *Reservoir* node.



An alternative approach is to use the flow in a link to trigger the conditional reservoir operating target, rather than the inflow node. In this case, the flow in link



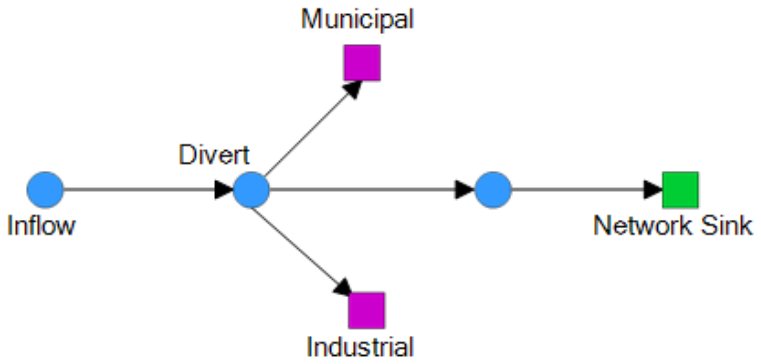
Inflow_Offstream will be used to change the reservoir operational rule. Entering the code shown below requires specifying the name of the link that is triggering the change in operating rule. Rather than returning to the GUI to obtain the link name, all the links (and nodes) for the network are listed in an area on the right-hand side of the **Custom Code Editor**. Notice that the variable `m_InflowLink.mlInfo.flow` is using the `mlInfo` class within the `Link` class. Also, there is no need to consider flows at various iterations since flow in link *Inflow_Offstream* is unaffected by any iterations occurring since it is directly conveying inflows to the network.

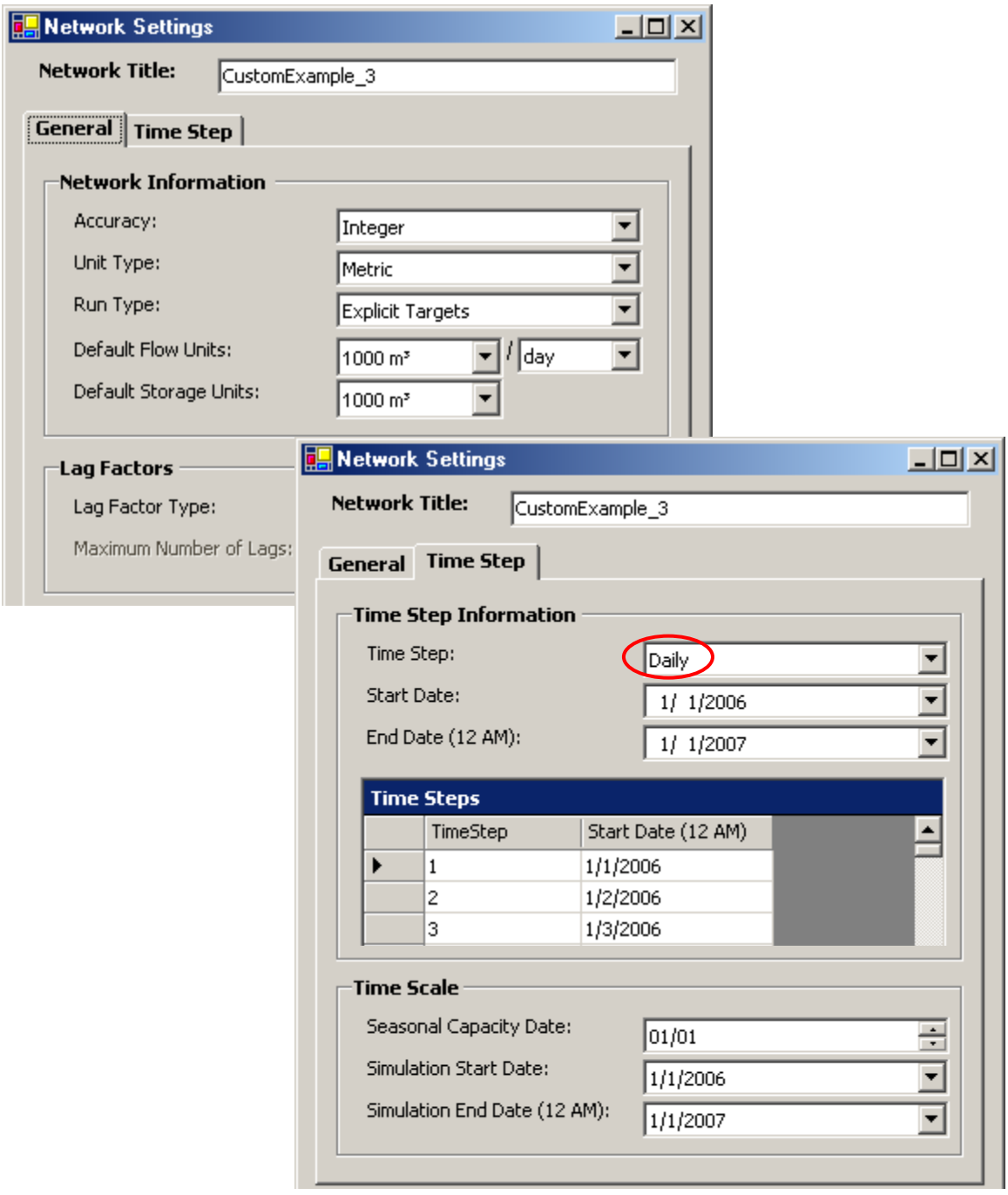
Saving this file as **Custom_2b.vb**, compiling, and running from the **Custom Code Editor** produces the same results for the *Reservoir* node.

Example #3: Changing Variable Link Capacities

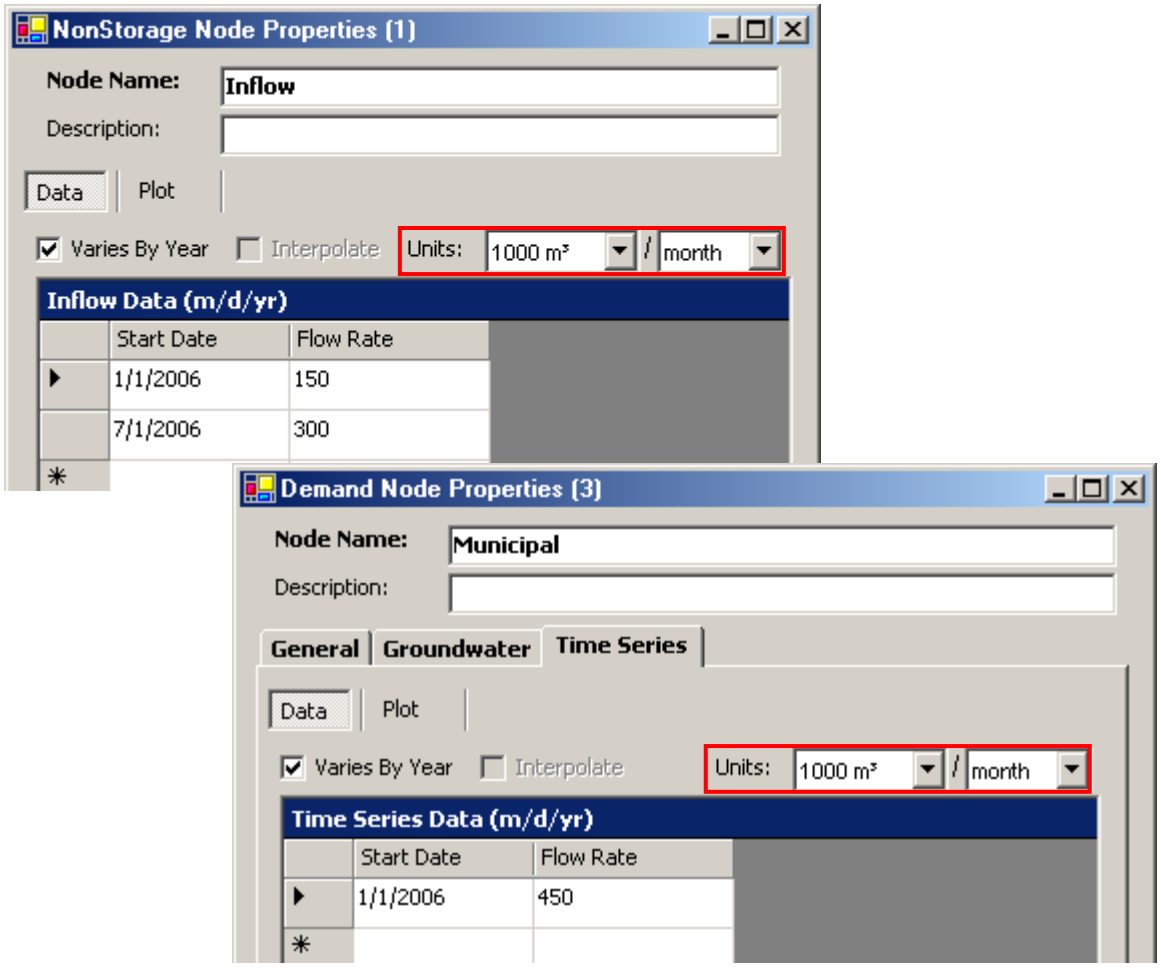
The objective of this exercise is show how custom code can be used to set the variable capacity of a link to a complex function. This may be useful for situations where, for example, the flows in a link are to be controlled for maintaining ideal spawning conditions in a river reach for certain species of endangered fish. In other cases, it can be useful for distributing flow to various demands based on complex timing of water requirements rather than strictly based on water rights priorities.

Create the following network with two demands and an inflow source. The Network Settings are similar to the previous examples, except that a Daily time step is selected for this example. The inflow time series for node *Inflow* is 150 (1000 m^3/mo) from 1/1/06 to 7/1/06, and then changes to 300 (1000 m^3/mo). Although the simulation time step is daily, MODSIM allows time series data to be entered for any desired time step or units and then automatically either interpolates or aggregates the data as necessary to conform to the desired simulation time step. The *Municipal* an *Industrial* demand nodes each


Municipal
Divert
Inflow
Network Sink
Industrial


Network Settings
Network Title:
CustomExample_3
General
Time Step
Network Information
Accuracy:
Integer
Unit Type:
Metric
Run Type:
Explicit Targets
Default Flow Units:
1000 m³
/
day
Default Storage Units:
1000 m³
Lag Factors
Lag Factor Type:
Maximum Number of Lags:
Network Settings
Network Title:
CustomExample_3
General
Time Step
Time Step Information
Time Step:
Daily
Start Date:
1/ 1/2006
End Date (12 AM):
1/ 1/2007
Time Steps
TimeStep
Start Date (12 AM)
1
1/1/2006
2
1/2/2006
3
1/3/2006
Time Scale
Seasonal Capacity Date:
01/01
Simulation Start Date:
1/1/2006
Simulation End Date (12 AM):
1/1/2007

have a demand of 450 (1000 m^3/mo), although actual delivery will be based on functions defined in the custom code. Both demands have the same default priority of 100. It should be noted that if several demands have the same priority, there is no guarantee in MODSIM that flows will be equalized between the two demands. In this case, equalization or sharing of shortages to the demands is accomplished by use of Hydrologic State tables, multi-links carrying incremental flows under changing costs, or development of custom code to control the distribution of flows. This example is an illustration of the latter method.



This network should be named "CustomExample_3" in the **Network Settings** and saved as **CustomExample_3.xy**.

For this illustration, it will be assumed that the demands share the available water based on a complimentary sine-cosine function that is applied on a daily basis, but resets every month. The upper bounds on the links conveying flows to the demands will be set using MODSIM custom code. For each day of the month, calculate the equivalent radian value.

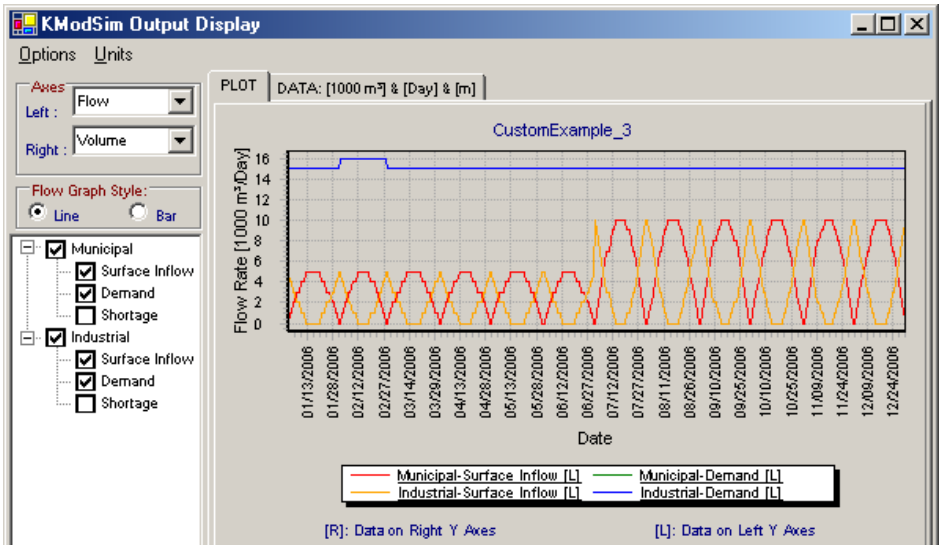
$$rad = (d - 1) \cdot \pi / (nd - 1)$$

where nd is the number of days in the month and d is the current day. The ***sin*** function is used to set positive bounds on the link conveying flows to one of the demands, with the

other demand receiving the complementary amount. Open the **Custom Code Editor**, create a **New** template, and insert the code below, which includes comments explaining the procedures.

```
'declare the variables to hold the links toMunicipal, toIndustrial,
'and inflowLink
Dim toMunicipal, toIndustrial, inflowLink As Link
Dim m_rad As Single
Private Sub OnInitilize()
'find the links in the network and assign to the link objects
    toMunicipal = myModel.FindLink("Divert_Municipal")
    toIndustrial = myModel.FindLink("Divert_Industrial")
    inflowLink = myModel.FindLink("Inflow_Divert")
End Sub
Private Sub OnIterationTop()
    'Date is a VB.NET class; the Index2Date Method in the TimeStepManager
    'class converts the currentModelTimeStepIndex to a Date, where
    'TypeIndexes.ModelIndex specifies that time step index begins at
    'simulation start date and continues until the simulation end date
    Dim currDate As Date = myModel.TimeStepManager.Index2Date _
        (myModel.mInfo.CurrentModelTimeStepIndex, TypeIndexes.ModelIndex)
    'declare nd as the variable holding the total days in the current month
    Dim nd as Integer = currDate.DaysInMonth(currDate.Year, currDate.Month)
    'PI is a field in the VB Math class
    m_rad = (currDate.Day - 1) * Math.PI / (nd - 1)
    'VB.NET CType function used to convert real no. to integer
    toMunicipal.mlInfo.hi = _
        CType(inflowLink.mlInfo.flow * Math.Sin(m_rad), Integer)
    toIndustrial.mlInfo.hi = _
        Ctype(inflowLink.mlInfo.flow - toMunicipal.mlInfo.hi, Integer)
End Sub
```

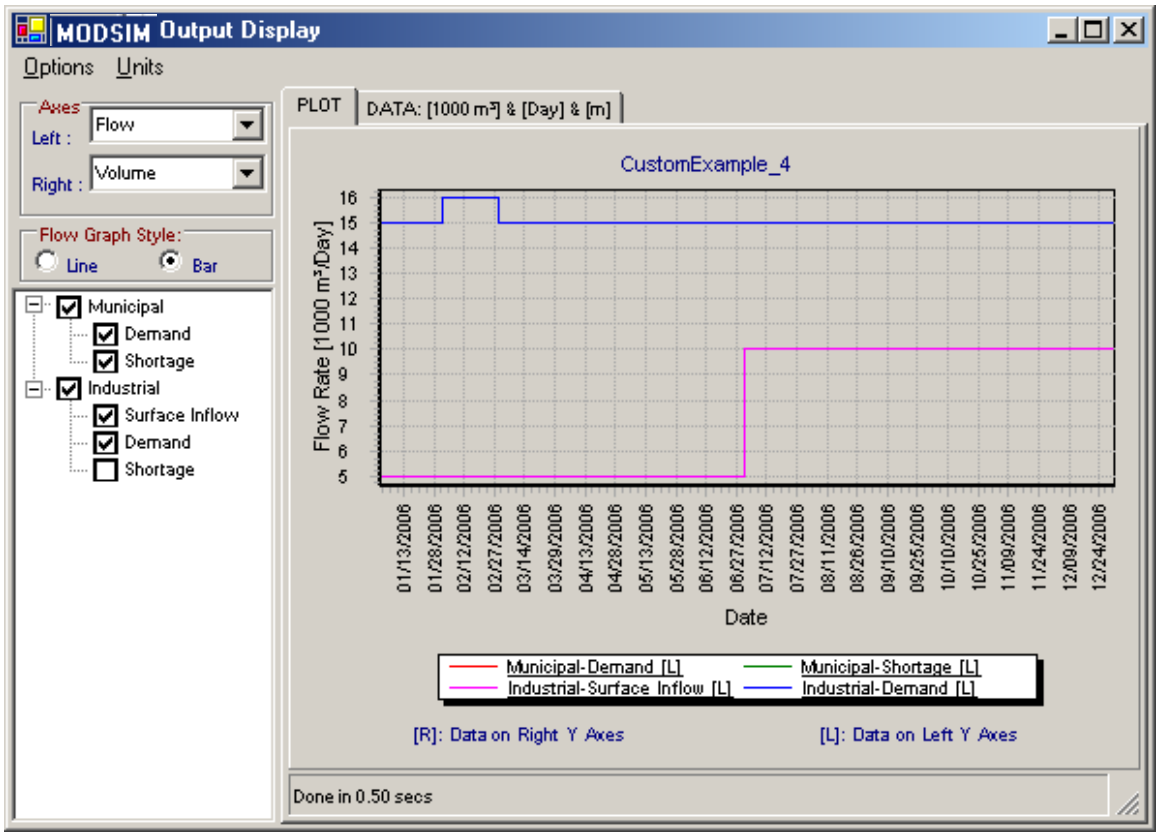
Saving this file as **Custom_3.vb**, compiling, and running from the **Custom Code Editor** produces the same results below when plotted in the MODSIM graphical user interface.



Example #4: Adaptive Adjustment of Link Costs

This example illustrates how custom code can be used to dynamically adjust costs assigned to links for a number of purposes. In this case, costs will be altered based on the flows in each link that are accumulated since the start of the simulation to the current time period. This could be utilized as a mechanism to balance total seasonal accumulations between several demands over the simulation period.

Open the previously saved daily network **CustomExample_3.xy**, which will also be used in this example. Save the network as **CustomExample_4.xy** and run from the MODSIM interface. The results shown below indicate that even though the two demands have the same priority, MODSIM arbitrarily gives all the available flow to *Industrial* and none to *Municipal*. This allocation could be easily reversed by assigning a higher priority to *Municipal*, but it is desired to achieve at least some degree of balance of total water deliveries over the entire simulation period.

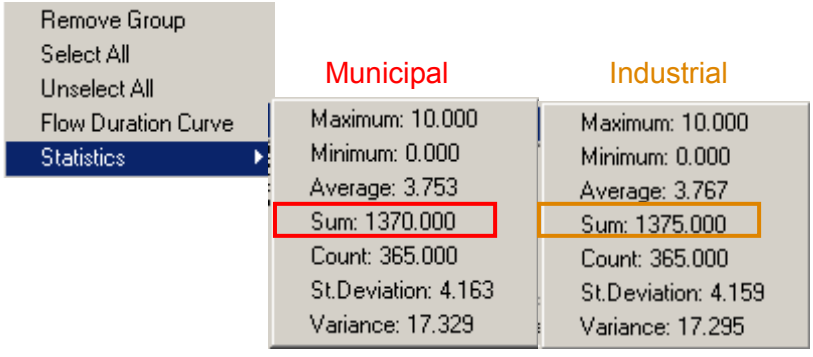


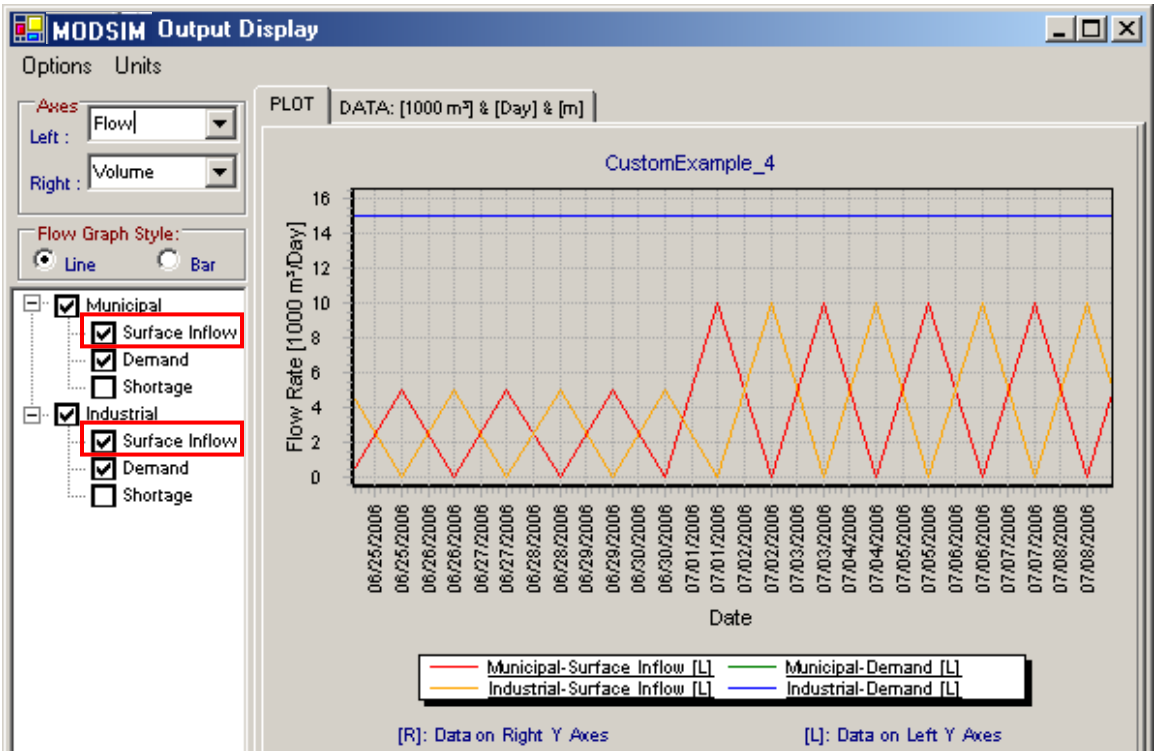
Open the **Custom Code Editor**, create a **New** template, and enter the Visual Basic code as shown below:

```
'declare the variables to hold the links toMunicipal, toIndustrial
Dim toMunicipal, toIndustrial As Link
'declare the variables accumulating the flow in each link
Dim accumMunicipal, accumIndustrial As Double
Private Sub OnInitilize()
'find the links in the network and assign to the link objects
    toMunicipal = myModel.FindLink("Divert_Municipal")
    toIndustrial = myModel.FindLink("Divert_Industrial")
End Sub
Private Sub OnIterationTop()
End Sub
Private Sub OnMessage(ByVal message As String)
End Sub
Private Sub OnIterationBottom()
End Sub
Private Sub OnIterationConverge()
    'accumulate the flow in each link
    accumMunicipal += toMunicipal.mlInfo.flow
    accumIndustrial += toIndustrial.mlInfo.flow
    'change costs based on accumulated flow relative to each link
    If accumMunicipal < accumIndustrial Then
        toMunicipal.mlInfo.cost = -10
        toIndustrial.mlInfo.cost = 0
    Else
        toMunicipal.mlInfo.cost = 0
        toIndustrial.mlInfo.cost = -10
    End If
End Sub
```

Notice that for this example, the module `OnIterationConverge()` is used since flows accessed here have converged to the final values in each time step. Saving this file as **Custom_4.vb**, compiling, and running from the **Custom Code Editor** produces the results below when plotted in the MODSIM graphical user interface.

Right-mouse click on Surface Inflow for both *Municipal* and *Industrial* in the **Output Display** table of contents produces statistical information indicating that total accumulated flow delivered to each demand is approximately equal (i.e., 1370 and 1375 (1000 m^3), respectively).

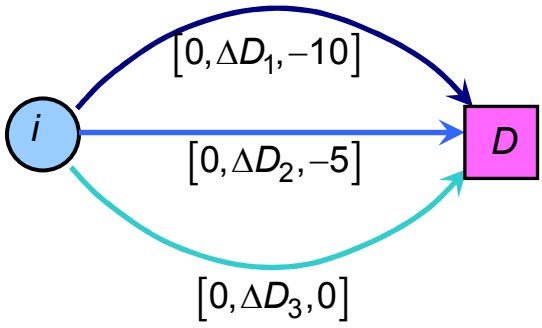




Example #5: Adjustment of Link Bounds for Equitable Flow Distribution

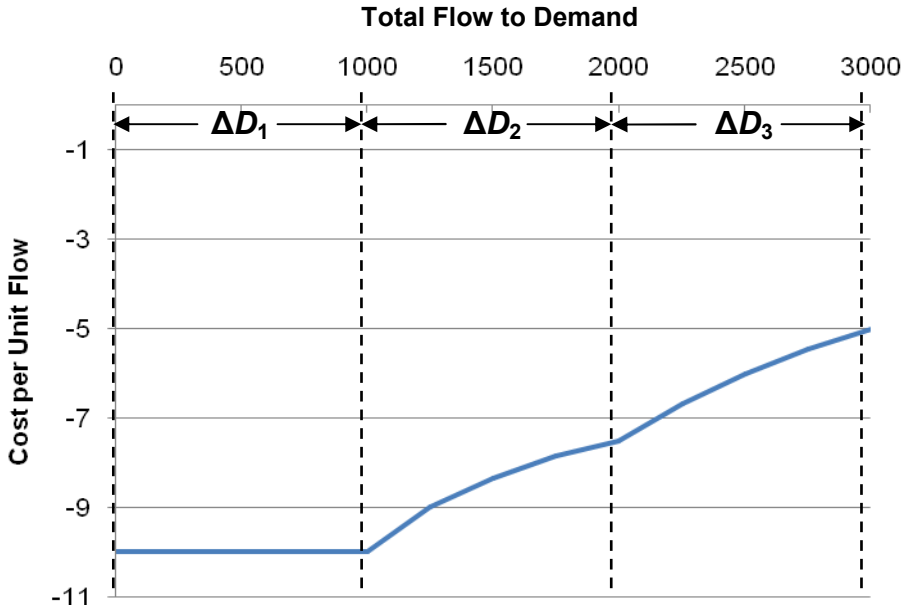
One method of equitably distributing flows to demands with the same priority is to create multiple links connecting each demand, where each link carries a fraction of the demand. By defining small, incrementally decreasing negative costs on each link of the multiple link set, the network optimization algorithm allocates the flows incrementally so as to balance the total deliveries to each demand. If multiple links are created linking demands with the same priority, as illustrated below, flows will first be distributed to the links connecting all the demands that have the most negative costs, with additional flows distributed to the higher cost links incrementally.

$$\left[l_k^n, u_k^n, c_k^n \right] \text{ for } n = 1,2,3$$



$$\Delta D_1 + \Delta D_2 + \Delta D_3 = D \text{ (total demand)}$$

With this structure, the distribution of flows to demands under the same priority is based on increasing unit costs as a function of total flow delivered to the demand.

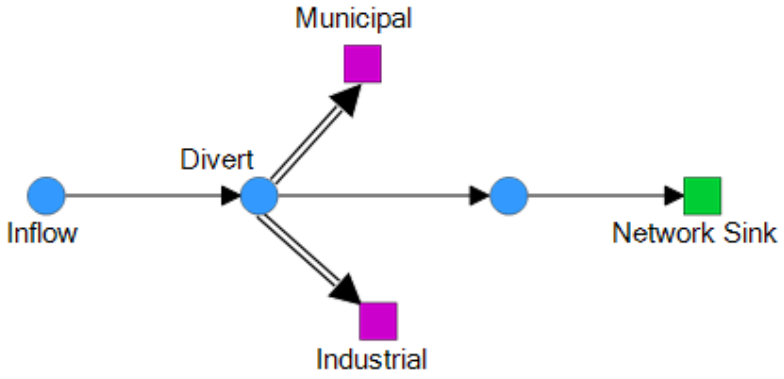


Open **CustomExample_4.xy** from the previous exercise and save it as **CustomExample_5.xy**. Open the **Custom Code Editor** and create a **New** template. A new procedure called `CreateLinks()` is included and called from the `Main` module. This procedure loops through all nodes in the network, starting with `myModel.firstNode` as the first created node in the network. If the node is a member of the class `NodeTypes.Demand`, then `dsNode` is defined as the origin node for the link connected to the demand node using the Public Field `m_node.in.link.from` (it is assumed that a single link already exists between `dsNode` and the demand node `m_node`). A `For` loop is used to create four additional links connecting the two nodes using the `ConnectFromNode` and `ConnectToNode` methods in the `Utils` class. Each new link is then assigned a unique name and an incremental cost in the Public Fields `Newlink.name` and `Newlink.m.cost`, respectively.

The Procedure `CreateLinks()` is called in the Main module since it actually changes the physical topology of the network. After creation of the network, it is written to a new file **CustomExample_5b.xy** using the `XYFileWriter.Write` method, with the argument for the string with the file name including the path to the file location. The file must be written prior to the units conversion routine for version 8.1 `m_81Units.Units_Conversion_Controller(myModel)` and execution of the model using `myModsim.RunSolver(myModel)` in the `Main` module. The following custom code listing should be inserted into the **New** template, saved as **Custom_5.vb**, compiled and run from the **Custom Code Editor**.

```
Dim myModel As New Model
Sub Main(ByVal CmdArgs() As String)
    Dim FileName As String = cmdargs(0)
    AddHandler myModel.Init, AddressOf OnInitilize
    AddHandler myModel.IterBottom, AddressOf OnIterationBottom
    AddHandler myModel.IterTop, AddressOf OnIterationTop
    AddHandler myModel.Converged, AddressOf OnIterationConverge
    AddHandler myModel.End, AddressOf OnFinished
    AddHandler myModel.OnMessage, AddressOf OnMessage
    AddHandler myModel.OnModsimError, AddressOf OnMessage
    XYFileReader.Read(myModel, FileName)
    Dim myModsim As Modsim
    'call CreateLinks before the network is initialized
    CreateLinks()
    'write the new *.xy file with the multilinks created
    XYFileWriter.Write(myModel, "c:\KModSim\Custom\CustomExample_5b.xy")
    Dim m_81Units As New Csu.Modsim.NetworkUtils.ManageUnits
    m_81Units.Unit_Conversion_Controller(myModel)
    myModsim.RunSolver(myModel)
End Sub
Private Sub CreateLinks()
'loop through all demand nodes starting at myModel.firstNode
    Dim m_node As Node = myModel.firstNode
    Do
        If m_node.NodeType = NodeTypes.Demand Then
        'declare dsNode as the origin node from which flow is conveyed
        'to the current demand node m_node
            Dim dsNode As Node = m_node.in.link.from
            Dim j As Integer
        'loop over additional 4 links connecting dsNode and m_node
            For j = 1 To 4
            'create each new link as part of the real network
                Dim Newlink As Link = myModel.AddNewLink(True)
            'insert the link between dsNode and m_node
                Utils.ConnectFromNode(Newlink, dsNode)
                Utils.ConnectToNode(Newlink, m_node)
            'give each new link a unique name
                Newlink.name = m_node.name & "Link_" & j
            'assign an arbitrary incremental cost to each link
                Newlink.m.cost = -5 * j
            Next
    End If
'go to the next demand node
    m_node = m_node.next
Loop Until m_node Is Nothing
Sub
```

After successful compilation and execution of the network **CustomExample_5.xy** using the custom code **Custom_5.vb** from the **Custom Code Editor**, the current network **CustomExample_5.xy** should be closed and the new network opened. With the opening of the new network **Custom_Example_5b.xy**, it is noted that the desired multilinks have indeed been created, as shown below.



Opening the link between nodes *Divert* and *Municipal* reveals that the incremental costs have been correctly assigned to each link.

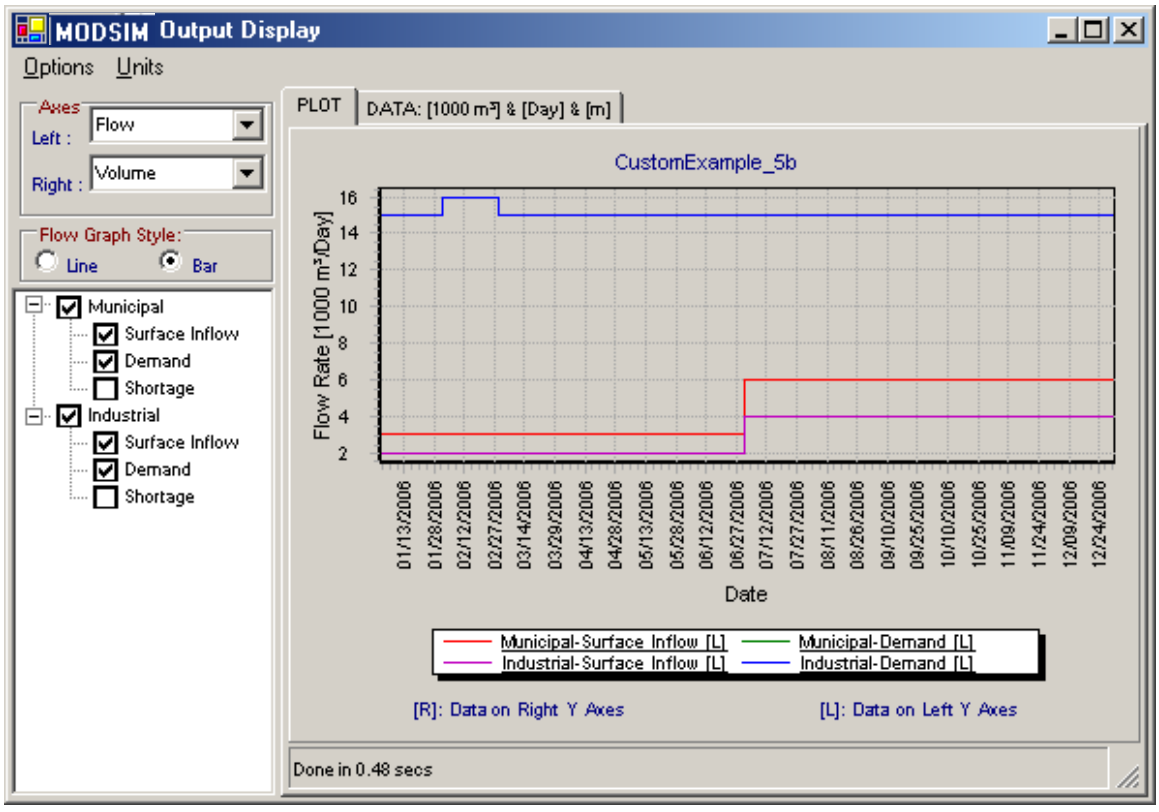
Links between Divert & Municipal

	Link Name	Link Type	Cost	Order of Use	Channel Type
13	MunicipalLink_4	Standard	-20	NA	Default
12	MunicipalLink_3	Standard	-15	NA	Default
11	MunicipalLink_2	Standard	-10	NA	Default
10	MunicipalLink_1	Standard	-5	NA	Default
3	Divert_Municipal	Standard	0	NA	Default

Again, open the **Custom Code Editor**, create a **New** template, and insert the following VB.NET code into the `OnIterationTop()` module. This custom code accesses the demand `currentDem` at the current time step and assigns the capacities of the five links now connected to each demand node as an amount equal to 1/5 of the demand. The `For Next` block loops over each demand node, with an upper limit in the `For` loop set as `myModel.minfo.demListLen - 1` so as to exclude the *Network Sink*, which is included in the `demListLen` of the total number of demand nodes in the network. A `Do Loop` block is used for looping over each of the five links connecting each demand node by updating `m_LinkList` to `m_LinkList.next` until the set of nodes in `m_LinkList` is empty. Save the custom code as **Custom_5b.vb**.

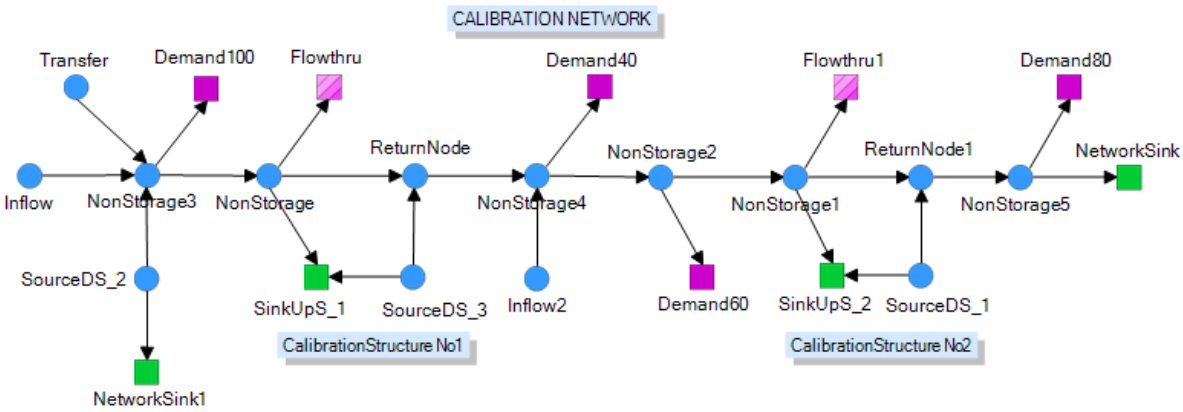
```
Private Sub OnIterationTop()
    Dim m_node As Node
    Dim i, j As Integer
    For i = 1 To myModel.mInfo.demListLen - 1
        m_node = myModel.mInfo.demList(i)
        Dim currentDem As Integer
        currentDem = m_node.mnInfo.nodedemand _
            (myModel.mInfo.CurrentModelTimeStepIndex, 0)
        Dim m_LinkList As LinkList = m_node.in
        Dim m_link As Link = m_LinkList.link
        Do
            m_LinkList.link.mlInfo.hi = currentDem / 5
            m_LinkList = m_LinkList.next
        Loop Until m_LinkList Is Nothing
    Next
End Sub
```

Compiling and executing from the **Custom Code Editor** produces the following results when plotted in the MODSIM graphical user interface. It can be seen that deliveries to the demand or more equitable, although not completely balanced. Increasing the number of links connected to the demand nodes will produce more equitable results since the flow discretization defined by the capacities assigned to each link will be smaller.



Advanced Example #6: Custom Network Calibration

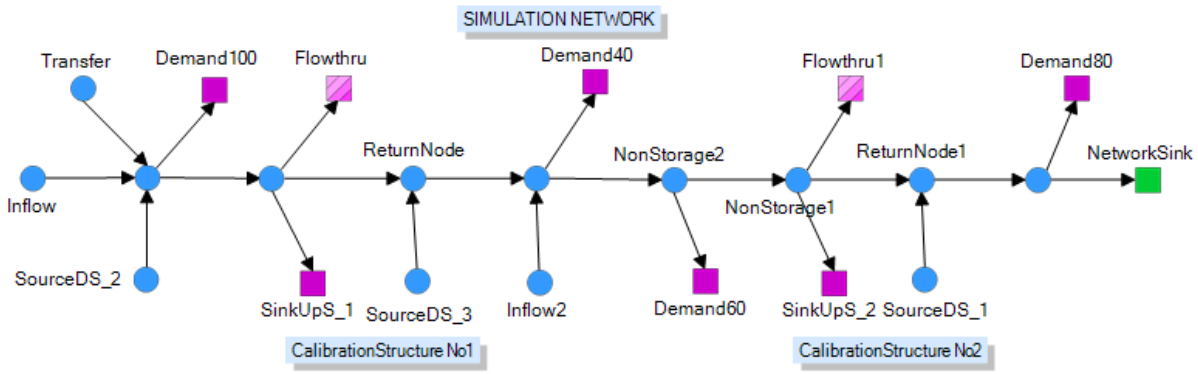
This example uses a calibration network to estimate gains and losses to the river reach based on actual measured flows at two gauging station locations. Custom code is used to import the calibration results into the simulation network. Use the provided calibration network file **CalibrationNetwork.xy**.



This calibration network includes actual measured flows at nodes *NonStorage* and *NonStorage1* representing gauging station locations by assigning these flows as flow-through demands at the nodes *Flowthru* and *Flowthru1*, respectively. The source nodes *SourceDS_1*, *SourceDS_2*, and *SourceDS_3* provide additional unmeasured gains to the river needed for network flows at the gauging station locations to match the actual measured flows. *NetworkSink1* removes excess flow originating from the node *SourceDS_2* that is not needed to provide the necessary calibration flows. The network sinks *SinkUpS_1* and *SinkUpS_2* serve to remove excess flows from source nodes *SourceDS_1* and *SourceDS_2*, respectively, that are not necessary for matching the measured flows. In addition, these sink nodes remove excess flows representing unmeasured losses along the river. These estimates of unmeasured river gains and losses that result in network simulation discharges matching the measured flows can then be imported into the network for simulation purposes, which is accomplished using custom code.

.

Execute the network **CalibrationNetwork.xy** and note the Microsoft Access database CalibrationNetworkOUTPUT.mbd that is created. The calibration flows representing gains and losses along the river will be imported into the **SimulationNetwork.xy** network using custom code. **SimulationNetwork.xy** is constructed from **CalibrationNetwork.xy** by removing *NetworkSink1*, and replacing network sinks *SinkUpS_1* and *SinkUpS_2* with demand nodes of the same name. The links *SourceDS_1_SinkUpS_1* and *SourceDS_2_SinkUpS_2* are also removed from the simulation network. It is also important to make sure that the links connected to the source and sink nodes have exactly the same names as the nodes they are connected to. For example, link *NonStorage_SinkUpS_1* should be simply renamed *SinkUpS_1*, etc. Make sure to set the time series of the Sink demands and Source inflows to zero, and set the cost on the links connected to *SinkUpS_1* and *SinkUpS_2* to a large negative value of



-70000, which serves to force the calibrated gains into the river system. Also, open the bypass links for the flow-through demands and close the links to the flow-through demands in order to neutralize the effect to these nodes.

Open the **Custom Code Editor** and create a **New** template. Add the reference

```
Imports Csu.Modsim.NetworkUtils
```

Declare global variables for the MSDB output class `MSDBOut` and the `CalibrationFileName` with the path to the file included.

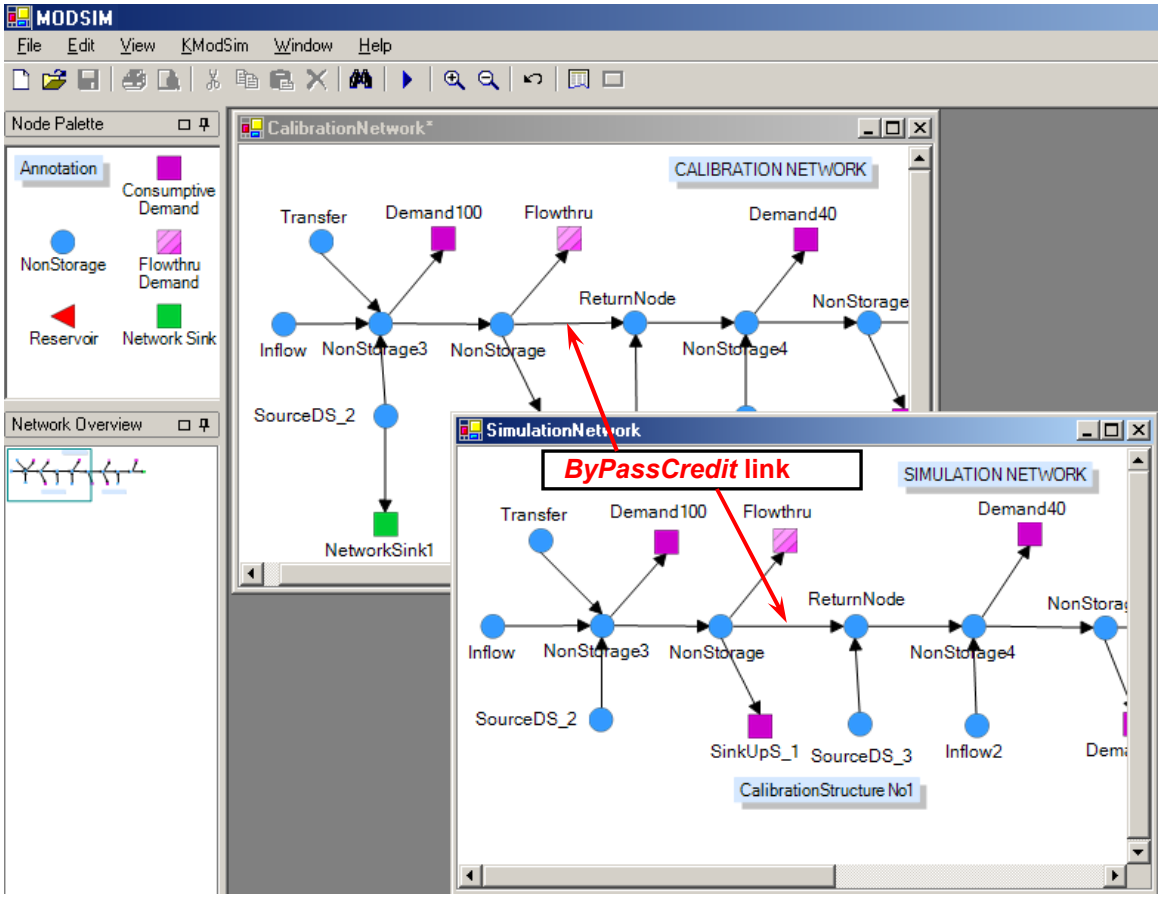
```
Imports Csu.Modsim.NetworkUtils
Imports Csu.Modsim.ModsimIO
Imports Csu.Modsim.ModsimModel
Imports System
Module CustomMOSDIM
    Sub Main(ByVal CmdArgs() As String)
        Dim FileName As String = cmdargs(0)
        AddHandler myModel.Init, AddressOf OnInitilize
        AddHandler myModel.IterBottom, AddressOf OnIterationBottom
        AddHandler myModel.IterTop, AddressOf OnIterationTop
        AddHandler myModel.Converged, AddressOf OnIterationConverge
        AddHandler myModel.End, AddressOf OnFinished
        AddHandler myModel.OnMessage, AddressOf OnMessage
        AddHandler myModel.OnModsimError, AddressOf OnMessage
        XYFileReader.Read(myModel, FileName)
        Dim myModsim As Modsim
        myModsim.RunSolver(myModel)
    End Sub
    Dim myModel As New Model
    Dim MSDBOut As ModelOutputMSDB
    Dim CalibrationFileName As String = _
        "c:\KModSim\Custom\CalibrationNetwork.xy"
    Private Sub OnInitilize()
        Dim m_CalModel As Model = New Model
        XYFileReader.Read(m_CalModel, CalibrationFileName)
        MSDBOut = New ModelOutputMSDB(m_CalModel)
    End Sub
```

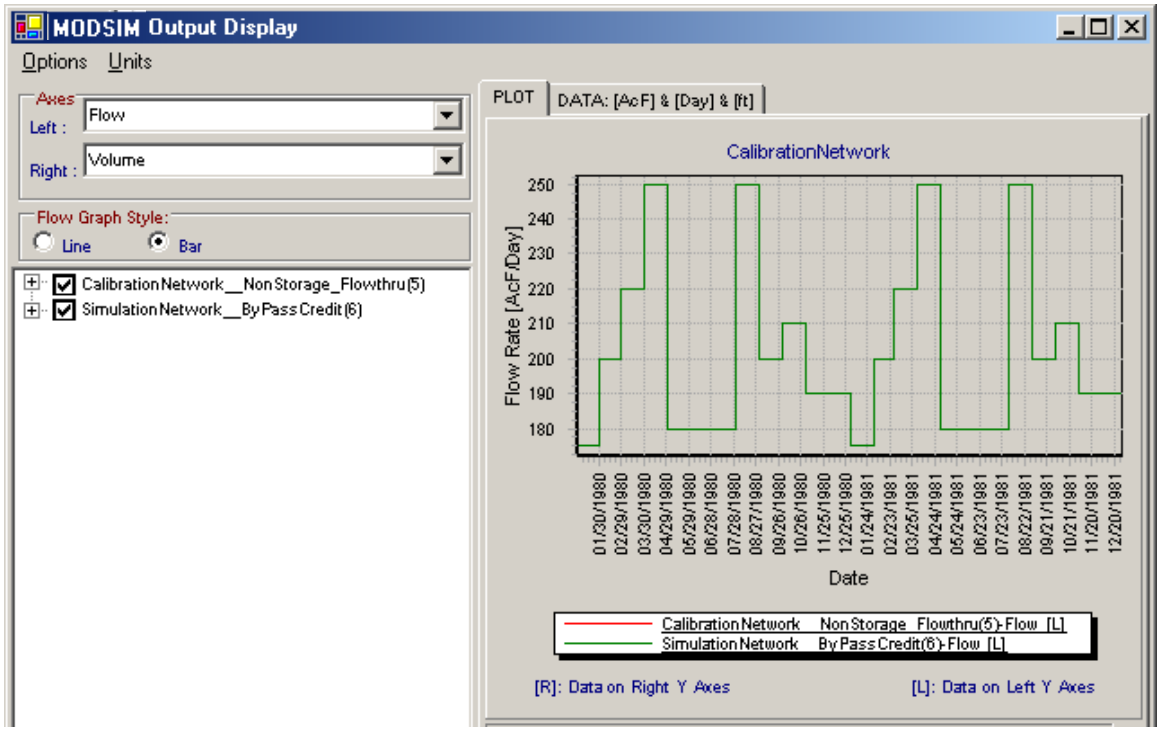
In the `OnInitilize()` procedure, the calibration network `m_CalModel` is declared as an instance of the Model class and is read using `XYFileReader.Read`. Assuming that **CalibrationNetwork.xy** has been successfully executed, MSDBOut is a new instance of the `ModelOutputMSDB` class which is populated with the data base output of `m_CalModel`.

Code is now added to the `OnIterationTop()` procedure to access flows from the output database produced by the **CalibrationNetwork.xy** execution for those links going to the sources and sinks (i.e., the nodes defining gains and losses to the network such that measured flows are matched by calculated flows in **CalibrationNetwork.xy**). These flows are then assigned to the source inflow and demand sink nodes in **SimulationNetwork.xy** that will provide the calculated gains and losses to the system that were estimated in the Calibration Network.

```
Private Sub OnIterationTop()
    If myModel.mInfo.Iteration = 0 Then
        Dim m_CurrIndex As Integer = myModel.mInfo.CurrentModelTimeStepIndex
        Dim m_node As Node = myModel.firstNode
        Do
            If Not m_node.name Is Nothing Then
                If m_node.name.StartsWith("SinkUpS_") Then
                    If m_node.mnInfo.nodedemand.Length = 0 Then
                        ReDim m_node.mnInfo.nodedemand _
                            (myModel.TimeStepManager.noModelTimeSteps, 1)
                    End If
                    m_node.mnInfo.nodedemand(m_CurrIndex, 0) = MSDBOut.LinkOutputQuery _
                        (ModelOutputMSDB.LinkOutputType.Flow, m_node.name, m_CurrIndex)
                End If
                If m_node.name.StartsWith("SourceDS_") Then
                    If m_node.mnInfo.inflow.Length = 0 Then
                        ReDim m_node.mnInfo.inflow _
                            `(myModel.TimeStepManager.noModelTimeSteps, 1)
                    End If
                    m_node.mnInfo.inflow(m_CurrIndex, 0) = MSDBOut.LinkOutputQuery _
                        (ModelOutputMSDB.LinkOutputType.Flow, m_node.name, m_CurrIndex)
                End If
            End If
            m_node = m_node.next
        Loop Until m_node Is Nothing
    End If
End Sub
```

Save this custom code as **Custom_6.vb**, compile, and run from the **Custom Code Editor**. Using the **Scenarios Analysis** tool (**MODSIM > Scenarios Analysis**) in the MODSIM graphical user interface and simultaneously plotting the output flows to links going to to the *Flowthru* and *Flowthru1* nodes in **CalibrationNetwork.xy** with the flows in links *BypassCredit* and *BypassCredit1*, respectively, in **SimulationNetwork.xy** reveals that the flows are exactly the same.


MODSIM
File Edit View KModSim Window Help
Node Palette
Annotation
Consumptive Demand
NonStorage
Flowthru Demand
Reservoir
Network Sink
Network Overview
CalibrationNetwork*
CALIBRATION NETWORK
Transfer
Demand100
Flowthru
Demand40
ReturnNode
NonStorage
Inflow
NonStorage3
NonStorage
NonStorage4
SourceDS_2
NetworkSink1
SimulationNetwork
ByPassCredit link
SIMULATION NETWORK
Transfer
Demand100
Flowthru
Demand40
ReturnNode
Inflow
NonStorage3
NonStorage
NonStorage4
SourceDS_2
SinkUpS_1
SourceDS_3
Inflow2
CalibrationStructure No1

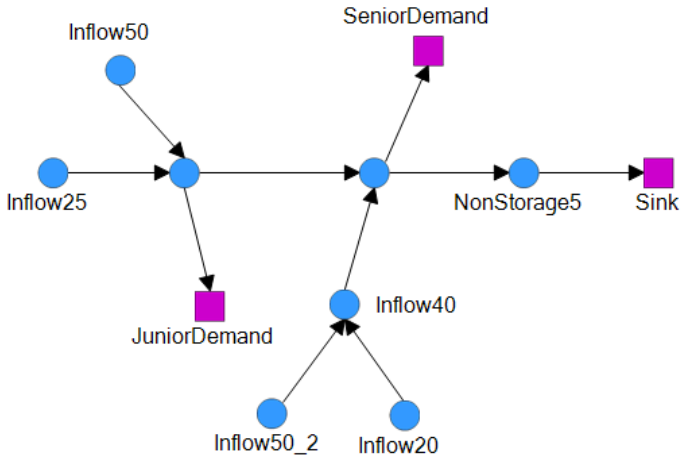

MODSIM Output Display
Options Units
Axes
Left : Flow
Right : Volume
Flow Graph Style:
Line
Bar
CalibrationNetwork__NonStorage_Flowthru(5)
SimulationNetwork__ByPassCredit(6)
PLOT
DATA: [AcF] & [Day] & [ft]
CalibrationNetwork
Flow Rate [AcF/Day]
250
240
230
220
210
200
190
180
01/30/1980
02/29/1980
03/30/1980
04/29/1980
05/29/1980
06/28/1980
07/28/1980
08/27/1980
09/26/1980
10/26/1980
11/25/1980
12/25/1980
01/24/1981
02/23/1981
03/25/1981
04/24/1981
05/24/1981
06/23/1981
07/23/1981
08/22/1981
09/21/1981
10/21/1981
11/20/1981
12/20/1981
Date
CalibrationNetwork NonStorage Flowthru(5)-Flow [L]
SimulationNetwork ByPassCredit(6)-Flow [L]
[R]: Data on Right Y Axes
[L]: Data on Left Y Axes

Advanced Example #7: Custom Water Quality Module

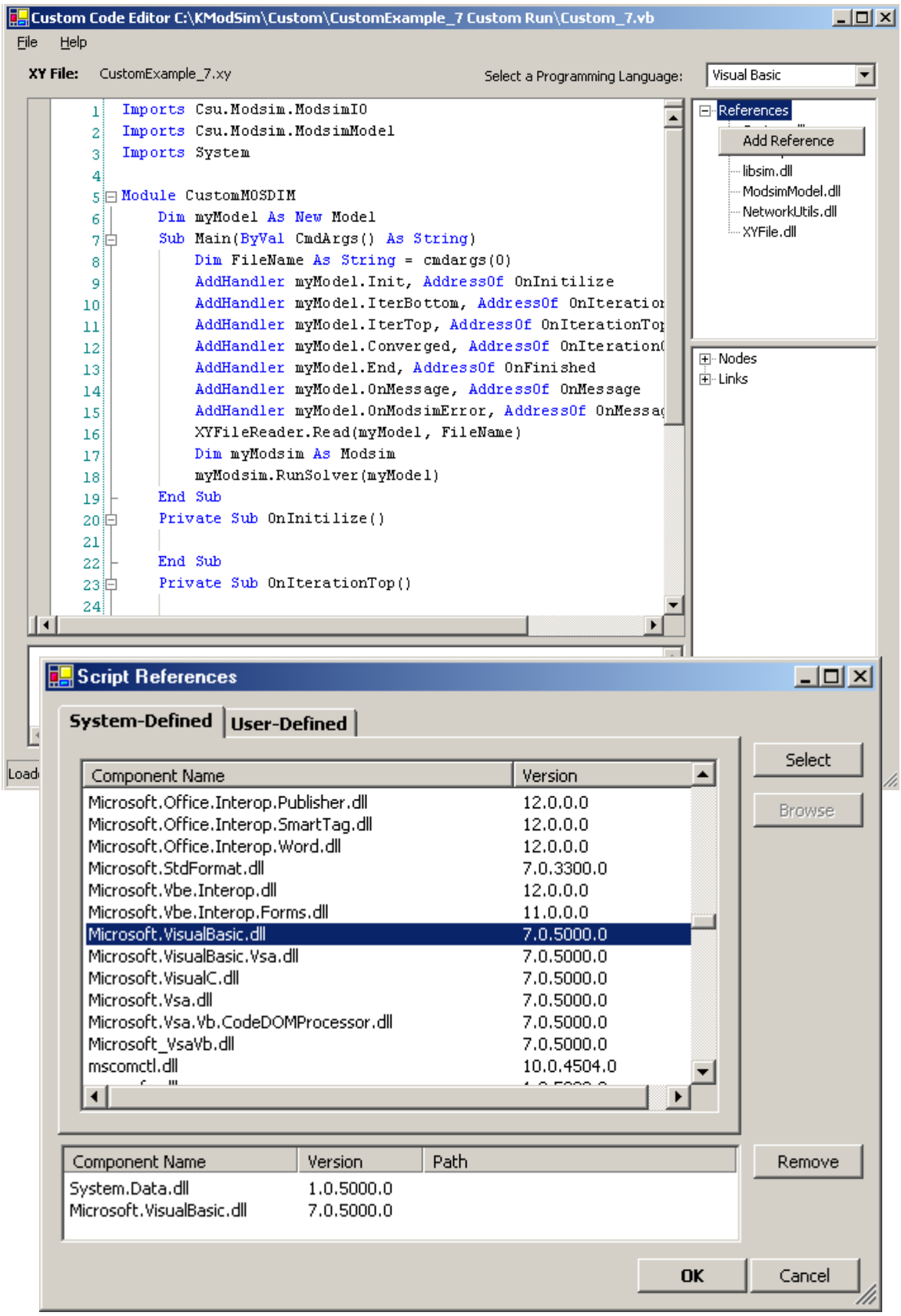
This example shows how to use the topological network tracing algorithm built into MODSIM to develop a basic water quality module that dynamically routes water quality concentrations throughout the network. Conservative assumptions are used for routing of flow concentrations in this example, which could represent total dissolved solids (TDS) in this case.

Create the test network shown below, where in this case the Network Sink is replaced with a standard demand node called *Sink*, since Network Sink nodes are not currently supported in the network tracing algorithm. Give *Sink* a large demand (e.g., 999999) and a low priority (e.g., 5000).

All inflows are set equal to 100 (1000 m^3/month), with a constant demand of 50 (1000 m^3/month) assigned to the *JuniorDemand* node and 25 (1000 m^3/month) to *SeniorDemand*. For this example, the measured water quality concentrations associated with the inflows are entered into **Description** field of **Nonstorage Node Properties** form of each inflow node. Enter the concentrations of the inflows as indicated by the node names; e.g., for *Inflow25* enter 0.25 in the **Description** field.



Open the Custom Code Editor, create a **New** template, and right-mouse click on References in the upper right-hand side area of the form, and click the context button **Add Reference** that appears. This opens the **Add References** form allowing selection of certain system-defined dll's needed for this example to be referenced in the custom code. Under the System-Defined tab, scroll to the *Component Name* "Microsoft.VisualBasic.dll" and click the **Select** button. Then scroll to the component System.Data.dll and again click the Select button. Notice that the selected components are now listed in the lower area of the Add References form. Clicking **OK** reveals that the selected components are now included in the list of references in the **Custom Control Editor**. Since the "Microsoft.dll" is not needed for this example, right-mouse click on this component and click the context button **Delete Reference** to remove it from the list.


Custom Code Editor C:\KModSim\Custom\CustomExample_7 Custom Run\Custom_7.vb
File
Help
XY File:
CustomExample_7.xy
Select a Programming Language:
Visual Basic
Imports Csu.Modsim.ModsimIO
Imports Csu.Modsim.ModsimModel
Imports System
Module CustomMOSDIM
Dim myModel As New Model
Sub Main(ByVal CmdArgs() As String)
Dim FileName As String = cmdargs(0)
AddHandler myModel.Init, AddressOf OnInitilize
AddHandler myModel.IterBottom, AddressOf OnIteratio
AddHandler myModel.IterTop, AddressOf OnIterationTo
AddHandler myModel.Converged, AddressOf OnIteration(
AddHandler myModel.End, AddressOf OnFinished
AddHandler myModel.OnMessage, AddressOf OnMessage
AddHandler myModel.OnModsimError, AddressOf OnMessa
XYFileReader.Read(myModel, FileName)
Dim myModsim As Modsim
myModsim.RunSolver(myModel)
End Sub
Private Sub OnInitilize()
End Sub
Private Sub OnIterationTop()
References
Add Reference
libsim.dll
ModsimModel.dll
NetworkUtils.dll
XYFile.dll
Nodes
Links
Load
Script References
System-Defined
User-Defined
Component Name
Version
Microsoft.Office.Interop.Publisher.dll 12.0.0.0
Microsoft.Office.Interop.SmartTag.dll 12.0.0.0
Microsoft.Office.Interop.Word.dll 12.0.0.0
Microsoft.StdFormat.dll 7.0.3300.0
Microsoft.Vbe.Interop.dll 12.0.0.0
Microsoft.Vbe.Interop.Forms.dll 11.0.0.0
Microsoft.VisualBasic.dll 7.0.5000.0
Microsoft.VisualBasic.Vsa.dll 7.0.5000.0
Microsoft.VisualC.dll 7.0.5000.0
Microsoft.Vsa.dll 7.0.5000.0
Microsoft.Vsa.Vb.CodeDOMProcessor.dll 7.0.5000.0
Microsoft_VsaVb.dll 7.0.5000.0
mscomctl.dll 10.0.4504.0
Select
Browse
Component Name
Version
Path
System.Data.dll 1.0.5000.0
Microsoft.VisualBasic.dll 7.0.5000.0
Remove
OK
Cancel

With the required references added, the Namespaces for those object classes are imported as listed below for convenience in accessing them in the code. The global variable `nodeCalcOrder`, which is `Private` to the `WaterQualityModule`, is declared as a Collection, which is a listing of objects. The `nodeCalcOrder` Collection is then populated by invoking the `NetTopology.findNetworkUpStream` method which is included in the object class `NetworkUtils`.

```
Imports Csu.Modsim.ModsimIO
Imports Csu.Modsim.ModsimModel
Imports Csu.Modsim.NetworkUtils
Imports System
Imports Microsoft.VisualBasic
Imports System.Collections
Imports System.Data
'change name to WaterQualityModule
Module WaterQualityModule
    Dim myModel As New Model
    Private nodeCalcOrder As Collection
    Sub Main(ByVal CmdArgs() As String)
        Dim FileName As String = CmdArgs(0)
        AddHandler myModel.Init, AddressOf OnInitilize
        AddHandler myModel.IterBottom, AddressOf OnIterationBottom
        AddHandler myModel.IterTop, AddressOf OnIterationTop
        AddHandler myModel.Converged, AddressOf OnIterationConverge
        AddHandler myModel.End, AddressOf OnFinished
        AddHandler myModel.OnMessage, AddressOf OnMessage
        AddHandler myModel.OnModsimError, AddressOf OnMessage
        XYFileReader.Read(myModel, FileName)
        'To trace the network, build list of nodes from most upstream to downstream
        nodeCalcOrder = NetTopology.findNetworkUpStream(myModel)
        Dim myModsim As Modsim
        myModsim.RunSolver(myModel)
    End Sub
```

A new class `QualityLinkData` is created *outside* of the `WaterQualityModule` to keep track of water quality concentrations in the network.

```
'new QualityLinkData class contains link water quality data placed in link.tag
Public Class QualityLinkData
    Public concentration As Single
    'Flag = true if concentration has been calculated
    Public valueSet As Boolean
    Sub New()
        valueSet = False
    End Sub
    Sub SetValue(ByVal aConc As Single)
        concentration = aConc
        valueSet = True
    End Sub
End Class
```

For this exercise, it is desired to create an additional column to the normal MODSIM output database giving water quality concentrations in the links. The `WithEvents` keyword in VB.NET allows the user to perform this action when the output event is triggered. The variable `m_OutputSupport` is declared as an instance of the `ModelOutputSupport` class in the `OnInitilize()` procedure. A `Handler` defines the address for the method that is launched when the output support event is triggered. The new Procedure `addLinkQualityOutput` assigns the new output to new Concentration column in the data base for each link and for each time step row in the data base.

```
'WithEvents keyword declares m_OutputSupport as instance of output
'event handler
Private WithEvents m_OutputSupport As ModelOutputSupport
Private Sub OnInitilize()
    'set the new output variable to the model output support class
    m_OutputSupport = myModel.OutputSupportClass
    'AddUserDefinedOutputVariable method with the ModelOutputSupport class;
    'parameters are (1) model, (2) name of new output, (3) True for links,
    '(4) False for nodes (5) string label on axis in output form
    m_OutputSupport.AddUserDefinedOutputVariable _
        (myModel, "Concentration", True, False, "Concentration [mg/L]")
    'Handler defines address for the method that is launched when output
    'support event is triggered
    AddHandler m_OutputSupport.AddCurrentUserLinkOutput, _
        AddressOf addLinkQualityOutput
End Sub
'new output assigned to column Concentration by link and row (time step)
Private Sub addLinkQualityOutput(ByVal m_link As Link, _
    ByRef m_row As DataRow)
    m_row("Concentration") = m_link.Tag.concentration
End Sub
```

The following routine is created *within* the module to add the new `QualityLinkData` class to each link tag object.

```
'new Procedure adds the new QualityLinkData class to each link tag object
Private Sub NewQualityInLinks()
    Dim cur_Link As Link
    cur_Link = myModel.firstLink
    Do
        cur_Link.Tag = New QualityLinkData
        cur_Link = cur_Link.next
    Loop Until cur_Link Is Nothing
End Sub
```

Once the flow calculations in MODSIM have converged for each time step, the water quality concentrations in each link must be dynamically calculated based on these flows. After the network has been initialized, the `OnIterationConverge()` procedure first calls the new procedure `NewQualityInLinks` that initializes the water quality concentrations of flows in the links. A `For…Next` block then loops through each node in the `nodeCalcOrder` collection from upstream to downstream, since this is the correct order in which the calculations must proceed (it is assumed that there are no loops in the network). and accesses the artificial link entering each inflow node. The variables `curNode` and curLink are declared to hold the current node in the loop and the artificial inflow link. The concentration value entered into the **Description** field in the **Node Properties** form is then set to a real number and assigned to the link tag for specifying the concentration of flow in that link. The total accumulated mass of the water quality constituent is then divided by the total accumulated inflow volume to the current node, which is then assigned as the concentration for flows in all links leaving the node. Conservative assumptions of complete mixing are assumed in these calculations.

Saving the custom code as **Custom_7.vb**, compiling, and executing in the **Custom Code Editor** gives the results shown below when displayed in the MODSIM graphical user interface. Notice that a new axis label "Concentration [mg/L]" has now been added as an option for display, and the correct concentrations are plotted for three selected links. Of course, the concentrations do not vary since the flows are constant in this example, but since the water quality is dynamically calculated, time variable concentrations if time variable inflow data are imported into the inflow nodes.

```
Private Sub OnIterationConverge()
    'call Procedure that initializes concentrations on the links
    NewQualityInLinks()
    Dim i As Integer
    'collection nodeCalcOrder defines order of tracing the nodes
    For i = 1 To nodeCalcOrder.Count
    'enumerate each item in the nodeCalcOrder collection
        Dim myEnumeratorUp As IEnumerator = nodeCalcOrder.Item(i).getenumerator
        While myEnumeratorUp.MoveNext
            Dim curLink As Link
            Dim curNode As Node = myModel.FindNode(myEnumeratorUp.Current)
            ' infLink holds the artificial inflow link coming to the inflow node
            curLink = curNode.mnInfo.infLink
            'assign concentrations to the inflows
            If curLink.mlInfo.flow > 0 Then
                'get the value from the Description field of the node and assign
                'to the concentration variable
                curLink.Tag.setvalue(CType(curNode.description, Single))
            End If
            Dim inMass As Single = 0
            Dim inWaterVol As Single = 0
            Dim m_InListLink As LinkList = curNode.in
            'loop through all links coming into current node to accumulate total
            'mass entering the node
```

```
            Do
                curLink = m_InListLink.link
                'only links with previously calculated concentration are used
                If curLink.Tag.valueset Then
                    'accumulate total constituent mass in the flow
                    inMass += curLink.Tag.concentration * curLink.mlInfo.flow
                    'accumulate total water volume
                    inWaterVol += curLink.mlInfo.flow
                End If
                m_InListLink = m_InListLink.next
            Loop Until m_InListLink Is Nothing
            'calculate concentration of flow leaving the node
            Dim outConcentration As Single = 0
            If inWaterVol > 0 Then outConcentration = inMass / inWaterVol
            Dim outLinkList As LinkList = curNode.out
            'assign concentrations to all links leaving the node--assumes
            'complete mixing
            Do
                curLink = outLinkList.link
                If Not curLink.mlInfo.isArtificial Then
                    If Not curLink.Tag.valueset Then
                        curLink.Tag.SetValue(outConcentration)
                    Else
                'error if concentration already assigned
                        Throw New Exception("ERROR!!! Concentration already _
                            assigned for this link")
                    End If
                End If
                outLinkList = outLinkList.next
            Loop Until outLinkList Is Nothing
        End While
    Next
End Sub
```