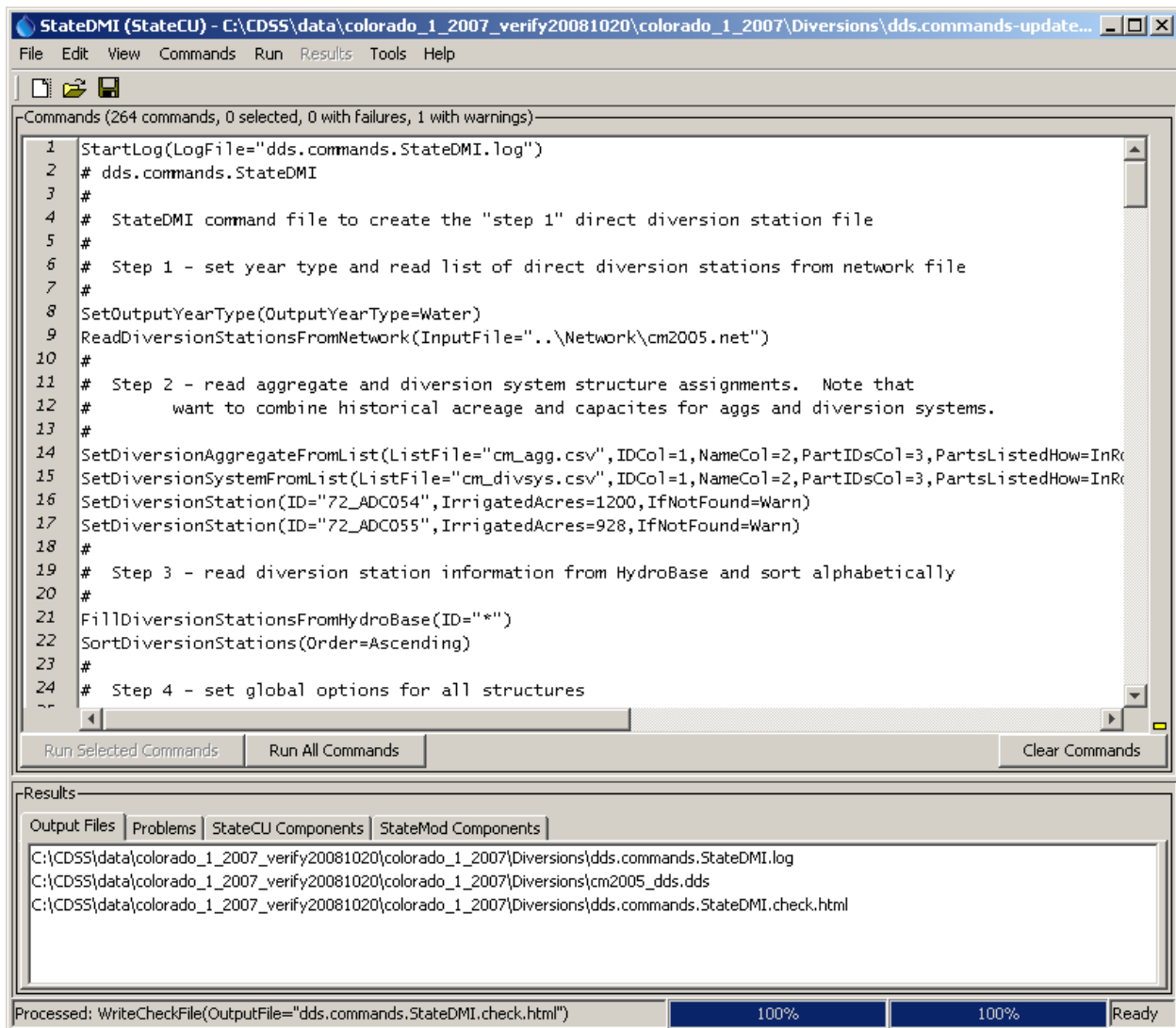


# StateDMI



**Colorado Department of Natural Resources  
Colorado Water Conservation Board  
Division of Water Resources**

*Developed by:*

 **Riverside Technology, inc.**

Version 03.11.00, 2010-08-11

This page is intentionally blank.

This document is formatted for double-sided printing.

Cover graphic is Main\_cover.

# Table of Contents

---

01_Cover.pdf	1
Blank Page	28
DISCLAIMER for CDSS Products	29
1 Acknowledgements	31
2 Introduction	33
2.1 How to Use this Documentation	34
2.2 CDSS Modeling Overview	35
2.3 Data Set Folder and File Conventions	36
2.4 Standard Procedures for Creating StateCU and StateMod Data Sets	40
2.5 Variations in StateMod Data Sets	40
2.6 Commands and Processing Sequence	43
3 Getting Started	45
3.1 Starting StateDMI	45
3.2 Select HydroBase Dialog	46
3.3 Main Interface	47
3.4 File Menu - Main Input and Output Control	55
3.5 Edit Menu – Editing Commands	58

# Table of Contents

---

3.6 View Menu – Enable/Disable Display Features_____	60
3.7 Commands Menu – Insert Commands for Processing Data Components_____	79
3.8 Run Menu – Running Commands_____	83
3.9 Results Menu – View Data Set and Command Results_____	84
3.10 Tools Menu_____	84
3.11 Help Menu_____	87
4 Creating StateCU Data Set Files_____	89
4.1 Control Data_____	89
4.2 Climate Station Data_____	90
4.3 Crop Characteristics/Coefficients Data_____	95
4.4 Delay Tables Data_____	99
4.5 CU Location Data_____	100
5 Creating StateMod Data Set Files_____	125
5.1 Control Data_____	126
5.2 Stream Gage Data_____	127
5.3 Delay Table Data_____	130
5.4 Diversion Data_____	133

## Table of Contents

---

5.5 Precipitation Data	161
5.6 Evaporation Data	161
5.7 Reservoir Data	162
5.8 Instream Flow Data	174
5.9 Well Data	182
5.10 Stream Estimate Data	198
5.11 River Network Data	205
5.12 Operational Data	210
5.13 San Juan Sediment Recovery Plan Data	210
5.14 Spatial Data	211
6 Troubleshooting	213
7 Quality Control	215
7.1 Quality Control for StateDMI Software	215
7.2 Using StateDMI and TSTool to Quality Control Data and Processes	225
Command Glossary	227
Command Reference: #	245
Command Reference: */	247

## Table of Contents

---

Command Reference: /*	249
Command Reference: AggregateWellRights ()	251
Command Reference: CalculateDiversionDemandTSMonthly()	257
Command Reference: CalculateDiversionDemandTSMonthlyAsMax()	261
Command Reference: CalculateDiversionStationEfficiencies()	263
Command Reference: CalculateStreamEstimateCoefficients()	265
Command Reference: CalculateWellDemandTSMonthly()	267
Command Reference: CalculateWellDemandTSMonthlyAsMax()	269
Command Reference: CalculateWellStationEfficiencies()	271
Command Reference: CheckBlaneyCriddle()	273
Command Reference: CheckClimateStations()	275
Command Reference: CheckCropCharacteristics()	277
Command Reference: CheckCropPatternTS()	279
Command Reference: CheckCULocations()	281
Command Reference: CheckDiversionDemandTSMonthly()	283
Command Reference: CheckDiversionHistoricalTSMonthly()	285
Command Reference: CheckDiversionRights()	287

## Table of Contents

---

Command Reference: CheckDiversionsStations()	289
Command Reference: CheckInstreamFlowDemandTSAverageMonthly()	291
Command Reference: CheckInstreamFlowRights()	293
Command Reference: CheckInstreamFlowStations()	295
Command Reference: CheckIrrigationPracticeTS()	297
Command Reference: CheckPenmanMonteith()	299
Command Reference: CheckReservoirRights()	301
Command Reference: CheckReservoirStations()	303
Command Reference: CheckRiverNetwork()	305
Command Reference: CheckStreamEstimateCoefficients()	307
Command Reference: CheckStreamEstimateStations()	309
Command Reference: CheckStreamGageStations()	311
Command Reference: CheckWellDemandTSMonthly()	313
Command Reference: CheckWellHistoricalPumpingTSMonthly()	315
Command Reference: CheckWellRights()	317
Command Reference: CheckWellStations()	319
Command Reference: CompareFiles()	321

## Table of Contents

---

Command Reference: CreateCropPatternTSForCULocations()	323
Command Reference: CreateIrrigationPracticeTSForCULocations()	325
Command Reference: CreateNetworkFromRiverNetwork()	327
Command Reference: CreateRegressionTestCommandFile()	329
Command Reference: CreateRiverNetworkFromNetwork()	333
Command Reference: Exit()	335
Command Reference: FillClimateStation()	337
Command Reference: FillClimateStationsFromHydroBase()	339
Command Reference: FillCropPatternTSConstant()	341
Command Reference: FillCropPatternTSInterpolate()	343
Command Reference: FillCropPatternTSRepeat()	345
Command Reference: FillCropPatternUsingWellRights()	347
Command Reference: FillCULocation()	351
Command Reference: FillCULocationClimateStationWeights ()	353
Command Reference: FillCULocationsFromHydroBase()	355
Command Reference: FillCULocationsFromList()	357
Command Reference: FillDiversionDemandTSMonthlyAverage()	359



## Table of Contents

---

Command Reference: FillDiversionDemandTSMonthlyConstant()	361
Command Reference: FillDiversionDemandTSMonthlyPattern()	363
Command Reference: FillDiversionHistoricalTSMonthlyAverage()	365
Command Reference: FillDiversionHistoricalTSMonthlyConstant()	369
Command Reference: FillDiversionHistoricalTSMonthlyPattern()	371
Command Reference: FillDiversionRight()	373
Command Reference: FillDiversionStation()	375
Command Reference: FillDiversionStationsFromHydroBase()	379
Command Reference: FillDiversionStationsFromNetwork()	381
Command Reference: FillInstreamFlowRight()	383
Command Reference: FillInstreamFlowStation()	385
Command Reference: FillInstreamFlowStationsFromHydroBase()	387
Command Reference: FillInstreamFlowStationsFromNetwork()	389
Command Reference: FillIrrigationPracticeTSAcreageUsingWellRights()	391
Command Reference: FillIrrigationPracticeTSInterpolate()	397
Command Reference: FillIrrigationPracticeTSRepeat()	403
Command Reference: FillNetworkFromHydroBase()	405

## Table of Contents

---

Command Reference: FillReservoirRight()	407
Command Reference: FillReservoirStation()	409
Command Reference: FillReservoirStationsFromNetwork()	413
Command Reference: FillReservoirStationsFromHydroBase()	415
Command Reference: FillRiverNetworkFromHydroBase()	417
Command Reference: FillRiverNetworkFromNetwork()	419
Command Reference: FillRiverNetworkNode()	421
Command Reference: FillStreamEstimateStation()	423
Command Reference: FillStreamEstimateStationsFromHydroBase()	425
Command Reference: FillStreamEstimateStationsFromNetwork()	427
Command Reference: FillStreamGageStation()	429
Command Reference: FillStreamGageStationsFromHydroBase()	431
Command Reference: FillStreamGageStationsFromNetwork()	433
Command Reference: FillWellDemandTSMonthlyAverage()	435
Command Reference: FillWellDemandTSMonthlyConstant()	437
Command Reference: FillWellDemandTSMonthlyPattern()	439
Command Reference: FillWellHistoricalPumpingTSMonthlyAverage()	441

## Table of Contents

---

Command Reference: FillWellHistoricalPumpingTSMonthlyConstant()	443
Command Reference: FillWellHistoricalPumpingTSMonthlyPattern()	445
Command Reference: FillWellRight()	447
Command Reference: FillWellStation()	449
Command Reference: FillWellStationsFromDiversionStations ()	453
Command Reference: FillWellStationsFromNetwork()	455
Command Reference: LimitDiversionDemandTSMonthlyToRights()	457
Command Reference: LimitDiversionHistoricalTSMonthlyToRights()	461
Command Reference: LimitWellDemandTSMonthlyToRights()	465
Command Reference: LimitWellHistoricalPumpingTSMonthlyToRights()	469
Command Reference: MergeListFileColumns()	473
Command Reference: MergeWellRights ()	475
Command Reference: OpenHydroBase()	481
Command Reference: PrintNetwork()	483
Command Reference: ReadBlaneyCriddleFromHydroBase()	485
Command Reference: ReadBlaneyCriddleFromStateCU()	487
Command Reference: ReadClimateStationsFromList()	489

## Table of Contents

---

Command Reference: ReadClimateStationsFromStateCU()	491
Command Reference: ReadCropCharacteristicsFromHydroBase()	493
Command Reference: ReadCropCharacteristicsFromStateCU()	495
Command Reference: ReadCropPatternTSFromHydroBase()	497
Command Reference: ReadCropPatternTSFromStateCU()	499
Command Reference: ReadCULocationsFromList()	501
Command Reference: ReadCULocationsFromStateCU()	503
Command Reference: ReadCULocationsFromStateMod()	505
Command Reference: ReadDelayTablesMonthlyFromStateMod()	507
Command Reference: ReadDiversionDemandTSMonthlyFromStateMod()	509
Command Reference: ReadDiversionHistoricalTSMonthlyFromHydro Base()	511
Diversion Comment “Not Used” Flag	511
Structure “Currently in Use” Flag	512
Command Reference: ReadDiversionHistoricalTSMonthlyFromStateMod()	519
Command Reference: ReadDiversionRightsFromHydroBase()	521
Command Reference: ReadDiversionRightsFromStateMod()	523
Command Reference: ReadDiversionStationsFromList()	525

## Table of Contents

---

Command Reference: ReadDiversionStationsFromNetwork()	527
Command Reference: ReadDiversionStationsFromStateMod()	529
Command Reference: ReadInstreamFlowDemandTSAverageMonthlyFromStateMod()	531
Command Reference: ReadInstreamFlowRightsFromHydroBase()	533
Command Reference: ReadInstreamFlowRightsFromStateMod()	535
Command Reference: ReadInstreamFlowStationsFromList()	537
Command Reference: ReadInstreamFlowStationsFromNetwork()	539
Command Reference: ReadInstreamFlowStationsFromStateMod()	541
Command Reference: ReadIrrigationPracticeTSFromHydroBase()	543
Command Reference: ReadIrrigationPracticeTSFromList()	549
Command Reference: ReadIrrigationPracticeTSFromStateCU()	551
Command Reference: ReadIrrigationWaterRequirementTSMonthlyFromStateCU()	553
Command Reference: ReadNetworkFromStateMod()	555
Command Reference: ReadPatternFile()	557
Command Reference: ReadPenmanMonteithFromHydroBase()	559
Command Reference: ReadPenmanMonteithFromStateCU()	561
Command Reference: ReadReservoirRightsFromHydroBase()	563

## Table of Contents

---

Command Reference: ReadReservoirRightsFromStateMod()	565
Command Reference: ReadReservoirStationsFromList()	567
Command Reference: ReadReservoirStationsFromNetwork()	569
Command Reference: ReadReservoirStationsFromStateMod()	571
Command Reference: ReadRiverNetworkFromStateMod()	573
Command Reference: ReadStreamEstimateCoefficientsFromStateMod()	575
Command Reference: ReadStreamEstimateStationsFromList()	577
Command Reference: ReadStreamEstimateStationsFromNetwork()	579
Command Reference: ReadStreamEstimateStationsFromStateMod()	581
Command Reference: ReadStreamGageStationsFromList()	583
Command Reference: ReadStreamGageStationsFromNetwork()	585
Command Reference: ReadStreamGageStationsFromStateMod()	587
Command Reference: ReadWellDemandTSMonthlyFromStateMod()	589
Command Reference: ReadWellHistoricalPumpingTSMonthlyFromStateCU()	591
Command Reference: ReadWellHistoricalPumpingTSMonthlyFromStateMod()	593
Command Reference: ReadWellRightsFromHydroBase()	595
Command Reference: ReadWellRightsFromStateMod()	603

## Table of Contents

---

Command Reference: ReadWellStationsFromList()	605
Command Reference: ReadWellStationsFromNetwork()	607
Command Reference: ReadWellStationsFromStateMod()	609
Command Reference: RemoveCropPatternTS()	611
Command Reference: RunCommands()	613
Command Reference: RunProgram()	617
Command Reference: RunPython()	621
Command Reference: SetBlaneyCriddle()	625
Command Reference: SetClimateStation()	627
Command Reference: SetCropCharacteristics()	629
Command Reference: SetCropPatternTS()	631
Command Reference: SetCropPatternTSFromList()	635
Command Reference: SetCULocation()	639
Command Reference: SetCULocationClimateStationWeights ()	641
Command Reference: setCULocationClimateStationWeightsFromHydroBase()	643
Command Reference: SetCULocationClimateStationWeightsFromList()	645
Command Reference: SetCULocationsFromList()	647

## Table of Contents

---

Command Reference: SetDebugLevel()	649
Command Reference: SetDiversionAggregate ()	651
Command Reference: SetDiversionAggregateFromList()	653
Command Reference: SetDiversionDemandTSMonthly()	657
Command Reference: SetDiversionDemandTSMonthlyConstant()	659
Command Reference: SetDiversionHistoricalTSMonthly()	661
Command Reference: SetDiversionHistoricalTSMonthlyConstant()	663
Command Reference: SetDiversionMultiStruct()	665
Command Reference: SetDiversionMultiStructFromList()	667
Command Reference: SetDiversionRight()	669
Command Reference: SetDiversionStation()	671
Command Reference: SetDiversionStationCapacitiesFromTS()	675
Command Reference: SetDiversionStationDelayTablesFromNetwork()	677
Command Reference: SetDiversionStationDelayTablesFromRTN()	679
Command Reference: SetDiversionStationsFromList()	681
Command Reference: SetDiversionSystem()	685
Command Reference: SetDiversionSystemFromList()	687



## Table of Contents

---

Command Reference: SetInstreamFlowDemandTSAverageMonthlyConstant()	691
Command Reference: SetInstreamFlowDemandTSAverageMonthlyFromRights()	693
Command Reference: SetInstreamFlowRight()	695
Command Reference: SetInstreamFlowStation()	697
Command Reference: SetIrrigationPracticeTS()	699
Command Reference: setIrrigationPracticeTSFromHydroBase()	703
Command Reference: SetIrrigationPracticeTSFromList()	711
Command Reference: setIrrigationPracticeTSMaxPumpingToRights()	717
Command Reference: SetIrrigationPracticeTSPumpingMaxUsingWell Rights()	721
Command Reference: SetIrrigationPracticeTSSprinklerAcreageFrom List()	727
Command Reference: SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()	731
Command Reference: SetOutputPeriod()	735
Command Reference: SetOutputYearType()	737
Command Reference: SetPenmanMonteith()	739
Command Reference: SetReservoirAggregate ()	741
Command Reference: SetReservoirAggregateFromList()	743
Command Reference: SetReservoirRight()	745

## Table of Contents

---

Command Reference: SetReservoirStation()	747
Command Reference: SetRiverNetworkNode()	751
Command Reference: SetStreamEstimateCoefficients()	753
Command Reference: SetStreamEstimateCoefficientsPFGage()	755
Command Reference: SetStreamEstimateStation()	757
Command Reference: SetStreamGageStation()	759
Command Reference: SetWarningLevel()	761
Command Reference: SetWellAggregate ()	763
Command Reference: SetWellAggregateFromList()	767
Command Reference: SetWellDemandTSMonthly()	771
Command Reference: SetWellDemandTSMonthlyConstant()	773
Command Reference: SetWellHistoricalPumpingTSMonthly()	775
Command Reference: SetWellHistoricalPumpingTSMonthlyConstant()	777
Command Reference: SetWellRight()	779
Command Reference: SetWellStation()	781
Command Reference: SetWellStationAreaToCropPatternTS ()	785
Command Reference: SetWellStationCapacitiesFromTS()	787

## Table of Contents

---

Command Reference: SetWellStationCapacityToWellRights ()	789
Command Reference: SetWellStationDelayTablesFromNetwork()	791
Command Reference: SetWellStationDelayTablesFromRTN()	793
Command Reference: SetWellStationDepletionTablesFromRTN()	795
Command Reference: SetWellStationsFromList()	797
Command Reference: SetWellSystem()	801
Command Reference: SetWellSystemFromList()	805
Command Reference: SetWorkingDir()	809
Command Reference: SortBlanneyCriddle()	811
Command Reference: SortClimateStations()	813
Command Reference: SortCropCharacteristics()	815
Command Reference: SortCropPatternTS()	817
Command Reference: SortCULocations()	819
Command Reference: SortDiversionDemandTSMonthly()	821
Command Reference: SortDiversionHistoricalTSMonthly()	823
Command Reference: SortDiversionRights()	825
Command Reference: SortDiversionStations()	827

## Table of Contents

---

Command Reference: SortInstreamFlowRights()	829
Command Reference: SortInstreamFlowStations()	831
Command Reference: SortIrrigationPracticeTS()	833
Command Reference: SortPenmanMonteith()	835
Command Reference: SortReservoirRights()	837
Command Reference: SortReservoirStations()	839
Command Reference: SortStreamEstimateStations()	841
Command Reference: SortStreamGageStations()	843
Command Reference: SortWellDemandTSMonthly()	845
Command Reference: SortWellHistoricalPumpingTSMonthly()	847
Command Reference: SortWellRights()	849
Command Reference: SortWellStations()	851
Command Reference: StartLog()	853
Command Reference: StartRegressionTestResultsReport()	855
Command Reference: TranslateBlaneyCriddle()	857
Command Reference: TranslateCropCharacteristics()	859
Command Reference: TranslateCropPatternTS()	861

## Table of Contents

---

Command Reference: TranslatePenmanMonteith()	863
Command Reference: WriteBlaneyCriddleToList()	865
Command Reference: WriteBlaneyCriddleToStateCU()	867
Command Reference: WriteCheckFile()	869
Command Reference: WriteClimateStationsToList()	871
Command Reference: WriteClimateStationsToStateCU()	873
Command Reference: WriteCropCharacteristicsToList()	875
Command Reference: WriteCropCharacteristicsToStateCU()	877
Command Reference: WriteCropPatternTSToDateValue()	879
Command Reference: WriteCropPatternTSToStateCU()	881
Command Reference: WriteCULocationsToList()	883
Command Reference: WriteCULocationsToStateCU()	885
Command Reference: WriteDelayTablesDailyToList()	887
Command Reference: WriteDelayTablesDailyToStateMod()	889
Command Reference: WriteDelayTablesMonthlyToList()	891
Command Reference: WriteDelayTablesMonthlyToStateMod()	893
Command Reference: WriteDiversionDemandTSMonthlyToStateMod()	895

## Table of Contents

---

Command Reference: WriteDiversionHistoricalTSMonthlyToStateMod()	897
Command Reference: WriteDiversionRightsToList()	899
Command Reference: WriteDiversionRightsToStateMod()	901
Command Reference: WriteDiversionStationsToList()	903
Command Reference: WriteDiversionStationsToStateMod()	905
Command Reference: WriteInstreamFlowDemandTSAverageMonthlyToStateMod()	907
Command Reference: WriteInstreamFlowRightsToList()	909
Command Reference: WriteInstreamFlowRightsToStateMod()	911
Command Reference: WriteInstreamFlowStationsToList()	913
Command Reference: WriteInstreamFlowStationsToStateMod()	915
Command Reference: WriteIrrigationPracticeTSToDateValue()	917
Command Reference: WriteIrrigationPracticeTSToStateCU()	919
Command Reference: WriteNetworkToStateMod()	921
Command Reference: WritePenmanMonteithToList()	923
Command Reference: WritePenmanMonteithToStateCU()	925
Command Reference: WriteReservoirRightsToList()	927
Command Reference: WriteReservoirRightsToStateMod()	929

## Table of Contents

---

Command Reference: WriteReservoirStationsToList()	931
Command Reference: WriteReservoirStationsToStateMod()	933
Command Reference: WriteRiverNetworkToList()	935
Command Reference: WriteRiverNetworkToStateMod()	937
Command Reference: WriteStreamEstimateCoefficientsToList()	939
Command Reference: WriteStreamEstimateCoefficientsToStateMod()	941
Command Reference: WriteStreamEstimateStationsToList()	943
Command Reference: WriteStreamEstimateStationsToStateMod()	945
Command Reference: WriteStreamGageStationsToList()	947
Command Reference: WriteStreamGageStationsToStateMod()	949
Command Reference: WriteWellDemandTSMonthlyToStateMod()	951
Command Reference: WriteWellHistoricalPumpingTSMonthlyToStateMod()	953
Command Reference: WriteWellRightsToList()	955
Command Reference: WriteWellRightsToStateMod()	957
Command Reference: WriteWellStationsToList()	959
Command Reference: WriteWellStationsToStateMod()	961
Appendix: StateDMI Installation and Configuration	963

# Table of Contents

---

1. Overview	963
2. Installing StateDMI as Part of CDSS	963
3. Installing StateDMI	964
4. Uninstalling StateDMI Software	973
Appendix: StateDMI Release Notes	975
StateDMI Version History	975
Known Limitations	978
Changes in Versions 3.11.00 – 3.11.01	978
Changes in Version 3.10.00	978
Changes in Versions 3.09.00 – 03.09.02	978
Changes in Versions 3.04.00 – 3.08.02	980
Changes in Versions 3.00.00 to 3.03.00	981
Changes in Version 2.18.00	983
Changes in Version 2.17.00	983
Changes in Version 2.16.00	983
Changes in Version 2.14.00	984
Changes in Version 2.02.00 – 2.13.00	984



## Table of Contents

---

Changes in Version 2.01.00	984
Changes in Version 2.00.00	985
Changes in Version 1.22.00	985
Changes in Version 1.21.00	985
Changes in Version 1.20.05	985
Changes in Version 1.20.04	985
Changes in Version 1.20.03	985
Changes in Version 1.20.02	985
Changes in Version 1.20.01	985
Changes in Version 1.20.00	986
Changes in Version 1.18.10	986
Changes in Version 1.18.09	986
Changes in Version 1.18.08	986
Changes in Version 1.18.07	986
Changes in Version 1.18.06	986
Changes in Version 1.18.05	986
Changes in Version 1.18.04	986

## Table of Contents

---

Changes in Version 1.18.03	987
Changes in Version 1.18.02	987
Changes in Version 1.18.01	987
Changes in Version 1.18.00	987
Changes in Version 1.17.21	987
Changes in Version 1.17.20	987
Changes in Version 1.17.19	987
Changes in Version 1.17.18	988
Changes in Version 1.17.17	988
Changes in Version 1.17.16	988
Changes in Version 1.17.15	988
Changes in Version 1.17.14	988
Changes in Version 1.17.13	989
Changes in Version 1.17.12	989
Changes in Version 1.17.11	989
Changes in Version 1.17.10	990
Changes in Version 1.17.09	990

## Table of Contents

---

Changes in Version 1.17.08	990
Changes in Version 1.17.07	990
Changes in Version 1.17.06	991
Changes in Version 1.17.05	991
Changes in Version 1.17.04	991
Changes in Version 1.17.03	991
Changes in Version 1.17.02	992
Changes in Version 1.17.01	992
Changes in Version 1.17.00	992
Changes in Version 1.16.03	992
Changes in Version 1.16.02	992
Changes in Version 1.16.01	993
Changes in Version 1.16.00	993
Changes in Version 1.15.02	993
Changes in Version 1.15.01	993
Changes in Version 1.15.00	993
99_Spine.pdf	995

---

## Blank Page

This page is intentionally blank.

---

# DISCLAIMER for CDSS Products

2002-02-16, Acrobat Distiller

CDSS products include data and software from State of Colorado sources and from external sources like the U. S. Geological Survey (USGS). The following disclaimer applies to CDSS products:

**CDSS products and associated access are under development at this time. Access is provided solely to test and demonstrate CDSS capabilities. In the future, this access may be restricted or offered for a fee. The State assumes no legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed herein. It is the user's responsibility to determine the fitness of the data for a particular purpose.**

This page is intentionally blank

---

# 1 Acknowledgements

Version 3.09.01, 2010-02-0

StateDMI has been developed by Riverside Technology, inc. (Riverside) with funding from the State of Colorado, Water Conservation Board as part of Colorado's Decision Support Systems (CDSS). StateDMI continues to be developed as part of the SPDSS (South Platte Decision Support System) and other projects.

Support for StateDMI can be contacted by emailing [cdss@state.co.us](mailto:cdss@state.co.us).

This page is intentionally blank.



---

## 2 Introduction

Version 03.10.00, 2010-05-11

StateDMI is a tool that can be used to process and format data for CDSS models, including the StateCU (consumptive use) and StateMod (surface water) models. The “DMI” corresponds to “Data Management Interface,” which is a general term for a tool that translates data from one form to another. TSTool is a DMI utility for processing time series. StateDGI and StatePP are other CDSS DMI utilities, which process GIS data and generate input for the MODFLOW groundwater model. StateDMI’s input data are read from the State of Colorado’s HydroBase database, spatial data files (e.g., ESRI shapefiles), text files, and existing StateMod and StateCU data files. Output is written to StateCU, StateMod, and text formats. StateDMI can be considered the middle application in the modeling process:

1. StateView and the CDSS web site are general HydroBase data-viewing tools, for initial data evaluation.
2. StateDMI processes model data from HydroBase and other sources into model files (see also TSTool, which performs a similar function for time series data).
3. The StateCU and StateMod models and graphical user interfaces are the final end-user applications for modeling.

StateDMI uses a workflow command language (similar to TSTool) to describe how data should be processed. The command language approach has a number of benefits:

1. It allows control of whether a data processing step occurs (or not).
2. It allows control of the order of data processing steps.
3. It allows complicated data processing sequences to be broken into manageable steps, which allows evaluation of different combinations and facilitates troubleshooting.
4. It allows data processing procedures to be saved and rerun at a later time. Consequently, complicated data processing steps can be remembered.
5. It allows data processing to be automated. For example, rather than interactively executing the same steps each time data need to be processed, an effort can be made once to determine data processing steps and record the steps in command files. The same steps can then be rerun later with little effort.
6. It allows comments to be inserted in the data processing procedures. For example, data that are read from HydroBase can be edited using commands and comments can be inserted with the commands to explain the reason for the edits. Consequently, data processes are self-documenting.
7. It allows commands to be updated and reused for other situations. For example, a sequence of commands that is appropriate for one geographic region may also be appropriate for another region. An existing command file can be read, modified slightly, and rerun for the new situation.
8. It facilitates extending software features. For example, a new model file format or database can be implemented by adding new commands within the existing framework.
9. It allows tests for command workflows to be automated, simplifying software and process testing.

In spite of these benefits, command workflows can be somewhat intimidating. To address this issue, the StateDMI interface provides a framework that provides interactive editors for commands and performs checks on input and results. Documentation is also available for all commands.

The following chapters are available in this documentation:

**Chapter 1 – Acknowledgements** – recognizes contributors to the development and maintenance of the StateDMI software.

**Chapter 2 - Introduction** (this chapter) provides background information about StateDMI and the CDSS modeling framework and procedures.

**Chapter 3 – Getting Started** provides an overview of the StateDMI interface features.

**Chapter 4 – Creating StateCU Data Set Files** provides guidelines and examples of how StateDMI can be used to create StateCU data set files.

**Chapter 5 – Creating StateMod Data Set Files** provides guidelines and examples of how StateDMI can be used to create StateMod data set files.

**Chapter 6 – Troubleshooting** provides troubleshooting information.

**Chapter 7 – Quality Control** provides information about how StateDMI software and modeling processes can be quality controlled.

The **Command Reference** provides a complete command reference with commands listed in alphabetical order. **Chapter 4** and **Chapter 5** summarize the use of commands for each product. The **Command Reference** is by far the longest part of the documentation. The **Command Glossary** at the start of the **Command Reference** provides a list of parameters that are used in commands, which promotes standardization of parameters.

The **Installation and Configuration Appendix** provides information about installing and configuring StateDMI.

The **Release Notes Appendix** summarizes important software changes for each StateDMI version.

See also the *doc\Training* folder under the software installation, which includes slideshows and example files for self-paced training.

## 2.1 How to Use this Documentation

The documentation is organized into chapters that provide overview material, with extensive reference material at the end of the documentation. It is recommended that the documentation be used as follows:

1. New users should review the **Introduction** and **Getting Started** chapters to understand in general how StateDMI operates.
2. When processing StateCU or StateMod files, review the introductory pages of the corresponding chapters (**Chapter 4** for StateCU and **Chapter 5** for StateMod) to gain an appreciation of the data files that will need to be processed.
3. To produce files for a specific data component (e.g., diversion stations), refer to the section in the model chapter corresponding to the data component. Review the example(s) that are provided and utilize similar steps when creating new commands files. The documentation provides examples taken from actual data sets and, although not universally applicable, provides a good starting point for new work. Refer to command files and documentation available with downloadable data sets for the most current examples of production work.

4. To fully understand how to use a command, whether in a new command file or an existing command file that is being updated, refer to the **Command Reference** section at the end of the documentation.

## 2.2 CDSS Modeling Overview

For CDSS, a major focus has been to develop an integrated data-centered system that can create basin-wide data sets for planning purposes. The end result is basin models with hundreds or thousands of model nodes, with associated water rights, time series data, etc. StateDMI breaks up the data processing into sessions that focus on specific model data components that have corresponding data files. A command file controls the creation of each model file. Although the overall modeling process is complicated, StateDMI is organized to help facilitate creating an entire data set and individual model files. See also the TSTool documentation – TSTool is used to process time series data in CDSS.

The primary purpose of the StateCU model is to estimate irrigation water requirement, although it does also estimate non-irrigation requirements. Several input files need to be prepared to run the StateCU model. The number of files depends on the complexity of the analysis. The StateCU documentation describes the StateCU model files in detail.

The StateMod model is used to simulate surface water use considering the Prior Appropriation Doctrine (first in time, first in right). Its primary purpose is to evaluate the water demand and supply in order to allocate water. Whereas StateCU data sets focus primarily on historical data, StateMod data sets can have several variations in order model various water allocation conditions and issues. The StateMod documentation describes the StateMod model files in detail.

For CDSS modeling, the StateCU and StateMod models have some interdependency. For example, to estimate acreage, water rights data (consistent with StateMod) can be used to turn parcels off if water rights did not exist. Similarly, StateMod depends on the demand data produced by StateCU. Typically, full StateCU data sets are prepared before StateMod data sets; however, as shown in the example above, there is a need to produce some StateMod files when creating a StateCU data set. StateDMI supports this by providing StateMod commands for products needed by StateCU. Once data sets for both models have been created for a basin, it becomes easier to share model files and update them over time.

StateMod data sets are typically created for historical conditions, calculated demands (using full supply demands), and baseline (the current system) cases. If historical or simulated diversions are available, they can be provided to StateCU to evaluate a water supply limited condition. See the next section for more information on various data sets types.

Given that StateCU and StateMod have numerous input files and a variety of run options, it can be difficult to understand and maintain data sets. StateDMI helps streamline data processing so that data flow is clearer.

When modeling, some efficiency can be gained by selecting key stream gages (those with a significant period of record) and determining for each structure type (diversion, reservoir, instream flows, and wells) the key and non-key structures. Key structures are modeled explicitly within StateCU and StateMod while non-key structures may be aggregated. The use of aggregation is discussed where appropriate in this documentation and is addressed in the model data set documentation.

## 2.3 Data Set Folder and File Conventions

The conventions used for StateCU and StateMod data set directories and files have changed over time, in particular as new modeling challenges have been faced (e.g., groundwater, augmentation plans). Older conventions are not discussed because CDSS data sets have generally been updated to current standards. If necessary, refer to the model data set documentation for older data sets.

CDSS model data sets are developed by the State and contractors and are provided on the CDSS web site. The data sets typically only contain input files in zip files and are named with an abbreviation of the basin and the year of release (e.g., *cm2005* for the Colorado basin data set released in 2005). Note that the ending year of the model data is often less than the year of the release. Output files may not be made available due to the size of the files; consequently, users will need to rerun the models to produce output and/or refer to the data set documentation. StateCU and StateMod data sets are typically provided separately and StateCU data sets are typically released earlier than StateMod data sets.

Folders under the main data set folder are described in the following table (adapted from “Recommended Data Structure”, Ray Bennett, September 19, 2005). These conventions may change – see model data set documentation and files for conventions used with specific data sets. Folders are listed alphabetically in the following table; however, the order of processing is indicated by StateDMI menus and is described in model data set documentation and command files. Guidelines for data sets are as follows:

- **Top-level Data Set Folder.** The top-level data set folder (e.g., *cm2005*) will include all data and results for the model data sets. An exception is GIS files, which may be located in a shared location like the *\cdss\gis* folder, allowing multiple data sets to share GIS files, which can be large. However, if possible, it is recommended that GIS files are included with a data set to allow for stand-alone data sets.
- **Relative Paths.** The “flat” organization of data set folders facilitates the use of relative paths. Model response files and command files should utilize relative paths when referring to folders (e.g., *..\Diversions\cm2005.dds*). This facilitates transport of data sets from one location/computer to another.
- **Final Model Folder.** The final model folder (e.g., *StateMod* for the StateMod model), will contain:
  1. input files produced by data processing,
  2. miscellaneous files that do not require processing (e.g., response and control files)
  3. output files from the model run
- **Folder Variations.** Folders in addition to those described in the following table may be used to simplify maintenance and use. For example, *Historic*, *Calculated*, and *Baseline* folders may be used under the *StateMod* folder to separate main model variations. Additional data folders for processing may be included if they clarify data management and processing.
- **Supporting Files.** Miscellaneous support files should be stored in folders with related data. For example, historical reservoir end of month time series files (in addition to data that will be queried from HydroBase) should be stored in the *Reservoirs* folder. If necessary, use a sub-folder to clarify data management.
- **Log Files.** The `StartLog( )` command can be used as the first command in a command file to record processing that is performed. The log file can facilitate troubleshooting and serve as a useful artifact if a data set needs to be reviewed at a later date.
- **Quality Control.** The complexity of modeling and the decisions that are made based on the results require that quality control measures are implemented. Data checks can be performed using the `Check*( )` commands. See also the **Quality Control** chapter of this documentation.

Performing quality control activities throughout modeling will help to minimize uncertainty about the validity of the model results.

- **Comments.** Hand-edited data files and command files should include comments of the top indicating the source and date for data. Comments should be included throughout command files to describe processing.

### CDSS Data Set Folder Conventions

Folder	Primary Application (1)	Description
C:\CDSS\data\Basin		Main folder where basin includes data set release date (e.g., cm2005).
		<b>Consumptive Use Application</b>
.\ClimateCU	CU	Climate stations; temperature, precipitation, and frost time series associated with StateCU.
.\Crops	CU	Crop characteristics and coefficients; crop pattern and irrigation practice time series.
.\DelayCU	CU	Delay tables and assignment for StateCU limited supply analysis.
.\DocCU	CU	Documentation associated with a consumptive use application.
.\LocationCU	CU	CU locations and support list files.
.\StateCU	CU	<b>StateCU model files (all input and output for a consumptive use application).</b>
		<b>Surface Water Application</b>
.\ClimateSW	SW	Precipitation and evaporation time series associated with StateMod.
.\DelaySW	SW	Delay tables (monthly and daily) associated with StateMod.
.\DocSW	SW	Documentation associated with the surface water application.
.\Diversion	CU, SW, GW	Diversion stations and rights, historical and demand time series (monthly and daily), surface water aggregate, system, and multi-structure lists.
.\Instream	SW	Instream flow stations and rights, demand time series (average monthly, monthly, and daily).
.\Network	SW	StateMod network, generalized XML network.
.\Reservoirs	CU, SW	Reservoir stations and rights, end of month content and target time series.
.\StateMod	SW	<b>StateMod model files (all input and output files for a surface water application).</b>
.\StreamSW	SW	Stream files associated with StateMod (stream stations, historical time series, stream estimate coefficients, etc.).
.\Wells	CU, SW	Well stations and rights, historical pumping and demand time series, aggregation and system lists.
		<b>Groundwater Application</b>
.\Agg	GW	Aggregate polygons for StatePP.
.\DocGW	GW	Documentation associated with a groundwater application.
.\Edge	GW	Boundary conditions.

Folder	Primary Application (1)	Description
.\MIPumping	GW	M&I pumping.
.\ModFate	GW	Fate of surface water returns.
.\Modflow	GW	MODFLOW files (all input and output files for a ground water application).
.\PptRecharge	GW	Precipitation recharge associated with MODFLOW.
.\Prop	GW	Aquifer properties (K, SS, Sy, L).
.\RimInflows	GW	Rim inflows.
.\StateDGI	GW	GIS processing.
.\StatePP	GW	MODFLOW preprocessor.
.\StreamGW	GW	Stream files associated with the MODFLOW stream package.
.\StreamInflow	GW	Stream inflow to the groundwater model.
.\Survey	GW	Stream survey data.
.\URF	GW	Unit response development.

(1) Primary Application: CU = consumptive use, SW = surface water allocation, GW = groundwater

Both StateCU and StateMod data sets include some files that are typically not automatically created. These files include the main response and control files and the StateMod operational right file. However, most other files can be created in an automated fashion. The processing of data files typically occurs in a sequential fashion. Although modelers may have different approaches, StateDMI menus and documentation are generally organized according to data component/product dependency. For example, if one file depends on concepts or data from another file, then the dependent file is listed after the independent file in menus and procedures. In this way, the creation of a file avoids “forward referencing” another file that has not yet been created. However, some circular dependencies do occur in data preparation and are discussed with examples.

Although StateDMI’s interface is organized based on a logical creation order of the StateCU and StateMod files, it does not strictly impose rules on the order of creating files. StateDMI does encourage the use of standard StateCU and StateMod file extensions, as described in each model’s documentation. It does so by displaying the standard extensions in file choosers, although in most cases the user can override with any file extension.

The above information describes the general folder structure for a data set. The guidelines for naming the main data set folders are described below. Standard names for basin data set directories have been adopted to promote consistency and simplify data review. This naming convention reflects the following aspects of a data set:

- basin name, typically as an abbreviation (e.g., “rg” for Rio Grande)
- scope or scenario for the data modeled (e.g., whether a fraction or 100% of the consumptive use is modeled)
- year that the data set was created (may not agree with the last year included in the model)

The naming convention has changed over time and therefore legacy data set names do not agree with current conventions. For example, early data sets modeled approximately 75% of the consumptive demand. The next iteration of data modeled 100% of the consumptive demand, using aggregate stations where necessary, and these data sets were designated with a “T”. Current conventions are to include all effects by default and not use any special indicator like “T”. Therefore, the current naming convention focuses on the year that the data set was prepared and it is assumed that the data set takes advantage of all modeling capabilities. Short names are used because of an 8.3 character file name length limitation in StateMod, although this limitation may be removed in the future.

The following table lists examples of standard data set names, based on currently available data sets:

**Standard Names for Baseline Data Sets**

<b>Basin</b>	<b>Data Set Name (1)</b>
Arkansas	No data sets have been produced (ar?)
Upper Colorado Main Stem	cm
Gunnison	gm
Rio Grande	rg
San Juan/San Miguel/Dolores	sj
South Platte	sp
White	wm
Yampa	ym

The data set name recommendations have evolved over time and should be evaluated for each data set. For example, to facilitate future updates (e.g., extending data sets by additional years of data), it may be useful to NOT include the year in individual file names, using the year only for the main directory. However, this practice may lead to confusion when comparing data files from different versions of data sets because the year will not be included in the name. Conventions for each CDSS modeling effort should be evaluated and discussed with State of Colorado project managers.

To generate a calibrated StateMod model includes developing three inter-related data sets (see **Section 2.5** below for more information):

1. historical (also referred to as historic)
2. calculated
3. baseline

#### Example StateMod File Base Names

Model Run (StateMod Response File)	Key Properties of Data Set
cm2005H.rsp	Historical data set with 100% consumptive use included. Demands are generally the historical diversions. Reservoir targets are generally the historical end of month contents. Because historical files are often shared with other data set variants, the H may be omitted.
cm2005C.rsp	Calculated data set with 100% consumptive use included. Demands are calculated to equal the estimated headgate requirement (e.g., maximum of StateCU irrigation water requirement divided by average monthly efficiency AND historical diversions). Reservoir targets are generally forecasted.
cm2005B.rsp	Baseline data set with 100% consumptive use included. Demands are the same as the calculated data set; however, municipal, industrial, and trans-basin demands are set to a present or future value and facilities constructed during the study period are estimated to be on-line for the entire simulation.

Many of the files used in the historic, calculated and baseline data are the same. It is common for all the data to be the same except for the diversion demands and reservoir targets files. Refer to model data set documentation for detailed information about variations in data sets.

## 2.4 Standard Procedures for Creating StateCU and StateMod Data Sets

The previous sections described standard conventions for organizing data sets, including naming directories and files within data sets. **Chapter 4 – Creating StateCU Data Set Files** and **Chapter 5 – Creating StateMod Data Set Files** describe how to create each of the files necessary for each model. The recommended standard procedure for creating model files for each data type is to follow the steps in these chapters, illustrated by working examples from actual data sets.

The steps described in **Chapters 4** and **5** provide general guidelines related to data analysis and formatting. The following sections provide additional information related to variations in StateMod data sets. These variations should be considered when determining the level of modeling to be performed for a basin.

## 2.5 Variations in StateMod Data Sets

**Chapters 4** and **5** discuss how to create all model files. However, some files (e.g., calculated demands) are used only in the calculated and baseline data sets. The following sections describe the differences between data sets.



### 2.5.1 Creating a Historical Data Set

A historical data set is used to calibrate the model and match historical conditions. Historical time series (e.g., diversions, well pumping) are used for demands. Differences between simulated results and the historical time series are minimized by adjusting return flow patterns, stream estimate proration factors, and other data. See the StateMod documentation for more information about historical data sets.

### 2.5.2 Creating a Calculated Data Set

A “calculated” data set is one that uses estimated demands, rather than simply using historical data (e.g., diversion time series and historical reservoir levels). To produce a calculated data set, revise the following files from those used in the historic data simulation:

- The calculated control file (*\*C.ctl*) is the same as the historical control file (*\*H.ctl*) except header cards are revised to indicate it is a calculated data set.
- The calculated diversion demand file (*\*C.ddm*) is similar to the historical diversion demand file (*\*H.ddm*) except agricultural demands equal the estimated diversion headgate requirement for full supply rather than historical diversions.
- The calculated well demand file (*\*C.wem*) is similar to the historical well demand file (*\*H.wem*) except agricultural demands equal the estimated well pumping requirement (full supply) rather than historical pumping.
- The calculated reservoir target file (*\*C.tar*) is similar to the historical reservoir target file (*\*H.tar*) except reservoir targets are typically set to forecasted values. For example, individual time series files stored in the supporting files directory may be combined into the complete file.

### 2.5.3 Creating a Baseline Data Set

A baseline data set represents current or future conditions, allowing an evaluation of the system for “what if?” scenarios. To create a baseline data set, revise the following files from those used in the calculated data simulation:

- The baseline control file (*\*B.ctl*) is the same as the calculated control file (*\*C.ctl*) except header cards are revised to indicate it is a baseline data set.
- The baseline diversion demand file (*\*B.ddm*) is similar to the calculated diversion demand file (*\*C.ddm*) except municipal, industrial and trans-basin demands are revised to equal the present or estimated future demand. In addition, any diversions that may have been constructed during the study period will be estimated to be on-line for the entire study period. Demands are typically implemented by creating replacement time series files that are combined into the final model file.
- The baseline well demand file (*\*B.wem*) is similar to the calculated well demand file (*\*C.wem*) except municipal, industrial and trans-basin demands are revised to equal the present or estimated future demand. In addition, any wells that may have been constructed during the study period will be estimated to be on-line for the entire study period. Demands are typically implemented by creating replacement time series files that are combined into the final model file.
- The baseline reservoir target file (*\*B.tar*) is similar to the calculated reservoir target file (*\*C.tar*) except any reservoirs that may have been constructed during the study period will be estimated to be on-line for the entire study period. These reservoir targets are typically implemented by creating replacement files by hand.
- The baseline reservoir station file (*\*B.res*) is similar to the calculated reservoir station file (*\*C.res*) except any reservoirs that have been constructed during the study period may have a different initial content value. These reservoir station files are typically implemented by using data resets in the initial content.

### 2.5.4 Creating a Data Set with Aggregated Structures

In CDSS projects, the approach to modeling 100% of a basin's consumptive use (CU) has been to explicitly model key structures that include approximately 75% of the basins CU and aggregate the remaining CU into aggregated stations. The model data sets are reviewed and enhanced over time to improve the model's representation of actual conditions. The aggregation process is typically implemented as follows (see data set documentation for details for each basin):

1. Aggregated irrigation structures are identified in GIS software (e.g., the CDSS Toolbox software) from an irrigated acreage coverage as those not explicitly modeled.
2. Aggregated irrigation groups are defined based on location and cumulative aggregated acreage. Often aggregated groups are selected to coincide with a streamflow gage.
3. Aggregated reservoirs are defined based on non-explicitly modeled reservoir water rights. Often aggregated groups are selected to coincide with a streamflow gage.
4. Aggregated M&I demands are defined based on non-explicitly modeled M&I demands based on regional population data and per capita use estimates. Often aggregated groups are selected to coincide with a streamflow gage.
5. Aggregated water right classes are defined based on class size and typical call dates in a basin. These call dates are typically identified from an evaluation of historical call records and basin interviews.
6. Aggregated irrigation, reservoir and M&I structures are added to the network file (\*.net).
7. Aggregated irrigation structures, reservoirs and M&I uses are often located on the main stem in order to include their CU without developing new hydrology data on small tributaries. StateDMI commands recognize aggregate stations and process data accordingly.

In addition to diversion aggregate nodes, “systems” and “MultiStruct” nodes may be utilized in modeling. See the StateMod diversion stations description for more information.

StateCU and StateMod model files do not include information to describe collections. Consequently, StateDMI relies on commands like `SetDiversionAggregateFromList()` to supply information to be used during processing. Neglecting to provide this information will impact the results (e.g., diversion time series will contain smaller values because the aggregation is not occurring).

### 2.5.5 Creating a StateMod Data Set with Daily Data

The steps necessary to create a daily historical data set from a monthly data set is described in detail in the **Frequently Asked Questions** section of the StateMod documentation.

### 2.5.6 Creating a StateMod Data Set with Wells

The steps necessary to create a data set with wells are described in detail in the **Frequently Asked Questions** section of the StateMod documentation.

## 2.6 Commands and Processing Sequence

The StateDMI interface allows a list of commands to be created, which when processed result in the creation of model data files and other output products. Several commands are often needed to create a single model file, as shown in the following example:

```
#
# StateDMI commands to create the Rio Grande Climate Stations File
#
# Step 1 - read climate stations
#
# The following reads from a list file...
ReadClimateStationsFromList(ListFile="climate.lst",IDCol="1")
#
# Step 2 - set data manually
#
SetClimateStation(ID="newid",Latitude=100,Elevation=1999,Region1="ADAMS",
    Name="my station",IfNotFound=Add)
#
# Step 3 - fill climate station information
#
FillClimateStationsFromHydroBase(ID="*")
#
# Step 4 - write the climate stations file
#
WriteClimateStationsToStateCU(OutputFile="rgTW.CLI")
#
# Step 5 - check data
#
CheckClimateStations(ID="*")
WriteCheckFile(OutputFile="cli.commands.StateDMI.check.html")
```

The general sequence of commands when creating a model file is:

1. Read data from an existing source (e.g., a list file, the HydroBase database, or a model file) using `Read*()` commands. Delimited list files typically contain an identifier column, and data are then often read from HydroBase. List files can be created from the model network, StateView, etc.
2. If appropriate, set additional data (e.g., add information that was not present after the first item) using `Set*()` commands. Existing or new data may be added.
3. If appropriate, fill data (e.g., fill all latitude values that have not been previously specified) using `Fill*()` commands. Missing data can be filled but new data objects are not created.
4. If appropriate, further process data with commands that perform calculations (e.g., limit filled diversion time series to water rights that were in effect at the time). Various data products require commands of varying complexity.
5. Write output to model files, using `Write*()` commands.
6. Perform checks to ensure that data are suitable for modeling using `Check*()` commands.

The menus that list commands to process a specific file are generally listed in the above order, to emphasize the order that commands should be used. In some cases, additional commands will be shown because of additional processing that is required. Although StateDMI lists menus in the general order that they would be used, commands should be used in the order that is appropriate to accomplish a task. In particular, there are no restrictions on setting or filling values after a calculation has occurred.

StateDMI commands are free-format, using the syntax:

```
CommandName(Param1=Value1,Param2="Value2",...)
```

The command name corresponds to the command menus and each command is documented in the **Command Reference** at the end of this manual. Parameters can be listed in any order, separated by commas. In many cases, parameters have default values and do not need to be specified. Parameter values that include white space or commas should be enclosed in double quotes. The StateDMI GUI command editor dialogs help edit all commands.

StateCU and StateMod files each typically correspond to lists of *objects*. For example, StateMod data sets include a list of diversion stations (corresponding to the *.dds* file). StateCU has a list of consumptive use locations (corresponding to the *.str* file). Relationships between data objects occur through shared data fields (e.g., station identifiers). For example, diversion historical time series use the diversion station identifier.

StateDMI maintains lists of these objects in memory and manipulates the objects as commands are processed. For example, a list of diversion stations can be read from a StateMod diversion station file (*dds*). Additional diversion stations may then be added to the list using “set” commands. Because it is possible that lists of objects may be created from multiple input sources, StateDMI usually allows lists of objects to be appended. For example, both StateMod diversion stations (*dds* file) and wells (*wes* file) may be considered as locations where irrigation water requirement should be estimated in StateCU. Such locations are collectively referred to as *CU Locations*. Sort commands are available for most data types to facilitate consistent output.

Because a model data set may contain many files, it is convenient to create the files in a logical order, separating the work of creating a data set by using multiple command files. The convention used in this documentation is to describe using one command file to create one model file. The model data set documentation describes the order and logic in creating each model file.

---

## 3 Getting Started

Version 3.09.01, 2010-02-11

This chapter provides an overview of the StateDMI graphical user interface (GUI). The StateDMI interface has the following main purposes:

1. Provide an organized list of menus to facilitate configuration of command lists (workflows) to create StateCU and StateMod data files.
2. Manage and run command workflows.
3. Provide general reusable tools to process StateCU and StateMod data.
4. Display and edit the model network used to define the connectivity of locations used in StateCU and StateMod data sets.
5. Display the results of command workflow processing.
6. Provide automation and quality control tools to streamline data set creation.

The remainder of this chapter provides an overview of the graphical user interface, generally in the order of the interface features and menus. Menu items are listed in alphabetical order or by functional order (i.e., in the order that model files should typically be created).

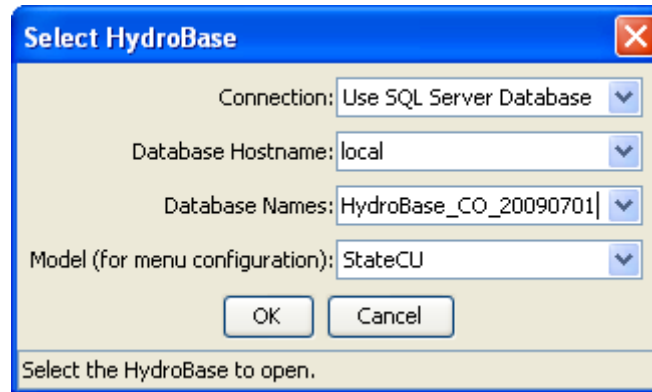
### 3.1 Starting StateDMI

StateDMI is a Java application and therefore is run using a Java Runtime Environment (JRE). The JRE is started using an executable file called *StateDMI.exe*, which is normally installed in `\CDSS\StateDMI-Version\bin`. This file can be run from a command shell, by selecting it from Windows Explorer, or more typically by selecting the **CDSS...StateDMI-Version** choice from the Start menu. The **Version** is the version of StateDMI to run. Multiple versions of StateDMI can be installed at the same time, to allow versions to remain with “frozen” data sets.

StateDMI provides features for StateCU and StateMod data sets, but not both at the same time. Start StateDMI for the appropriate model as shown in the next section, or use the **File...Switch to StateMod** or **File...Switch to StateCU** menus. If data files from one model are used by the other, StateDMI will provide appropriate features for both models for the specific data files.

### 3.2 Select HydroBase Dialog

Because one of the main input sources for StateDMI is the State of Colorado's HydroBase database, at startup you are requested to select a HydroBase database. A HydroBase database can also be selected from the **File...Open...HydroBase...** menu.



SelectHydroBase

**Select HydroBase Database Dialog**

HydroBase is available on DVD from the Division of Water Resources. Future software updates may allow StateDMI to access the database over the internet. In 2005 the Microsoft Access version of HydroBase was phased out in favor of MSDE (did not allow a single database to contain all State data), which has subsequently been replaced with SQL Server Express (allows the full Colorado database to be distributed).

If using an old Microsoft Access HydroBase database, you should have already configured a HydroBase ODBC DSN. You can select a local database and appropriate ODBC DSN, or, if you have access to a SQL Server HydroBase server, you can select **Use SQL Server Database**, in which case the database hostname is automatically determined from a predefined list – type in a new name if appropriate. You can also cancel the login, in which case HydroBase features will be disabled but you will be able to work with other data sources.

If a previous HydroBase connection has been made, then **Cancel** reverts to that connection.

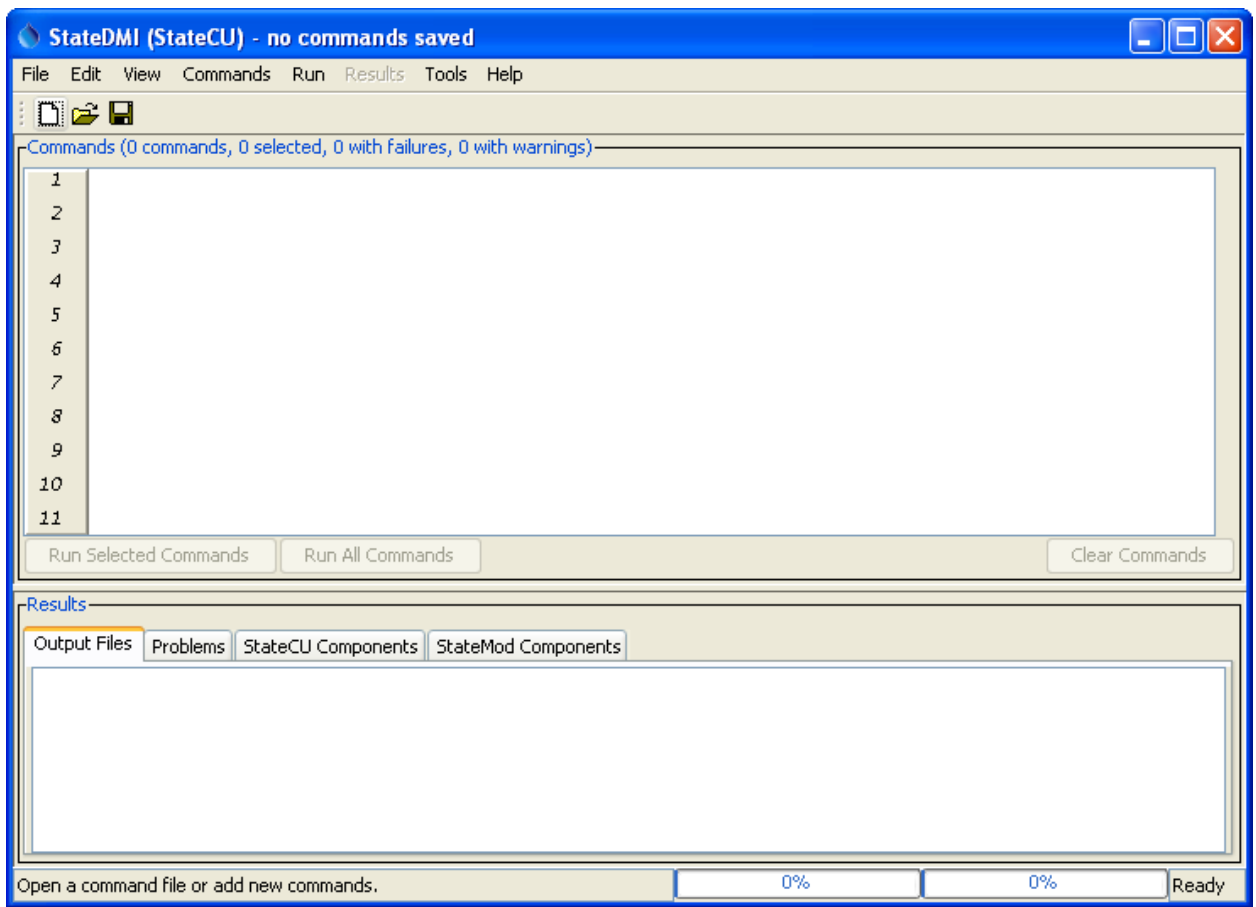
See also the **Installation and Configuration Appendix** for information about the CDSS configuration file, which can be used to set HydroBase selection defaults.

### 3.3 Main Interface

The StateDMI main interface is divided into the following areas:

- **Title Bar** (top)
- **Menu Bar** (below title)
- **Tool Bar** (below menu bar)
- **Commands list** (middle)
- **Results** (below commands)
- **Status Message areas** (bottom)
- **Map** (under development, as separate window)
- **Model Network** (as a separate window)

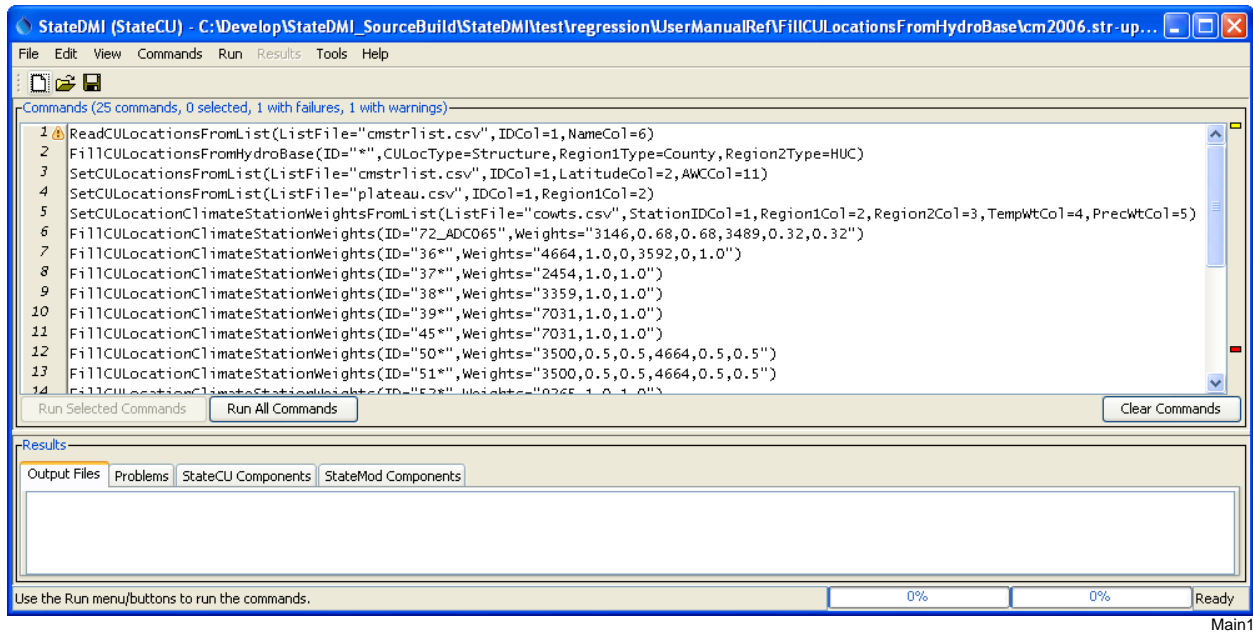
After starting the software, the main interface will be blank, as shown in the following figure:



StateDMI Interface After Startup

At this point, new commands can be added to the **Commands** list using the **Commands** menu, or an existing command file can be opened (**File...Open...Command File**). Commands, once visible, can be edited and run.

After opening a command file, the interface will appear similar to the following (the title bar displays the name of the command file and the **Commands** area title displays the status of the current commands list).



**StateDMI Interface after Loading a Commands File**

After loading the command file, StateDMI executes an initialization phase where each command is checked for basic errors. In the above example, a warning (yellow marker) and failure (red marker) are shown because the indicated commands reference files or folders that do not exist. These issues will need to be addressed before a complete run can occur. The following sections summarize the main GUI features.

### 3.3.1 Title Bar

The title bar at the top of the StateDMI interface indicates whether StateCU or StateMod commands are being edited, indicates the name of the command file, and whether changes to commands have been made.



**StateDMI Title Bar**



### 3.3.2 Menu Bar

The menu bar provides access to all the StateDMI features. StateDMI menus are generally consistent with standard Windows software and are summarized below. Each menu is described in detail starting with **Section 3.4**.

Menu	Description
<b>File</b>	Open and save data sets and commands files, select databases, select whether StateCU or StateMod files are being processed. See <b>Section 3.4</b> .
<b>Edit</b>	Cut/copy/paste and delete commands. See <b>Section 3.5</b> .
<b>View</b>	Toggle visual components (e.g., map, network) on/off. See <b>Section 3.6</b> .
<b>Commands</b>	Insert and edit commands to generate StateCU and StateMod model files. The sub-menus are specific to the model, although some general commands are present for each model's menus. See <b>Section 3.7</b> .
<b>Run</b>	Run commands to produce model output files and other data products. See <b>Section 3.8</b> . Users typically use the run buttons at the bottom of the <b>Commands</b> area.
<b>Results</b>	Display the results of processing commands. The menus are currently disabled. Instead, the overall results are typically written to model files and can be viewed as files or in tabular format by selecting a component at the bottom of the main window. See <b>Section 3.9</b> .
<b>Tools</b>	Display diagnostics. See <b>Section 3.10</b> .
<b>Help</b>	Display program version and support information. See <b>Section 3.11</b> .

### 3.3.3 Tool Bar

The tool bar provides graphical shortcuts to facilitate common actions:



**StateDMI Tool Bar**

Main\_Toolbar

The following tools are available in the toolbar:



Open a new blank command file. If changes to the current command file have occurred, the user is given the option of saving the current file. This is the same as **File...New...Command File**.



Open an existing command file. If changes to the current command file have occurred, the user is given the option of saving the current file. This is the same as **File...Open...Command File**.



Save changes to the current command file. This is the same as **File...Save...Command File**.

### 3.3.4 Command List

The **Commands** list occupies the middle part of the main interface and contains commands that can be processed to create StateCU and StateMod data files.



**Commands List**

The title for the **Commands** list indicates the number of commands, the number of commands selected from, and whether any commands have failures or warnings. Some interface features (e.g., editing, inserting new commands) operate on selected commands. The **Commands** list behaves according to Windows conventions:

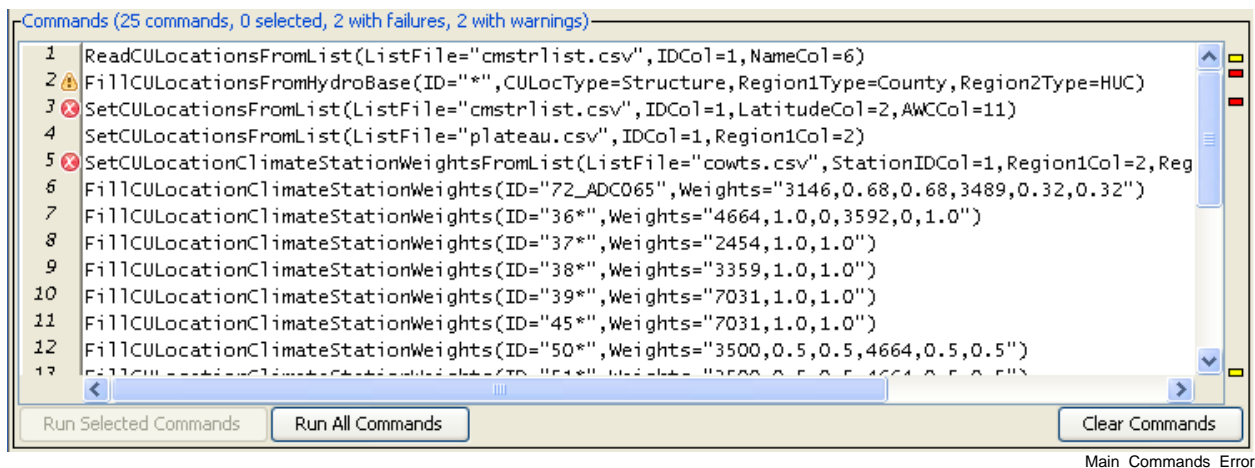
- Single-click to select one item.
- Ctrl-single-click to additionally select an item.
- Shift-single-click to select everything between the previous selection and the current selection.
- Double-click to display the command editor for the selected command.

Right-clicking over the **Commands** list displays a pop-up menu with useful command manipulation choices, some of which are further described in following sections (e.g., edit menu choices are discussed in **Section 3.5 - Edit Menu**). A summary of the pop-up menu choices is as follows:

Menu Choice	Description
<b>Show Command Status (Success/Warning/Failure)</b>	Displays a summary of problems encountered with the command, and recommendations for correcting the problems.
<b>Edit</b>	Edit the selected command using an edit dialog, which provides error checks and formats the commands. Double-clicking on a command will also display the command editor.
<b>Cut</b>	Cut the selected commands for pasting.
<b>Copy</b>	Copy the selected commands for pasting.
<b>Paste (After Selected)</b>	Paste commands that have been cut/copied, pasted after the selected row.
<b>Delete</b>	Delete the selected commands (currently the same as <b>Cut</b> ).
<b>Find commands(s) using substring...</b>	Find commands in the command list using a substring. This displays a dialog to enter the substring; press <b>Enter</b> and then the right-click in the found items list to go to or select found items.

Menu Choice	Description
<b>Find command using line number...</b>	Find a command using a line number. This is useful when correcting a command that generated an error during processing.
<b>Select All</b>	Select all the commands.
<b>Deselect All</b>	Deselect all the commands. This is useful when inserting commands at the end of the list.
<b>Convert Selected Commands to # Comments</b>	Convert selected commands to # comments.
<b>Convert Selected Commands from # Comments</b>	Convert # comments to commands.
<b>Run All Commands (create all output)</b>	Run all commands and create output (e.g., files). This is equivalent to using the <b>Run All Commands</b> in the <b>Commands</b> list area.
<b>Run All Commands (ignore output commands)</b>	Run all commands but skip any output commands. This is useful for testing data processing steps but final output is not yet needed.
<b>Run Selected Commands (create all output)</b>	Run selected commands and create output (e.g., files). This is equivalent to using the <b>Run Selected Commands</b> in the <b>Commands</b> list area.
<b>Run Selected Commands (ignore output commands)</b>	Run selected commands but skip any output commands. This is useful for testing data processing steps but final output is not yet needed.
<b>Cancel Command Processing</b>	If commands are currently being processed, this allows the processing to be cancelled. The current command being processed will finish before action is taken.

Commands are numbered to simplify editing. The command list also includes left and right gutters to display graphics that help with error handling. The following figure illustrates a command workflow with errors.



Command List Illustrating Error

The following error handling features are available:

- Clicking on the left gutter will hide and un-hide the gutter.
- The graphic in the left gutter indicates the severity of a problem (see below for full explanation).
- The colored box on the right indicates the severity of a problem and, when clicked on, positions the visible list of commands to display the command corresponding to the problem.

- Commands have three phases: 1) initialization, 2) discovery, 3) run. Initialization occurs when reading a command file or adding a new command. The discover phase is executed only for commands that generate information for other commands needed during editing, such as lists of identifiers (discovery is not often used in StateDMI but is used extensively in the TSTool software). The run phase is when commands are processed to generate results.
- Positioning the mouse over a graphic in the left or right gutter will show a popup message with the problem information. The popup is only visible for a few seconds so use the right-click popup menu **Show Command Status (Success/Warning/Failure)** for a dialog that does not automatically disappear. See also the **Results** area **Problems** tab.

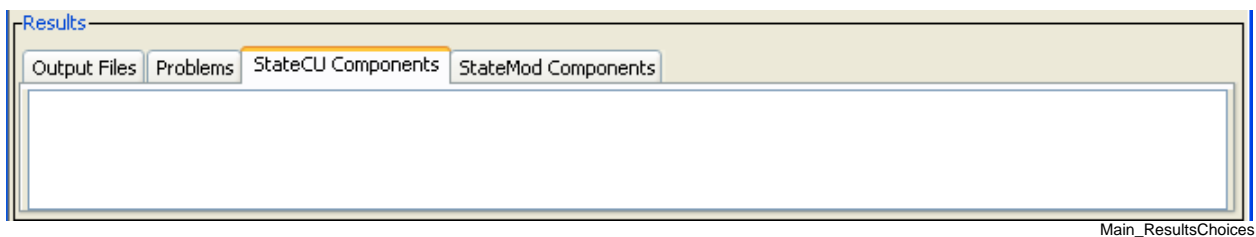
The meaning of the gutter symbols is described in the following table.

**Command List Error Handling Graphics**

Command List Left Gutter Graphic	Description
No graphic	Command is successful (a warning or failure has not been detected).
■	The status is unknown, typically because the status will not be known until a command runs.
🔒	The command has a problem that has been classified as non-fatal. For example, a command to fill data may be used but results in no data being filled. In general, commands with warnings need to be fixed unless work is preliminary.
❌	The command has failed, meaning that output is likely incomplete. A problem summary and recommendation to fix the problem are available in the status information. Commands with failures generally need to be fixed. Software support should be contacted if the fix is not evident.

### 3.3.5 Results

The **Results** area shows the results of processing commands.



**StateDMI Results Area**

Results can generally be displayed as output files and a component table, and a summary of problems is also provided. See below for more information.

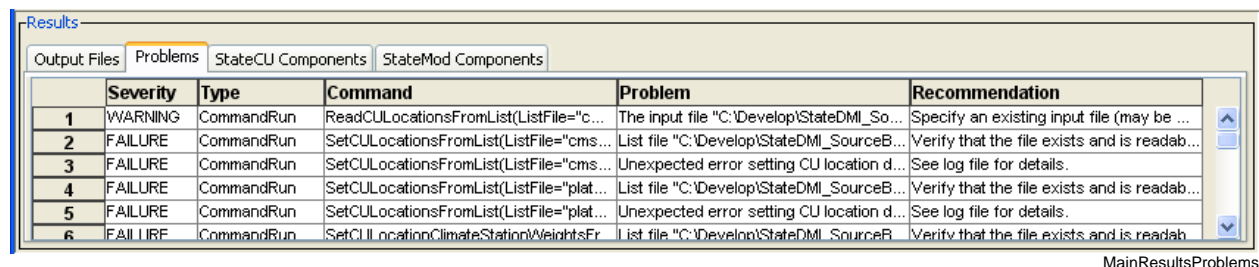
#### **Results – Output Files**

The main purpose of StateDMI is to prepare data set files with command workflows. The resulting model files may be long, complex, and difficult to review. However, an experienced user may simply want to scroll through the StateDMI output files and visually scan the data for completeness and accuracy. To facilitate this approach, the list of files created during commands processing is displayed and can be selected from the **Output Files** tab in the **Results** area. After selecting an output file, Notepad is used to

display the file. Additional files can be selected if desired, with each being displayed in a separate Notepad window. Currently, only files created during processing are listed (additional input files are not listed).

### Results – Problems

The Problems tab in the results area displays a summary of problems from all commands.



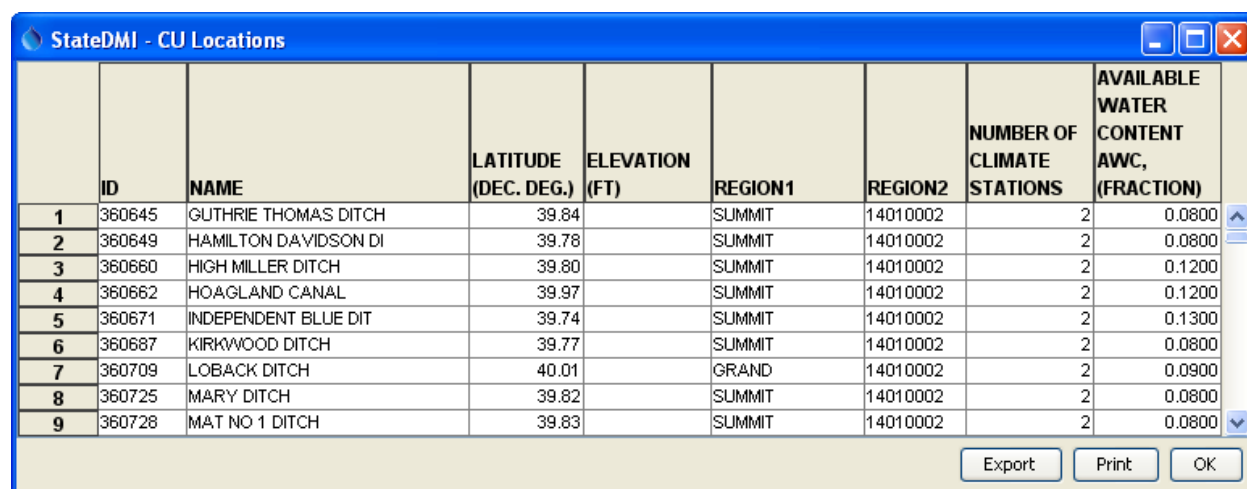
	Severity	Type	Command	Problem	Recommendation
1	WARNING	CommandRun	ReadCULocationsFromList(ListFile="c...	The input file "C:\Develop\StateDMI_So...	Specify an existing input file (may be ...
2	FAILURE	CommandRun	SetCULocationsFromList(ListFile="cms...	List file "C:\Develop\StateDMI_SourceB...	Verify that the file exists and is readab...
3	FAILURE	CommandRun	SetCULocationsFromList(ListFile="cms...	Unexpected error setting CU location d...	See log file for details.
4	FAILURE	CommandRun	SetCULocationsFromList(ListFile="plat...	List file "C:\Develop\StateDMI_SourceB...	Verify that the file exists and is readab...
5	FAILURE	CommandRun	SetCULocationsFromList(ListFile="plat...	Unexpected error setting CU location d...	See log file for details.
6	FAILURE	CommandRun	SetCULocationClimateStationWeightsFr	List file "C:\Develop\StateDMI_SourceB...	Verify that the file exists and is readab...

MainResultsProblems

This summary may be easier to use than individually displaying the status for each command with a problem. In the future, functionality may be enabled to click on a row and select the offending command in the command list. See also the `Check*( )` commands and the `WriteCheckFile( )` command, which will create a check file in CSV and HTML format.

### Results – StateCU and StateMod Components

Another option for viewing data is to display tabular records of the results, by data set component. To do so, select from the lists in the **StateCU Components** and **StateMod Components** tabs. For most command files, only one list will have choices but in some cases both lists may have choices. StateDMI internally manages data for each model. After making a selection, a simple tabular display will be shown, as in the following figure. The columns are typically shown in the order listed in the model documentation, in order to agree with model file output.



	ID	NAME	LATITUDE (DEC. DEG.)	ELEVATION (FT)	REGION1	REGION2	NUMBER OF CLIMATE STATIONS	AVAILABLE WATER CONTENT AWC, (FRACTION)
1	360645	GUTHRIE THOMAS DITCH	39.84		SUMMIT	14010002	2	0.0800
2	360649	HAMILTON DAVIDSON DI	39.78		SUMMIT	14010002	2	0.0800
3	360660	HIGH MILLER DITCH	39.80		SUMMIT	14010002	2	0.1200
4	360662	HOAGLAND CANAL	39.97		SUMMIT	14010002	2	0.1200
5	360671	INDEPENDENT BLUE DIT	39.74		SUMMIT	14010002	2	0.1300
6	360687	KIRKWOOD DITCH	39.77		SUMMIT	14010002	2	0.0800
7	360709	LOBACK DITCH	40.01		GRAND	14010002	2	0.0900
8	360725	MARY DITCH	39.82		SUMMIT	14010002	2	0.0800
9	360728	MAT NO 1 DITCH	39.83		SUMMIT	14010002	2	0.0800

Main\_Results\_CULocations

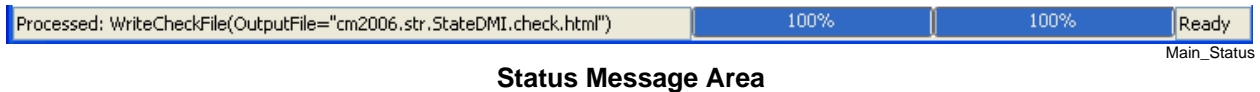
### Example Tabular Results Display

The data shown in the table can be viewed, copied to other applications, saved to a file, and printed. Printing may not provide the best representation of the data, especially if the table is very wide. Consequently, the file representation of results (in the **Output Files** tab) may be more appropriate for

printing. Columns can be sorted by right clicking on a column heading and picking the sort order. Note that some output files may correspond to multiple components. This occurs when a file has a complex structure that cannot easily be flattened into a single table.

### 3.3.6 Status Message Areas

The title bar and status message areas provide useful information about the current state of the interface and command list.



The status message area at the bottom of the StateDMI interface is split into three parts:

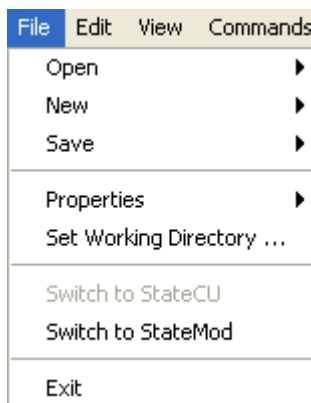
1. The left-most part is used to display general messages. For example, if commands are being run, this area indicates the command that is being processed.
2. The second part shows two progress bars that are updated when processing commands. The left progress bar shows the overall progress in the command file (percent of commands that have been processed). The right progress bar shows the progress within the command – this capability is only enabled in some commands that take longer to run.
3. The right-most status field provides a one-word status indicating when you should wait. The command processor is implemented as a separate thread in the program. Consequently, when commands are being processed, the application does not totally freeze while work occurs. Because it is possible to perform other tasks while the commands are being processed, an hourglass cursor is not used during processing and instead the progress meter and the one-word status should be used to know if commands are currently being processed.

### 3.3.7 Map (Under Development)

The map interface uses a general mapping component available in other CDSS software. See the **GeoView Mapping Tools Appendix** in the StateView and TSTool software for more information. The map interface is not currently integrated with StateDMI commands but is used to provide a reference of features that may be modeled with StateCU and StateMod. To display a map, use the **File...Show Map** menu described below. Then select a GeoView project file (.gvp). For example, select the same project file used by the StateMod GUI. The use of the map interface is being evaluated.

### 3.4 File Menu - Main Input and Output Control

The File menu provides standard input and output features as described below.

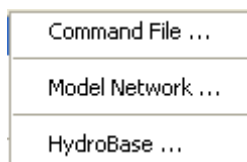


Menu\_File

#### File Menu

#### 3.4.1 File...Open Menu

The **File...Open** menu allows opening input sources.



Menu\_File\_Open

#### File...Open Menu

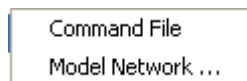
The **File...Open...Command File** menu allows an existing command file to be opened. A new command file can be started using the **File...New...Command File** menu item or corresponding tool on the tool bar.

The **File...Open...Model Network** allows a model network to be viewed and saved (see **Section 3.6.1** and later).

The **File...Open...HydroBase** menu opens a connection the HydroBase database (see **Section 3.2**).

#### 3.4.2 File...New

The **File...New** menu allows creation of a new command file and model network (see also **Section 3.6.1** and later for more information about the network).

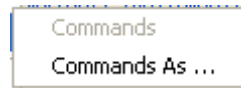


MenuFile\_New

#### File...New Menu

### 3.4.3 File...Save

The **File...Save** menu saves the contents of the **Commands** list.



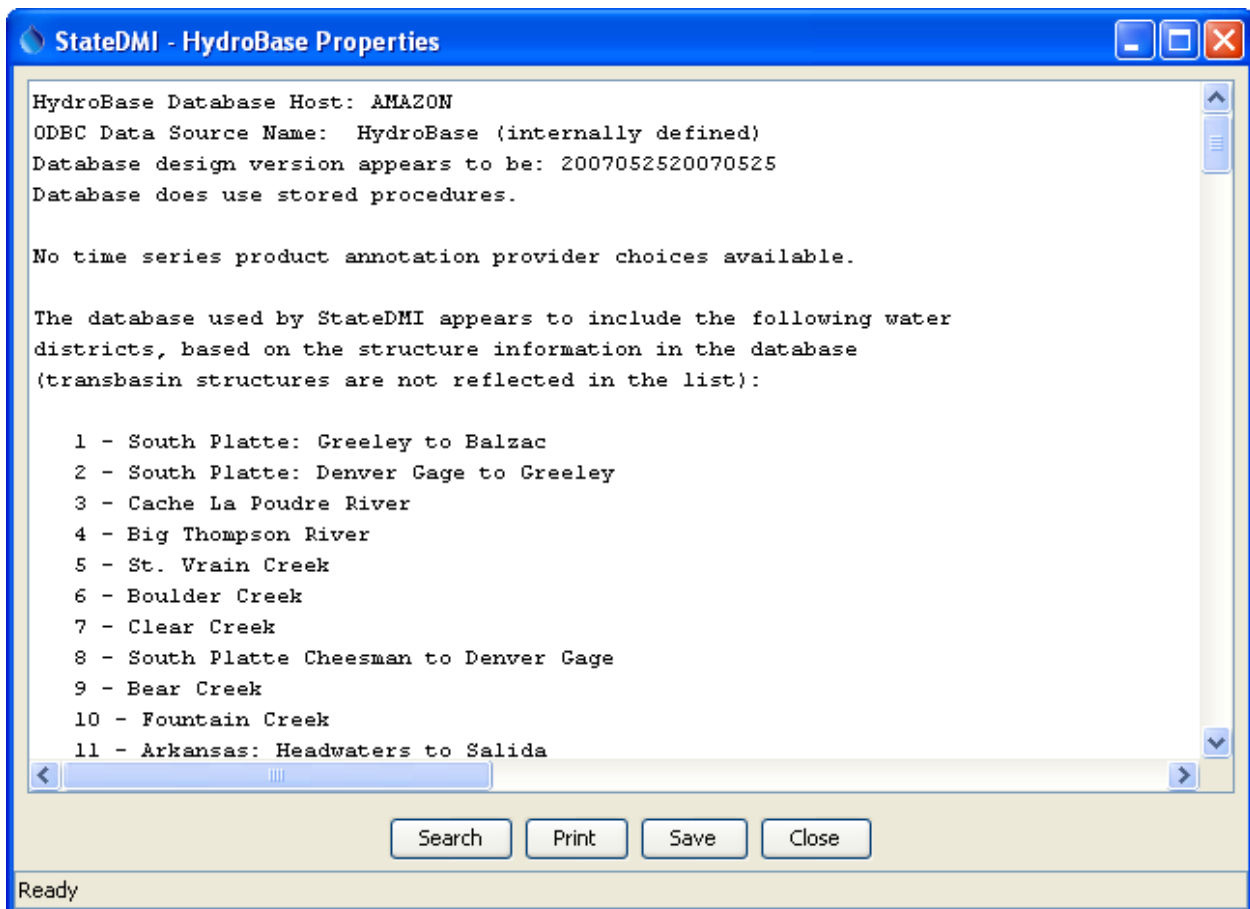
MenuFile\_Save

**File...Save Menu**

If a new command file has been started, you are prompted to specify a file name to save. The commands can also be saved to a new file.

### 3.4.4 File...Properties

The **File...Properties...HydroBase** menu displays the following dialog, which is available if a HydroBase connection has been made. The properties show HydroBase database information, including the database that is being used, database version, and the water districts that are in the database being queried. The water districts are determined from the structure table in HydroBase. The information that is shown is consistent with that shown by other State of Colorado tools and is useful for troubleshooting.



Menu\_File\_HydroBaseProperties

**HydroBase Properties**

### 3.4.5 File...Set Working Directory



The **File...Set Working Directory** menu item displays a file chooser dialog that allows you to select the working directory. The working directory, when set properly, can greatly simplify command files because relative file paths can be used for input and output. The working directory is normally set in one of the following ways, with the current setting being defined by the most recent item that has occurred:

1. The startup directory for the StateDMI program,
2. The directory where a command file was opened,
3. The directory where a command file was saved,
4. The directory specified by a `SetWorkingDir()` command,
5. The directory specified by **File...Set Working Directory**.

The menu item is provided to allow the working directory to be set before a command file has been saved (or opened) and it typically eliminates the need for `SetWorkingDir()` commands in command files.

### 3.4.6 File...Switch to StateCU and File...Switch to StateMod

The **File...Switch to StateCU** menu switches the StateDMI interface to operate on a StateCU data set. The **File...Switch to StateMod** menu switches the StateDMI interface to operate on a StateMod data set. These menus are necessary because StateDMI is designed to only show one model's features at a time. A noticeable change in behavior is that the **Commands** menu choices will reflect commands for the active model.

### 3.4.7 File...Exit

The **File...Exit** menu exits StateDMI. You will be prompted to confirm the exit. If you have edited the command list you will be prompted to save the commands.

### 3.5 Edit Menu – Editing Commands

The **Edit** menu can be used to edit the **Commands** list. Specific edit features are described below. Right clicking over the **Commands** list provides a popup menu with many choices described below.



Menu\_Edit

**Edit Menu**

#### 3.5.1 Cut/Copy/Paste/Delete

The **Edit...Cut** and **Edit...Copy** menu items are enabled if there are items in the **Commands** list. **Cut** deletes the selected item(s) from the **Commands** list and saves its information in memory. **Copy** just saves the information in memory. After **Cut** or **Copy** is executed, select an item in the **Commands** list and use **Paste** (see below). **Currently, these features do not allow interaction with other applications. However, Ctrl-C and Ctrl-V do work with many text entry fields in StateDMI.**

**Paste** is enabled if one or more items from the **Commands** list has been cut or copied. To paste the item, select an item in the **Commands** list and press **Edit...Paste Command(s) (After Selected)**. The new item will be added after the selected item(s). To insert at the front of the list, you must paste after the first item, and then cut and paste the first item to reverse the order.

The **Delete** choice currently works exactly like the **Cut** choice.

#### 3.5.2 Select All/Deselect All Commands

The **Edit...Select All Commands** and **Edit...Deselect All Commands** menu items are enabled if there are items in the **Commands** list. Use these menus to facilitate editing. Refer to the **Commands** list title to see how many commands are currently selected.

#### 3.5.3 Edit Command

The **Edit...Command...** menu can be used to edit an individual command. StateDMI will determine the command that is being edited and will display the editor dialog for that command, performing data checks. This feature is also accessible by right clicking on the **Commands** list and selecting the **Edit Command...** menu item.

### 3.5.4 Edit Command File

The ***Edit...Command File*** menu choice can be used to edit a commands file using **Notepad**. Currently, there is no way to change the editor. You must re-read the command file into StateDMI after using the editor in order for StateDMI to recognize the commands in the file.

### 3.5.5 Convert Selected Commands To/From Comments

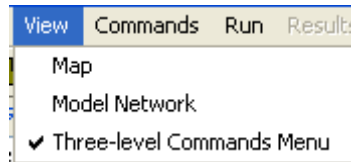
The ***Edit...Convert selected commands to # comments*** menu can be used to toggle selected commands in the **Commands** list to comments (lines that begin with #). This is useful when temporarily disabling commands, rather than deleting them.

The ***Edit...Convert selected commands from # comments*** menu can be used to toggle selected commands in the **Commands** list from comments back to active commands. This is useful when re-enabling commands that were temporarily disabled.

Multi-line `/* */` comment notation can be inserted using the ***Commands...General – Comments*** menu.

### 3.6 View Menu – Enable/Disable Display Features

The **View** menu enables and disables important StateDMI display features.



MenuView

View Menu

The **View...Map** menu item can be used to display a map. Currently this feature is under development. Use the file selector dialog to open a GeoView project file (*gvp*). See the **GeoView Mapping Tools Appendix** in StateView and TSTool documentation for more information about mapping tools. Map features are envisioned to be enhanced in future software releases. GeoView project files are available for StateView and StateMod data sets and can be selected to display in StateDMI.

The **View...Model Network** menu item displays a StateMod model network and allows edits to the network. A model network represents the rivers and model nodes in a diagram, where the geographical representation of rivers have been straightened and oriented to facilitate presentation of the network. StateDMI commands can then extract station lists from the network for processing into data files. See **Section 3.6.5** for more information.

The **Three-level Commands Menu** option allows switching the command menu format. This option is available primarily for developers and the default setting should not normally be changed.

#### 3.6.1 Updating an old Makenet Network to New Format

Previously, the Makenet program was used to process a model network and produce StateMod model files. The disadvantage of this approach was that the network file needed to be manually edited (there was no graphical user interface) and the format of the file sometimes resulted in errors. StateDMI understands how to read the old Makenet network file; consequently, the older files should be updated to the new convention to take advantage of new features and simplify maintenance. To update an old Makenet file to the new format:

1. If necessary, for an existing data set, rename the old *\*.net* file to another name (e.g., *XXX\_orig.net*). In many cases, StateDMI will be used to create an updated data set and therefore a rename is unnecessary because old and new files are in different directories.
2. Select the **File...Open...Model Network** menu and select the old Makenet *\*.net* file.
3. The StateDMI software will read the Makenet network file and display the network in a diagram window. During this process, a number of pieces of information are lost, including stream labels (now drawn with annotations), and page size (now setup as a layout). See the next section for information about editing the network. Also during this step, adjustments to the network are made. For example, blank nodes are removed since they are no longer needed. Confluence nodes are explicitly represented in the network because they are needed for visualization. Some node types like *Import* are converted to *Other* – all node types in the network now correspond to a station type in StateMod data sets. The coordinates that are used after this step are those defined in the Makenet file – it is envisioned that the coordinates could be scaled to physical coordinates like UTM to allow overlaying spatial data layers.

4. After the network is displayed, use the **Save XML Network File** tool in the network editor to save the representation to a new network file. The file name can adhere to the same naming convention as before (use \*.net).
5. To modify the network, use the features described in **Section 3.6.5**.

Below is an example from a new generalized network file. The file format is XML (eXtensible Markup Language), which is free format and allows new properties to be added as needed. Although the file can be modified with an editor, the graphical network editor should be used in most cases in order to enforce data conventions. The following example serves as the documentation for the network file format and the format is described in the comments at the top of the file.

```
<!--
#>
#>  StateMod XML Network File
#>
#>  File generated by...
#>  program:      StateDMI 3.08.00 (2009-06-10)
#>  user:         rrb
#>  date:         Mon Jun 15 17:37:05 MDT 2009
#>  host:         DWRDENRRBXPPC2
#>  directory:    D:\Cdss\Data\Sp2008L\StreamSW
#>  command line: StateDMI
#>
#> The StateMod XML network file is a generalized representation
#> of the StateMod network. It includes some of the information
#> in the StateMod river network file (*.rin) but also includes
#> spatial layout information needed to produce a diagram of the
#> network. The XML includes top-level properties for the
#> network, and data elements for each node in the network.
#> Each network node is represented as a single XML element
#> Node properties are stored as property = "value".
#>
#> Node connections are specified by either a
#>     <DownstreamNode ID = "Node ID"/>
#> or
#>     <UpstreamNode ID = "Node ID"/>
#> tag. There may be more than one upstream node, but at most
#> one downstream node.
#>
#> The XML network is typically created in one of three ways:
#>
#> 1) An old "makenet" (*.net) file is read and converted to
#> XML (e.g., in StateDMI). In this case, some internal
#> identifiers (e.g., for confluence nodes) will be defaulted in
#> order to have unique identifiers, and the coordinates will be
#> those from the Makenet file, in order to preserve the diagram
#> appearance from in the original Makenet file.
#>
#> 2) A StateMod river network file (*.rin) file is converted to
#> XML (e.g., by StateDMI). In this case, confluence nodes will
#> not be present and StateDMI can be used to set the coordinates
#> to actual physical coordinates (e.g., UTM). The coordinates
#> in the diagram will need to be repositioned to match a
#> straight-line representation, if such a representation is
#> desired.
#>
#> 3) A new network is created entirely within the StateDMI or
#> StateModGUI interface. In this case, the positioning of nodes
#> can occur as each node is defined in the network, or can occur
#> at the end.
```

```

#>
#> Once a generalized XML network is available, StateDMI can be
#> used to create StateMod station files. The node type and the
#> "IsNaturalFlow" property are used to determine lists of
#> stations for various files.
#>
#> The following properties are used in this file. Elements are
#> indicated in <angle brackets> with element properties listed
#> below each element.
#>
#> NOTE:
#>
#> If any of the following have an ampersand (&), greater than (>)
#> or less than (<) in them, these values MUST be escaped (see
#> below):
#>   - Page Layout ID
#>   - Node ID
#>   - Downstream Node ID
#>   - Upstream Node ID
#>   - Link Upstream Node ID
#>   - Link Downstream Node ID
#>   - Annotation Text
#>
#> The escape values are the following. These are automatically
#> inserted by the network-saving software, if the characters are
#> inserted when editing a network programmatically, but if the
#> network is edited by hand they must be inserted manually.
#>
#>   &   ->   &amp;
#>   >   ->   &gt;
#>   <   ->   &lt;
#>
#> <StateMod_Network>      Indicates the bounds of network
#>                          definition
#>
#>   XMin                  The minimum X coordinate used to
#>                          display the network, determined from
#>                          node coordinates.
#>
#>   YMin                  The minimum Y coordinate used to
#>                          display the network, determined from
#>                          node coordinates.
#>
#>   XMax                  The maximum X coordinate used to
#>                          display the network, determined from
#>                          node coordinates.
#>
#>   YMax                  The maximum Y coordinate used to
#>                          display the network, determined from
#>                          node coordinates.
#>
#>   LegendX               The X coordinate of the lower-left point
#>                          of the legend.
#>
#>   LegendY               The Y coordinate of the lower-left point
#>                          of the legend.
#>
#>   <PageLayout>          Indicates properties for a page layout,
#>                          resulting in a reasonable representation
#>                          of the network in hard copy. One or
#>                          more page layouts may be provided in
#>                          order to support printing on various
#>                          sizes of paper.

```

```

#>
#>      IsDefault      Indicates whether the page layout is the
#>                      one that should be loaded automatically
#>                      when the network is first displayed.
#>                      Only one PageLayout should have
#>                      this with a value of "True".
#>                      Recognized values are:
#>                      True
#>                      False
#>
#>      PaperSize      Indicates the paper size for a page
#>                      layout.  Recognized values are:
#>                      11x17      - 11x17 inches
#>                      A          - 8.5x11 inches
#>                      B          - 11x17 inches
#>                      C          - 17x22 inches
#>                      D          - 22x34 inches
#>                      E          - 34x44 inches
#>                      Executive - 7.5x10 inches
#>                      Legal      - 8.5x14 inches
#>                      Letter     - 8.5x11 inches
#>
#>      PageOrientation Indicates the orientation of the printed
#>                      page.  Recognized values are:
#>                      Landscape
#>                      Portrait
#>
#>      NodeLabelFontSize Indicates the size (in points) of the
#>                      font used for node labels.
#>
#>      NodeSize        Indicates the size (in points) of the
#>                      symbol used to represent a node.
#>
#>      <Node>          Data element for a node in the network.
#>
#>      ID              Identifier for the node, matching the
#>                      label on the diagram and the identifier
#>                      in the StateMod files.  It is assumed
#>                      that the station identifier and river
#>                      node identifier are the same.  The
#>                      identifier usually matches a State of
#>                      Colorado WDID, USGS gage ID, or other
#>                      standard identifier that can be queried.
#>                      Aggregate or "other" nodes use
#>                      identifiers as per modeling procedures.
#>
#>      Area            The natural flow contributing area.
#>
#>      AlternateX       The physical coordinates for the node,
#>      AlternateY       typically the UTM coordinate taken from
#>                      HydroBase or another data source.
#>
#>      Description      A description/name for the node,
#>                      typically taken from HydroBase or
#>                      another data source.
#>
#>      IsNaturalFlow    If "true", then the node is a location
#>                      where stream flows will be estimated
#>                      (and a station will be listed in the
#>                      StateMod stream estimate station file).
#>                      This property replaces the old IsBaseflow property.
#>
#>      IsImport         If "true", then the node is an import

```

```

#> node, indicating that water will be
#> introduced into the stream network at
#> the node. This is commonly used to
#> represent transbasin diversions. This
#> property is only used to indicate how
#> the node should be displayed in the
#> network diagram.
#>
#> LabelPosition The position of the node label, relative
#> to the node symbol. Recognized values
#> are:
#>     AboveCenter
#>     UpperRight
#>     Right
#>     LowerRight
#>     BelowCenter
#>     LowerLeft
#>     Left
#>     UpperLeft
#>     Center
#>
#> Precipitation The natural flow contributing area precipitation .
#>
#> Type The node type. This information is used
#> by software like StateDMI to extract
#> lists of nodes, for data processing.
#> Recognized values are:
#>     Confluence
#>     Diversion
#>     Diversion and Well
#>     End
#>     Instream Flow
#>     Other
#>     Reservoir
#>     Streamflow
#>     Well
#>     XConfluence
#>
#> X The coordinates used to display the node
#> Y in the diagram. These coordinates may
#> match the physical coordinates exactly,
#> may be interpolated from the coordinates
#> of neighboring nodes, or may be the
#> result of an edit.
#>
#> <DownstreamNode> Information about nodes downstream
#> from the current node. This information
#> is used to connect the nodes in the
#> network and is equivalent to the
#> StateMod river network file (*.rin)
#> "cstadn" data. Currently only one
#> downstream node is allowed.
#>
#> ID Identifier for the node downstream from
#> the current node.
#>
#> <UpstreamNode> Information about nodes upstream from the
#> current node. Repeat for all nodes
#> upstream of the current node.
#>
#> ID Identifier for the node upstream from
#> the current node.
#>
#>

```



```
#> <Annotation>      Data element for a network annotation.
#>
#>      FontName      The name of the font in which the
#>                    annotation is drawn.  Recognized values
#>                    are:
#>                        Arial
#>                        Courier
#>                        Helvetica
#>
#>      FontSize      The size of the font in which the
#>                    annotation is drawn.
#>
#>      FontStyle      The style of the font in which the
#>                    annotation is drawn.  Recognized values
#>                    are:
#>                        Plain
#>                        Italic
#>                        Bold
#>                        BoldItalic
#>
#>      Point          The point at which to draw the
#>                    annotation.  The value of "Point"
#>                    must be two numeric values separated by
#>                    a single comma.  E.g:
#>                        Point="77.44,9.0"
#>
#>      ShapeType      The type of shape of the annotation.
#>                    The only recognized value is:
#>                        Text
#>
#>      Text           The text to be drawn on the network.
#>
#>      TextPosition   The position the text will be drawn,
#>                    relative to the "Point" value.
#>                    Recognized values are:
#>                        AboveCenter
#>                        UpperRight
#>                        Right
#>                        LowerRight
#>                        BelowCenter
#>                        LowerLeft
#>                        Left
#>                        UpperLeft
#>                        Center
#>
#> <Link>             Data element for a network link.
#>
#>      FromNodeID     The ID of the node from which the link
#>                    is drawn.
#>
#>      LineStyle       The style in which the link line is
#>                    drawn.  The only recognized value is:
#>                        Dashed
#>
#>      ShapeType       The type of shape being drawn.  The only
#>                    recognized value is:
#>                        Link
#>
#>      ToNodeID        The ID of the node to which the link
#>                    is drawn.
#>
#>
#> EndHeader
```

```

-->
<StateMod_Network
  XMin = "-550.000000"
  YMin = "-425.000000"
  XMax = "1650.000000"
  YMax = "1275.000000"
  LegendX = "1274.000000"
  LegendY = "-63.000000">
  <PageLayout ID = "Page Layout #1"
    IsDefault = "true"
    PaperSize = "E"
    PageOrientation = "Landscape"
    NodeLabelFontSize = "12"
    NodeSize = "14"/>
  <PageLayout ID = "Page Layout #2"
    IsDefault = "False"
    PaperSize = "D"
    PageOrientation = "Landscape"
    NodeLabelFontSize = "10"
    NodeSize = "20"/>
  <PageLayout ID = "Page Layout #3"
    IsDefault = "False"
    PaperSize = "B"
    PageOrientation = "Portrait"
    NodeLabelFontSize = "10"
    NodeSize = "14"/>
  <PageLayout ID = "Page Layout #4"
    IsDefault = "False"
    PaperSize = "C"
    PageOrientation = "Landscape"
    NodeLabelFontSize = "10"
    NodeSize = "20"/>
  <PageLayout ID = "Page Layout #5"
    IsDefault = "False"
    PaperSize = "D"
    PageOrientation = "Landscape"
    NodeLabelFontSize = "10"
    NodeSize = "20"/>
  <Node ID = "64_AWP003"
    AlternateX = "-999.0"
    AlternateY = "-999.0"
    Description = ""
    IsBaseflow = "false"
    IsNaturalFlow = "false"
    IsImport = "false"
    LabelPosition = "AboveCenter"
    Type = "Well"
    X = "1523.958333"
    Y = "565.291667">
    <DownstreamNode ID = "06764000"/>
  </Node>

  <Node ID = "64_AWP002"
    AlternateX = "-999.0"
    AlternateY = "-999.0"
    Description = ""
    IsBaseflow = "false"
    IsNaturalFlow = "false"
    IsImport = "false"
    LabelPosition = "BelowCenter"
    Type = "Well"
    X = "1527.083333"
    Y = "380.916667">

```

```
<DownstreamNode ID = "64_AWP004"/>
</Node>

<Node ID = "64_AWP004"
  AlternateX = "-999.0"
  AlternateY = "-999.0"
  Description = ""
  IsBaseflow = "false"
  IsNaturalFlow = "false"
  IsImport = "false"
  LabelPosition = "AboveCenter"
  Type = "Well"
  X = "1526.562500"
  Y = "413.208333">
  <DownstreamNode ID = "64_AWP005"/>
  <UpstreamNode ID = "64_AWP002"/>
</Node>

... many nodes omitted ...

<Node ID = "END"
  AlternateX = "-999.0"
  AlternateY = "-999.0"
  Description = ""
  IsBaseflow = "false"
  IsNaturalFlow = "false"
  IsImport = "false"
  LabelPosition = "AboveCenter"
  Type = "End"
  X = "1600.000000"
  Y = "524.041667">
  <UpstreamNode ID = "64999999"/>
</Node>

<Annotation
  ShapeType="Text"
  Text="SPDSS Lower South Platte River Basin Water Resources Planning Model"
  Point="594.431373,1053.161477"
  TextPosition="Center"
  FontName="Helvetica"
  FontStyle="Plain"
  FontSize="72" />

... many annotations omitted ...

<Link
  ShapeType="Link"
  LineStyle="Dashed"
  FromNodeID="0100513"
  ToNodeID="Jackson_I" />

... many links omitted ...

</StateMod_Network>
```

### 3.6.2 Manually Creating a New StateMod Generalized Network

**If a new model data set is being prepared (or a network for an existing data set cannot be created in an automated way), a generalized network can be manually created using the following steps. In this process, each node must be added to the network.**

1. Before creating the network in StateDMI, it is useful to have an idea of the general layout of the network, where the streams in the data set follow the general geographical orientation. If the river basin runs north south, then a portrait page orientation should be used. If the basin runs east/west, then a landscape page orientation should be used.
2. After StateDMI has started, use the **File...New...Model Network** menu item. A network editor window will be shown, with only a page outline, legend, and end node.
3. The network editor requires that a page size and orientation be specified (see the **Section 3.6.5 Page Properties** information for details). To start, pick a page layout that will be used for editing and hardcopy review. If the network has many nodes, it may be necessary to pick a page size for a plotter (if a plotter is available). If the network has only a few nodes, then 8.5x11 or 11x17 page size may be sufficient.
4. Add a node by right clicking on the end node and selecting **Add Upstream Node**. Repeat as many times as necessary to complete the network. During this process, it may be necessary to change the printed node and font sizes appropriate for the hardcopy network. See also other network editor features described in **Section 3.6.5**, which may be used to position nodes.
5. Use the **Save As XML** tool at the top of the network editor to save the network file. This file can then be used by StateDMI commands and can be opened later with **File...Open...Model Network**.

The above procedure initializes a StateMod generalized network. Once created, the network editor features can be used to change the network.

### 3.6.3 Automatically Creating a New StateMod Generalized Network

Features to automate creation of a network have been tested during StateDMI development. However, various technical issues still remain and these features are not available for production work. The basic procedure is envisioned to use the following steps:

1. Determine a list of stations to be modeled (e.g., from HydroBase).
2. Query location coordinates (e.g., latitude/longitude or UTM) and upstream/downstream relationships (e.g., from HydroBase and/or NHD [National Hydrography Dataset]) and create a network based on physical coordinates (stored in the “alternate coordinates” in the network file).
3. As appropriate, utilize existing and new network editor features to adjust the network diagram to be more readable and suitable for modeling. For example, separate nodes that may be too close together to read labels. The network node coordinates will therefore reflect user edits, but the original “alternate” coordinates will still be available and could be used to draw a geographical representation of the network.

Use of NHD may facilitate referencing diversion, reservoir, stream gages, and other locations to rivers, thus allowing automated determination of upstream to downstream relationships. However, this information is currently available in CDSS only on a limited basis and therefore the automated creation of the network has not been possible.

### 3.6.4 Creating a New StateMod Generalized Network from an Existing StateMod River Network File

If an existing StateMod data set has no corresponding Makenet \*.net file, it is possible to create a generalized network file from the StateMod river network file (\*.rin). However, StateDMI features to do so have been tested only during development and technical issues remain. The basic procedure is envisioned to use the following steps:

1. Read the list of stations to be modeled from the StateMod \*.*rin* river network file. This supplies upstream/downstream relationships but does not provide coordinates for the network.
2. Query location coordinates (e.g., latitude/longitude or UTM) from HydroBase and create a network based on physical coordinates (stored in the “alternate coordinates” in the network file). Interpolate missing coordinates.
3. As appropriate, utilize existing and new network editor features to adjust the network diagram to be more readable and suitable for modeling. For example, separate nodes that may be too close together to read labels.

The above capabilities are available on a limited basis with current StateDMI commands. However, all technical issues have not been resolved and therefore these features are currently not utilized in production.

### 3.6.5 StateMod Model Network Editor

The **View...Model Network** menu item displays the editor window for the StateMod generalized model network (\*.*net*). This editor is available in StateDMI to make adjustments to the model network before file generation. It is also available in the StateMod GUI, for small adjustments to the data set. It is envisioned that the network editor will continue to be used with StateDMI for configuration model networks and be used to a lesser extent in the StateMod GUI for editing. It is also envisioned that additional tools will be added to the network editor to allow for more targeted use in StateDMI, and StateMod GUI, for example to display the return flow locations, and to display the stations that are referenced in an operating rule.

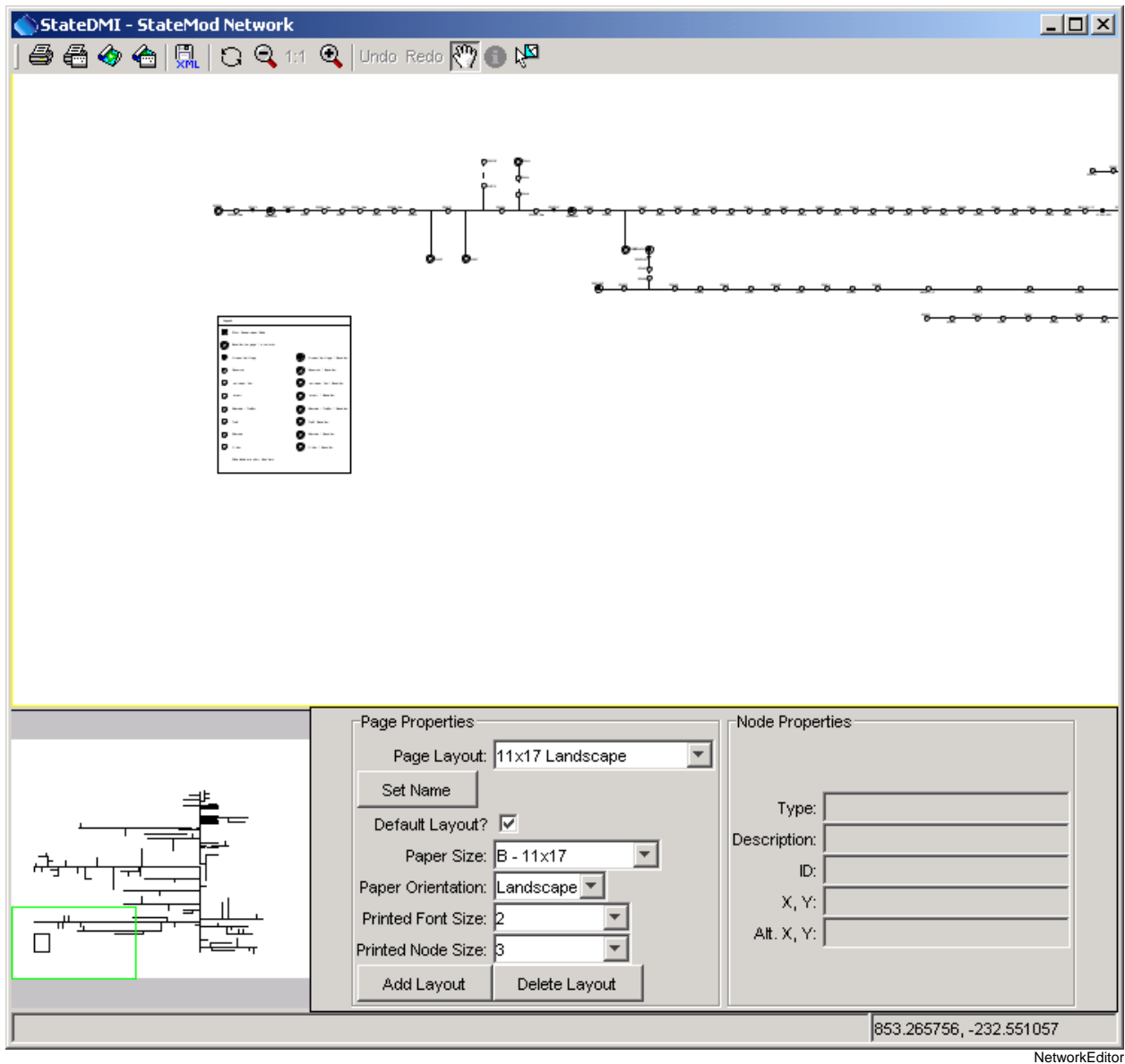
To use the network editor to adjust an existing model network, use the following basic steps:

1. Select the **View...Model Network** menu item and select the network (\*.*net*) file to be edited.
2. The network file will be read and displayed in the editor window (see below).
3. Use the editor to add, delete, or move stations (nodes), or change the information associated with the nodes. Also add annotations for stream names and main titles (see below for more information).
4. Use the **Save XML Network File** tool to resave the file. This file can then be used with `ReadXXXStationsFromNetwork( )` commands when processing data.
5. Repeat any of the steps, as necessary.

Several issues must currently be considered when using the network editor:

1. When the XML file is written, the header contains the last commandsfile that is run. If these commands contain strings that are prohibited in XML, errors may occur when the network file is read for processing. In particular, lines of dashes “-----” are prohibited, even in comments in the commands file. StateDMI will try to remove offending text when writing the XML file, but additional cases may arise. The workaround is to edit the XML file and remove the commands from the header.
2. It is envisioned that an integrated approach can be taken where the network that is opened can be used in modeling without supplying a file name for the network file. Therefore, some commands will process the in memory network if it has been opened. This approach is being evaluated. However, if a command reads a network file during processing and the network display is open, the network display is not currently automatically refreshed. Although it is envisioned that the visual representation of the network is fully integrated with commands processing, keeping the steps separate at this time is probably wise, to avoid confusion. In other words, edit the network interactively and save the result, and then specify the file name in commands.

The following figure shows the network editor after a network file has been read and displayed:
















**Network Editor**

The network editor consists of the following areas:

- Tools (top) – initiate actions (e.g., printing), switch mode, edit tools
- Main canvas (middle) – area where editing occurs
- Overview/reference window (lower left) – indicates the current view as a subset of the total network
- Page properties (lower middle) – the settings used for the network display, if printed
- Node properties (lower right) – the properties of the node that was last selected.

## Tools

The tools that are available include the following:

	Print the entire network using the selected layout (page size, orientation, etc.) This is useful for generation of final products.
	Print the visible network using letter-sized paper. This is useful for troubleshooting or reviewing specific parts of the network.
	Save the entire network to an image file.
	Save the visible network extent to an image file. This is useful for creating inserts for documents.
	Save the network to the XML file.
	Refresh the network (redraw).
	Zoom out by 50%, based on the current extent.
	Reset the scale to match the layout.
	Zoom in by 50%, based on the current extent.
	If a node position has changed, allow it to be undone (or redone).
	Pan the visible extent of the network – currently this is the default when clicking on other than a node.
	Information tool – currently unused. It is envisioned that this tool could be enabled to show model-related data from a data set.
	Select a feature – currently this is the default when clicking on a node.

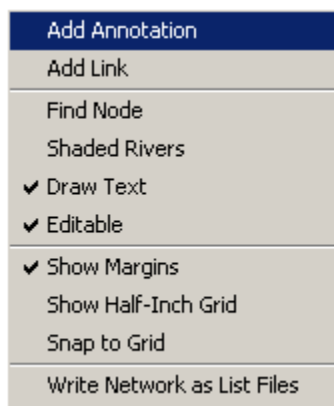
## Main Canvas

The main canvas displays the network for the current scale and location. Use the tools to scroll, pan, or zoom to a specific region.

To move an existing node, select it with the mouse and drag to the new location. Use the **Undo/Redo** tool if necessary to discard a change.

See sections below for information about adding/moving/deleting nodes and other actions.

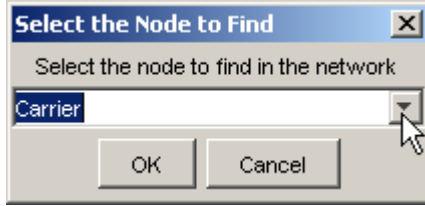
Right-clicking on the canvas (not near a node), displays the following menu:



NetworkEditor\_Popup

The actions for the menu items are described in the following table.

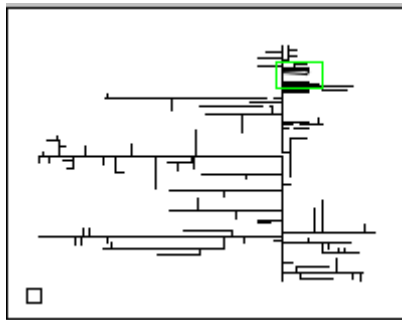
### Network Editor Popup Menu Items

Menu Item	Action
<b>Add Annotation</b>	Add an annotation at the point where the mouse was clicked. See <b>Section 3.6.5.2</b> below.
<b>Add Link</b>	Add a link between nodes. See <b>Section 3.6.5.3</b> below.
<b>Find Node</b>	<p>Display the following dialog, listing all nodes in the network.</p>  <p style="text-align: right; font-size: small;">NetworkEditor_Popup_FindNode</p> <p>After selecting a node and pressing <b>OK</b>, the network will scroll so that the selected node is in the center of the network window.</p>
<b>Shaded Rivers</b>	If selected, shade the rivers based on stream order. This is useful to emphasize upstream to downstream progression.
<b>Draw Text</b>	If selected, draw text labels on the network. Text can be turned off if only the lines need to be printed.
<b>Editable</b>	If selected, the network is editable. If it is important to protect a network from editing, the network can be made non-editable. Editing actions will then be prohibited in the session.
<b>Show Margins</b>	If selected, the page margins are shown, representing an approximate boundary within which drawing should be limited. It is recommended that network features not extend into the margins.
<b>Show Half-Inch Grid</b>	If selected, a grid of lines will be drawn at half-inch intervals. This is useful for layout purposes.
<b>Snap to Grid</b>	If selected, nodes will be restricted to being positioned on grid lines.
<b>Write Network as List Files</b>	<p>Prompt for a base file name and then write delimited list files for each station type, to be used as lists of stations with commands files. Each file is listed in order of upstream to downstream. This recognizes that it can be more generic to use list files with StateDMI processing, rather than reading from the network itself. This approach is being evaluated as list files are used. Issues to be resolved include:</p> <ol style="list-style-type: none"> <li>1. DIV and D&amp;W nodes both exist in the network and are written as separate lists. Therefore two commands may be needed when processing the lists.</li> <li>2. Stream gages (FLO nodes) are written as one list and baseflow stations (FLO and other stations where baseflow is True) are written as separate lists. Users must decide which list to use.</li> </ol>



### **Overview/Reference Window**

The overview window indicates the current extent of the network in the main canvas.

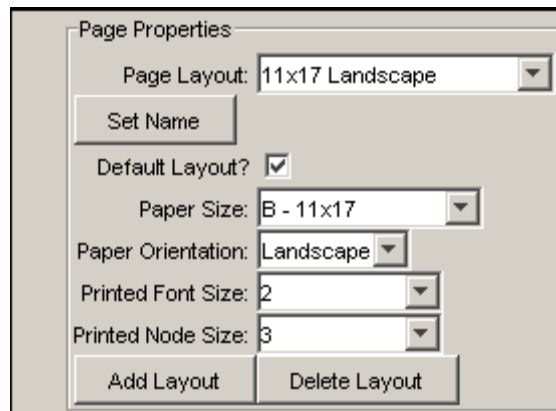


NetworkEditor\_Overview

Click anywhere in the overview window to center the main canvas view on that point. Or, drag the overview window extent box to a new location to reposition the network in the main canvas.

### **Page Properties**

The page properties can be set for multiple layouts using the **Page Properties** settings.



NetworkEditor\_PageProperties

Because one of the primary products related to the network is a printed network diagram, the network is essentially configured as a document. Therefore, the graphics and text on the diagram are scaled (unlike some map and graph displays where the text point size is constant even when the data scale changes).

Modelers responsible for data sets should define one or more layouts for the network to allow printing on common page sizes. Often, there is so much detail on the network that a hard copy can only be printed on large paper sizes. However, more unreadable versions may be appropriate for review. Once layouts are defined, only minor changes should be required. It is recommended that the **Page Layout** name include the page size and orientation.

Network editing should typically occur using the page layout that will be used in production printouts. Differences in the relative dimensions of page sizes can cause some scaling in output when switching between layouts.

### Node Properties

The node properties area in the network editor shows the node properties for the most recently selected node.

The Node Properties dialog box displays the following information for the selected node:

Type:	Diversion
Description:	PETERSON D 1
ID:	250628
X, Y:	1652.714286, 1272.211803
Alt. X, Y:	121758.500000, 4216182.000000

NetworkEditor\_NodeProperties

This is useful when scanning network node information. See the next section for information about changing node properties.

#### 3.6.5.1 Adding/Deleting/Changing a Node

To add a node, select a node, right click, and press **Add Upstream Node**. The following dialog is then used to enter information about the new node (see below for information about changing node properties).

The Add Node dialog box contains the following fields and options:

- Existing Nodes:**
  - Downstream Node: 201828
  - Upstream Node: CONFL\_12 (dropdown menu)
- New Node Data:**
  - Node ID: (empty text field)
  - Node Type: FLOW - Streamflow (dropdown menu)
  - Is Baseflow: ☐
  - Is Import: ☐

Buttons: OK, Cancel

NetworkEditor\_Add

**Add Node Dialog**

To delete a node, select a node, right-click, and press **Delete Node**. Currently you are not given the chance to cancel and the **Undo/Redo** tool does not apply.

A node is moved by selecting the node on the network and dragging to a new location. To move multiple nodes draw a box around nodes and then move the group. Node properties for an existing node are edited by selecting a node in the network, right clicking, and pressing the **Properties** menu item, which will display a dialog similar to the following:

The dialog box is titled "Node Properties - 200922TM". It contains the following fields and controls:

- ID: 200922TM
- Type: FLOW - Streamflow (dropdown menu)
- Description: WEMINUCHE PASS D
- X: 21.332379
- Y: 793.361957
- Is Baseflow: ☒
- Is Import: ☐
- Area: 0.010000
- Precipitation: 0.010000
- Label Position: Left (dropdown menu)
- Downstream node: 200922 (model node)  
200922 (diagram node)
- Buttons: Apply, OK, Cancel

NetworkEditor\_Popup\_NodeProperties

**Node Properties Dialog**

The node types correspond either to StateMod station types or to node types needed for visualization (e.g., confluences), which are not transferred to StateMod files. Although Makenet allowed **Import** and **Baseflow** node types, these types are no longer supported. Instead, node types correspond to StateMod station types, with the **Other** node type used where needed. The **Is Baseflow** check indicates that **Area** and **Precipitation** information are available for the node – these data are used when processing stream estimate stations.

### 3.6.5.2 Adding/Deleting/Changing Annotations

Annotations are text labels that can be drawn on the network. They are typically used for title, author, revision date, stream names, etc., using font sizes appropriate for the information.

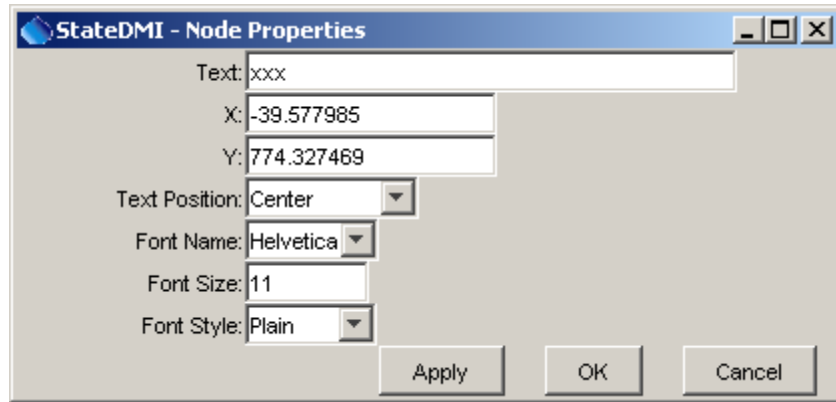
To add an annotation, right-click at a point of interest (not near a node) and select the **Add Annotation** menu item, which will display the following dialog:

The dialog box is titled "Enter the annotation text". It contains the following fields and controls:

- Enter the annotation text:
- Some text
- Buttons: OK, Cancel

NetworkEditor\_Popup\_AddAnnotation

Pressing **OK** displays the annotation text centered at the point where the mouse was clicked. Once an annotation is added, it can be moved and its properties can be set by right clicking on the annotation anchor point and pressing **Properties**:



NetworkEditor\_Popup\_AnnotationProperties

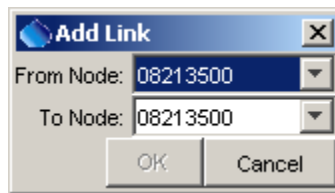
An annotation can be moved by selecting the annotation and dragging it to the new location.

An annotation can be deleted by right clicking on the annotation and pressing the **Delete Annotation** menu item.

### 3.6.5.3 Adding/Deleting Links

Links are dashed lines between nodes, typically used to represent an operational relationship between nodes (e.g., to represent carrier ditches). Annotations can be placed next to links to describe the link.

To add a link, right-click on the network (not near a node) and use the **Add Link** menu item. The following dialog will be shown:





NetworkEditor\_Popup\_AddLink

After selecting nodes and pressing **OK**, the link will be drawn between the nodes as a straight dashed line.

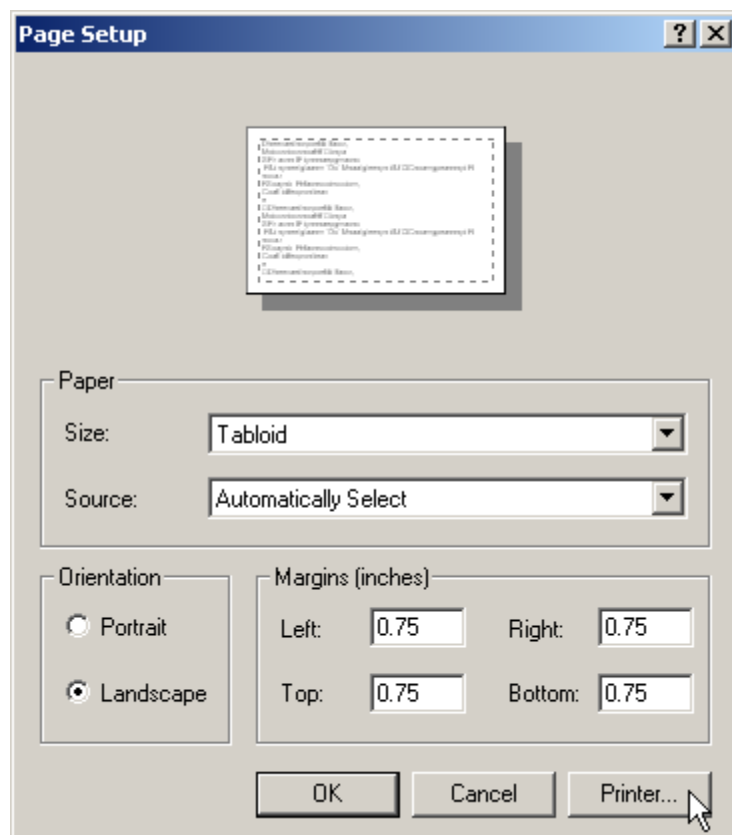
To delete the link, select one of the nodes involved in the link, right-click and select **Delete Link**. If the node is involved in more than one link, a list of links will be shown.

### 3.6.5.4 Printing the Network

To print the entire network, use the  tool and follow the procedure described below. To save the visible network as an image, use the  tool and follow the procedure described below. Note that when printing, curved graphics are drawn using a technique called “anti-aliasing,” where curves are created by using shades of gray. This may result in graphics that are difficult to read for some page sizes.

When the print tools are used, several dialogs are shown, as required by the Java and Microsoft environments. Although options are available in various dialogs, the following approach is recommended (improvements are being evaluated):

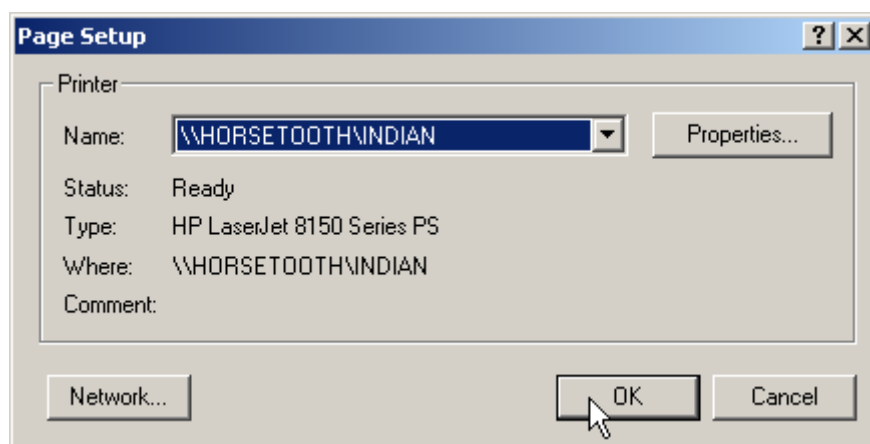
1. After selecting one of the tools mentioned above, a Java **Page Setup** dialog will be shown (this should be the same regardless of Windows version):



NetworkEditor\_Print1

Select the printer of interest by using the **Printer...** button, as discussed in the next item.

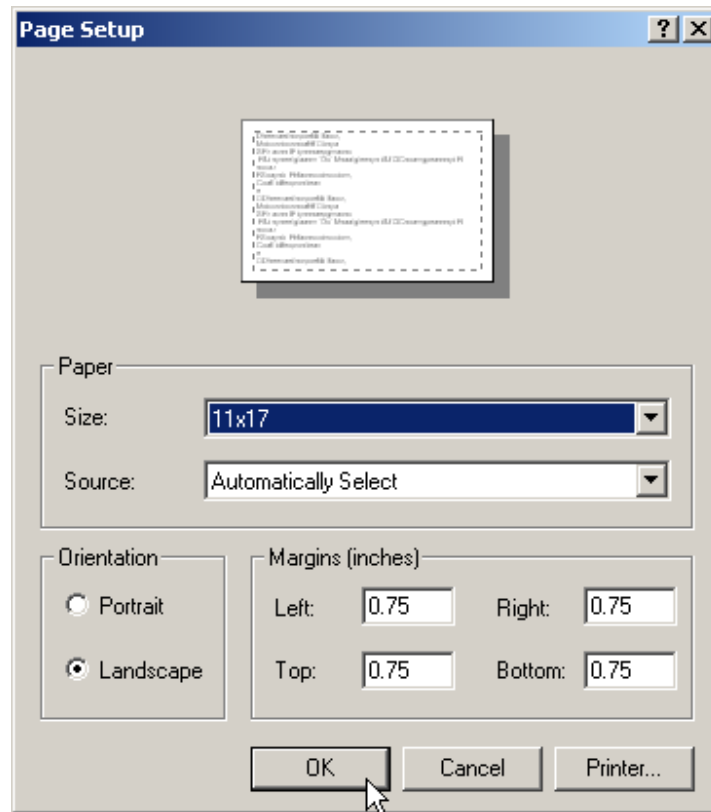
2. A Windows **Page Setup** dialog will be shown:



NetworkEditor\_Print2

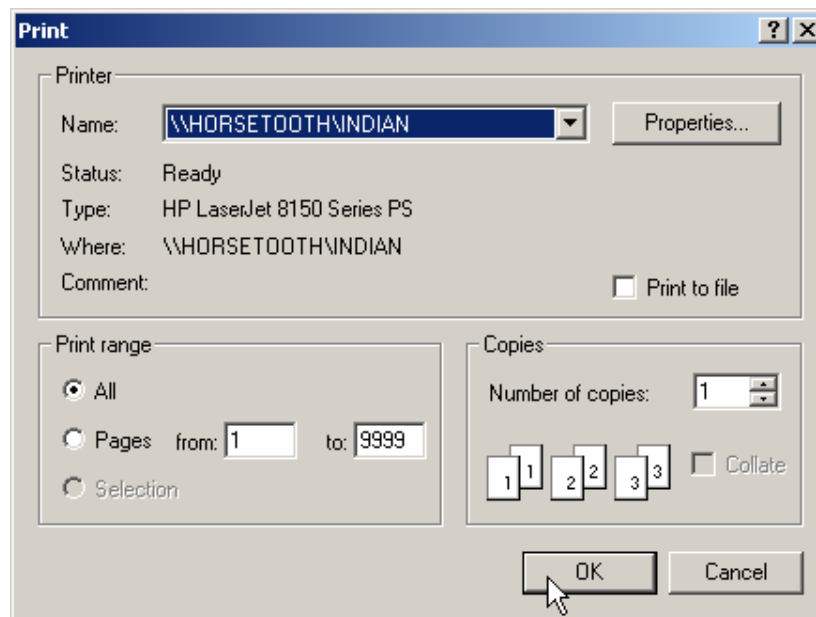
Pick a printer that can handle the page size specified in the current network editor page layout and press **OK**.

3. In the original dialog, select the paper size to match the current network layout and press **OK**:



NetworkEditor\_Print3



4. A Windows **Print** dialog will be shown:



NetworkEditor\_Print4

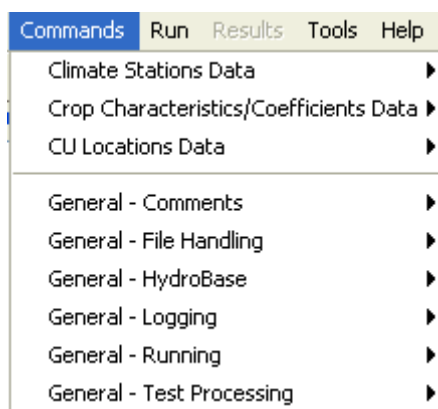
DO NOT change the printer settings. Simply press **OK** to finish printing.

### 3.6.5.5 Saving the Network as an Image

To save the entire network as an image, use the  tool and select an image file. To save the visible network as an image, use the  tool and select an image file.

## 3.7 Commands Menu – Insert Commands for Processing Data Components

The **Commands** menu lists groups of related commands that can be used to process model data. The contents of the **Commands** menu will be appropriate for each model. For example, the top level menu for StateCU is as follows:



MenuCommands\_StateCU

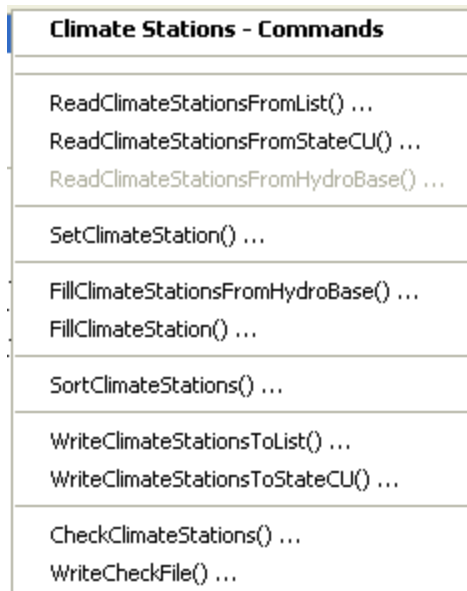
**Commands Menu for StateCU**

The general guidelines for data and menus are:

- Components are grouped according to physical data and identify a primary component for each group, which will supply the identifiers and names for individual data objects. For example, for **Climate Stations Data**, StateCU has a climate stations file, which has identifiers and names for climate stations. This component is the primary component in the “Climate Stations Data” group. The secondary components are time series data at each station.
- As much as possible, groups and components are listed according to dependency and processing order. For example, for StateCU, the **CU Locations Data** includes files that use crop types. The definitions of crop types are stored in a separate file. Because the CU Locations files use the crop types, and therefore depend on their definitions, crop data are listed before CU Locations data. The recommended order is not required; however, it provides some structure to creating a data set.

In some cases, selecting a data component menu will display a dialog indicating that the files for that component cannot be prepared with StateDMI and instead should be prepared with TSTool, a spreadsheet, or some other software. The intent of the StateDMI menus is to show all data components in order to help the user create a complete data set; however, other software may be required.

The sub-menus for a data component provide specific commands for the file that is being processed. Each sub-menu lists commands that can be inserted into the **Commands** list, which can then be processed to produce output. For example, the menu for **Climate Stations** is:



MenuCommands\_ClimateStations

### Commands...Climate Stations Data...Climate Stations Menu

The menus for a specific data component typically include commands to read the list of objects, set additional information, fill missing data, perform calculations (if appropriate), write output, and check the data.

To edit an existing command, select the command in the **Commands** list and then use the right-click **Edit** menu or the **Edit...Command** menu (or double-click on the command). This will display a command editor specific to the command. See the **Commands Reference** at the end of this documentation.

To insert a new command at the end of the **Commands** list:

1. Make sure that no commands are selected in the **Commands** list (see the title above the **Commands** list, which indicates if commands are selected).
2. Select the appropriate command menu and edit the command. After pressing **OK** in the command dialog, the command will be inserted at the end of the **Commands** list.

To insert a new command before an existing command in the **Commands** list:

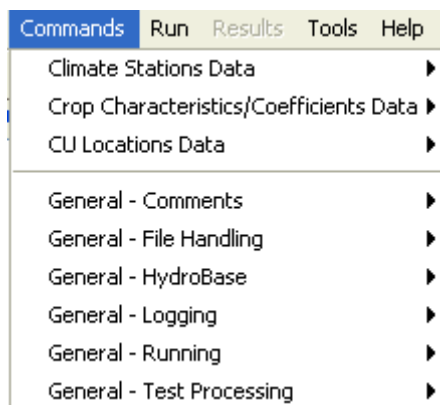
1. Select the command in the **Commands** list to insert before.
2. Select the appropriate command menu and edit the command. After pressing **OK** in the command dialog, the command will be inserted before the first selected command in the **Commands** list.

**Chapter 4 Creating StateCU Data Set Files** and **Chapter 5 Creating StateMod Data Set Files** discuss the sequence of commands that can be used to create model files. The **Commands Reference** describes each command and the dialog that is used to edit the command.



### 3.7.1 General Commands

General commands are listed under the **Commands...General – ...** menus and can be used with any model.

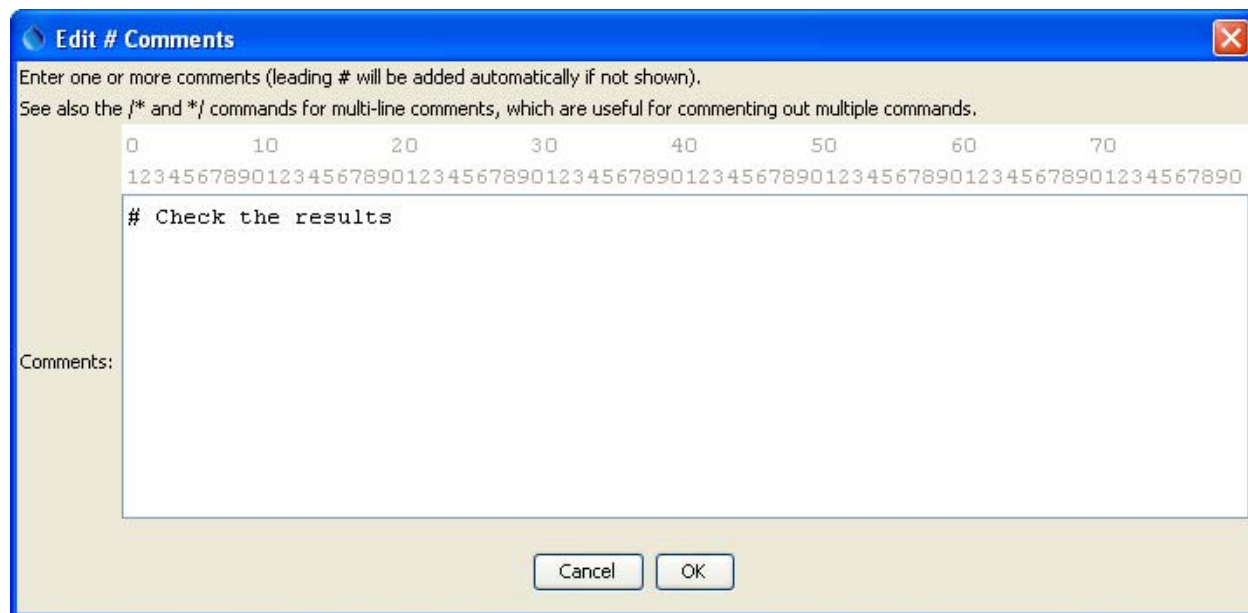


MenuCommands\_General

**Commands...General Menu**

#### **General Commands – Comments**

Single-line comments in commands files start with the # character and can be used to document commands. Multi-line comments start with the /\* characters and end with \*/ (a convention used in C, C++, C#, Java, and other programming languages). Multi-line comments are useful for commenting out blocks of commands. The following dialog is used to edit one or more # comment lines:



c\_comment

**# Comment Dialog**

A menu choice is also available to insert a #@readOnly comment – this will alert StateDMI to warn the user if they try to save the file. This special comment is useful for protecting command files that should not be edited.

### ***General Commands – File Handling***

File handling commands are useful for testing and other data management tasks.

The `MergeListFileColumns()` command is useful when processing list files. For example, the StateView software can be used to export a list of structures, where the identifiers use separate WD and ID columns. These columns can then be merged to produce a single WDID column, which can be processed by StateDMI to create model files.

### ***General Commands – HydroBase***

The `OpenHydroBase()` command programmatically opens a connection to a HydroBase database. This is useful if data from two databases need to be combined (open a connection, read data, open a new connection, read from the second database).

### ***General Commands – Logging***

The `StartLog()` command can be used to start a log file, which records processing steps and is useful in troubleshooting. Saving a specific log file also allows a comparison of data processing at different times. It is recommended that log files have the same name as the command file, with an optional date/time and the additional file extension `*.log`.

The `SetDebugLevel()` and `SetWarningLevel()` commands are usually only used in troubleshooting.

### ***General Commands – Running***

The `RunCommands()` command can be used to run one command file within another. This is useful for automated testing.

The `RunProgram()` and `RunPython()` commands are used to run external programs.

The `Exit()` command is useful for skipping over the last commands in a workflow, without having to comment them out.

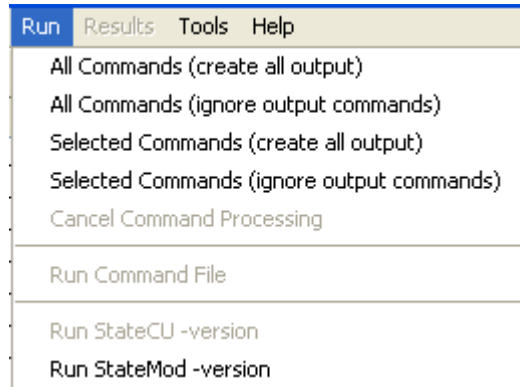
The `SetWorkingDir()` command is generally not used but is provided for backward compatibility.

### ***General Commands – Test Processing***

Test processing commands are used to validate the StateDMI software and standard workflow processes. See the **Quality Control** chapter for more information.

### 3.8 Run Menu – Running Commands

The **Run** menu processes the commands in the **Commands** list. Menu items similar to the following are also available in a popup menu by right clicking on the **Commands** list.



MenuRun

Run Menu

The **Run...All Commands (create all output)** menu will process the commands in the Commands list and create output if appropriate. For example, the `Write*( )` commands will write the data objects that are in memory to files.

The **Run...All Commands (ignore output commands)** menu will process the commands in the **Commands** list, ignoring commands that generate output products. This is useful when testing data processing commands and the (usually) slow write commands can be skipped.

The **Run...Selected Commands (create all output)** and **Run...Selected Commands (create all output)** menus are similar to the above; however, only commands that are selected will be run.

The **Run...Cancel Command Processing** menu item is enabled if commands are currently being processed. Use this menu item to cancel processing (e.g., if the commands result in excessive output or processing time). Processing will stop after the currently running command finishes.

The **Run...Command File** choice will run a commands file without making the results available in the interface. **This feature is not yet implemented.**

The **Run...StateCU -version** menu runs the StateCU model in order to display its version. This is useful when troubleshooting problems. **This menu item is currently disabled because the StateCU model does not have a version option.**

The **Run...StateMod -version** menu runs the StateMod model in order to display its version. This is useful when troubleshooting problems. However, it relies on StateMod being in the PATH, which may not be the case.

Select the **Help...About** menu to determine the version of StateCU and StateMod that was used when developing StateDMI. Changes to the model file formats for other versions may not be recognized in StateDMI.

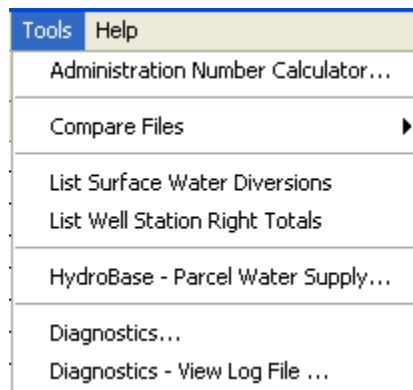
### 3.9 Results Menu – View Data Set and Command Results

The **Results** menu is currently disabled. It is envisioned as a way to view data set components from a data set or commands processing.

The alternative is to select results in **Results** area in the bottom of the main window, which provides access to all results.

### 3.10 Tools Menu

The **Tools** menu lists tools that perform useful tasks. Some of the menu items have been added to help during development.



MenuTools

Tools Menu

The **Tools...Administration Number Calculator...** menu can be used to convert between the State of Colorado's administration numbers and appropriation dates. Administration numbers are used by StateMod to determine the seniority of water rights.

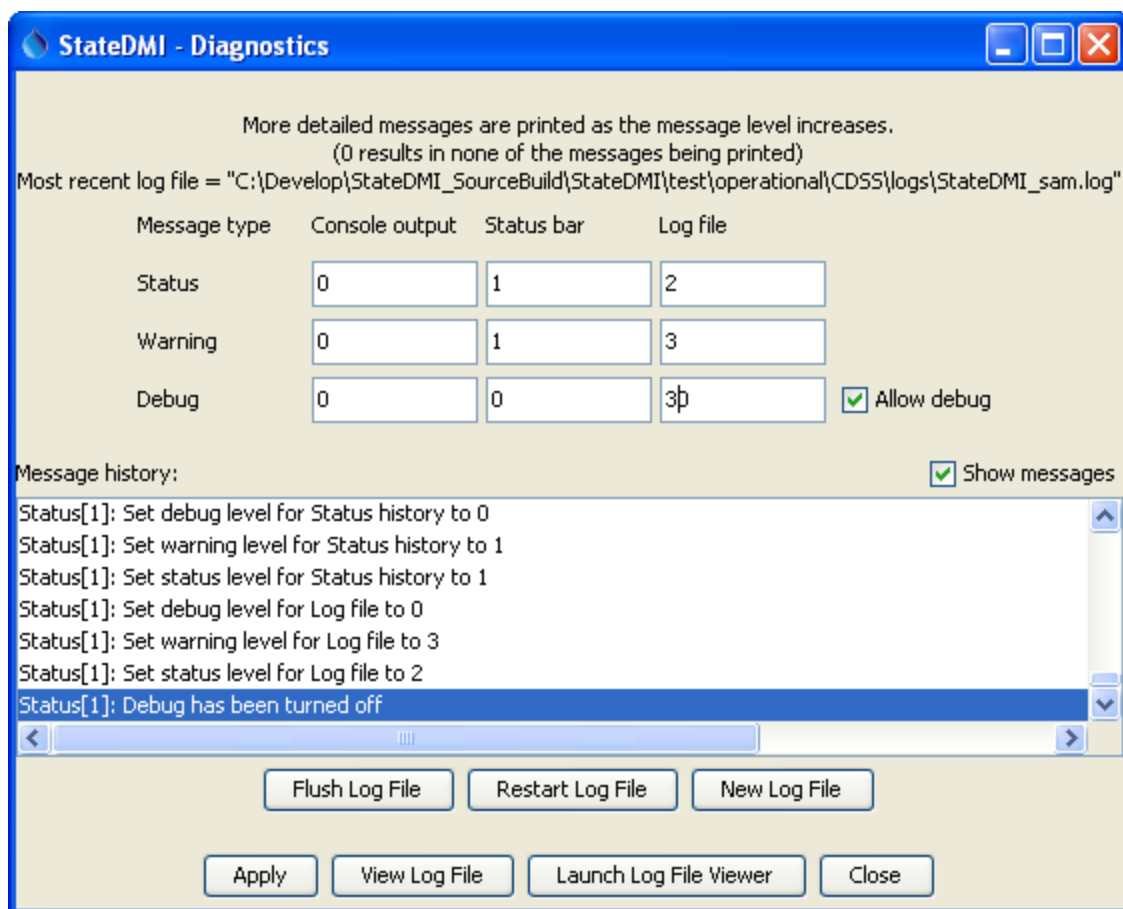
The **Tools...Compare Files** menu provides tools for comparing files, in particular used by developers during testing.

The **Tools...List Surface Water Diversions** tool can be used to list diversions from a StateMod diversion stations file that ONLY have surface water supply.

The **Tools...List Well Station Right Totals** tool can be used to list well station right totals by station.

The **Tools...Merge List File Columns** tool can be used to interactively select a delimited file and merge one or more columns to create a new column. This is useful, for example, when merging the WD and ID columns from StateView exports, to create a WDID column in a list file that is used with modeling. See also the companion `MergeListFileColumns()` command.

The **Tools...Diagnostics** menu displays the diagnostics interface, which is used to set message levels and view messages as StateDMI processes data. This is useful for tracking data problems, which result in warnings in display and analysis routines. Specify the level of detail for messages printed to various output locations by changing the values in the diagnostics window. Higher levels result in more output and slower performance.

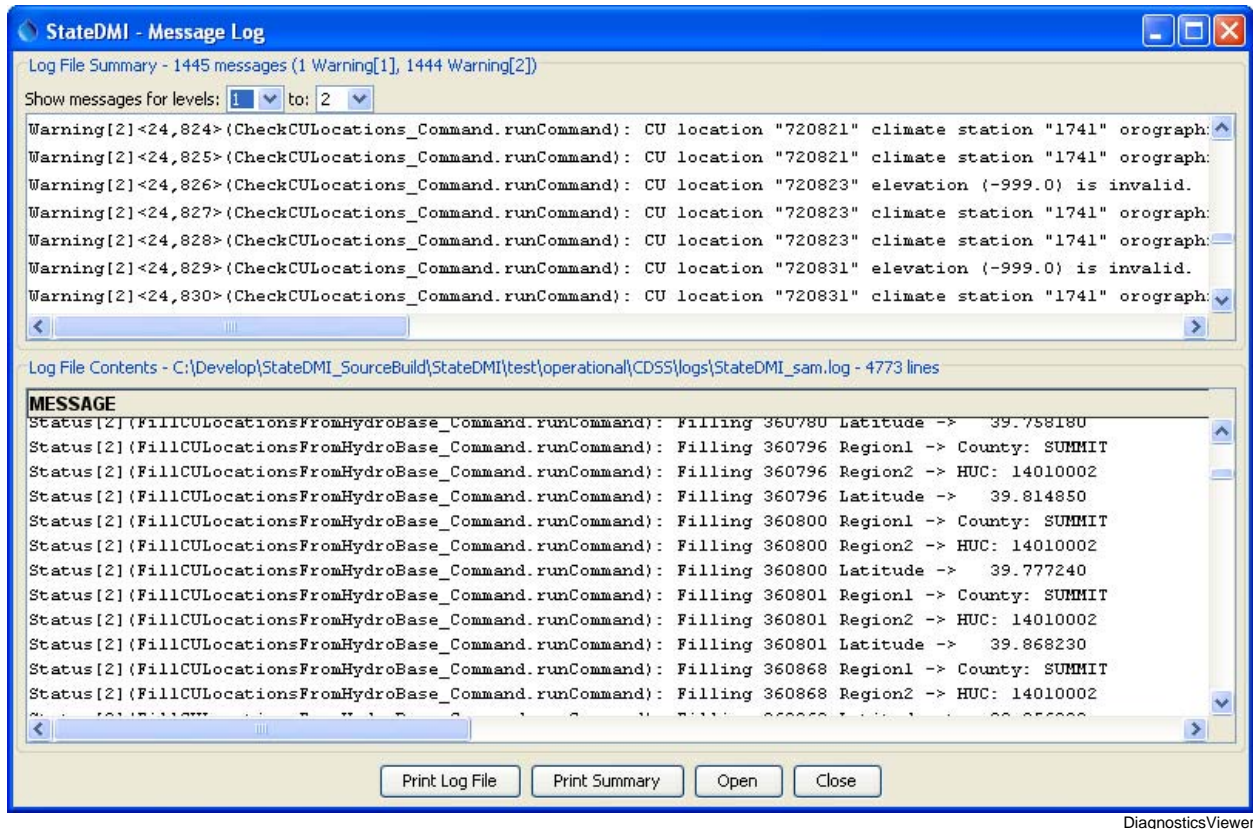


Diagnostics

### Diagnostics

Review the messages in the status bar at the bottom of the main window if output is not as expected. For more information, consult the log file or use the log file viewer (see next section), which contains these messages as well as more detailed information. The log file is named *StateDMI\_USER.log* and is created in the logs directory under the StateDMI installation directory. The user name is consistent with your system login. The **View Log File** and **Launch Log File Viewer** buttons will be enabled if the log file has been created. The former will display the log file in a new window, as described below. The latter will display the log file in Notepad.

Selecting the **View Log File** button in the **Tools...Diagnostics** tool or selecting the **Tools...Diagnostics – View Log File** menu will display the message log file viewer window:



DiagnosticsViewer

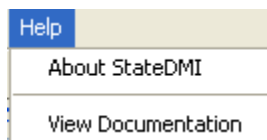
**Log File Viewer Window**

The log file viewer provides a summary of important warning messages in the top of the window. Selecting a message and right clicking provides options to go to the message in the main log file (bottom of the window) or go to the command in the main window.

The log file is useful for reviewing the detailed sequential steps of processing. However, the status information and check file output created by the `WriteCheckFile()` command are generally easier to use when troubleshooting workflow processing.

### 3.11 Help Menu

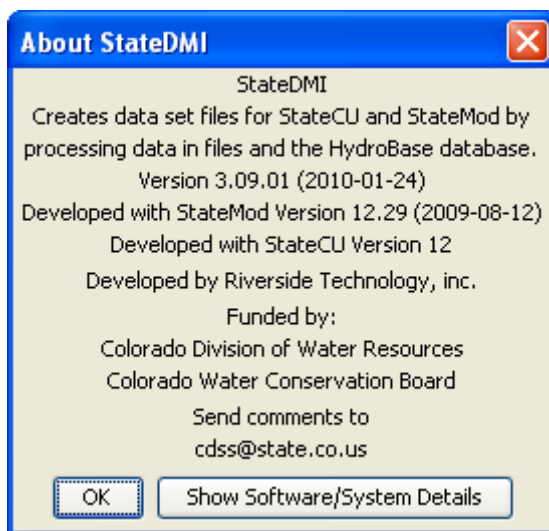
The help menu displays the StateDMI version and support information.



MenuHelp

#### Help Menu

The **Help...About StateDMI** menu displays the program version number, as shown in the following figure. Indicate the version number when reporting problems or suggestions.



MenuHelpAbout

#### Help...About StateDMI Dialog

If **Tools...Diagnostics** has been used to turn on debugging, then the above dialog will include a button labeled **Show Software/System Details**, which can be used to display information about the computer and StateDMI software. This information may be requested during troubleshooting.

The **Help...View Documentation** menu displays the software documentation in a web browser. Use the navigable table of contents to jump to a specific section.

This page is intentionally blank.

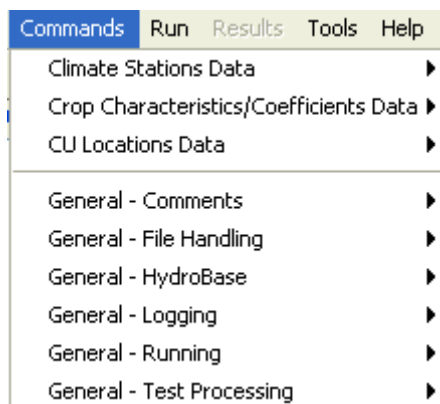


---

## 4 Creating StateCU Data Set Files

Version 3.10.00, 2010-04-02

When StateDMI is used to process StateCU data set files, the **Commands** menu lists the StateCU data groups (use **File...Switch to StateCU** if necessary to see the StateCU command menus):



MenuCommands\_StateCU

### Commands Menu when Used with StateCU Data Set Files

Each item corresponds to a data component group, under which are specific data components (products). Each data product corresponds to a model input file and is discussed in the following sections. The **General** commands are useful at any time (e.g., add comments). The top-level data groups utilize unique data identifiers shared among the products in the group. For example, the CU Locations Data are all referenced using a CU Location identifier (e.g., a ditch identifier).

Examples of StateCU model files are not included in this documentation. Refer to the StateCU model documentation for detailed information about model file formats. Example command files are included for each product and are taken from existing data sets. Command file logic may vary by data set and existing data sets should be consulted if available. Data sets typically fall into two categories: those that include groundwater (e.g., Arkansas, Rio Grande, and South Platte), and those that do not (e.g., Colorado, Gunnison, San Juan, Yampa, White).

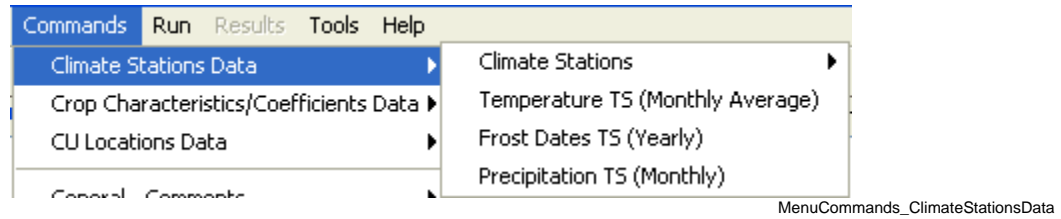
A StateCU analysis estimates water requirement at locations. StateDMI uses general terminology and refers to the locations as “CU Locations”, although StateCU data sets may focus on structures, climate stations, or other types of locations. Each CU Location is associated with climate data, crop patterns (either determined from actual irrigated lands or unit areas), and irrigation practice data. The CDSS data sets have in the past used the concept of County/HUC (Hydrologic Unit Code) to associate structures with climate stations. StateDMI uses a more general Region1/Region2 notation (e.g., the actual regions might be “County” and “” [no Region2]). The command editor dialogs provide information to help explain the key data that are used to associate the various data components.

### 4.1 Control Data

StateCU control data (the response and control file) are currently not processed by StateDMI, although commands may be added in the future. Currently you must create the StateCU control data using a text editor or copy and modify an existing file.

## 4.2 Climate Station Data

Climate station data consists of:

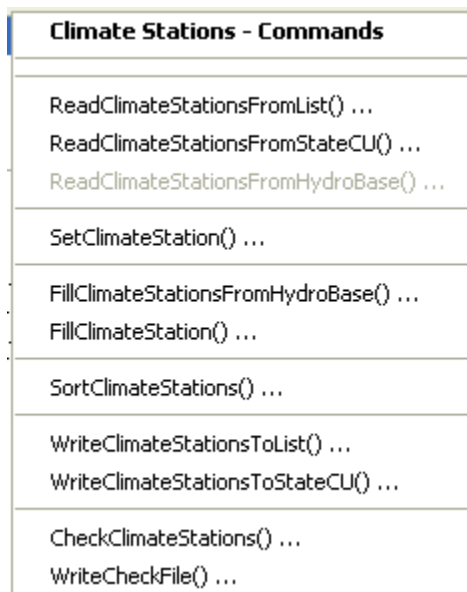


- Climate stations
- Temperature time series (monthly)
- Frost date time series (yearly)
- Precipitation time series (monthly)

Each of the above data types is stored in a separate file, using the climate station identifier as the primary identifier. Climate station weights are included in CU Location data. The processing of each data file is discussed below.

### 4.2.1 Climate Stations

Climate stations used with StateCU often are selected by reviewing available climate time series data to find stations with acceptable periods of record. TSTool or other software can be used to identify acceptable climate stations. The **Commands...Climate Stations Data...Climate Stations** menus insert commands to process climate station data:



MenuCommands\_ClimateStations

**Commands...Climate Stations Data...Climate Stations Data Menu**

The following table summarizes the use of each command:

### Climate Stations Data Commands

Command	Description
<code>ReadClimateStationsFromList()</code>	Read from a delimited list file the list of climate stations to be included in the data set.
<code>ReadClimateStationsFromStateCU()</code>	Read from a StateCU climate stations file the list of climate stations to be included in the data set.
<code>ReadClimateStationsFromHydroBase()</code>	Currently disabled. Read from HydroBase a list of climate stations to be included in the data set. It is envisioned that a county name or some other region would be supplied to help select climate stations. Instead, use the <code>FillClimateStationsFromHydroBase()</code> command.
<code>SetClimateStation()</code>	Set the data for, and optionally add, climate stations.
<code>FillClimateStationsFromHydroBase()</code>	Fill missing data for defined climate stations, using data from HydroBase.
<code>FillClimateStation()</code>	Fill missing data for defined climate stations, user user-supplied values.
<code>SortClimateStations()</code>	Sort the climate stations by station identifier.
<code>WriteClimateStationsToList()</code>	Write defined climate stations to a delimited list file.
<code>WriteClimateStationsToStateCU()</code>	Write defined climate stations to a StateCU file.
<code>CheckClimateStations()</code>	Check climate stations data for problems.
<code>WriteCheckFile()</code>	Write the results of data checks to a file.

An example command file is shown below (adapted from the Colorado cm2006 StateCU data set):

```
# StateDMI commands to create Colorado model climate stations file
#
# Step 1 - read climate stations from a list
#
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
#
# Step 2 - fill climate stations from HydroBase
#
FillClimateStationsFromHydroBase(ID="*")
#
# Step 3 - set/fill additional data not found in HydroBase
#
SetClimateStation(ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation(ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation(ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation(ID="0484",Region1="MOFFAT",IfNotFound=Add)
#
# Step 4 - write the file
#
WriteClimateStationsToStateCU(OutputFile="..\StateCU\COclim2006.cli")
#
# Step 5 - check results
#
CheckClimateStations(ID="*")
WriteCheckFile(OutputFile="CO.cli.StateDMI.check.html")
```

## 4.2.2 Temperature Time Series (Monthly)

Monthly temperature time series are not created by StateDMI. Instead, use TSTool or other software to create the time series file. An example TSTool command file is shown below (adapted from the Rio Grande data set). Refer to the TSTool documentation for current software features.

```
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2002")
SetOutputYearType(OutputYearType=Calendar)
#
# 2184 - DEL NORTE 2 E
2184.NOAA.TempMean.Month~HydroBase
#
#
# 0130 - ALAMOSA SAN LUIS VALLEY RGNL
0130.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="2184.NOAA.TempMean.Month",IndependentTSID="0130.NOAA.TempMean.Month",
    NumberOfEquations=OneEquation)
# perform regress operation on the following
#
FillRegression(TSID="0130.NOAA.TempMean.Month",IndependentTSID="2184.NOAA.TempMean.Month",
    NumberOfEquations=OneEquation)
TS AlamosaFill = Copy(TSID="0130.NOAA.TempMean.Month",NewTSID="0130.NOAA.TempMean.Month.copy")
#
#
# 0776 - BLANCA
0776.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="0776.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
# 1458 - CENTER 4 SSW
1458.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="1458.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
# 3541 - GREAT SAND DUNES N M
3541.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="3541.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
# 3951 - HERMIT 7 ESE
3951.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="3951.NOAA.TempMean.Month",IndependentTSID="0130.NOAA.TempMean.Month",
    NumberOfEquations=OneEquation)
#
#
# 5322 - MANASSA
5322.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="5322.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
# 5706 - MONTE VISTA 2 W
5706.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="5706.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
# 7337 - SAGUACHE
7337.NOAA.TempMean.Month~HydroBase
FillRegression(TSID="7337.NOAA.TempMean.Month",IndependentTSID="AlamosaFill",NumberOfEquations=OneEquation)
#
#
Free(TSList=LastMatchingTSID,TSID="AlamosaFill")
#
WriteStateMod(TSList=AllTS,OutputFile="..\StateCU\temp2002.stm")
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="rg2002_tmp.TSTool.check.html")
```

### 4.2.3 Frost Date Time Series (Yearly)

Yearly frost date time series are not created by StateDMI. Instead, use TSTool or other software to create the time series file. Note that older versions of TSTool internally treated frost date time series as a special time series with four frost dates per year. However, this representation could not be handled generically by TSTool's data filling and analysis features. Consequently, the current TSTool treats frost dates as Julian days since the beginning of the year (day 1 = January 1), allowing data to be filled with any of the standard commands, and time series to be graphed similar to other data. The following TSTool command file excerpt illustrates how to create the StateCU frost date file (adapted from the Rio Grande data set). Refer to the TSTool documentation for current software features.

```
SetOutputPeriod(OutputStart="1950",OutputEnd="2002")
#
# 0130 - ALAMOSA SAN LUIS VALLEY RGNL
0130.NOAA.FrostDateL28S.Year~HydroBase
0130.NOAA.FrostDateL32S.Year~HydroBase
0130.NOAA.FrostDateF32F.Year~HydroBase
0130.NOAA.FrostDateF28F.Year~HydroBase
#
#
# 0776 - BLANCA
0776.NOAA.FrostDateL28S.Year~HydroBase
0776.NOAA.FrostDateL32S.Year~HydroBase
0776.NOAA.FrostDateF32F.Year~HydroBase
0776.NOAA.FrostDateF28F.Year~HydroBase
#
#
# 1458 - CENTER 4 SSW
1458.NOAA.FrostDateL28S.Year~HydroBase
1458.NOAA.FrostDateL32S.Year~HydroBase
1458.NOAA.FrostDateF32F.Year~HydroBase
1458.NOAA.FrostDateF28F.Year~HydroBase
#
#
FillHistYearAverage(TSList=AllMatchingTSID,TSID="*")
#
#
WriteStateCU(OutputFile="..\StateCU\Frost2002.stm")
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="rg2002_frost.TSTool.check.html")
```

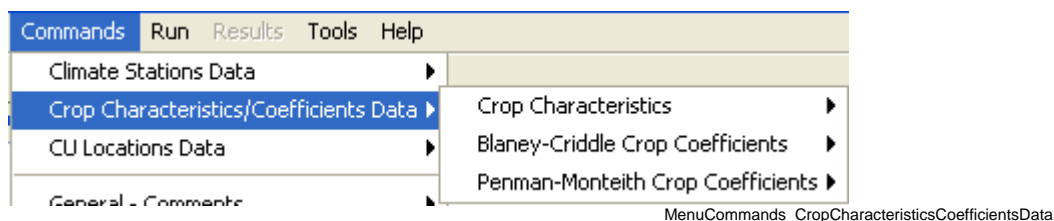
#### 4.2.4 Precipitation Time Series (Monthly)

Monthly precipitation time series are not created by StateDMI. Instead, use TSTool or other software to create the time series file. The following TSTool command file excerpt illustrates how to create the StateCU precipitation time series file (adapted from the Rio Grande data set). Refer to the TSTool documentation for current software features.

```
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2002")
SetOutputYearType(OutputYearType=Calendar)
#
# _____
# 0130 - ALAMOSA SAN LUIS VALLEY RGNL
0130.NOAA.Precip.Month~HydroBase
#
# _____
# 0776 - BLANCA
0776.NOAA.Precip.Month~HydroBase
#
# _____
# 1458 - CENTER 4 SSW
1458.NOAA.Precip.Month~HydroBase
#
# _____
# 2184 - DEL NORTE 2 E
2184.NOAA.Precip.Month~HydroBase
#
# _____
# 3541 - GREAT SAND DUNES N M
3541.NOAA.Precip.Month~HydroBase
#
# _____
# 3951 - HERMIT 7 ESE
3951.NOAA.Precip.Month~HydroBase
#
# _____
# 5322 - MANASSA
5322.NOAA.Precip.Month~HydroBase
#
# _____
# 5706 - MONTE VISTA 2 W
5706.NOAA.Precip.Month~HydroBase
#
# _____
# 7337 - SAGUACHE
7337.NOAA.Precip.Month~HydroBase
#
# _____
FillHistMonthAverage(TSList=AllTS)
#
# _____
WriteStateMod(TSList=AllTS,OutputFile="..\StateCU\Ppt2002.stm")
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="rg2002_precip.TSTool.check.html")
```

### 4.3 Crop Characteristics/Coefficients Data

StateCU crop characteristics and coefficients files are small files that provide information about crops, independent of irrigation practice.



The crop characteristics/coefficients data primary identifier is crop name (type), for example ALFALFA.TR21. The information after the period is associated with an analysis method. Crop data may be adjusted for high altitude or other local calibration efforts. The irrigated lands crop data (i.e., the data in HydroBase) are typically saved as ALFALFA, etc., because these data are independent of the use of the data. To make the crop names consistent during modeling, it is typical to use a `Translate*()` command before writing the data. For example, translate the more generic names to the longer names before writing the crop pattern time series to a file, specifying ID patterns to translate by location if necessary. Translate commands are available for data products that include the crop names. In documentation and software, crop “name”, “type”, and “identifier” are used interchangeably.

#### 4.3.1 Crop Characteristics

Crop characteristics include information about crop types that are used in an analysis, including planting, harvesting, and root depth data. Although only a few crops are typically used in an analysis in a basin, it is often convenient to provide information for many crop types. Crop characteristics should be defined before CU Locations because the crop types are used in the crop pattern time series file associated with CU Locations.

The **Commands...Crop Characteristics/Coefficients...Crop Characteristics** menu inserts commands to process the StateCU crop characteristics file:

Crop Characteristics - Commands
ReadCropCharacteristicsFromStateCU() ...
ReadCropCharacteristicsFromHydroBase() ...
SetCropCharacteristics() ...
TranslateCropCharacteristics() ...
SortCropCharacteristics() ...
WriteCropCharacteristicsToList() ...
WriteCropCharacteristicsToStateCU() ...
CheckCropCharacteristics() ...
WriteCheckFile() ...

MenuCommands\_CropCharacteristics

The following table summarizes the use of each command:

### Crop Characteristics Commands

Command	Description
<code>ReadCropCharacteristicsFromStateCU()</code>	Read from a StateCU file the crop characteristics to include in the data set.
<code>ReadCropCharacteristicsFromHydroBase()</code>	Read from HydroBase the crop characteristics to include in the data set.
<code>SetCropCharacteristics()</code>	Set the data for, and optionally add, crop characteristics data.
<code>TranslateCropCharacteristics()</code>	Translate crop characteristics name for specific modeling conventions, such as locally calibrated coefficients.
<code>SortCropCharacteristics()</code>	Sort the crop characteristics by crop name.
<code>WriteCropCharacteristicsToList()</code>	Write defined crop characteristics to a delimited list file.
<code>WriteCropCharacteristicsToStateCU()</code>	Write defined crop characteristics to a StateCU file.
<code>CheckCropCharacteristics()</code>	Check crop characteristics data for problems.
<code>WriteCheckFile()</code>	Write the results of data checks to a file.

An example command file is shown below (adapted from the Rio Grande data set).

```

StartLog(LogFile="Crops_CCH.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Crop Characteristics File
#
# Step 1 - read data from HydroBase
#
# Read the general TR-21 characteristics first and then override with Rio Grande
# data.
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 2 - adjust crop characteristics if needed
#   No resets are needed.
#
# Step 3 - write the file
#
WriteCropCharacteristicsToStateCU(OutputFile="rg2007.cch")
#
# Check the results
#
CheckCropCharacteristics(ID="*")
WriteCheckFile(OutputFile="Crops_CCH.StateDMI.check.html")

```



### 4.3.2 Blaney-Criddle Crop Coefficients

Blaney-Criddle crop coefficients estimate crop irrigation water requirement during the year or growing season, based on reference conditions. For daily (perennial) crop curves, 25 values are required, corresponding to the days of the year for month start/end and midpoints. For percent of season (annual) crop curves, 21 values are required, corresponding to 0, 5, ..., 100 percent of the growing season. The **Commands...Crop Characteristics/Coefficients...Blaney-Criddle Crop Coefficients** menu inserts commands to process the StateCU Blaney-Criddle crop coefficients file:

Blaney-Criddle Crop Coefficients - Commands
ReadBlaneyCriddleFromStateCU() ... ReadBlaneyCriddleFromHydroBase() ...
SetBlaneyCriddle() ... TranslateBlaneyCriddle() ...
SortBlaneyCriddle() ...
WriteBlaneyCriddleToList() ... WriteBlaneyCriddleToStateCU() ...
CheckBlaneyCriddle() ... WriteCheckFile() ...

MenuCommands\_BlaneyCriddle

The following table summarizes the use of each command:

**Blaney-Criddle Crop Coefficient Commands**

Command	Description
ReadBlaneyCriddleFromStateCU()	Read from a StateCU file the Blaney-Criddle coefficient data to include in the data set.
ReadBlaneyCriddleFromHydroBase()	Read from HydroBase the Blaney-Criddle coefficient data to include in the data set.
SetBlaneyCriddle()	Set the data for, and optionally add, Blaney-Criddle coefficient data.
TranslateBlaneyCriddle()	Translate crop name in Blaney-Criddle data, for specific modeling conventions, such as locally calibrated coefficients.
SortBlaneyCriddle()	Sort the Blaney-Criddle data by crop name.
WriteBlaneyCriddleToList()	Write defined Blaney-Criddle data to a delimited list file.
WriteBlaneyCriddleToStateCU()	Write defined Blaney-Criddle data to a StateCU file.
CheckBlaneyCriddle()	Check Blaney-Criddle data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file is shown below:

```
StartLog(LogFile="Crops_KBC.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Blaney-Criddle coefficients file
#
# Step 1 - read data from HydroBase
#
# Read the general Blaney-Criddle coefficients first and then override with Rio Grande
# data.
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 3 - write the file
#
SortBlaneyCriddle(Order=Ascending)
WriteBlaneyCriddleToStateCU(OutputFile="rg2007.kbc")
#
# Check the results
#
CheckBlaneyCriddle(ID="*")
WriteCheckFile(OutputFile="Crops_KBC.StateDMI.check.html")
```

### 4.3.3 Penman-Monteith Crop Coefficients

Penman-Monteith crop coefficients estimate crop irrigation water requirement during one or more growth stages, with coefficients specified at 10 percent intervals (0... 100 per growth stage). ALFALFA crops require 33 percent/coefficient pairs (3 growth stages), GRASS\_PASTURE requires 11 (1 growth stage), and all other crops require 22 (2 growth stages). The **Commands...Crop Characteristics/Coefficients...Penman-Monteith Crop Coefficients** menu inserts commands to process the StateCU Penman-Monteith crop coefficients file:

Penman-Monteith Crop Coefficients - Commands
ReadPenmanMonteithFromStateCU() ... ReadPenmanMonteithFromHydroBase() ...
SetPenmanMonteith() ... TranslatePenmanMonteith() ...
SortPenmanMonteith() ...
WritePenmanMonteithToList() ... WritePenmanMonteithToStateCU() ...
CheckPenmanMonteith() ... WriteCheckFile() ...

MenuCommands\_PenmanMonteith

The following table summarizes the use of each command:

### Penman-Monteith Crop Coefficient Commands

Command	Description
ReadPenmanMonteithFromStateCU()	Read from a StateCU file the Penman-Monteith coefficient data to include in the data set.
ReadPenmanMonteithFromHydroBase()	Read from HydroBase the Penman-Monteith coefficient data to include in the data set.
SetPenmanMonteith ()	Set the data for, and optionally add, Penman-Monteith coefficient data.
TranslatePenmanMonteith ()	Translate crop name in Penman-Monteith data, for specific modeling conventions, such as locally calibrated coefficients.
SortPenmanMonteith ()	Sort the Penman-Monteith data by crop name.
WritePenmanMonteithToList()	Write defined Penman-Monteith data to a delimited list file.
WritePenmanMonteithToStateCU()	Write defined Penman-Monteith data to a StateCU file.
CheckPenmanMonteith ()	Check Penman-Monteith data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file is shown below:

```
StartLog(LogFile="Crops_KPM.StateDMI.log")
#
# StatedMI commands to create the Penman-Monteith crop coefficients file
#
# Step 1 - read data from HydroBase
#
# Read the general ASCE standardized coefficients
ReadPenmanMonteithFromHydroBase(PenmanMonteithMethod="PENMAN-MONTEITH_ALFALFA")
#
# Step 3 - write the file
#
SortPenmanMonteith (Order=Ascending)
WritePenmanMonteithToStateCU(OutputFile="rg2007.kpm")
#
# Check the results
#
CheckPenmanMonteith (ID="*")
WriteCheckFile(OutputFile="Crops_KPM.StateDMI.check.html")
```

## 4.4 Delay Tables Data

Delay tables data were used previously with StateCU when modeling river depletions. This approach is no longer used. StateDMI features related to the delay tables data group have been disabled in StateDMI.

## 4.5 CU Location Data

The term *CU Location* is used to define a location where a consumptive use estimate is being determined. Consumptive use is determined for the following locations:

1. Diversion structures with only surface water supply.
2. Diversion structures with surface and groundwater supply. In this case, the wells are identified as an aggregate/system by ditch identifiers.
3. Wells or well fields with only groundwater supply. Agricultural locations are typically specified as an aggregate/system by parcel identifiers. Municipal single wells and well fields can also be modeled and are often defined as aggregate/systems by well identifiers.

The StateCU model, files, documentation, and interface primarily focus on consumptive use at structures and terminology is dominated by “structure”. For example, one of the main input files to StateCU is the structure (.str) file. However, to allow for more general application of StateCU, StateDMI uses the more general term *CU Location* in its menus and documentation. The current StateDMI features do focus on structure locations; however, the design allows for other types of locations. Examples of possible CU Locations are:

- ditches (diversion structures) and wells
- climate stations
- water district
- county
- parcel of land (e.g., irrigated parcel)
- any location specified by a coordinate

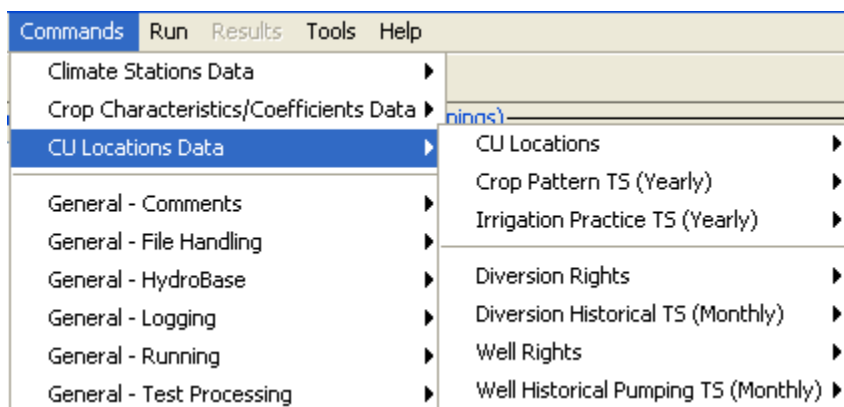
CU Locations are the entry point into several StateCU data set files. Once CU Locations are defined, other data objects, including crop patterns and irrigation practice, can be defined sequentially. In most cases, the CU Location identifier (e.g., a structure identifier) is used in related files. Therefore, these identifiers must be unique and are a primary key in all data processing.

StateCU previously managed CU Location data using county and HUC (Hydrologic Unit Code) identifier combinations. For example, the StateCU .str file includes fields for *county* and *HUC*. These fields can be treated generically as *Region 1* and *Region 2* because there is no real limitation to use county and HUC within StateCU. Therefore, StateDMI uses the terms *Region1* and *Region2* for these fields. Commands and the corresponding edit dialogs currently offer options only for county and HUC data but have been configured to allow future enhancements for other types of regions (for example where *Region 1* is “climate station” and *Region 2* is blank). More recent CU modeling is not tied to the County/HUC convention.

Data associated with CU Locations using the location ID are:

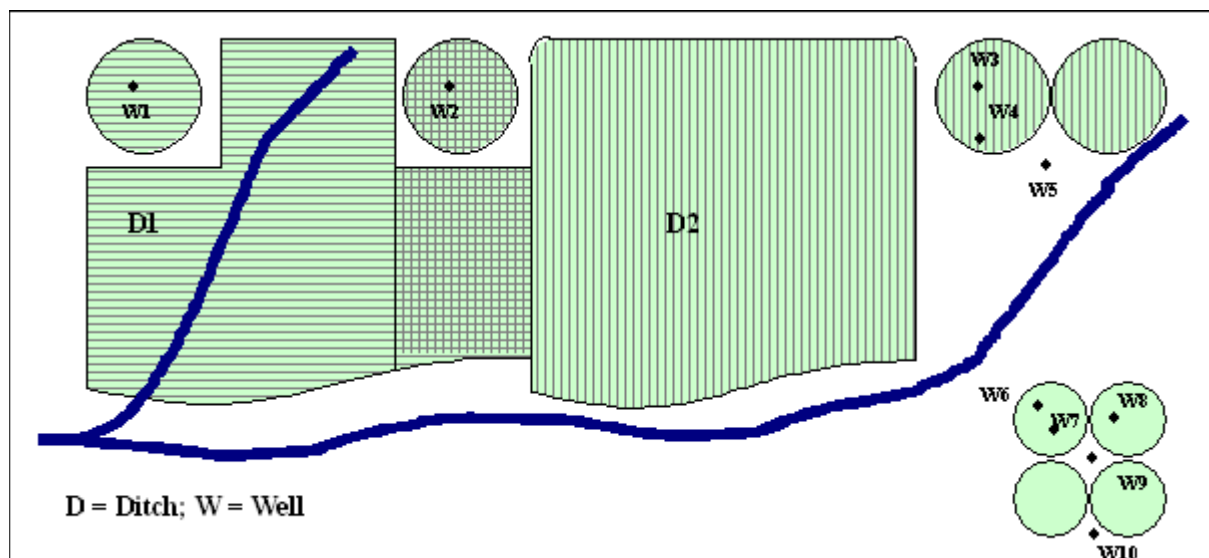
- CU Locations
- Crop pattern time series (yearly)
- Irrigation practice (parameter) time series (yearly)
- Diversion rights – water supply limited analysis only
- Diversion historical time series (monthly) – water supply limited analysis only
- Well Rights – used to limit groundwater acreage
- Well Historical Pumping Time Series – can be limited to well rights

The menu to access commands for each data component associated with CU Locations is shown below:



MenuCommands\_CULocationsData

The following figure illustrates possible ditch and well water supply for parcels.



ParcelSupplyDiagram

**Example Supply for Parcels**

In this example, two ditches (D1 and D2, each represented with different hatching) provide surface water supply to the indicated parcels. In some cases, only one ditch provides supply. Between the ditches, both supply water to shared parcels. Wells can supplement surface water supply (parcels above the river) or can be the sole supplier of water (lower right) and wells do not need to be physically located on a parcel to provide supply to the parcel.

In addition to explicit locations (e.g., single ditch), CU Locations may consist of a collection of individual parts. Currently, two main types of collections are recognized, as historically used in StateMod modeling:

- **Aggregate** – a group of diversions and/or wells where the water rights in the collection are aggregated (the original distinct rights are not individually accessible in the data set files). Aggregation reduces the number of water rights in model files, thereby decreasing the amount of output and model run times. Aggregation of well rights was used in Rio Grande modeling.

- **System** – a group of diversions and/or wells where water rights in the collection are not aggregated (each right is accessible in the data set files). For example, well systems are used in the South Platte data set, where individual rights are related to augmentation plans (StateMod plan stations). Output and model run times increase when individual rights are modeled.

In both cases, StateDMI assumes that the CU Location list includes all locations to be modeled. Any locations that are aggregates or systems must be defined using the appropriate commands (see `Set*Aggregate( )` and `Set*System( )` commands below). Diversions are grouped by specifying a list of the individual ditch identifiers (e.g., D1 in the above figure may be an aggregate of more than one ditch). Irrigation wells are grouped by indicating the parcel identifiers associated with wells (e.g., W6 – W10 in the above figure may be grouped into a single location for modeling, using the parcel identifiers to group the data). Municipal wells can be grouped by well identifier.

Aggregate and system identifier conventions are described in the **Introduction** chapter. In general, StateCU data sets should use the same conventions as defined in a related StateMod data set. In particular, when referencing a well station, use `aggregate/system` commands for well stations and when referencing a diversion station, use `aggregate/system` commands for diversion stations.

### 4.5.1 CU Locations

The **Commands...CU Locations Data...CU Locations** menu inserts commands to process the CU Locations (structure) file:

CU Locations - Commands
ReadCULocationsFromList() ... ReadCULocationsFromStateCU() ... ReadCULocationsFromStateMod() ...
SetCULocation() ... SetCULocationsFromList() ... SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ... SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
SortCULocations() ...
FillCULocationsFromList() ... FillCULocationsFromHydroBase() ... FillCULocation() ...
SetCULocationClimateStationWeights() ... SetCULocationClimateStationWeightsFromList() ... [Legacy] SetCULocationClimateStationWeightsFromHydroBase() ... FillCULocationClimateStationWeights() ...
WriteCULocationsToList() ... WriteCULocationsToStateCU() ...
CheckCULocations() ... WriteCheckFile() ...

MenuCommands\_CULocations

The following table summarizes the use of each command, in the order of the menu:

### CU Location Commands

Command	Description
ReadCULocationsFromList()	Read from a delimited list file the CU Locations to include in the data set.
ReadCULocationsFromStateCU()	Read from a StateCU structure file the CU Locations to include in the data set.
ReadCULocationsFromStateMod()	Read from a StateMod diversion or well station file the CU Locations to include in the data set.
SetCULocation()	Set data for an existing CU Location or optionally add a new CU Location.
SetCULocationsFromList()	Read and set CU Location data from a delimited list file.
SetDiversionAggregate()	For a diversion CU Location, indicate the parts that comprise an aggregate diversion.
SetDiversionAggregatesFromList()	For diversion CU Locations, indicate the parts that comprise aggregate diversions, using data in a delimited list file.
SetDiversionSystem()	For a diversion CU Location, indicate the parts that comprise a diversion system.
SetDiversionSystemsFromList()	For diversion CU Locations, indicate the parts that comprise diversion systems, using data in a delimited list file.
SetWellAggregate()	For a well CU Location, indicate the parts that comprise an aggregate well.
SetWellAggregatesFromList()	For well CU Locations, indicate the parts that comprise aggregate wells, using data in a delimited list file.
SetWellSystem()	For a well CU Location, indicate the parts that comprise a well system.
SetWellSystemsFromList()	For well CU Locations, indicate the parts that comprise well systems, using data in a delimited list file.
SortCULocation()	Sort the CU Locations. This is useful to force consistency between files.
FillCULocationsFromList()	Fill missing CU Location data, using data in a delimited list file.
FillCULocationsFromHydroBase()	Fill missing CU Location data, using data in HydroBase.
FillCULocation()	Fill missing CU Location data, using user-supplied data.
SetCULocationClimateStationWeights()	Set climate station weight data for a CU Location, using user-supplied data.
SetCULocationClimateStationWeightsFromList()	Set climate station weight data for a CU Location, using data in a delimited list file.
SetCULocationClimateStationWeightsFromHydroBase()	Set climate station weight data for a CU Location, using data in HydroBase. <b>Legacy command – not currently used.</b>
FillCULocationClimateStationWeights()	Fill climate station weight data for a CU location, using user-supplied data.
WriteCULocationsToList()	Write defined CU Locations data to a delimited list file.
WriteCULocationsToStateCU()	Write defined CU Locations data to a StateCU file.
CheckCULocations()	Check CU Location data for problems.
WriteCheckFile()	Write the results of data checks to a file.



An example command file is shown below (from preliminary South Platte Sp2008L data set). Lists of locations in this case have been generated from the StateMod network (see the StateMod chapter) and separate lists are maintained for various surface and groundwater locations.

```
# Sp2008L_STR.StateDMI
# South Platte Decision Support System
# Historic Consumptive Use Model
# Structure File (*.str)
#
# Step 1 - Read Structure List File (WDID, Name)
#
# Structure List includes Key Structures from Task 3, Aggregate GW, and Aggregate SW
ReadCULocationsFromList(ListFile="Sp2008L_StructList.csv",IDCol=1,NameCol=3)
#
# Step 2 - Read structure information from HydroBase (Latitude, County, HUC)
FillCULocationsFromHydroBase(ID="*",CULocType=Structure,Region1Type=County,Region2Type=HUC)
#
# Step 3 - Assign AWC values based on Task 57, generate using the CDSS Toolbox
#
# # Key Structure AWC Values
SetCULocationsFromList(ListFile="AWC_2001.csv",IDCol=1,AWCCol=2)
#
# # GW AGG Structure AWC Values
SetCULocationsFromList(ListFile="AWC_Agg_GW.csv",IDCol=1,AWCCol=2)
#
# # SW AGG Structure AWC Values
SetCULocationsFromList(ListFile="AWC_Agg_SW.csv",IDCol=1,AWCCol=2)
#
# Step 4 - Assign Elevation
FillCULocationsFromList(ListFile="Key_Elev.csv",IDCol=1,ElevationCol=3)
#
# Step 5 - Set Demand Structure Information based on Demand Carrier
SetCULocation(ID="0100503_I",Latitude=40.38,Elevation=4533.00,
  Region1="WELD",Region2="10190003",AWC=0.1375,IfNotFound=Warn)
#
SetCULocation(ID="6400526",AWC=0.1393,IfNotFound=Warn)
#
# Missing values assigned to Diversion Systems
SetCULocation(ID="0100503_D",Latitude=40.28567,Region1="MORGAN",IfNotFound=Warn)
# DivSys and Aggregate use weighted latitude from climate station assignments
# County and HUC information not assigned to DivSys or Aggregate Structures
#
# Step 6 - Read structure climate weights from list created from the CDSS Toolbox Climate Tool
SetCULocationClimateStationWeightsFromList(ListFile="Climate_2001.csv",IDCol=1,
  StationIDCol=2,TempWtCol=3,PrecWtCol=3)
#
# Step 8 - Fill Key Climate Station
#
FillCULocationClimateStationWeights(ID="01*",IncludeOrographicTempAdj=False,
  IncludeOrographicPrecAdj=False,Weights="0945,1.0,1.0")
#
# Step 7 - Write Structure File
SortCULocations()
WriteCULocationsToStateCU(OutputFile="SP2008L.str")
#
# Check the results
CheckCULocations(ID="*")
WriteCheckFile(OutputFile="SP2008L.str.check.html")
```

The following command file illustrates creation of the CU Location file for a basin without groundwater (taken from Colorado cm2006 data set):

```
ReadCULocationsFromList(ListFile="cmstrlist.csv",IDCol=1,NameCol=6)
FillCULocationsFromHydroBase(ID="*",CULocType=Structure,Region1Type=County,Region2Type=HUC)
SetCULocationsFromList(ListFile="cmstrlist.csv",IDCol=1,LatitudeCol=2,AWCCol=11)
SetCULocationsFromList(ListFile="plateau.csv",IDCol=1,Region1Col=2)
SetCULocationClimateStationWeightsFromList(ListFile="cowts.csv",
    StationIDCol=1,Region1Col=2,Region2Col=3,TempWtCol=4,PrecWtCol=5)
FillCULocationClimateStationWeights(ID="72_ADC065",Weights="3146,0.68,0.68,3489,0.32,0.32")
FillCULocationClimateStationWeights(ID="36*",Weights="4664,1.0,0.3592,0,1.0")
FillCULocationClimateStationWeights(ID="37*",Weights="2454,1.0,1.0")
FillCULocationClimateStationWeights(ID="38*",Weights="3359,1.0,1.0")
FillCULocationClimateStationWeights(ID="39*",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="45*",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="50*",Weights="3500,0.5,0.5,4664,0.5,0.5")
FillCULocationClimateStationWeights(ID="51*",Weights="3500,0.5,0.5,4664,0.5,0.5")
FillCULocationClimateStationWeights(ID="52*",Weights="9265,1.0,1.0")
FillCULocationClimateStationWeights(ID="53*",Weights="9265,1.0,1.0")
FillCULocationClimateStationWeights(ID="70*",Weights="0214,1.0,1.0")
FillCULocationClimateStationWeights(ID="72*",Weights="1741,1.0,1.0")
FillCULocationClimateStationWeights(ID="950001",Weights="3146,0.68,0.68,3489,0.32,0.32")
FillCULocationClimateStationWeights(ID="950010",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="950011",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="950050",Weights="3146,0.68,0.68,3489,0.32,0.32")
WriteCULocationsToStateCU(OutputFile="..\Statecu\cm2006.str",WriteHow=OverwriteFile)
# Check the results
CheckCULocations(ID="*")
WriteCheckFile(OutputFile="cm2006.str.StateDMI.check.html")
```

#### 4.5.2 Crop Pattern Time Series (Yearly)

Crop pattern time series indicate the annual crops and their acreage for each CU Location. The crop pattern file contains a time series of crop patterns for CU Locations, over the period that is being modeled. The crop pattern data include crop type names and area associated with the crop for the year. It is not required that all CU Locations include crops but this is often the case. If a crop is added for a CU Location in any year, StateDMI will output a value in each year. Consequently, a full time series will be available for each location/crop combination, even if many years have zeros. It is therefore important to fill such data appropriately such that missing data (e.g., -999) are removed from output.

The crop characteristics/coefficients data primary identifier is crop name (type), for example ALFALFA.TR21. The information after the period is associated with an analysis method. Crop data may be adjusted for high altitude or other local calibration efforts. The irrigated lands crop data are typically saved as ALFALFA, etc. in HydroBase, because these data are independent of the use of the data. To make the crop names consistent, use a TranslateCropPatternTS( ) command before writing the data. For example, translate the more generic names from HydroBase to the longer names before writing the crop pattern time series to a file, specifying ID patterns to translate by location if necessary.

The crop pattern time series file format was originally defined by legacy software in which a total acreage and fraction by crop is reported. Because the fraction has three significant figures, the resulting acreage by crop, when computed from the total, is only accurate to 3 significant figures. In current StateDMI software, the actual copy acreage is used in computations and the total and fraction are written to files only for information purposes and to retain the historical file format. Consequently, comparing acreage from old and new files may be slightly different due to the precision issue.

The **Commands...CU Locations Data...Crop Patterns TS (Yearly)** menu inserts commands to process the crop patterns:

Crop Pattern TS (Yearly) - Commands
SetOutputPeriod() ...
ReadCULocationsFromList() ... ReadCULocationsFromStateCU() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ... SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
CreateCropPatternTSForCULocations() ...
ReadCropPatternTSFromStateCU() ... SetCropPatternTSFromList() ... ReadCropPatternTSFromHydroBase() ...
SetCropPatternTS() ... TranslateCropPatternTS() ... RemoveCropPatternTS() ...
FillCropPatternTSConstant() ... FillCropPatternTSInterpolate() ... FillCropPatternTSRepeat() ... [Legacy] FillCropPatternTSUsingWellRights() ...
SortCropPatternTS() ...
WriteCropPatternTSToStateCU() ... WriteCropPatternTSToDateValue() ...
CheckCropPatternTS() ... WriteCheckFile() ...

MenuCommands\_CropPatternTS

The following table summarizes the use of each command, in the order of the menu items:

### Crop Pattern Time Series Commands

Command	Description
SetOutputPeriod()	Set the output period for crop pattern time series.
ReadCULocationsFromList()	Read CU Locations from a list file. Identifiers should be specified and other columns may be needed for data filling.
ReadCULocationsFromStateCU()	Read from a StateCU file the CU Locations to include in the data set.
SetDiversionAggregate()	For a diversion CU Location, indicate the parts that comprise an aggregate diversion.
SetDiversionAggregatesFromList()	For diversion CU Locations, indicate the parts that comprise aggregate diversions, using data in a delimited list file.
SetDiversionSystem()	For a diversion CU Location, indicate the parts that comprise a diversion system.
SetDiversionSystemsFromList()	For diversion CU Locations, indicate the parts that comprise diversion systems, using data in a delimited list file.
SetWellAggregate()	For a well CU Location, indicate the parts that comprise an aggregate well.
SetWellAggregatesFromList()	For well CU Locations, indicate the parts that comprise aggregate wells, using data in a delimited list file.
SetWellSystem()	For a well CU Location, indicate the parts that comprise a well system.
SetWellSystemsFromList()	For well CU Locations, indicate the parts that comprise well systems, using data in a delimited list file.
CreateCropPatternTSForCULocations()	Create empty crop pattern time series data for each CU Location. The resulting data can be updated with other commands.
ReadCropPatternTSFromStateCU()	Read crop pattern data from a StateCU file and update the StateDMI information.
SetCropPatternTSFromList()	Set crop pattern data from a list file, in order to supplement data that are not in HydroBase. A list file should be specified for each year of irrigated lands data. The data can be processed with HydroBase data as if they were parcels.
ReadCropPatternTSFromHydroBase()	Read crop pattern data from HydroBase.
SetCropPatternTS()	Set crop pattern data using user-supplied values.
TranslateCropPatternTS()	Change a crop type in crop pattern data.
RemoveCropPatternTS()	Remove a specific crop pattern time series.
FillCropPatternTSConstant()	Fill missing crop pattern data with a constant value.
FillCropPatternTSInterpolate()	Fill missing crop pattern data using interpolation.
FillCropPatternTSRepeat()	Fill missing crop pattern data by repeating values.

Command	Description
FillCropPatternTSUsingWellRights()	Fill crop pattern time series using well rights. This is used to turn off groundwater only parcels back in time during the early data period. <b>This legacy command is typically no longer used.</b>
SortCropPatternTS	Sort crop pattern time series by location identifier.
WriteCropPatternTSToStateCU()	Write defined crop pattern data to a StateCU file.
WriteCropPatternTSToDateValue()	Write defined crop pattern data to a DateValue file.
CheckCropPatternTS()	Check crop pattern data for problems.
WriteCheckFile()	Write the results of data checks to a file.

There are several ways to define crop pattern data in StateDMI:

1. Read a CU Locations file using ReadCULocationsFromStateCU() or ReadCULocationsFromList() and then read the associated crop patterns from HydroBase using ReadCropPatternTSFromHydrobase(). This is typically used if irrigated lands data have been populated in HydroBase and is the standard approach.
2. Read crop patterns from an existing crop patterns time series file using the ReadCUCropPatternsFromStateCU() command. This is typically only used if an existing file needs to be adjusted (e.g., by extending the period with fill options).
3. Utilize data that are not in HydroBase by using the SetCropPatternTSFromList() command. This may be appropriate for new development where data have not yet been loaded into HydroBase.

Once crop patterns are defined with the above commands, crop patterns for specific CU Locations can be edited using SetCropPatternTS() and SetCropPatternTSFromList() commands. These commands can also be used to supply values for specific locations, to be considered when irrigated lands are processed from a database. For example, acreage can be assigned to a structure that is part of an aggregate (but which does not have irrigated parcels in the database), and the supplied value will be included in the aggregate when the irrigated lands from the database are processed. Because determining crop patterns is a data- and labor-intensive effort, data are not typically available for each year in a modeling period. Therefore, crop patterns known for specific years are often extended or interpolated for other years using the FillCropPatternTSRepeat() and FillCropPatternTSInterpolate() commands. An attempt was made in the Rio Grande to relate crop patterns to agricultural statistics (crop planting and harvest data); however, this approach proved to be inaccurate and the more straightforward methods are typically used. Finally, output can be written using the WriteCropPatternsTSToStateCU() command.

An example commands file is shown below (from the Colorado cm2006 data set). This illustrates the major steps in the standard approach.

```
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,NameCol=2,
    PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,NameCol=2,
    PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
# Step 4 - Need to translate crops out of HB to include TR21 suffix
# Translate all crops from HB to include .TR21 suffix
TranslateCropPatternTS(ID="*",OldCropType="GRASS_PASTURE",NewCropType="GRASS_PASTURE.TR21")
TranslateCropPatternTS(ID="*",OldCropType="CORN_GRAIN",NewCropType="CORN_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SMALL_GRAINS",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="VEGETABLES",NewCropType="VEGETABLES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WO_COVER",NewCropType="ORCHARD_WO_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WITH_COVER",NewCropType="ORCHARD_WITH_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="DRY_BEANS",NewCropType="DRY_BEANS.TR21")
TranslateCropPatternTS(ID="*",OldCropType="GRAPES",NewCropType="GRAPES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="WHEAT",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SUNFLOWER",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SOD_FARM",NewCropType="GRASS_PASTURE.TR21")
# Step 5 - Translate crop names
# use high-altitude coefficients for structures with more than 50% of irrigated
# acreage above 6500 feet
TranslateCropPatternTS(ListFile="cm2005_HA.lst",IDCol=1,OldCropType="GRASS_PASTURE.TR21",
    NewCropType="GRASS_PASTURE.DWHA")
# Step 6 - Fill Acreage
# Fill SW structure acreage backward from 1999 to 1950
# Fill acreage forward for all structures from 2000 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1993,FillDirection=Backward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1993,FillEnd=1999,FillDirection=Forward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2000,FillEnd=2006,FillDirection=Forward)
# Step 7 - Write final *.cds file
WriteCropPatternTSToStateCU(OutputFile="..\StateCU\cm2006.cds",WriteCropArea=True)
# Check the results
CheckCropPatternTS(ID="*")
WriteCheckFile(OutputFile="cm2006.cds.StatedMI.check.html")
```

The following command file illustrates how to process crop characteristics in a basin with groundwater supply (from preliminary South Platte Sp2008L data set). The main difference is that lists of locations are defined using aggregate/system wells.

```
#
# Sp2008L_CDS.StatedMI
#
#
# StartLog(LogFile="Sp2008L_CDS.log")
# Crop Distribution File (*.cds) for the SPDSS Consumptive Use Model
#
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\LocationCU\SP2008L.str")
#
# Step 2 - Read SW aggregates, divsys, demandsys, and GW aggregates
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
#
```

```

SetWellSystemFromList(ListFile="..\SP_GWAGG_1956.csv",Year=1956,Div=1,PartType=Parcel,
  IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAGG_1976.csv",Year=1976,Div=1,PartType=Parcel,
  IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAGG_1987.csv",Year=1987,Div=1,PartType=Parcel,
  IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAGG_2001.csv",Year=2001,Div=1,PartType=Parcel,
  IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAGG_2005.csv",Year=2005,Div=1,PartType=Parcel,
  IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
#
# # Step 4 - Read well rights and determine gw-only structure acreage in 1950
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer")
FillCropPatternTSUsingWellRights(ID="*",IncludeSurfaceWaterSupply=False,CropType="*",
  FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 5 -
#   Fill SW structure acreage backward from 1956 to 1950
#   Linearly interpolate acreage for all structures between 1956, 1976, 1987, 2001, and 2005
#   Fill acreage forward for all structures from 2005 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1956,FillDirection=Backward)
FillCropPatternTSInterpolate(ID="*",CropType="*",FillStart=1956,FillEnd=1976)
FillCropPatternTSInterpolate(ID="*",CropType="*",FillStart=1976,FillEnd=1987)
FillCropPatternTSInterpolate(ID="*",CropType="*",FillStart=1987,FillEnd=2001)
FillCropPatternTSInterpolate(ID="*",CropType="*",FillStart=2001,FillEnd=2005)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2005,FillEnd=2006,FillDirection=Forward)
#
# Step 6 - Set to Missing and Fill primary WDID of Demand Structure = 0
SetCropPatternTS(ID="0100503_D",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
SetCropPatternTS(ID="0100507_D",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
SetCropPatternTS(ID="0100687",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
SetCropPatternTS(ID="0200834",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
#
SetCropPatternTS(ID="6400511_D",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
# Step 7 - No Acreage in HydroBase, Set to Missing = 0
SetCropPatternTS(ID="0100501",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
SetCropPatternTS(ID="0100513",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
SetCropPatternTS(ID="0100829",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
#
SetCropPatternTS(ID="6400519",SetStart=1950,SetEnd=2006,
  CropPattern="ALFALFA,0,CORN_GRAIN.TR21,0,SUGAR_BEETS,0,GRASS_PASTURE,0,VEGETABLES,0,SPRING_GRAIN.TR21,0",
  IrrigationMethod=Flood,SupplyType=Ground,ProcessWhen=Now)
# Step 8 - Translate crop names to Locally Calibrated based on structure location and elevation
# Source: Translate.xls (20070809)
# Alfalfa
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
TranslateCropPatternTS(ListFile="SP2008_CCLP.csv",IDCol=1,OldCropType="ALFALFA.TR21",
  NewCropType="ALFALFA.CCLP")
TranslateCropPatternTS(ListFile="SP2008_CCUP.csv",IDCol=1,OldCropType="ALFALFA.TR21",
  NewCropType="ALFALFA.CCUP")

```

```

# CORN_GRAIN
TranslateCropPatternTS( ID="*", OldCropType="CORN", NewCropType="CORN_GRAIN.TR21" )
TranslateCropPatternTS( ListFile="SP2008_CCLP.csv", IDCol=1, OldCropType="CORN_GRAIN.TR21",
    NewCropType="CORN_GRAIN.CCLP" )
TranslateCropPatternTS( ListFile="SP2008_CCUP.csv", IDCol=1, OldCropType="CORN_GRAIN.TR21",
    NewCropType="CORN_GRAIN.CCUP" )
# DRY_BEANS
TranslateCropPatternTS( ID="*", OldCropType="DRY_BEANS", NewCropType="DRY_BEANS.TR21" )
TranslateCropPatternTS( ListFile="SP2008_CCLP.csv", IDCol=1, OldCropType="DRY_BEANS.TR21",
    NewCropType="DRY_BEANS.CCLP" )
TranslateCropPatternTS( ListFile="SP2008_CCUP.csv", IDCol=1, OldCropType="DRY_BEANS.TR21",
    NewCropType="DRY_BEANS.CCUP" )
# GRASS_PASTURE
TranslateCropPatternTS( ID="*", OldCropType="GRASS_PASTURE", NewCropType="GRASS_PASTURE.TR21" )
TranslateCropPatternTS( ListFile="SP2008_CCLP.csv", IDCol=1, OldCropType="GRASS_PASTURE.TR21",
    NewCropType="GRASS_PASTURE.CCLP" )
TranslateCropPatternTS( ListFile="SP2008_CCUP.csv", IDCol=1, OldCropType="GRASS_PASTURE.TR21",
    NewCropType="GRASS_PASTURE.CCUP" )
TranslateCropPatternTS( ListFile="SP2008_DWHA_OroAdj.csv", IDCol=1, OldCropType="GRASS_PASTURE.TR21",
    NewCropType="GRASS_PASTURE.DWHA" )
# SMALL_GRAINS
TranslateCropPatternTS( ID="*", OldCropType="SMALL_GRAINS", NewCropType="SPRING_GRAIN.TR21" )
TranslateCropPatternTS( ListFile="SP2008_CCLP.csv", IDCol=1, OldCropType="SPRING_GRAIN.TR21",
    NewCropType="SPRING_GRAIN.CCLP" )
TranslateCropPatternTS( ListFile="SP2008_CCUP.csv", IDCol=1, OldCropType="SPRING_GRAIN.TR21",
    NewCropType="SPRING_GRAIN.CCUP" )
# SUGAR_BEETS
TranslateCropPatternTS( ID="*", OldCropType="SUGAR_BEETS", NewCropType="SUGAR_BEETS.TR21" )
TranslateCropPatternTS( ListFile="SP2008_CCLP.csv", IDCol=1, OldCropType="SUGAR_BEETS.TR21",
    NewCropType="SUGAR_BEETS.CCLP" )
TranslateCropPatternTS( ListFile="SP2008_CCUP.csv", IDCol=1, OldCropType="SUGAR_BEETS.TR21",
    NewCropType="SUGAR_BEETS.CCUP" )
# SUGAR_BEETS
TranslateCropPatternTS( ID="*", OldCropType="VEGETABLES", NewCropType="VEGETABLES.TR21" )
# SOD_FARM
TranslateCropPatternTS( ID="*", OldCropType="SOD_FARM", NewCropType="BLUEGRASS.POCHOP" )
# ORCHARD_WO_COVER
TranslateCropPatternTS( ID="*", OldCropType="ORCHARD_WO_COVER", NewCropType="ORCHARD_WO_COVER.TR21" )
#
# Step 9 - Write final *.cds file
WriteCropPatternTSToStateCU( OutputFile="..\StateCU\Historic\SP2008L.cds", WriteHow=OverwriteFile )
WriteCropPatternTSToStateCU( OutputFile="SP2008L.cds", WriteHow=OverwriteFile )

```



### 4.5.3 Irrigation Practice Time Series (Yearly)

The irrigation practice (parameter) time series file contains CU Location parameter data that are available as yearly time series. These data are also used as input to StateMod for use in groundwater and variable-efficiency modeling. The data in the file include the following for each year:

- Maximum delivery efficiencies, which may be specified with `SetIrrigationPracticeTS()` or `SetIrrigationPracticeTSFromList()` commands.
- Maximum flood irrigation efficiencies, which may be specified with `SetIrrigationPracticeTS()` or `SetIrrigationPracticeTSFromList()`. This applies to low efficiency irrigation methods such as flood and furrow.
- Maximum sprinkler irrigation efficiencies, which may be specified with `SetIrrigationPracticeTS()` or `SetIrrigationPracticeTSFromList()`. This applies to high efficiency irrigation methods such as sprinkler and drip.
- Acres irrigated from surface water only with flood irrigation (low efficiency irrigation). Data are typically read from HydroBase and then estimated with interpolation or repeat. See the example below.
- Acres irrigated from surface water only with sprinkler irrigation (high efficiency irrigation). Data are typically read from HydroBase and then estimated with interpolation or repeat. See the example below.
- Acres irrigated that have ground water supply (may also have surface water supply), flood irrigation (low efficiency irrigation). Data are typically read from HydroBase and then estimated with interpolation or repeat. See the example below.
- Acres irrigated that have ground water supply (may also have surface water supply), sprinkler irrigation (high efficiency irrigation). Data are typically read from HydroBase and then estimated with interpolation or repeat. See the example below.
- Maximum monthly pumping (ACFT), determined from summing the well yields/decrees for the wells associated with the location, using the permit and right dates to turn on wells. The data are usually processed with the `SetIrrigationPracticeTSPumpingMaxToWellRights()` command.
- Groundwater use mode, typically changed from defaults using the `SetIrrigationPraticeTS()` command.
- Total acres for location. These numbers should be an exact duplicate of the total acreage from the crop pattern time series. See the `SetIrrigationPracticeTSTotalAcreageFromCropPatternTSTotalAcreage()` command.

The definition of CU Locations as well/diversion system/aggregate is important because the logic to process each type of location is different. **The only way for StateDMI to know whether a CU location is groundwater only is to check for an aggregate/system that is specified as a list of parcels.** This is because CU Locations data in the StateCU files have no indicator of whether a location is a diversion, well or diversion supplemented by wells. This information could be determined from the irrigation practice file; however, creating this file is the subject of this section and the file is not available as input!

Because the irrigation practice time series file contains multiple time series, the `CreateIrrigationPracticeTSForCULocations()` command is used to create blank time series for each CU location, each filled with missing data. Appropriate `SetIrrigationPracticeTS*()` commands can then be used to define data values. Fill commands can be used to fill in missing values during the output period.

The crop pattern time series file should have been previously created and is utilized as the “baseline” for acreage by supplying the total acreage. The total acreage is maintained during irrigation practice data filling, adjusting acreage parts as appropriate. General guidelines on setting acreage, as implemented by commands discussed in this section, are as follows:

1. Crop pattern time series total acreage is relied on for the total acreage. Where inconsistencies occur (e.g., groundwater acres are higher than total acres), the crop pattern total takes precedence.
2. Groundwater acreage takes precedence next because of data availability for groundwater supply for parcels (well to parcel relationships). Total groundwater acreage is made consistent with the total acreage, and may cause a cascade of acreage adjustments described in following items. In cases where there is no groundwater supply, groundwater acreage is zero and surface water acreage takes precedence.
  - a. Groundwater acreage irrigated by sprinklers takes precedence over flood irrigation, based on irrigated lands irrigation method identification.
  - b. Groundwater acreage irrigated by flood is the remainder within the groundwater acreage.
3. Surface water only acreage (no groundwater supply) is set to total acres minus groundwater supply acres.
  - a. Surface water acreage irrigated by sprinklers takes precedence over flood irrigation, based on irrigated lands irrigation method identification.
  - b. Surface water acreage irrigated by flood irrigation is the remainder within the surface water only acreage.

Consequently, the following acreage relationships are maintained, and are used at checks at various locations during processing:

$$Total = Groundwater + SurfaceOnly$$

$$Total = (GroundwaterOnly_{Sprinkler} + GroundwaterOnly_{Flood}) + (SurfaceOnly_{Sprinkler} + SurfaceOnly_{Flood})$$

The above logic is necessary to make the solution of acreage terms determinate and may result in a cascade of acreage adjustments as values are set. If missing values remain after processing, it is necessary to utilize more data to set observations (e.g., using well rights to limit acreage in the early part of the period), or to use fill commands to fill gaps (which will result in the cascade of calculations described above). During initial data set development, it may be useful to implement one command at a time and review the results, studying the impacts of each command in filling in gaps.

The **Commands... CU Locations Data...Irrigation Practice TS (Yearly)** menu inserts commands to process the StateCU irrigation practice file:

<b>Irrigation Practice TS (Yearly) - Commands</b>
SetOutputPeriod() ...
ReadCULocationsFromList() ... ReadCULocationsFromStateCU() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ... SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
CreateIrrigationPracticeTSForCULocations() ... ReadIrrigationPracticeTSFromStateCU() ... ReadIrrigationPracticeTSFromHydroBase() ... ReadIrrigationPracticeTSFromList() ...
1: ReadCropPatternTSFromStateCU() ... 2: SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage() ... SetIrrigationPracticeTSPumpingMaxUsingWellRights() ... SetIrrigationPracticeTSSprinklerAcreageFromList() ... SetIrrigationPracticeTS() ... SetIrrigationPracticeTSFromList() ...
1: ReadWellRightsFromStateMod() ... 2: FillIrrigationPracticeTSAcreageUsingWellRights() ...
FillIrrigationPracticeTSInterpolate() ... FillIrrigationPracticeTSRepeat() ...
SortIrrigationPracticeTS() ...
WriteIrrigationPracticeTSToDateValue() ... WriteIrrigationPracticeTSToStateCU() ...
CheckIrrigationPracticeTS() ... WriteCheckFile() ...

MenuCommands\_IrrigationPracticeTS

The following table summarizes the use of each command:

### Irrigation Practice Time Series Commands

Command	Description
SetOutputPeriod()	Set the output period for irrigation practice time series.
ReadCULocationsFromList()	Read CU Locations from a list file. Identifiers should be specified and other columns may be needed for data filling.
ReadCULocationsFromStateCU()	Read from a StateCU file the CU Locations to include in the data set.
SetDiversionAggregate()	For a diversion CU Location, indicate the parts that comprise an aggregate diversion.
SetDiversionAggregateFromList()	For diversion CU Locations, indicate the parts that comprise aggregate diversions, using data in a delimited list file.
SetDiversionSystem()	For a diversion CU Location, indicate the parts that comprise a diversion system.
SetDiversionSystemFromList()	For diversion CU Locations, indicate the parts that comprise diversion systems, using data in a delimited list file.
SetWellAggregate()	For a well CU Location, indicate the parts that comprise an aggregate well.
SetWellAggregateFromList()	For well CU Locations, indicate the parts that comprise aggregate wells, using data in a delimited list file.
SetWellSystem()	For a well CU Location, indicate the parts that comprise a well system.
SetWellSystemFromList()	For well CU Locations, indicate the parts that comprise well systems, using data in a delimited list file.
CreateIrrigationPracticeTS ForCULocations()	Create empty irrigation practice time series data for each CU Location. The resulting data can be updated with other commands.
ReadIrrigationPracticeTS FromStateCU()	Read irrigation practice time series data from a StateCU file.
ReadIrrigationPracticeTS FromHydroBase()	Read irrigation practice acreage values from HydroBase.
ReadIrrigationPracticeTSFromList()	Read irrigation practice time series data from a list, optionally to combine with HydroBase data.
ReadCropPatternTSFromStateCU()	Read crop pattern time series from a StateCU file, in order to set the acreage total in the irrigation practice time series.
SetIrrigationPracticeTS TotalAcreage ToCropPatternTSTotalAcreage()	Set the irrigation practice total acreage to the crop pattern total acreage. This should be done after reading acreage data from HydroBase and before any other acreage filling occurs because the total is used as a check and is maintained in final results.
SetIrrigationPracticeTSPumpingMax UsingWellRights()	Set the irrigation practice pumping maximum time series to well rights. See also ReadWellRightsFromStateMod().

Command	Description
SetIrrigationPracticeTSSprinklerAcreageFromList()	Set the irrigation practice sprinkler acreage time series from a list file.
SetIrrigationPracticeTS()	Set irrigation practice data using user-supplied values.
SetIrrigationPracticeTSFromList()	Set irrigation practice data from a delimited list file.
ReadWellRightsFromStateMod()	Read a StateMod well rights file, for use with SetIrrigationPracticeTSPumpingMax UsingWellRights() and FillIrrigationPracticeTS().
FillIrrigationPracticeTS UsingWellRights()	Fill the irrigation practice acreage time series using well rights. This is only applied to lands with groundwater supply and is used in the early data period.
FillIrrigationPracticeTS Interpolate()	Fill missing irrigation practice data using interpolation.
FillIrrigationPracticeTSRepeat()	Fill missing irrigation practice data by repeating values.
SortIrrigationPracticeTS()	Sort irrigation practice time series by location identifier.
WriteIrrigationPracticeTS ToDateValue()	Write defined irrigation practice time series to a DateValue file. This is useful if the data are to be used with the TSTool software.
WriteIrrigationPracticeTS ToStateCU()	Write defined irrigation practice time series to a StateCU file.
CheckIrrigationPracticeTS()	Check crop pattern data for problems.
WriteCheckFile()	Write the results of data checks to a file.

The following example command file illustrates creating the irrigation practice file in a basin where groundwater supply is not included (from Colorado cm2006 data set):

```
# Step 1 - Set output period and read CU locations from structure file
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,NameCol=2,
    PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,NameCol=2,
    PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
# Set Max SW Eff = 1.0
SetIrrigationPracticeTS(ID="*",SurfaceDelEffMax=1.0,FloodAppEffMax=.60,SprinklerAppEffMax=.80,
    PumpingMax=0,GWMode=2)
SetIrrigationPracticeTSFromList(ListFile="cmstrlist.csv",ID="*",SetStart=1950,
    SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="7",FloodAppEffMaxCol="8",SprinklerAppEffMaxCol="9")
# Step 6 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Year="1993,2000",Div="5")
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="..\StateCU\cm2006.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
# Step 9 - Fill all land use acreage
# Fill groundwater acreage first
# Fill surface water sprinkler and flood 1950-2006
# Fill ground water sprinkler and flood 1950-2006
# Step 9a - estimate total GW and total SW
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1993",FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2000",FillEnd="2006",FillDirection="Forward")
# Step 9b - fill remaining irrigation method values
```

```

FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1993",FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2000",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="1993",FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="2000",FillEnd="2006",FillDirection="Forward")
# Step 10 - Write final ipy file
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\cm2006.ipy")
# Check the results
CheckIrrigationPracticeTS(ID="")
WriteCheckFile(OutputFile="cm2006.ipy.StateDMI.check.html")

```

An example command file that considers groundwater is shown below (from Rio Grande data set):

```

StartLog(LogFile="LocationCU_IPY.StateDMI.log")
#
# LocationCU_Ipy.StateDMI
# Used for all three scenarios:
#   1. rg2007      Division 3
#   2. rg2007_SW   Division 3 plus New Mexico
#   3. rg2007_GW   Ground Water Basin Only
#
# rrb 2007/10/09; Revised rgdssall_2007.csv = rgdssall_STR.csv
# to add the following based on new URF coverage
# 27URF28
# 20URF00
# 24URF00
# 24IRF00
# 35URF00
# 26URF21
#
# DOES INCLUDE 7 structures in New Mexico (90*).
# These are:
#           ID="90ACEQM",Name="Acequia Madre, NM")
#           ID="90AMALIA",Name="Amalia Area")
#           ID="90CERRO",Name="Cerro and Association")
#           ID="90CERTO",Name="Cerrito Canal")
#           ID="90METRJ",Name="ME Trujillo")
#           ID="90PAPEN",Name="Plaza Arriba and Penasquita")
#           ID="90PDMD",Name="Plaza Del Media")
#
#
# rrb 2007/09/18; File provided by Sam via Email on 9/19/2007. Revised as follows:
#
#           Revised to exclude 2002 data until Agro does additional analysis
#           Revised NoGIS_1936 to exclude the following structures already
#           in the 1936 coverage per Sam Email on 9/19.
#           Reset Taos No 3 (220639) because it sold in 1974
#           Removed extra output include by SAM
#           Revised sprinkler file from sprink_Acreage.csv to
#           Sprink_Acreage_2007.csv to get include 2003-2005 data
#
# rrb 2007/08/07; Revised Ditches with Recharge decrees to use GW method 3 from method 1
# rrb 2007/06/27; Comment out Excelsior (200627) so that it uses SW first
# rrb 2007/06/27; Add San Luis Valley (200829) to use GW first
#
#
# rrb 2007/06/18; Copied from Sam via FTP per Email 6/19/2007.
# rrb 2007/06/18; Revised to exclude 2002 until questions answered by Agro
#
#
# rrb 2007/06/20; Added 2002 back in because of the following enhancement
# rrb 2007/06/20; To be consistent between years and the URF coverage

```

```

#           Revised GW only lands from:
#       Nosurf_1936.csv to 1936_GWonly_Agg.csv
#       Nosurf_1998.csv to 1998_GWonly_Agg.csv
#       Nosurf_2002.csv to 2002_GWonly_Agg.csv
#
# Step 1 - read locations
#       Read locations with irrigation from a list file produced with the STR file
#
ReadCULocationsFromList(ListFile="rgdssall_STR.csv",IDCol=1,NameCol=2,LatitudeCol=3,
    ElevationCol=4,Region1Col=5,Region2Col=6,AWCCol=7)
#
# Step 2 - define aggregates and systems
#       Diversions are collections using a list of WDIDs, and the list of IDs is
#       constant through the model period.
#       Aggregates will result in well rights being aggregated.
#       Systems will be modeled with all well rights (no aggregation).
#       Well-only lands are collections using a list of parcel identifiers, and
#       the lists are specified for each year where data are available because the
#       parcel identifiers change from year to year.
#
# Diversions with and without groundwater supply...
SetDiversionAggregateFromList(ListFile="..\Diversions\rgTW_divaggregates.csv",
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
SetDiversionSystemFromList(ListFile="rg2007_divsystems_Acres.csv",IDCol=1,
    PartIDsCol=2,PartsListedHow=InRow)
#
# Wells with groundwater only supply..
SetWellSystemFromList(ListFile="..\Wells\1936_GWonly_agg.csv",Year=1936,Div=3,PartType=Parcel,IDCol=1,
    PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Wells\1998_GWonly_agg.csv",Year=1998,Div=3,PartType=Parcel,IDCol=1,
    PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 3 - create irrigation practice time series...
#       Specify a start of 1936 to use 1936 data in filling.
#       Only 1950 to 2005 will be output at the end.
#       The createIrrigationPracticeTSForCULocations() command creates empty time
#       series for each location, so that they can be manipulated with following
#       commands.
#
SetOutputPeriod(OutputStart="1936",OutputEnd="2005")
CreateIrrigationPracticeTSForCULocations(ID="*")
#
# Step 4 - fill/set data that are straightforward to set
#
# Step 4a - set efficiency limits for all structures.
#       These values are not in HydroBase.
#       Question - where do these come from?  StateCU?  Circular?
#
setIrrigationPracticeTSFromList(ListFile="eff.csv",IDCol="1",SurfaceDelEffMaxCol="2",FloodAppEffMaxCol="3",
    SprinklerAppEffMaxCol="4")
SetIrrigationPracticeTSFromList(ListFile="eff_2007.csv",ID="*",IDCol="1",SurfaceDelEffMaxCol="2",
    FloodAppEffMaxCol="3",SprinklerAppEffMaxCol="4")
#
# Step 4b - set the GWMode
#       The default is mutual ditch (GWMode=2).
#       Set GWMode for structures using the maximum supply mode (GWMode=1).
#       Does this have any impact on the order of importance of acreage below?
#
SetIrrigationPracticeTS(ID="200812",GWMode=3)
SetIrrigationPracticeTS(ID="200631",GWMode=3)
SetIrrigationPracticeTS(ID="200798",GWMode=3)
SetIrrigationPracticeTS(ID="200829",GWMode=3)
#
# Step 5 - set the pumping maximum for all locations using well rights
#
# Set the maximum well pumping to well water rights from the StateMod merged
# rights, which contains merged rights from the multiple years of irrigated lands.
# The number of days per month (30.4) is specified to convert CFS to AF/M and
# agrees with the data processing done in Phase 4.
# The full period will be set, including zeros at the beginning if no well

```

```

# rights are available.
# Locations that only have surface water (no well rights) will be set to zero
# throughout the set period.
ReadWellRightsFromStateMod(InputFile="..\Wells\rg2007.wer",Append=False)
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
  IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
#
# Step 6 - read the 1936 and 1998 Acreage/IrrigationMethod/SupplyType data
#
# Step 6a - provide supplemental data to be used - not in HydroBase
#
SetIrrigationPracticeTSFromList(ListFile="..\Crops\NoGIS_1998.csv",ID="*",SetStart=1998,
  SetEnd=1998,IDCol="1",AcresTotalCol="3")
SetIrrigationPracticeTSFromList(ListFile="..\Crops\NoGIS_1936.csv",ID="*",SetStart=1936,
  SetEnd=1936,IDCol="1",AcresTotalCol="3")
#
# Step 6b - read the data from HydroBase
#
#   Read 1936 and 1998 irrigated parcel data (area, irrigation method, supply
#   type[groundwater or not]) for each location.
#
#   After this step:
#   1. All acreage values for 1936 and 1998 will be set.
#   2. All other years will be missing.
#
ReadIrrigationPracticeTSFromHydroBase(ID="*",Year="1936,1998",Div="3")
#
# Step 7 - read the crop pattern total acreage and set as the IPY total acreage
#
# Step 7a - read CDS total acreage and set in IPY
#   The CDS total is used in all cases for the full period - include 1936 to
#   facilitate data checks and review trends. The extreme year 2002 is
#   omitted so as to not impact data filling at the end of the period. It is
#   read at the end of processing and superimposed on the results.
#   The period 1950+ is written at the end.
#
#   After this step:
#   1. All acreage values for 1936 and 1998 will be set.
#   2. All other years will be missing.
#   3. Total acreage will be set for the entire period.
#
ReadCropPatternTSFromStateCU(InputFile="..\Crops\rg2007_With1936.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*",SetStart=1936,SetEnd=2005)
#
# Step 8 - fill groundwater acreage time series
#
# Step 8a - first limit the groundwater acreage using well rights
#
#   Fill the groundwater acreage data prior to 1998 using the 1998 parcels
#   and associated water rights.
#   Turn off parcels when a water right is not available for
#   the year. This uses the water right file before it is merged for
#   multiple years because the unmerged 1998 rights are needed (therefore
#   the water rights file from max pumping CANNOT be reused here).
#
#   After this step:
#   1. All groundwater acreage prior to 1998 will have been estimated by using
#   the 1998 well data. The irrigation method will therefore be controlled
#   by the 1998 data (This may result in overestimating sprinkler acres
#   prior to 1970, before which sprinkler acreage should be zero) and will
#   need to be further refined below.
#   2. Surface water only total acreage will have been estimated for all
#   locations as Total - GW.
#   3. Surface water acreage by irrigation method will have been set to zero
#   for groundwater only locations. For other locations additional
#   processing will occur below (from user-supplied sprinkler data and/or
#   interpolate/repeat of irrigation method time series).
#
ReadWellRightsFromStateMod(InputFile="..\Wells\rg2007_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
  IncludeGroundwaterOnlySupply="True",FillStart=1937,FillEnd=1997,ParcelYear=1998)

```



```

#
# Step 8b - fill the groundwater total acres for years that could not be
# set from well rights.
#
# Since 1998 was used as the year for well rights, all years up to and
# including 1998 will have groundwater total acres set. This will also
# have resulted in surface water only total acres being set. Therefore,
# just repeat the 1998 values forward in time to the end of the period.
# Fill each irrigation method since the values will be set in 1998 and the
# information needs to be retained. The groundwater total acres will be
# computed from these values, and consequently the surface water total will
# be computed.
#
# After this step:
# 1. Groundwater total acreage will be set for all locations for the full
# period.
# 2. Groundwater acreage by irrigation method will still be missing for
# the years filled in this step, unless the groundwater total was zero,
# in which case the irrigation method parts will also be zero. See
# the step below to use RCWCD data to fill the irrigation method time
# series.
# 3. Surface water only total acreage time series will be computed as
# Total - GW acres.
# 4. Surface water only acreage by irrigation method will still be missing
# in some cases until the RCWCD is read below and/or repeat/interpolation
# of irrigation method time series occurs below.
# Filling the total acres is NOT NORMALLY NEEDED. However, this will
# fill in the 2002 data so that when groundwater acreage parts are set, they will
# be able to compare and adjust to the total. Filling over 2002 is needed to
# complete the standard process but 2002 will be superimposed on the end.
#
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-GroundWaterFlood",
    FillStart="1998",FillEnd="2005",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-GroundWaterSprinkler",
    FillStart="1998",FillEnd="2005",FillDirection="Forward")
#
# Step 8c - use RGWCD sprinkler data to adjust irrigation method
# The sprinkler acreage in the list file input to the command below should be filled for the
# period 1950 to 2005, interpolating between observed data and carried
# forward from 1998 to 2005. This file indicates only sprinkler acreage
# but not whether the acreage is for groundwater or surface-water only.
# The above commands have focused on resolving groundwater acreage
# total, utilizing 1998 data to determine whether FLOOD or SPRINKLER
# for estimated years, the following command redistributes the acreage
# within the groundwater total first, possibly resulting in a different
# irrigation method mix than from above. A summary of the steps is as
# follows:
# 1. GWSprinkler = min(Sprinkler_FromListFile,GWTotal)
# where GWTotal has resulted from the above processing steps
# and Sprinkler_ListFile is sprinkler acreage from the list file (no
# assumption about ground/surface water yet - the focus is on
# resolving the irrigation method within groundwater).
# 2. GWflood = GWtotal - GWSprinkler
# 2. SWSprinkler = min(Sprinkler_FromListFile - GWSprinkler, SWtotal)
# 4. SWflood = SWtotal - SWSprinkler.
#
# Note: This step is used in the Rio Grande.
# An alternative in basins like the South Platte is to use a set
# command to explicitly set GWSprinkler and SWSprinkler to zero
# in the early study period.
#
# After this step:
# 1. All acreage terms should be set except where no sprinkler data were
# available (in this case use a set command to set to zero if necessary) -
# see the following step.
# 2. The end of the period, where sprinkler was not set, may need to be
# repeated, interpolated from the last observation.
#
SetIrrigationPracticeTSSprinklerAcreageFromList(ListFile="..\SprinklerAcreage\sprink_acreage_2007.csv",
    ID="",YearCol=2,IDCol="1",AcresSprinklerCol="3")
#

```

```

# Step 8d - set sprinkler data to zero where RGWCD were not available in
# the early period.
#
# This will remove missing data from the early period.
#
# After this step:
# 1) The only missing data should be at the end of the period where
# sprinkler data were not provided from observations.
#
SetIrrigationPracticeTS(ID="*",SetStart=1936,SetEnd=1970,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 9 - fill surface water acres
#
# Step 9a - fill before 1998 using 1936
# Use interpolation between the 1936 and 1998 snapshots to fill in surface
# water sprinkler and flood acres - this defines the split between
# SWflood and SWSprinkler.
# After the initial interpolation, the values are adjusted so that
# TotalAcres - GWacres = SWflood + SWSprinkler
# If necessary, SWflood and SWSprinkler are prorated up/down to
# satisfy the above.
#
# Note for the South Platte, since there is not a bounding year with data
# at the start of the period, use fill repeat backwards at the period start.
#
# Step 9b - fill acreage after 1998 by repeating 1998
# Fill repeat after 1998 for all IPY acreage columns
# Or should this go after Step 10 below?
#
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-SurfaceWaterOnlySprinkler",
    FillStart="1998",FillEnd="2005",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-SurfaceWaterOnlyFlood",
    FillStart="1998",FillEnd="2005",FillDirection="Forward")
#
#
# Step 10 - read the 2002 Acreage/IrrigationMethod/SupplyType data
#
# Step 10a - provide supplemental data to be used - not in HydroBase
#
# Step 10b - read the data from HydroBase
#
# Read 2002 data.
#
#
# Step 10b; Replace Taos No 3 (220639
SetIrrigationPracticeTS(ID="220639",SetStart=1936,SetEnd=1973,AcresSWFlood=911.05,
    AcresSWSprinkler=0.0,AcresGWFlood=0.0,AcresGWSprinkler=0.0,AcresTotal=911.05)
SetIrrigationPracticeTS(ID="220639",SetStart=1974,SetEnd=2005,AcresSWFlood=109.24,
    AcresSWSprinkler=0.0,AcresGWFlood=0.0,AcresGWSprinkler=0.0,AcresTotal=109.24)
#
# Step 11 - write the StateCU IPY file(s)
#
# Step 11.a - write old format for Phase 4 comparison
# First write old format to allow comparison with Phase 4 and use with
# StateMod (not yet updated to version 12).
# Problem - this may not be possible given the adjustments that are made
# above - SAM will see if the old GW and Sprinkler data can be computed.
#
SortIrrigationPracticeTS()
WriteIrrigationPracticeTSToStateCU(OutputFile="..\CompareWithPhase4\Current\rg2007.Version10.ipy",
    OutputStart="1950",OutputEnd="2005",Version="10")
#
# Step 11.b - write the final results for use with StateCU and StateMod
# This uses the StateCU version 12+ output because all acreage
# computations require that the 4 acreage columns add up to the total.
#
WriteIrrigationPracticeTSToStateCU(OutputFile="rg2007_With1936.ipy")
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\rg2007.ipy",
    OutputStart="1950",OutputEnd="2005")
#
# Store results in source directory and StateMod

```

```
WriteIrrigationPracticeTSToStateCU(OutputFile="rg2007.ipy",OutputStart="1950",
    OutputEnd="2005")
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\Historic\rg2007.ipy",
    OutputStart="1950",OutputEnd="2005")
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateMod\Historic\rg2007.ipy",
    OutputStart="1950",OutputEnd="2005")
#
# Check the results
CheckIrrigationPracticeTS(ID="*")
WriteCheckFile(OutputFile="LocationCU_IPY.StateDMI.check.html")
```

#### 4.5.4 Diversion Water Rights

Diversion water rights, when used with StateCU, are used for a water supply limited by water rights analysis and are typically copied from a StateMod data set. The relevant commands are included with StateCU commands to facilitate creating the diversion rights file independent of a StateMod data set. Refer to the StateMod chapter for information about creating the diversion water rights file.

#### 4.5.5 Diversion Time Series

Diversion time series, when used with StateCU, are used for a water supply limited analysis and are typically copied from a StateMod data set. The relevant commands are included with StateCU commands to facilitate creating the diversion demand time series file independent of a StateMod data set. Refer to the StateMod chapter for information about creating the diversion demand time series file.

#### 4.5.6 Well Water Rights

Well water rights are used to set the maximum groundwater pumping data in the irrigation practice time series. The relevant commands are included with StateCU commands to facilitate creating the well water rights file independent of a StateMod data set. Refer to the StateMod chapter for information about creating the well water rights file.

#### 4.5.7 Well Historical Pumping Time Series (Monthly)

Historical pumping time series are typically produced by StateCU and are input to StateMod. The commands to process well demand time series are included as a copy of those available with StateMod data processing. However, these commands may not be suitable based on data availability. Refer to the StateMod chapter for a description of these commands and approaches for creating the historical pumping file for StateMod.

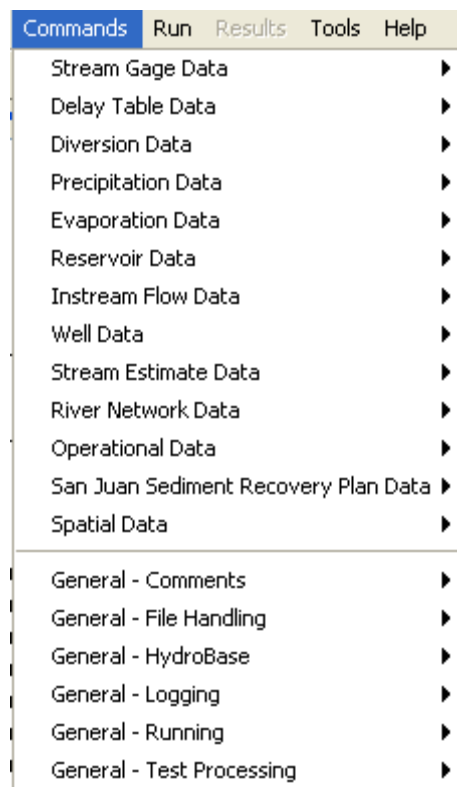
This page is intentionally blank.

---

## 5 Creating StateMod Data Set Files

Version 03.09.01, 2010-02-12

The **Commands** menu lists StateMod data components and groups when StateDMI is used to process StateMod data set files (use **File...Switch to StateMod** if necessary to see the StateMod command menus).



MenuCommands\_StateMod

**Commands Menu for StateMod**

Each menu corresponds to a data component group. Each sub-menu corresponds to a StateMod data set component and input file and is discussed in the following sections. The top-level data groups utilize unique data identifiers shared among products in the group. For example, Diversion Data are all referenced using a diversion station identifier. **General Commands** are useful at any time (e.g., add comments) and are discussed in the **Getting Started** chapter.

Examples of StateMod model files are not included in this documentation. Refer to the StateMod model documentation for detailed information about model file formats. Command file examples from CDSS data sets are included in documentation; however, refer to the current data sets for current examples because there may have been refinements in the approach.

The StateMod model is used to perform water allocation studies for a river basin. Most data files focus on data groups that include primary data files (e.g., station files) and secondary data files (e.g., water rights, time series). Some files provide more basic data (e.g., return flow patterns) and others provide more complex data (e.g., operating rules and river network). The organization of the StateDMI **Commands** menu is meant to facilitate creating data files in a logical order. However, there is generally

no limitation that prevents a user from combining commands in any desired order or working on files in other than the order shown.

## 5.1 Control Data

StateMod control data consists of:

- Response file
- Control file
- Output control file

Control data are currently not processed by StateDMI, although commands may be added in the future (e.g., to update the response file when a data file is written, so that the file names agree). Background information about each file is provided in the following sections.

### 5.1.1 Response File

A StateMod response file (\*.rsp) lists all the data files that are used in a model run and is specified to StateMod on the command line (see the StateMod documentation). The StateMod GUI also uses the response file when opening a data set. The response file is generally copied from an existing data set and hand-edited as appropriate. The base name of the response file (the part before the extension) should adhere to current modeling standards (see **Section 2.2 – Data Set Directory and File Conventions**). A separate response file for each run (e.g., for historical, calculated, baseline, daily runs) is usually created rather than editing the response file between runs. Note that some files can be specified as empty files, in which case StateMod will ignore the input type. The convention is to use an empty “dummy” file in these cases.

Recent updates to the StateMod model have introduced a free-format response file that allows data set files to be listed in any order, or be omitted altogether. This simplifies the management of a data. It is recommended that the newer free-format response file be used for StateMod because it allows more flexibility and reduces errors. See the StateMod documentation for information about the response file.

It is recommended that the files in a StateMod response file be specified using only file names (no paths) and that relative paths be used if necessary (e.g., ..\StateCU\rgTW.ddc). This allows the data set to be moved from one location to another without requiring edits to response files.

Rather than requiring a response file during processing, StateDMI provides commands that directly read needed files. For example, to process diversion station efficiencies, commands are provided to read the irrigation water requirement and historical diversion time series files.

### 5.1.2 Control File

The control file (\*.ctl) is a fixed-format file that specifies many of the run-time parameters to StateMod, including the simulation period (note the simulation period may be less than the input data period to shorten execution time) and parameters that control the execution (e.g., whether the run is for monthly or daily data). See the StateMod documentation for a full explanation of control file parameters. The meaning of data in some data files requires referencing the control file. For example, monthly efficiencies in the diversion and well stations files are listed according to the year type (calendar, water, or irrigation year) in the control file.

Rather than requiring that a control file is available during processing, StateDMI commands allow parameters to be specified as needed. For example, the `SetOutputYearType( )` command indicates whether output should be calendar or water year.

### 5.1.3 Output Control File

The output control file (\*.out or \*.xou) contains data that will limit the extent of selected output file requests when running StateMod in report mode. It is generated by StateMod using the `-check` option, which assumes you will want to review historical streamflow stations only. The output control file can be edited manually or with the StateMod GUI to add or remove additional structures for detailed output review. The output control file is not used by StateDMI.

## 5.2 Stream Gage Data

StateMod uses water supply from streamflow data to satisfy demands (it does not simulate run-off from precipitation). Stream gage data consists of:

- Stream gage stations
- Historical flow time series (monthly, daily)
- Natural flow time series (monthly, daily)

Each of the above data types is stored in a separate file, using the stream gage station identifier as the primary identifier. The term “River” and “Stream” are sometimes used interchangeably in StateMod documentation; however, StateDMI uses “Stream” in most cases. StateMod now supports separate stream gage and stream estimate data (see **Section 5.10 – Stream Estimate Data**). Stream gage stations correspond to locations where historical data are available, and when using separate stream gage and estimate station files should not include stream estimate stations. However, until modeling conventions begin utilizing separate stream gage and estimate station files, StateDMI also allows a combined stream gage/estimate station file (in which case the stream estimate station file is not used).

The processing of each data file is discussed below.

### 5.2.1 Stream Gage Stations

Stream gage stations used with StateMod often are selected by reviewing available stream gage historical time series data to find stations with acceptable periods of record. TSTool or other software can be used to identify acceptable stream gage stations.

Stream gage station identifiers are typically USGS or other agency identifiers. These identifiers correspond to data in HydroBase and other sources and therefore allow data to be located in the original source.

The **Commands...Stream Gage Data...Stream Gage Stations** menus insert commands to process stream gage station data:

Stream Gage Stations - Commands
ReadStreamGageStationsFromList() ...
ReadStreamGageStationsFromNetwork() ...
ReadStreamGageStationsFromStateMod() ...
SetStreamGageStation() ...
SortStreamGageStations() ...
FillStreamGageStationsFromHydroBase() ...
1: ReadNetworkFromStateMod() ...
2: FillStreamGageStationsFromNetwork() ...
FillStreamGageStation() ...
WriteStreamGageStationsToList() ...
WriteStreamGageStationsToStateMod() ...
CheckStreamGageStations()...
WriteCheckFile() ...

MenuCommands\_StreamGageStations

### Commands...Stream Gage Data...Stream Gage Stations Menu

The following table summarizes the use of each command:

#### Stream Gage Station Commands

Command	Description
ReadStreamGageStationsFromList()	Read from a delimited list file the list of stream gage stations to be included in the data set.
ReadStreamGageStationsFromNetwork()	Read from a StateMod network file a list of stream gage stations to be included in the data set.
ReadStreamGageStationsFromStateMod()	Read from a StateMod stream gage stations file the list of stream gage stations to be included in the data set.
SetStreamGageStation()	Set the data for, and optionally add, stream gage stations.
SortStreamGageStations()	Sort the stream gage stations. This is useful to force consistency between files.
FillStreamGageStationsFromHydroBase()	Fill missing data for defined stream gage stations, using data from HydroBase. For example, retrieve the station names.
ReadNetworkFromStateMod()	Read the network file for use in filling.
FillStreamGageStationsFromNetwork()	Fill missing data for defined stream gage stations, using data from a StateMod network file. This is useful when the station names are not found in HydroBase and numerous



Command	Description
	SetStreamGageStation() commands would otherwise be required.
FillStreamGageStation()	Fill missing data for defined stream gage stations, user user-supplied values.
WriteStreamGageStationsToList()	Write defined stream gage stations to a delimited list file.
WriteStreamGageStationsToStateMod()	Write defined stream gage stations to a StateMod file.
CheckStreamGageStations()	Check stream gage stations data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the stream gage station file, including stream estimate stations, is shown below (from the Colorado cm2005 data set):

```
StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase, fill in missing names from
#         the network file
#
FillStreamGageStationsFromHydroBase(ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
SetStreamGageStation(ID="954683",DailyID="09047500",IfNotFound=Warn)
SetStreamGageStation(ID="09046600",DailyID="09047500",IfNotFound=Warn)
... many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")
```

### 5.2.2 Stream Historical Time Series (Monthly, Daily)

StateDMI does not process stream historical time series. Instead, use TSTool, a spreadsheet, or other software to create the monthly and daily historical streamflow time series files. For simple models, use TSTool's `CreateFromList()` command to specify a list of station identifiers and create time series identifiers for HydroBase time series. The following TSTool command file excerpt illustrates how to create a historical monthly streamflow time series (from the Colorado cm2005 data set):

```
# rih.commands.TSTool
#
# creates historical streamflow file for the Colorado River Basin.
#
# step 1 - Extract data from Hydrobase or read *.stm files as noted below
#
SetInputPeriod(InputStart="10/1908",InputEnd="9/2005")
# COLORADO R BELOW BAKER GULCH, NR GRAND LAKE, CO.
09010500.USGS.Streamflow.Month~HydroBase
# COLORADO RIVER NEAR GRAND LAKE, CO.
09011000.USGS.Streamflow.Month~HydroBase
# COLORADO RIVER NEAR GRANBY, CO.
09019500...MONTH~StateMod~09019500.stm
# WILLOW CK BL WILLOW CK RESERVOIR
09021000...MONTH~StateMod~09021000.stm
# FRASER RIVER NEAR WINTER PARK, CO.
09024000.USGS.Streamflow.Month~HydroBase
# VASQUEZ CREEK AT WINTER PARK, CO.
09025000.USGS.Streamflow.Month~HydroBase
# ST. LOUIS CREEK NEAR FRASER, CO.
09026500.USGS.Streamflow.Month~HydroBase
...many similar commands omitted...
#
# Combine the two historic gages that sit on the Blue River above Dillon
#
# BLUE RIVER NEAR DILLON, CO.
09046600.USGS.Streamflow.Month~HydroBase
# Blue River at Dillon, CO
09047000.USGS.Streamflow.Month~HydroBase
FillFromTS(TSList=LastMatchingTSID,TSID="09046600.USGS.Streamflow.Month",
    IndependentTSList=LastMatchingTSID,
    IndependentTSID="09047000.USGS.Streamflow.Month")
Free(TSList=LastMatchingTSID,TSID="09047000.USGS.Streamflow.Month")
#
# SNAKE RIVER NEAR MONTEZUMA, CO.
09047500.USGS.Streamflow.Month~HydroBase
...many similar commands omitted...
#
# Use Homestake Creek near Red Cliff to fill missing values in Homestake Creek at
Gold Park
#
# HOMESTAKE CREEK AT GOLD PARK, CO.
09064000.USGS.Streamflow.Month~HydroBase
# 09064500 - HOMESTAKE CREEK NEAR RED CLIFF, CO.
09064500.USGS.Streamflow.Month~HydroBase
FillRegression(TSID="09064000.USGS.Streamflow.Month",
    IndependentTSID="09064500.USGS.Streamflow.Month",
    NumberOfEquations=MonthlyEquations,Transformation=Log)
Free(TSList=LastMatchingTSID,TSID="09064500.USGS.Streamflow.Month")
#
# Cross Creek nr Minturn, CO
09065100.USGS.Streamflow.Month~HydroBase
...many similar commands omitted
```

```
#
# Imports from other basins-replacement files created from 1909-2005 historical
diversions
404657...MONTH~StateMod~404657.stm
504600...MONTH~StateMod~504600.stm
950040...MONTH~StateMod~950040.stm
954001...MONTH~StateMod~954001.stm
#
# step 2 - Set output period and year type
SetOutputYearType(OutputYearType=water)
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
#
# step 3 - write output file
#
WriteStateMod(TSList=AllTS,OutputFile="..\StateMod\cm2005.rih",Precision=0)
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="rih.commands.StateDMI.check.html")
```

### 5.2.3 Stream Natural Flow Time Series (Monthly, Daily)

The stream natural flow file contains streamflows from which have been removed the impacts of historical diversions, return flows, well pumping, and reservoir storage, release, evaporation and seepage. It is normally generated by StateMod using the `-baseflow` option (the term “natural flow” has replaced “baseflow”, although software and documentation may still use the older term in places). To process natural flow time series, it is necessary to create station and historical time series files, but not water rights or demands. Stream natural flow time series for stream gage stations are not processed by StateDMI. Instead, use StateMod’s baseflow module, TSTool, or other software to create monthly and daily natural flow time series files.

When historical data are provided that allow 100% of human impacts to be removed, the natural flows generated by StateMod are the same as true natural flows. When historical data are provided that represent less than 100% of human impacts, it is implicitly assumed that the historical diversion and reservoir impacts that are left in the gage will not change significantly under a "What If" scenario.

The monthly natural flow time series file created by StateMod is automatically named (\*.xbm). However, it is commonly renamed (\*.rim) to ensure that a simulation scenario can be reproduced and allow input data sets to be distributed without having to rerun the baseflow module.

## 5.3 Delay Table Data

Delay table data consists of:

- Monthly delay tables
- Daily delay tables

Delay tables indicate the pattern for return flows (for diversion and well stations) and depletions (for well stations) and therefore should be available before processing diversion or well stations. Delay patterns represent how a unit of water is distributed by percent over time. Each delay table has a unique numerical identifier, and the identifier can be shared between monthly and daily files. StateDMI does not currently provide tools to generate delay tables (see the **Introduction** chapter for background on CDSS tools). However, commands are available to manipulate existing files.

### 5.3.1 Delay Tables (Monthly)

Monthly delay tables are typically produced manually (e.g., simple delay patterns) or by using the Glover method, Stream Depletion Functions (SDFs), or Unit Response Functions (URFs). In CDSS, the MakeRTN GIS tool has been used to develop return flow and depletion data for URF zones in the Rio Grande.

Delay table identifiers have traditionally been assigned sequential integer identifiers because StateMod does not support character identifiers for delay tables. Simple delay tables (e.g., return 100% of return flow in the first month) have lower delay table numbers.

The **Commands...Delay Tables Data...Delay Tables (Monthly)** menu items insert commands to process monthly delay table data:

Delay Tables (Monthly) - Commands
ReadDelayTablesMonthlyFromStateMod() ...
WriteDelayTablesMonthlyToList() ...
WriteDelayTablesMonthlyToStateMod() ...

MenuCommands\_DelayTablesMonthly

#### Commands...Delay Tables Data...Delay Tables (Monthly) Menu

The following table summarizes the use of each command:

#### Delay Table (Monthly) Commands

Command	Description
ReadDelayTablesMonthlyFromStateMod( )	Read monthly delay tables from a StateMod delay tables file, optionally scaling the delay values (e.g., to convert fraction to percent).
WriteDelayTablesMonthlyToList( )	Write monthly delay tables to a delimited list file.
WriteDelayTablesMonthlyToStateMod( )	Write monthly delay tables to a StateMod delay tables file.

### 5.3.2 Delay Tables (Daily)

The **Commands...Delay Tables Data...Delay Tables (Daily)** menu items insert commands to process daily delay table data:

Delay Tables (Daily) - Commands
ReadDelayTablesDailyFromStateMod() ...
WriteDelayTablesDailyToList() ...
WriteDelayTablesDailyToStateMod() ...

MenuCommands\_DelayTablesDaily

#### Commands...Delay Tables Data...Delay Tables (Daily) Menu

The following table summarizes the use of each command:

**Delay Table (Daily) Commands**

Command	Description
<code>ReadDelayTablesDailyFromStateMod()</code>	Read daily delay tables from a StateMod delay tables file, optionally scaling the delay values (e.g., to convert fraction to percent).
<code>WriteDelayTablesDailyToList()</code>	Write daily delay tables to a delimited list file.
<code>WriteDelayTablesDailyToStateMod()</code>	Write daily delay tables to a StateMod delay tables file.

## 5.4 Diversion Data

Diversion data consists of:

- Diversion stations
- Diversion rights
- Historical flow time series (monthly, daily)
- Demand time series (monthly, monthly override, average monthly, daily)
- Irrigation practice (yearly)
- Consumptive water requirement (monthly, daily)
- Soil moisture time series (yearly)

Each of the above data types is stored in a separate file, using the diversion station identifier as the primary identifier.

The processing of each data file is discussed below.

### 5.4.1 Diversion Stations

Each diversion station used with StateMod can be one of four types:

1. Explicit diversion, where no aggregation or special treatment occurs – this type is used for key structures that need to be explicitly modeled. The diversion station diverts from a single point on a water body. The diversion station identifier is usually a 7-character water district identifier (6-character for old data sets) or fabricated identifier that starts with the water district number.
2. Diversion “MultiStruct,” used to represent two or more diversion stations that divert from different tributaries but which serve the same lands. In this case, multiple diversion stations are grouped and one is assigned as the primary diversion station. To model historical conditions, each diversion station is represented in the network (e.g., using the WDID as the station identifier) and diversion records, water rights, and capacities correspond to each diversion station. To estimate average efficiencies (when evaluating demand time series), the total demand and historical time series are considered. Additionally, when estimating demand time series, the total demand is assigned to the primary structure and the demands for secondary structures are set to zero. Operating rules are required to control the exchange of water between diversions in the MultiStruct. This modeling construct should be defined using the `SetDiversionMultiStruct*()` commands and only need to be defined when processing demands.

3. Diversion system (a type of collection), where the characteristics (capacity, historical diversion, demand) of multiple diversions are summed at one location and water rights are modeled explicitly – this type is used when related diversion structures operate as a system to divert water from a single water source. Only the diversion system identifier is included in the model network and this identifier should be different from the parts in the collection. The naming convention for modeling in CDSS is to use a primary ditch in the collection for the modeled node or select an identifier that includes the district and “MS” or similar. Diversion systems should be defined using the `SetDiversionSystem* ( )` commands and need to be defined when processing all diversion station files (if diversion systems are used).
4. Diversion aggregate (a type of collection), which is the same as a diversion system except that water rights are aggregated into classes. Aggregation of the water rights occurs when the `ReadDiversionRightsFromHydroBase ( )` command is executed. The naming convention for modeling in CDSS is to use an identifier similar to 20\_ADCNNN, where the leading 20 indicates the water district, ADC indicates aggregate diversion, and NNN is a number to allow multiple diversion aggregates in a water district. This convention allows summary of demand and supply for basins. Diversion aggregates should be defined using the `SetDiversionAggregate* ( )` commands and need to be defined when processing all diversion station files (if aggregates are used).

The determination of the diversion station type for each diversion station is usually made by reviewing available data (e.g., water rights), and discussing administrative data with knowledgeable persons (e.g., water commissioners). Typically, key diversions have large capacities, irrigate larger acreage totals, and/or have important water rights and administrative roles. Minor diversions, or groups of diversions for which independent data are difficult to determine, may be lumped together in an aggregate or system. Grouping diversions into aggregates reduces the overall number of model nodes and output. Various commands refer to “collection type” when discussing aggregates and systems, in order to simplify documentation.

The diversion stations file may be updated several times, as follows:

1. Initial creation (see this section).
2. Adjust diversion station capacities based on historical diversions (see **Section 5.4.3**).
3. Adjust diversion monthly efficiencies based on estimates from consumptive water requirement (see **Section 5.4.5**).

However, it is also possible to create the secondary files using an initial list of diversion stations, and then create the StateMod diversion stations file with one command file.

The **Commands...Diversion Data...Diversion Stations** menus insert commands to process diversion station data:

<b>Diversion Stations - Commands</b>
SetOutputYearType() ...
ReadDiversionStationsFromList() ... ReadDiversionStationsFromNetwork() ... ReadDiversionStationsFromStateMod() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ...
SetDiversionStation() ... SetDiversionStationsFromList() ...
SortDiversionStations() ...
FillDiversionStationsFromHydroBase() ... FillDiversionStationsFromNetwork() ... FillDiversionStation() ...
SetDiversionStationDelayTablesFromNetwork() ... SetDiversionStationDelayTablesFromRTN() ...
WriteDiversionStationsToList() ... WriteDiversionStationsToStateMod() ...
CheckDiversionStations() ... WriteCheckFile() ...

MenuCommands\_DiversionStations

### Commands...Diversion Data...Diversion Stations Menu

The following table summarizes the use of each command:

#### Diversion Stations Commands

<b>Command</b>	<b>Description</b>
SetOutputYearType( )	Set the output year type. For diversion stations, this indicates the order of monthly efficiencies in the diversion stations data.
ReadDiversionStationsFromList( )	Read from a delimited list file the list of diversion stations to be included in the data set.
ReadDiversionStationsFromNetwork( )	Read from a StateMod network file a list of diversion stations to be included in the data set.
ReadDiversionStationsFromStateMod( )	Read from a StateMod diversion stations file the list of diversion stations to be

Command	Description
	included in the data set.
SetDiversionAggregate()	Specify that a diversion station is an aggregate and define its parts.
SetDiversionAggregateFromList()	Specify that one or more diversion stations are aggregates and define their parts, using a delimited list file.
SetDiversionSystem()	Specify that a diversion station is a system and define its parts.
SetDiversionSystemFromList()	Specify that one or more diversion stations are systems and define their parts, using a delimited list file.
SetDiversionStation()	Set the data for, and optionally add, diversion stations.
SetDiversionStationsFromList()	Set the data for diversion stations from a delimited list file.
SortDiversionStations()	Sort the diversion stations. This is useful to force consistency between files.
FillDiversionStationsFromHydroBase()	Fill missing data for defined diversion stations, using data from HydroBase. For example, retrieve the station names, and capacities.
FillDiversionStationsFromNetwork()	Fill missing data for defined diversion stations, using data from the network.
FillDiversionStation()	Fill missing data for defined diversion stations, using user-supplied values.
SetDiversionStationDelayTablesFromNetwork()	Set default delay table information using network relationships.
SetDiversionStationDelayTablesFromRTN()	Set delay table information using information in a return flow file.
WriteDiversionStationsToList()	Write defined diversion stations to a delimited list file.
WriteDiversionStationsToStateMod()	Write defined diversion stations to a StateMod file.
CheckDiversionStations()	Check diversion stations data for problems.
WriteCheckFile()	Write the results of data checks to a file.

If a multi-step process is used to create the diversion stations file, it is recommended that during initial creation of the diversion stations file, suitable default values are assigned to complete as much information as possible, including:

- capacity
- default monthly efficiencies
- acreage
- use and demand type
- delay tables

The following command file example (from the Colorado cm2005 data set) illustrates how to create a diversion station file. The output file will in this case be updated with historical diversion time series in



subsequent processing but could be updated in one step if the time series file is created first (e.g., by reading the diversion stations from a list file or the network when processing the time series file).

```

StartLog(LogFile="dds.commands.StateDMI.log")
# dds.commands.StateDMI
#
# StateDMI command file to create the "step 1" direct diversion station file
#
# Step 1 - set year type and read list of direct diversion stations from network file
#
SetOutputYearType(OutputYearType=Water)
ReadDiversionStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - read aggregate and diversion system structure assignments. Note that
#         want to combine historical acreage and capacities for aggs and diversion systems.
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionStation(ID="72_ADC054",IrrigatedAcres=1200,IfNotFound=Warn)
SetDiversionStation(ID="72_ADC055",IrrigatedAcres=928,IfNotFound=Warn)
#
# Step 3 - read diversion station information from HydroBase and sort alphabetically
#
FillDiversionStationsFromHydroBase(ID="*")
SortDiversionStations(Order=Ascending)
#
# Step 4 - set global options for all structures
#
SetDiversionStation(ID="*",RiverNodeID="ID",OnOff=1,ReplaceResOption=-
1,DailyID="4",DemandType=1,UseType=1,DemandSource=1,EffAnnual=60,IfNotFound=Warn)
SetDiversionStationDelayTablesFromNetwork(ID="*",DefaultTable=1)
#
# Step 5 - overwrite downstream return flow location, efficiencies and delay patterns based
#         on return flow file: read annual average irrigation efficiencies from StateCU (*.def)
#
SetDiversionStationDelayTablesFromRTN(InputFile="cm2005.rtn",SetEfficiency=True)
SetDiversionStationsFromList(ListFile="cm2005.def",IDCol="1",EffMonthlyCol="2",
    Delim="Space",MergeDelim=True)
#
# Step 6 - override HydroBase capacities and demand sources
#
# Transbasin Diversions - demscr=6 & resreplace=0 (does not get Green Mtn. replacement)
SetDiversionStation(ID="364626",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="364684",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="364685",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="374614",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="374641",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="371091",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="374648",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="364683",Capacity=500.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="364699",Capacity=77.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="954683",Name="Continental_Hoosier_Tunnel",Capacity=500.0,
    ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="384613",Capacity=120,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="384617",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="384625",Capacity=1000.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="954699",Name="Boustead_Summary",Capacity=1600.0,
    ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="514625",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="514601",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="514603",Capacity=500.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="514634",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="514655",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="724721",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="724715",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="384717",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
#
# The following are carriers to transbasin tunnel collections - demscr=7
#     Missouri Tunnel - Carrier to Homestake Tunnel
SetDiversionStation(ID="374643",Capacity=600.0,ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)

```

```

# Hunter Tunnel - Carrier to Bousted Tunnel
SetDiversionStation(ID="381594",Capacity=310.0,ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
# Moffat Tunnel Carriers
SetDiversionStation(ID="510728",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
SetDiversionStation(ID="511310",Name="Vasquez_Creek",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
SetDiversionStation(ID="511309",Name="St_Louis_Cr",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
SetDiversionStation(ID="510639",Name="Jim_Creek",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
SetDiversionStation(ID="511269",Name="Ranch_Creek",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#
# StateDMI expects monthly values to be entered in Calendar Year.
#
# The following are municipal and industrial diversions - demsrc=6
# Rankin No. 1 Ditch, Dillon Valley W&SD
SetDiversionStation(ID="360784",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Straight Creek Ditch, Town of Dillon
SetDiversionStation(ID="360829",Capacity=3.5,DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Climax Demands
SetDiversionStation(ID="360841",Capacity=53.19,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Keystone Snow Line Ditch (Snowmaking)
SetDiversionStation(ID="360908",Capacity=2.5,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Breckenridge Snowmaking
SetDiversionStation(ID="360989",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Breckenridge Municipal
SetDiversionStation(ID="361008",Capacity=4.87,DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Copper Mtn. Snowmaking
SetDiversionStation(ID="361016",Capacity=2.5,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Metcalf Ditch - Upper Eagle Valley Water Authority
SetDiversionStation(ID="370708",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Town of Rifle Pump and Pipeline
SetDiversionStation(ID="390967",Capacity=8.5,DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# East Snowmass Brush Creek Pipeline
SetDiversionStation(ID="381441",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Glenwood L Water Company System
SetDiversionStation(ID="531051",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="530585",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Maroon Ditch - Aspen
SetDiversionStation(ID="380854",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Midland Flume Ditch - Aspen
SetDiversionStation(ID="380869",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Carbondale Water System and Pipeline
SetDiversionStation(ID="381052",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Redlands Power Canal
SetDiversionStation(ID="420541",Capacity=610.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Henderson Mine Water System
SetDiversionStation(ID="511070",Capacity=8.8,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Shoshone Power Plant
SetDiversionStation(ID="530584",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Grand Junction Demands
SetDiversionStation(ID="950051",Name="Grand Junction
Demands",Capacity=21.0,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Grand Junction Gunnison Pipeline
SetDiversionStation(ID="420520",ReplaceResOption=0,DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Grand Junction Colorado River Pipeline
SetDiversionStation(ID="720644",DemandSource=6,
EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
# Molina Power Plant
SetDiversionStation(ID="720807",Capacity=50.0,ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
# Palisade Town Pipeline

```

```

SetDiversionStation(ID="720816",Capacity=5.0,DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
#   The following meet municipal demands for the Ute WCD
SetDiversionStation(ID="720920",Capacity=50.0,DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="721339",DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="950020",Name="Ute Water Treatment",Capacity=17.0,IrrigatedAcres=0,
  DemandSource=6,EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="950030",Name="Mason Eddy-Ute",Capacity=7.0,IrrigatedAcres=0,DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="721329",Name="Rapid Creek PP
DivSys",DemandSource=6,EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
#   Keystone Municipal
SetDiversionStation(ID="955002",Name="Keystone Municipal",Capacity=2.0,IrrigatedAcres=0,DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
#   Vail Municipal Use
SetDiversionStation(ID="955001",Name="Vail Valley Consolidated-Senior",Capacity=11.2,IrrigatedAcres=0,
  DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
SetDiversionStation(ID="955003",Name="Vail Valley Consolidated-Non Irr",Capacity=13.0,IrrigatedAcres=0,
  DemandSource=6,
  EffMonthly="10,12,14,44,55,62,61,56,44,26,0,10",IfNotFound=Warn)
#   Green Mtn. Hydro-Electric
SetDiversionStation(ID="360881",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
#   Williams Fork Power Conduit
SetDiversionStation(ID="511237",ReplaceResOption=0,DemandSource=6,IfNotFound=Warn)
#   Green Mtn. Contract Water Users (Baseline Scenario only)
SetDiversionStation(ID="950060",Name="Green_Mtn_Contract_Dem.",Capacity=999,ReplaceResOption=0,
  IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
#   Redlands Power Canal Irrigation (acres from 724713)
SetDiversionStation(ID="950050",Name="Redlands Power Canal-
Irr",Capacity=140.0,ReplaceResOption=0,IrrigatedAcres=4297,DemandSource=8,IfNotFound=Warn)
#
# The following are reservoir carrier structures
#   Elliott Creek Feeder - carrier to Green Mtn. Res
SetDiversionStation(ID="360606",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#   Wolcott Pumping Pipeline - carrier to Wolcott Res
SetDiversionStation(ID="371146",Capacity=500,ReplaceResOption=0,DemandSource=7,
  EffAnnual=0,IfNotFound=Warn)
#   CBT Willow Creek Feeder
SetDiversionStation(ID="510958",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#   Windy Gap Pump Pipeline Canal - carrier up to Shadow Mtn and Granby
SetDiversionStation(ID="514700",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#
# The following are project-specific diversions - demsrc and resreplace can vary
#   Silt Project
#   Grass Valley Canal
SetDiversionStation(ID="390563",ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,IfNotFound=Warn)
#   Silt Pump Canal - secondary structure in MS setup
SetDiversionStation(ID="390663",ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,
  EffAnnual=0,IfNotFound=Warn)
#   Dry Elk Valley Demands
SetDiversionStation(ID="950010",Name="Dry Elk Valley Irr",Capacity=45.0,
  IrrigatedAcres=2590,IfNotFound=Warn)
#   Irrigation Demands below Harvey Gap Reservoir - primary structure of MS setup
SetDiversionStation(ID="950011",Name="Farmers Irrigation
Comp",Capacity=72.0,IrrigatedAcres=2906,IfNotFound=Warn)
#
#   Collbran Project
#   Bonham Branch Pipeline
SetDiversionStation(ID="720542",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#   Cottonwood Branch Pipeline
SetDiversionStation(ID="720583",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#   Leon Park Feeder Canal
SetDiversionStation(ID="720746",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#   Park Creek Ditch (Vega)
SetDiversionStation(ID="720820",ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,IfNotFound=Warn)
#   Southside Canal
SetDiversionStation(ID="720879",ReplaceResOption=0,DemandSource=7,IfNotFound=Warn)
#

```

```

# Cameo Demand/Grand Valley Area EW - Why resreplace set to 1 for these structures?
# Grand Valley Irrigation Canal
SetDiversionStation(ID="720645",Capacity=650.0,ReplaceResOption=1,IfNotFound=Warn)
# Orchard Mesa Irrigation District - primary structure in MS Setup
SetDiversionStation(ID="720813",Capacity=461.0,ReplaceResOption=0,DemandSource=3,IfNotFound=Warn)
SetDiversionStation(ID="950004",Name="OMID Hydraulic Pump",Capacity=272.0,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
# Grand Valley Project
SetDiversionStation(ID="720646",Capacity=1620.0,IrrigatedAcres=0,DemandSource=7,IfNotFound=Warn)
SetDiversionStation(ID="950001",Name="Grand Valley
Project",Capacity=850.0,ReplaceResOption=0,IrrigatedAcres=28900,DemandSource=8,IfNotFound=Warn)
# Colorado River Pumping Plant - secondary source for OMID irrigation MS setup
SetDiversionStation(ID="721330",DemandSource=5,IfNotFound=Warn)
# USA Power Plant
SetDiversionStation(ID="950002",Name="USA Power Plant",Capacity=800.0,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
# Orchard Mesa Check
SetDiversionStation(ID="950003",Name="Orchard Mesa Check",Capacity=1072.0,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="950005",Name="OMID Pre-1985 Bypass",Capacity=1072.0,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="950006",Name="OMID Post-1985
Bypass",Capacity=1072.0,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
# 15-Mile Fish Requirement
SetDiversionStation(ID="952001",Name="15-Mile Fish
Requirement",Capacity=999,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
#
# The following structure is an aggregate M & I node -
# this node is included (despite zero demand) to maintain consistency with other basins and for
# potential future use.
SetDiversionStation(ID="72_AMC001",Name="72_AMC001 Colorado River nr Stateline",Capacity=999,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
#
# The following structures are used for a dataset-specific scenario
# Leonard Rice - 2 structures (Calculated and Baseline datasets only!)
SetDiversionStation(ID="950007",Name="USA PP-Winter-OM
Stip",Capacity=850.35,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="950008",Name="USA PP-Summer-OM
Stip",Capacity=850.35,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="950061",Name="Green Mtn Annual_Rep_Est.",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953001",Name="Ruedi Rnd 1-Muni Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953002",Name="Ruedi Rnd 1-Ind Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953003",Name="Ruedi Rnd 2-Muni Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953004",Name="Ruedi Rnd 2-Ind Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953005",Name="Ruedi Addl Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
# Woford Mtn Reservoir Demand (Baseline dataset only!)
SetDiversionStation(ID="953101",Name="Woford Fraser Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953102",Name="Woford MidPark Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="953103",Name="Woford Market Demand",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
# C1 - structures (Calculated and Baseline datasets only!)
SetDiversionStation(ID="956001",Name="Future Depletion #1",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
SetDiversionStation(ID="956002",Name="Future Depletion #2",Capacity=999,ReplaceResOption=0,
IrrigatedAcres=0,DemandSource=6,IfNotFound=Warn)
#
# Demand nodes to release excess HUP water from Homestake, Dillon, Williams Fork, and
# Woford Reservoirs
SetDiversionStation(ID="954516D",Name="HUP Release
Node",OnOff=1,Capacity=99999,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,
EffAnnual=0,IfNotFound=Warn)
SetDiversionStation(ID="954512D",Name="HUP Release
Node",OnOff=1,Capacity=99999,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,

```

```

EffAnnual=0,IfNotFound=Warn)
SetDiversionStation(ID="953709D",Name="HUP Release
Node",OnOff=1,Capacity=99999,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,
EffAnnual=0,IfNotFound=Warn)
SetDiversionStation(ID="953668D",Name="HUP Release
Node",OnOff=1,Capacity=99999,ReplaceResOption=0,IrrigatedAcres=0,DemandSource=7,
EffAnnual=0,IfNotFound=Warn)
#
# The following are structures that need alternate return location definitions
# 510848 - change return flow pattern to mimic portion of returns that occur in the same month
SetDiversionStation(ID="510848",Returns="510546,40,4",IfNotFound=Warn)
#
# StateDMI expects monthly values to be entered in Calendar Year.
#
# Step 7 - setting efficiencies for specific structures
# Acreage during the study period for the following 22 structures is different than what
# it is today (the value in HydroBase). Crop water requirements calculated by
# the CU Model are incorrect for the structures during the study period, but are correct
# for the baseline scenario.
# To avoid incorrect efficiencies being calculated by StateDMI (crop water requirement /
# historical diversion),
# we are setting the efficiencies for these structures equal to the basin-wide
# efficiency (3/9/99), ra
# Updated by James Heath (heath@lrcwe.com) with updated basin wide efficiencies (2/23/2006)
#
SetDiversionStation(ID="360687",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360725",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360728",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360729",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360765",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360780",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="360800",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="370519",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="370571",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="370723",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="370848",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="380528",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="380572",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="380663",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="380939",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="380996",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="381062",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
SetDiversionStation(ID="381078",EffMonthly="5,4,9,16,25,29,30,25,18,9,5,6",IfNotFound=Warn)
#
# Step 8 - create "step 1" direct diversion station file
#
WriteDiversionStationsToStateMod(OutputFile="cm2005_dds.dds")
#
# Check the results.
CheckDiversionStations(ID="*")
WriteCheckFile(OutputFile="dds.commands.StateDMI.check.html")

```

## 5.4.2 Diversion Rights

Diversion rights correspond to the diversion stations, using the diversion station identifier to relate the data. Diversion right identifiers are typically the diversion station identifier followed by .NN, where NN is a sequential number starting with 01. Rights for diversion aggregate/system stations have rights corresponding to water right classes.

The **Commands...Diversion Data...Diversion Rights** menu items insert commands to process diversion rights data:

Diversion Rights - Commands
ReadDiversionStationsFromList() ... ReadDiversionStationsFromStateMod() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ...
ReadDiversionRightsFromHydroBase() ... ReadDiversionRightsFromStateMod() ...
SetDiversionRight() ...
SortDiversionRights() ...
FillDiversionRight() ...
WriteDiversionRightsToList() ... WriteDiversionRightsToStateMod() ...
CheckDiversionRights() ... WriteCheckFile() ...

MenuCommands\_DiversionRights

### Commands...Diversion Data...Diversion Rights Menu

The following table summarizes the use of each command:

#### Diversion Rights Commands

Command	Description
ReadDiversionStationsFromList()	Read from a delimited file the list of diversion stations to be included in the data set – the list indicates the stations for which to process rights.
ReadDiversionStationsFromStateMod()	Read from a StateMod diversion stations file the list of diversion stations to be included in the data set – the list indicates the stations for which to process rights.
SetDiversionAggregate()	Specify that a diversion is an aggregate and define its parts.
SetDiversionAggregateFromList()	Specify that one or more diversions are aggregates and define their parts, using a delimited list file.
SetDiversionSystem()	Specify that a diversion is a system and define its parts.
SetDiversionSystemFromList()	Specify that one or more diversions are systems and define their parts, using a delimited list file.
ReadDiversionRightsFromHydroBase()	For each diversion station, read the corresponding diversion rights from HydroBase.
ReadDiversionRightsFromStateMod()	Read diversion rights from a StateMod diversion rights file.
SetDiversionRight()	Set the data for, and optionally add, diversion rights.

Command	Description
SortDiversionRights()	Sort the diversion rights. This is useful to force consistency between files.
FillDiversionRight()	Fill missing data for defined diversion rights, using user-supplied values.
WriteDiversionRightsToList()	Write defined diversion rights to a delimited file.
WriteDiversionRightsToStateMod()	Write defined diversion rights to a StateMod file.
CheckDiversionRights()	Check diversion rights data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the diversion rights file is shown below (from the Colorado cm2005 data set):

```

StartLog(LogFile="ddr.commands.StateDMI.log")
# ddr.commands.StateDMI
#
# StateDMI command file to create the direct diversion rights file for the Colorado model
#
# Step 1 - read structures from preliminary direct diversion station file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 2 - read aggregate and diversion system structure assignments. Note that
#         want to combine water rights for aggs and diversion systems, but
#         water rights are assigned to primary and secondary components of multistructures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
#
# Step 3 - read diversion rights from HydroBase and define water rights classes
#         used for aggregate structures - but NOT for diversion systems
#
ReadDiversionRightsFromHydroBase(ID="*",OnOffDefault=1,
    AdminNumClasses="14854.00000,20427.18999,22729.21241,
    30895.21241,31258.00000,32023.28989,39095.38998,43621.42906,46674.00000,48966.00000,99999.")
#
# Step 4 - set water rights for structure IDs different from or not included in HydroBase
#
# Grand Valley Area - many rights obtain water through operations
SetDiversionRight(ID="720646.02",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.03",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.05",Name="USA Power
Plant",StationID="ID",Decree=800.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.07",Name="Grand Valley
Proj",StationID="ID",AdministrationNumber=22729.19544,
    Decree=40.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.08",Name="USA_PP_Winter_OM-
Stip",StationID="ID",AdministrationNumber=30895.21241,Decree=800.00,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.09",Name="USA_PP_SummerSr_OM-
Stip",StationID="ID",AdministrationNumber=30895.21241,Decree=490,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.10",Name="USA_PP_SummerJr_OM-
Stip",StationID="ID",AdministrationNumber=100000.1000,Decree=999.00,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720813.01",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="950001.01",Name="Grand Valley Proj -
Irr",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950002.01",Name="USA Power Plant",StationID="ID",AdministrationNumber=99999.99999,
    Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950003.01",Name="Orchard Mesa
Check",StationID="ID",AdministrationNumber=999999.0000,Decree=640.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950004.01",Name="OMID Hydraulic
Pump",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950005.01",Name="OMID Pre-1985

```

```

Bypass",StationID="ID",AdministrationNumber=999998.0000,Decree=1100.0,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950006.01",Name="OMID Post-1985
Bypass",StationID="ID",AdministrationNumber=30895.23492,Decree=1100.0,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950007.01",Name="USA PP Winter OM-
Stip",StationID="ID",AdministrationNumber=99999.90009,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950008.01",Name="USA PP Summer OM-
Stip",StationID="ID",AdministrationNumber=100000.1000,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
... similar commands omitted...
#
#
# Municipal Water Rights
SetDiversionRight(ID="955002.01",Name="Snake R Water Dist Well
1",StationID="ID",AdministrationNumber=18181.00000,Decree=0.03,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="955002.02",Name="Snake R Water Dist Well
1",StationID="ID",AdministrationNumber=32075.25333,Decree=0.12,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="955002.03",Name="Snake R Water Dist Well
1",StationID="ID",AdministrationNumber=44741.00000,Decree=1.23,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="955001.01",Name="Vail Valley Water -
Irr",StationID="ID",AdministrationNumber=15646.00000,Decree=11.2,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="955003.01",Name="Vail Valley Water -
NonIrr",StationID="ID",AdministrationNumber=42420.41366,Decree=13.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950051.01",Name="City of Grand Jct",StationID="ID",AdministrationNumber=1.00000,
  Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.01",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=12753.00000,Decree=4.03,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.02",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=30895.12724,Decree=1.95,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.03",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=38895.24260,Decree=0.74,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.04",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=32811.00000,Decree=2.12,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.05",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=38847.00000,Decree=20.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.06",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=46751.46599,Decree=11.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.07",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=46995.00000,Decree=4.1,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950020.08",Name="Ute Water Treatment
Plant",StationID="ID",AdministrationNumber=41791.00000,Decree=15.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="720816.01",Name="Palisade Town
Pipeline",StationID="ID",AdministrationNumber=12797.00000,Decree=1.44,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="720816.02",Name="Palisade Town
Pipeline",StationID="ID",AdministrationNumber=14222.00000,Decree=3.55,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
...similar commands omitted...
#
# Industrial Water Rights
SetDiversionRight(ID="360989.01",Name="Maggie Pond
Snowmaking",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="361016.01",Name="Copper Mtn
Snowmaking",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,
  IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="720807.01",Name="Molina Power
Plant",StationID="ID",AdministrationNumber=99999.99999,Decree=999.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
# TenMile Diversion No. 1 - set diversion b/c it has been "Transferred From" in 1996 database
SetDiversionRight(ID="360841.01",Name="TenMile Diversion
No.1",StationID="ID",AdministrationNumber=31566.00000,Decree=35.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
#
# Redlands Power Canal and Irrigation rights (420541 has 3 rights of which only the first is modified,
# James Heath (heath@lrcwe.com))
SetDiversionRight(ID="420541.01",Name="Redlands Power
Canal",StationID="ID",AdministrationNumber=22283.20300,Decree=610.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="950050.01",Name="Redlands Power Canal-

```



```

Irr",StationID="ID",AdministrationNumber=22283.20300,Decree=60.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950050.02",Name="Redlands Power Canal-
Irr",StationID="ID",AdministrationNumber=34419.33414,Decree=80.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
#
# Silt Project default water rights - water obtained through operations
SetDiversionRight(ID="950010.01",Name="Dry Elk Valley Irr",StationID="ID",
AdministrationNumber=99999.99999,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="950011.01",Name="Farmers Irrigation
Comp",StationID="ID",AdministrationNumber=99999.99999,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
#
# 15-Mile Reach - LR-2
SetDiversionRight(ID="952001.01",Name="15-Mile Fish
Require",StationID="ID",AdministrationNumber=99999.91000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
#
# Excess HUP Releases from Homestake, Dillon, Williams Fork, and WOLFORD Reservoirs Water Rights
SetDiversionRight(ID="954516D.01",Name="HUP Release Node",StationID="ID",
AdministrationNumber=99999.99999,Decree=0.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="954512D.01",Name="HUP Release Node",StationID="ID",
AdministrationNumber=99999.99999,Decree=0.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="953709D.01",Name="HUP Release Node",StationID="ID",
AdministrationNumber=99999.99999,Decree=0.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="953668D.01",Name="HUP Release Node",StationID="ID",
AdministrationNumber=99999.99999,Decree=0.0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
#
# WOLFORD MOUNTAIN RESERVOIR DEMAND
SetDiversionRight(ID="953101.01",Name="Wolford_Fraser_Dem",StationID="ID",
AdministrationNumber=99999.00000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="953102.01",Name="Wolford_MidPark_Dem",StationID="ID",
AdministrationNumber=99999.00000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="953103.01",Name="Wolford_Market_Dem",StationID="ID",
AdministrationNumber=99999.00000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
...similar commands omitted...
# FUTURE DEPLETIONS
SetDiversionRight(ID="956001.01",Name="Future_Depletion_#1",StationID="ID",
AdministrationNumber=99999.00000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetDiversionRight(ID="956002.01",Name="Future_Depletion_#2",StationID="ID",
AdministrationNumber=99999.00000,Decree=0,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
#
# Cliff Ditch - both water rights reside under WDID 500539 - set 12 cfs of second water right to 500731
# and reduce to 12 cfs at 500539 - this water right serves both 500539 & 500731
SetDiversionRight(ID="500731.01",Name="Cliff Ditch Hdg No
2",StationID="ID",AdministrationNumber=20676.19665,Decree=12.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="500539.02",Name="Cliff
Ditch",StationID="ID",AdministrationNumber=20676.19665,Decree=12.0,OnOff=1,IfNotFound=Add,IfFound=Set)
#
#
# Step 5 - Add Free water rights for structures historically diverting more than water rights
# Example from San Juan - replace section when we get a list of free river water rights
SetDiversionRight(ID="360662.99",Name="HOAGLAND CANAL
SPRUCE",StationID="360662",AdministrationNumber=99999.99999,Decree=999.00,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="360729.99",Name="MAT NO 2
DITCH",StationID="360729",AdministrationNumber=99999.99999,Decree=999.00,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="360734.99",Name="MCKAY
DITCH",StationID="360734",AdministrationNumber=99999.99999,Decree=999.00,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="360765.99",Name="PALMER-MCKINLEY
DITCH",StationID="360765",AdministrationNumber=99999.99999,Decree=999.00,IfNotFound=Add,IfFound=Set)
...similar commands omitted...
#
# Step 6 - add municipal aggregate rights - this agg node water right is set to zero as no
# M&I uses need to be aggregated and accounted for.
# the node is included to maintain consistency with other basins and for potential future use

```

```
#
SetDiversiionRight(ID="72_AMC001.01",Name="72_AMC001 Colorado River nr
Stateline",StationID="ID",AdministrationNumber=1.00000,Decree=0.0,IfNotFound=Add,IfFound=Set)
#
# Step 7 - create direct diverison rights file
#
WriteDiversiionRightsToStateMod(OutputFile="..\STATEMOD\cm2005.ddd")
#
# Check the results
CheckDiversiionRights(ID="*")
WriteCheckFile(OutputFile="ddd.commands.StateDMI.check.html")
```

### 5.4.3 Diversion Historical Time Series (Monthly)

Diversion historical time series (monthly) correspond to each diversion station, using the station identifier to relate the data.

The **Commands...Diversion Data...Diversion Historical TS (Monthly)** menus insert commands to process diversion historical time series (monthly) data (and also update the diversion stations file because of changes to the capacity data):

Diversion Historical TS (Monthly) - Commands
SetOutputPeriod() ... SetOutputYearType() ...
ReadDiversionStationsFromList() ... ReadDiversionStationsFromStateMod() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ...
ReadDiversionHistoricalTSMonthlyFromHydroBase() ... ReadDiversionHistoricalTSMonthlyFromStateMod() ...
SetDiversionHistoricalTSMonthly() ... SetDiversionHistoricalTSMonthlyConstant() ...
FillDiversionHistoricalTSMonthlyAverage() ... FillDiversionHistoricalTSMonthlyConstant() ... 1: ReadPatternFile() ... 2: FillDiversionHistoricalTSMonthlyPattern() ...
1: ReadDiversionRightsFromStateMod() ... 2: LimitDiversionHistoricalTSMonthlyToRights() ...
SortDiversionHistoricalTSMonthly() ... WriteDiversionHistoricalTSMonthlyToStateMod() ...
SetDiversionStationCapacitiesFromTS() ... SetDiversionStation() ... SetDiversionStationsFromList() ... WriteDiversionStationsToStateMod() ...
CheckDiversionHistoricalTSMonthly() ... WriteCheckFile() ...

MenuCommands\_DiversionHistoricalTSMonthly

### Commands...Diversion Data...Diversion Historical TS (Monthly) Menu

The following table summarizes the use of each command:

### Diversión Historical Time Series (Monthly) Commands

Command	Description
SetOutputPeriod()	Set the output period. Time series are automatically extended to this period if necessary.
SetOutputYearType()	Set the output year type, which is used when writing the files and for monthly efficiency data order.
ReadDiversiónStationsFromList()	Read from a delimited file the list of diversion stations to be included in the data set.
ReadDiversiónStationsFromStateMod()	Read from a StateMod diversion stations file the list of diversion stations to be included in the data set.
SetDiversiónAggregate()	Specify that a diversion is an aggregate and define its parts.
SetDiversiónAggregateFromList()	Specify that one or more diversions are aggregates and define their parts, using a delimited list file.
SetDiversiónSystem()	Specify that a diversion is a system and define its parts.
SetDiversiónSystemFromList()	Specify that one or more diversions are systems and define their parts, using a delimited list file.
ReadDiversiónHistoricalTSMonthlyFromHydroBase()	Read diversion historical time series (monthly) from HydroBase, filling and adding aggregate/system part time series if necessary.
ReadDiversiónHistoricalTSMonthlyFromStateMod()	Read diversion historical time series (monthly) from a StateMod file.
SetDiversiónHistoricalTSMonthly()	Set the data for a diversion historical time series (monthly) by reading another time series (e.g., from a file). This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
SetDiversiónHistoricalTSMonthlyConstant()	Set the data for a diversion historical time series (monthly) to a constant value. This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
FillDiversiónHistoricalTSMonthlyAverage()	Fill missing data in diversion historical time series (monthly) to the historical monthly average values. If an aggregate/system, the historical average is computed from the total.
FillDiversiónHistoricalTSMonthlyConstant()	Fill missing data in diversion historical time series (monthly) to a constant value.
ReadPatternFile()	Read the pattern file used with FillDiversiónHistoricalTSMonthlyPattern() commands.
FillDiversiónHistoricalTSMonthlyPattern()	Fill missing data in diversion historical time series (monthly) to the historical monthly average values, using wet/dry/average values.
ReadDiversiónRightsFromStateMod()	Read the diversion rights file for use with the LimitDiversiónHistoricalTSMonthlyToRights() command.

Command	Description
LimitDiversionHistoricalTSMonthlyToRights()	Limit the diversion historical time series (monthly) to the water rights that were available at each point in time.
SortDiversionHistoricalTSMonthly()	Sort the diversion historical time series (monthly). This is useful to force consistency between files.
WriteDiversionHistoricalTSMonthlyToStateMod()	Write defined diversion historical time series (monthly) to a StateMod file.
SetDiversionStationCapacitiesFromTS()	Set the diversion station capacities to the maximum historical time series value.
SetDiversionStation()	Set diversion station information (e.g., to override capacity changes from the previous step).
SetDiversionStationsFromList()	Set diversion station information from a delimited file (e.g., to override capacity changes from the previous step).
WriteDiversionStationsToStateMod()	Write diversion stations data to a StateMod diversion stations file (use if the capacities have been updated).
CheckDiversionHistoricalTSMonthly()	Check diversion historical monthly time series data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the diversion historical time series (monthly) file is shown below (from the Colorado cm2005 data set). Note that aggregate part time series are filled before being added to the total for the aggregate station, and explicit diversion time series are filled separately after reading.

```

StartLog(LogFile="ddh.commands.StateDMI.log")
# ddh.commands.StateDMI
#
# StateDMI command file to create the historical diversion file
# and the "step 2" direct diversion structure file, updated so structure
# capacity = maximum historical diversion
#
# Step 1 - set time-series period and year type
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read structure list from preliminary direct diversion structure file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 3 - read aggregate and diversion system structure assignments. Note that
# want to combine historical diversions for aggs and diversion systems, but
# historical diversions are separate for primary and secondary components
# of multistructures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
PartsListedHow=InRow)
#
# Step 4 - read historical diversions from HydroBase. Note that want individual structures
# in aggregates and diversion systems to be filled first, then diversions combined.
#
ReadDiversionHistoricalTSMonthlyFromHydroBase(ID="*",IncludeCollections=False,
UseDiversionComments=True)
#
# Step 5 - read fill pattern file, and assign patterns to water districts
#
ReadPatternFile(InputFile="fill2005.pat")

```

```

ReadDiversiOnHistoricalTSMonthlyFromHydroBase(ID="36*",IncludeExplicit=False,
  UseDiversiOnComments=True,
  PatternID="09037500",FillPatternOrder=1,FillAverageOrder=2)
#
# Step 6 - assign transbasin diversions from streamflow gages
#
SetDiversiOnHistoricalTSMonthly(ID="364626",TSID="09047300.DWR.Streamflow.Month~HydroBase")
...similar commands omitted...
# note that adams tunnel streamgage ID changed in 10/1996 from 09013000 to ADANETCO
SetDiversiOnHistoricalTSMonthly(ID="514634",TSID="514634...MONTH~StateMod~514634.stm")
# Con-Hoosier System - Blue River Diversion, driven by operating rules to con-hoosier
summary demand
SetDiversiOnHistoricalTSMonthly(ID="364683",TSID="364683...MONTH~StateMod~zero.stm")
SetDiversiOnHistoricalTSMonthly(ID="364699",TSID="364699...MONTH~StateMod~zero.stm")
# Fryingpan-Arkansas Project
SetDiversiOnHistoricalTSMonthly(ID="381594",TSID="381594...MONTH~StateMod~381594.stm")
SetDiversiOnHistoricalTSMonthly(ID="384625",TSID="384625...MONTH~StateMod~384625.stm")
SetDiversiOnHistoricalTSMonthly(ID="954699",TSID="954699...MONTH~StateMod~zero.stm")
...similar commands omitted...
#
# Step 7 - set diversions from external time-series files
#
# The following commands are added to access Task 11.2 replacement files
SetDiversiOnHistoricalTSMonthly(ID="380757",TSID="380757...MONTH~StateMod~380757.stm")
...similar commands omitted...#
# The following structures are set for Municipal and Industrial Diversions
SetDiversiOnHistoricalTSMonthly(ID="360784",TSID="360784...MONTH~StateMod~360784.stm")
...similar commands omitted...
#
# Set transbasin diversions to "0" prior to construction
#
# Wurtz Ditch
SetDiversiOnHistoricalTSMonthlyConstant(ID="374648",Constant=0,SetEnd="01/1929")
...similar commands omitted...
#
# Step 8 - fill historical diversion using pattern approach
#
FillDiversiOnHistoricalTSMonthlyPattern(ID="36*",PatternID="09034500")
...similar commands omitted...
#
# Step 9 - Fill remaining missing with month average
#
FillDiversiOnHistoricalTSMonthlyAverage(ID="*")
#
# Step 10 - Limit filled diversion to water rights. Exceptions include structure
# receiving significant reservoir supply, carrier structures, etc.
#
LimitDiversiOnHistoricalTSMonthlyToRights(InputFile="..\statemod\cm2005.ddr",
  ID="*",IgnoreID="954683,952001,950010,950011")
#
# Step 11 - sort structures and create historical diversion file
#
SortDiversiOnHistoricalTSMonthly(Order=Ascending)
WriteDiversiOnHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
#
# Step 12 - update capacities and create final direct diversion station file
#
SetDiversiOnStationCapacitiesFromTS(ID="*")
WriteDiversiOnStationsToStateMod(OutputFile="..\statemod\cm2005.dds")
#
# Check the results.
CheckDiversiOnHistoricalTSMonthly(ID="*")
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")

```

#### 5.4.4 Diversion Historical Time Series (Daily)

StateDMI does not process daily diversion historical time series. TSTool, a spreadsheet, or other software can be used to create the data. More commonly, the monthly demand data can be distributed to daily time series internally by StateMod by specifying the appropriate daily station identifier.

#### 5.4.5 Diversion Demand Time Series (Monthly)

Diversion demand time series (monthly) correspond to each diversion station, using the station identifier to relate the data. Current modeling practices use variable monthly efficiency, computed by StateCU. Average monthly efficiencies are also typically set from StateCU results (as a list file input when defining diversion stations) but can also be computed by dividing irrigation water requirement time series from StateCU by historical diversion time series. In CDSS, demands are typically computed for three different data sets, as follows:

##### ***Historical Demand***

Filled demands are limited by the water rights on-line at the time. Historical measured diversions are not limited. Free water rights are assumed to either be on for the entire period, or beginning with the earliest water right. In this case the demands are the same as the historical diversions, typically just a copy of the historical diversions time series file. This approach can be accomplished by using the `LimitDiversionHistoricalTSMonthlyToRights()` command.

##### ***Calculated Demand***

Irrigation demands are calculated based on  $IWR/Eff_{ave}$ . The entire period is limited by water rights on-line at the time. Free water rights are assumed to either be on for the entire period, or beginning with the earliest water right. This approach can be accomplished by using the `LimitDiversionDemandTSMonthlyToRights()` command.

##### ***Baseline Calculated Demands***

Demands are treated the same as for the calculated demand case. However, the entire period is limited by the current water rights (to simulate current conditions). This approach can be accomplished by using the `LimitDiversionDemandTSMonthlyToRights(..., LimitToCurrent=True, ...)` command.

##### ***Special Considerations – Conditional Rights***

Conditional water rights may be included in StateMod rights files and be turned off for the historical demands by setting the rights switch to 0 (zero) in the historical data files. Conditional rights, if considered in the Baseline data set, can be turned on. This requires that a different rights file be used with the calculated data set files.

##### ***Special Considerations – Comparing Calculated and Historical Demands***

For some data sets, it may be appropriate to use the `CalculateDiversionDemandTSMonthlyAsMax()` command to set the diversion demands to the maximum of calculated and historical demands. Using this approach can improve calibrations, for example:

- If the demand equals the historical value, then the diversion station at times operates at a significantly lower efficiency than the average efficiency.

- If the demand equals  $IWR/Eff_{ave}$ , then the diversion station may be water short and will try to divert at least enough water to operate at an average efficiency.

Modelers should consider the above issues when deciding how to prepare data for a particular data set.

The **Commands...Diversion Data...Diversion Demand TS (Monthly)** menus insert commands to process diversion demand time series (monthly) data (and optionally the diversion stations, to save estimated efficiencies):

Diversion Demand TS (Monthly) - Commands
SetOutputPeriod() ... SetOutputYearType() ...
ReadDiversionStationsFromList() ... ReadDiversionStationsFromStateMod() ...
SetDiversionAggregate() ... SetDiversionAggregateFromList() ... SetDiversionSystem() ... SetDiversionSystemFromList() ... SetDiversionMultiStruct()... SetDiversionMultiStructFromList()...
1: ReadIrrigationWaterRequirementTSMonthlyFromStateCU() ... 2: ReadDiversionHistoricalTSMonthlyFromStateMod() ... [Legacy] 3: CalculateDiversionStationEfficiencies() ... SetDiversionStation() ... SetDiversionStationsFromList() ... WriteDiversionStationsToStateMod() ...
CalculateDiversionDemandTSMonthly() ... CalculateDiversionDemandTSMonthlyAsMax() ... ReadDiversionDemandTSMonthlyFromStateMod() ...
FillDiversionDemandTSMonthlyAverage() ... FillDiversionDemandTSMonthlyConstant() ... 1: ReadPatternFile() ... 2: FillDiversionDemandTSMonthlyPattern() ...
LimitDiversionDemandTSMonthlyToRights() ... SetDiversionDemandTSMonthly() ... SetDiversionDemandTSMonthlyConstant() ...
SortDiversionDemandTSMonthly() ... WriteDiversionDemandTSMonthlyToStateMod() ...
CheckDiversionDemandTSMonthly() ... WriteCheckFile() ...

MenuCommands\_DiversionDemandTSMonthly

### Commands...Diversion Data...Diversion Demand TS (Monthly) Menu



The following table summarizes the use of each command:

### Diversion Demand Time Series (Monthly) Commands

Command	Description
SetOutputPeriod()	Set the output period. Time series are automatically extended to this period if necessary.
SetOutputYearType()	Set the output year type, which is used when writing the files and for determining the monthly efficiency order in station data.
ReadDiversionStationsFromList()	Read from a delimited file the list of diversion stations to be included in the data set.
ReadDiversionStationsFromStateMod()	Read from a StateMod diversion stations file the list of diversion stations to be included in the data set.
SetDiversionAggregate()	Specify that a diversion is an aggregate and define its parts.
SetDiversionAggregateFromList()	Specify that one or more diversions are aggregates and define their parts, using a delimited list file.
SetDiversionSystem()	Specify that a diversion is a system and define its parts.
SetDiversionSystemFromList()	Specify that one or more diversions are systems and define their parts, using a delimited list file.
SetDiversionMultiStruct()	Specify that a diversion is a “MultiStruct” and define its parts.
SetDiversionMultiStructFromList()	Specify that one or more diversions are “MultiStruct”’s and define their parts, using a delimited list file.
ReadIrrigationWaterRequirementTSMonthlyFromStateCU()	Read irrigation water requirement (IWR) time series generated by the StateCU model.
ReadDiversionHistoricalTSMonthlyFromStateMod()	Read diversion historical time series (monthly) from a StateMod file.
CalculateDiversionStationEfficiencies()	Calculate diversion station average monthly efficiencies as IWR/Diversions.
SetDiversionStation()	Set diversion station data, in particular efficiency data to override the result from the previous command.
SetDiversionStationsFromList()	Set diversion station information from a delimited file (e.g., to override capacity changes from the previous step).
WriteDiversionStationsToStateMod()	Write diversion stations to StateMod – the data will include updated average efficiencies.
CalculateDiversionDemandTSMonthly()	Calculate the diversion demand time series (monthly) using IWR/Eff <sub>ave</sub> and historical diversion time series.

Command	Description
<code>CalculateDiversionDemandTSMonthlyAsMax()</code>	Calculate the diversion demand time series (monthly) as the maximum of the demand (see previous command) and the diversion historical time series.
<code>ReadDiversionDemandTSMonthlyFromStateMod()</code>	Read the diversion demand time series (monthly) from a StateMod file, if a previous result is being modified.
<code>FillDiversionDemandTSMonthlyAverage()</code>	Fill missing data in diversion demand time series (monthly) to the monthly average values. If an aggregate/system, the average is computed from the total.
<code>FillDiversionDemandTSMonthlyConstant()</code>	Fill missing data in diversion demand time series (monthly) to a constant value.
<code>ReadPatternFile()</code>	Read the pattern file used with <code>FillDiversionDemandTSMonthlyPattern()</code> commands.
<code>FillDiversionDemandTSMonthlyPattern()</code>	Fill missing data in diversion demand time series (monthly) to the monthly average values, using wet/dry/average values.
<code>LimitDiversionDemandTSMonthlyToRights()</code>	Limit the diversion demand time series (monthly) to the water rights that were available at each point in time.
<code>SetDiversionDemandTSMonthly()</code>	Set the data for a diversion demand time series (monthly). This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set). Use this after the other commands to ensure that values will remain set.
<code>SetDiversionDemandTSMonthlyConstant()</code>	Set the data for a diversion demand time series (monthly) to a constant value. This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
<code>SortDiversionDemandTSMonthly()</code>	Sort the diversion demand time series (monthly). This is useful to force consistency between files.
<code>WriteDiversionDemandTSMonthlyToStateMod()</code>	Write diversion demand time series (monthly) to a StateMod file.
<code>CheckDiversionDemandTSMonthly()</code>	Check diversion demand monthly time series data for problems.
<code>WriteCheckFile()</code>	Write the results of data checks to a file.

An example command file to create the diversion demand time series (monthly) file for the historical case is shown below (adapted from Colorado cm2005 data set):

```
StartLog(LogFile="Hddm.commands.StateDMI.log")
# Hddm.commands.StateDMI - Creates Upper Colorado River Historical Demand file
#
# Step 1 - set the output period, used to compute averages...
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="9/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read historical diversion file as demand - defined structures for *.ddm file
#
ReadDiversionDemandTSMonthlyFromStateMod(InputFile="..\statemod\cm2005.ddh")
#
# Step 3 - override specific demands with time series...
#
SetDiversionDemandTSMonthly(ID="720807",TSID="720807..DivTotal.Month~StateMod~720807.stm")
# Set carrier structures to zero
SetDiversionDemandTSMonthlyConstant(ID="360606",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720542",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720583",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720746",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720820",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720879",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="724721",Constant=0)
# Set GVP specific demands
SetDiversionDemandTSMonthlyConstant(ID="950003",Constant=100000,SetStart="11/1926")
SetDiversionDemandTSMonthlyConstant(ID="950005",Constant=60000,SetEnd="9/1984")
SetDiversionDemandTSMonthlyConstant(ID="950006",Constant=60000,SetStart="10/1984")
# Set Excess HUP node demands for Homestake, Dillon, Williams Fork, and Wolford Reservoirs
SetDiversionDemandTSMonthlyConstant(ID="954516D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="954512D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="953709D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="953668D",Constant=999999)
#
# Step 4 - write the time series to the StateMod file...
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\statemod\cm2005H.ddm")
#
# Check the results.
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Hddm.commands.StatedDMI.check.html")
```

The following example illustrates how to create the calculated data set diversion demand time series (from the Colorado cm2005 data set):

```
StartLog(LogFile="Cddm.commands.StateDMI.log")
# Cddm.commands.StateDMI
#
# StateDMI command file to create the Calculated demand file
#
# Step 1 - set the output period, used to compute averages...
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read historical diversion file -defines structures for *.ddm file
#           plus read *.ddh file
#
ReadDiversionStationsFromStateMod(InputFile="..\StateMod\cm2005.dds")
ReadDiversionHistoricalTSMonthlyFromStateMod(InputFile="..\StateMod\cm2005.ddh")
#
# Step 3 - read StateCU *.iwr and *.def files (irrigation requirements and average efficiencies)
#
ReadIrrigationWaterRequirementTSMonthlyFromStateCU(InputFile="..\StateMod\cm2005.iwr")
```

```

# CalculateDiversiOnStationEfficiencies(ID="",EffMin=0,EffMax=60,EffCalcStart=10/1974,
  EffCalcEnd=9/2004,LEZeroInAverage=False)
SetDiversiOnStationsFromList(ListFile="cm2005.def",IDCol="1",EffMonthlyCol="2",Delim="Space",
  MergeDelim=True)
#
# Step 4 - determine calculated demand =iwr/efficiency
#         - take max of calculated demand and historical diversion
#
CalculateDiversiOnDemandTSMonthly(ID="")
CalculateDiversiOnDemandTSMonthlyAsMax(ID="")
#
# Step 5 - set carriers nodes demand to 0, set full demand and summary demand nodes
#
# set carrier "transbasin" diversion to Divide Creek to "0", use operating rules to satisfy demand
SetDiversiOnDemandTSMonthlyConstant(ID="724721",Constant=0)
# place summary demand at the Moffat Tunnel, zero out collection points
SetDiversiOnDemandTSMonthly(ID="514655",TSID="514655..DivTotal.Month~StateMod~514655.stm")
SetDiversiOnDemandTSMonthlyConstant(ID="510639",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="510728",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="511269",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="511309",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="511310",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="514603",Constant=0)
# place summary demand at the Boustead Summary node, zero out collection points
SetDiversiOnDemandTSMonthly(ID="954699",TSID="954699..DivTotal.Month~StateMod~954699.stm")
SetDiversiOnDemandTSMonthlyConstant(ID="381594",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="384625",Constant=0)
# Homestake - Zero Missouri Tunnel and drive by Homestake Reservoir Demend
SetDiversiOnDemandTSMonthlyConstant(ID="374643",Constant=0)
# Collbran Project Feeder/Supply Canals
SetDiversiOnDemandTSMonthlyConstant(ID="720879",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="720820",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="720746",Constant=0)
# Grand Valley Project Carrier (Roller Dam)
SetDiversiOnDemandTSMonthlyConstant(ID="720646",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="950001",Constant=0,SetEnd="09/1915")
# Molina Power Project
SetDiversiOnDemandTSMonthlyConstant(ID="720583",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="720542",Constant=0)
SetDiversiOnDemandTSMonthly(ID="720807",
  TSID="720807..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
# Silt Project / Grass Valley / Rifle Gap
SetDiversiOnDemandTSMonthlyConstant(ID="390663",Constant=0)
SetDiversiOnDemandTSMonthlyConstant(ID="390563",Constant=0)
# Elliot Feeder to Green Mountain Res
SetDiversiOnDemandTSMonthlyConstant(ID="360606",Constant=0)
#
# set demands for OMID Multi Structure - need to change demand calculation for 720813 in the future
# when 721330 is operational. At that point an stm file will need to be created with the total
diversions
# of structures 720813 and 721330.
SetDiversiOnDemandTSMonthlyConstant(ID="721330",Constant=0)
SetDiversiOnDemandTSMonthly(ID="720813",TSID="720813..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
#
# Step 6 - set calculated demand to historic for structures whose historical acreage is different
#         from current
#
SetDiversiOnDemandTSMonthly(ID="360687",TSID="360687..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360725",TSID="360725..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360728",TSID="360728..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360729",TSID="360729..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360765",TSID="360765..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360780",TSID="360780..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="360800",TSID="360800..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="370519",TSID="370519..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="370571",TSID="370571..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="370723",TSID="370723..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="370848",TSID="370848..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="380528",TSID="380528..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="380572",TSID="380572..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")
SetDiversiOnDemandTSMonthly(ID="380663",TSID="380663..DivTotal.MONTH~StateMod~..\\StateMod\\cm2005H.ddm")

```

```

SetDiversionDemandTSMonthly(ID="380939",TSID="380939..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="380996",TSID="380996..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="381062",TSID="381062..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="381078",TSID="381078..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="950005",TSID="950005..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="950006",TSID="950006..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
#
# Set Ute WCD demand node structure and set other structures to zero
SetDiversionDemandTSMonthly(ID="950020",TSID="950020..DivTotal.Month~StateMod~950020.stm")
SetDiversionDemandTSMonthlyConstant(ID="950030",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="721339",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="720920",Constant=0)
SetDiversionDemandTSMonthlyConstant(ID="721329",Constant=0)
#
# Set Orchard Mesa Check
SetDiversionDemandTSMonthly(ID="950003",TSID="950003..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
#
# Set Excess HUP node demands for Homestake, Dillon, Williams Fork, and Wolford Reservoirs
SetDiversionDemandTSMonthlyConstant(ID="954516D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="954512D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="953709D",Constant=999999)
SetDiversionDemandTSMonthlyConstant(ID="953668D",Constant=999999)
# Step 7 - write out calculated demand file
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005C.ddm")
#
# Check the results
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Cddm.commands.StateDMI.check.html")

```

The following example illustrates how to create the baseline data set diversion demand time series (from the Colorado cm2005 data set):

```

StartLog(LogFile="Bddm.commands.StateDMI.log")
# Bddm.commands.StateDMI
#
# StateDMI command file to create the Baseline demand file
#
# Step 1 - set time-series period and year type
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read calculated demand file
#
ReadDiversionDemandTSMonthlyFromStateMod(InputFile="..\statemod\cm2005C.ddm")
#
# Step 3 - set baseline demand that vary from calculated demand
#
# TRANSBASIN DIVERSIONS
#
# Con-Hoosier Transbasin Demands
SetDiversionDemandTSMonthly(ID="954683",TSID="954683..DivTotal.MONTH~StateMod~954683_baseline.stm")
# Boustead Transbasin Demands
SetDiversionDemandTSMonthly(ID="954699",TSID="954699..DivTotal.MONTH~StateMod~954699_baseline.stm")
# Moffat Transbasin Demands
SetDiversionDemandTSMonthly(ID="514655",TSID="514655..DivTotal.MONTH~StateMod~514655_baseline.stm")
# Vidler Tunnel
SetDiversionDemandTSMonthly(ID="364626",TSID="364626..DivTotal.MONTH~StateMod~364626_baseline.stm")
# Robert's Tunnel
SetDiversionDemandTSMonthly(ID="364684",TSID="364684..DivTotal.MONTH~StateMod~364684_baseline.stm")
# BOREAS PASS DITCH
SetDiversionDemandTSMonthly(ID="364685",TSID="364685..DivTotal.MONTH~StateMod~364685_baseline.stm")
# EWING DITCH AT TENNESSEE PASS, CO.
SetDiversionDemandTSMonthly(ID="371091",TSID="371091..DivTotal.MONTH~StateMod~371091_baseline.stm")
# HOMESTAKE PROJ TUNNEL
SetDiversionDemandTSMonthly(ID="374614",TSID="374614..DivTotal.MONTH~StateMod~374614_baseline.stm")
# COLUMBINE DITCH

```

```

SetDiversionDemandTSMonthly(ID="374641",TSID="374641..DivTotal.MONTH~StateMod~374641_baseline.stm")
# WARREN E WURTS DITCH
SetDiversionDemandTSMonthly(ID="374648",TSID="374648..DivTotal.MONTH~StateMod~374648_baseline.stm")
# BUSK - IVANHOE TUNNEL
SetDiversionDemandTSMonthly(ID="384613",TSID="384613..DivTotal.MONTH~StateMod~384613_baseline.stm")
# INDEPENDENCE PASS TM DVR TUNNEL NO 1
SetDiversionDemandTSMonthly(ID="384617",TSID="384617..DivTotal.MONTH~StateMod~384617_baseline.stm")
# BERTHOUD CANAL TUNNEL
SetDiversionDemandTSMonthly(ID="514625",TSID="514625..DivTotal.MONTH~StateMod~514625_baseline.stm")
# ADAMS TUNNEL
SetDiversionDemandTSMonthly(ID="514634",TSID="514634..DivTotal.MONTH~StateMod~514634_baseline.stm")
# WILLOW CREEK FEEDER
SetDiversionDemandTSMonthly(ID="510958",TSID="510958..DivTotal.MONTH~StateMod~510958_baseline.stm")
# WINDY GAP PUMP
SetDiversionDemandTSMonthly(ID="514700",TSID="514700..DivTotal.MONTH~StateMod~514700_baseline.stm")
# WEST THREE MILE DITCH
SetDiversionDemandTSMonthly(ID="384717",TSID="384717..DivTotal.MONTH~StateMod~384717_baseline.stm")
#
# MUNICIPAL AND INDUSTRIAL
#
# DILLON_VALLEY_W&SD_(RANKIN_NO._1_DITCH)
SetDiversionDemandTSMonthly(ID="360784",TSID="360784..DivTotal.MONTH~StateMod~360784_baseline.stm")
# TOWN_OF_DILLON_(Straight_Creek_Ditch)
SetDiversionDemandTSMonthly(ID="360829",TSID="360829..DivTotal.MONTH~StateMod~360829_baseline.stm")
# TENMILE DIVERSION NO 1 (Climax)
SetDiversionDemandTSMonthlyConstant(ID="360841",Constant=0)
# Keystone Resort Snowmaking
SetDiversionDemandTSMonthly(ID="360908",TSID="360908..DivTotal.MONTH~StateMod~360908_baseline.stm")
# TOWN_OF_BRECKENRIDGE_(Breckenridge_Pipeline)
SetDiversionDemandTSMonthly(ID="361008",TSID="361008..DivTotal.MONTH~StateMod~361008_baseline.stm")
# COPPER_MOUNTAIN_SKI_AREA_SNOWMAKING
SetDiversionDemandTSMonthly(ID="361016",TSID="361016..DivTotal.MONTH~StateMod~361016_baseline.stm")
# UPPER_EAGLE_VALLEY_WATER_AUTHORITY_(Metcalf_Ditch)
SetDiversionDemandTSMonthly(ID="370708",TSID="370708..DivTotal.MONTH~StateMod~370708_baseline.stm")
# CARBONDALE WTR SYS & PL
SetDiversionDemandTSMonthly(ID="381052",TSID="381052..DivTotal.MONTH~StateMod~381052_baseline.stm")
# Snowmass Water and Utility
SetDiversionDemandTSMonthly(ID="381441",TSID="381441..DivTotal.MONTH~StateMod~381441_baseline.stm")
# RIFLE TOWN OF PUMP & PL
SetDiversionDemandTSMonthly(ID="390967",TSID="390967..DivTotal.MONTH~StateMod~390967_baseline.stm")
# GRAND JCT GUNNISON P-L
SetDiversionDemandTSMonthly(ID="420520",TSID="420520..DivTotal.MONTH~StateMod~420520_baseline.stm")
# REDLANDS POWER CANAL
SetDiversionDemandTSMonthly(ID="420541",TSID="420541..DivTotal.MONTH~StateMod~420541_baseline.stm")
# HENDERSON MINE WTR SYS
SetDiversionDemandTSMonthly(ID="511070",TSID="511070..DivTotal.MONTH~StateMod~511070_baseline.stm")
# SHOSHONE POWER PLANT
SetDiversionDemandTSMonthly(ID="530584",TSID="530584..DivTotal.MONTH~StateMod~530584_baseline.stm")
# GLENWOOD L WATER CO SYS
SetDiversionDemandTSMonthly(ID="530585",TSID="530585..DivTotal.MONTH~StateMod~530585_baseline.stm")
# TOWN_OF_CLIFTON_(Grand_Junction_Colorado_River_PL)
SetDiversionDemandTSMonthly(ID="720644",TSID="720644..DivTotal.MONTH~StateMod~720644_baseline.stm")
# MOLINA POWER PLANT
SetDiversionDemandTSMonthly(ID="720807",TSID="720807..DivTotal.MONTH~StateMod~720807_baseline.stm")
# PALISADE_TOWN_PIPELINE_(720816)_(TREATED_PLANT_FLOW)
SetDiversionDemandTSMonthly(ID="720816",TSID="720816..DivTotal.MONTH~StateMod~720816_baseline.stm")
# Ute Water Treatment
SetDiversionDemandTSMonthly(ID="950020",TSID="950020..DivTotal.MONTH~StateMod~950020_baseline.stm")
# CITY_OF_GRAND_JUNCTION
SetDiversionDemandTSMonthly(ID="950051",TSID="950051..DivTotal.MONTH~StateMod~950051_baseline.stm")
# VAIL VALLEY CONSOLIDATED WATER DISTRICT - irr. season
SetDiversionDemandTSMonthly(ID="955001",TSID="955001..DivTotal.MONTH~StateMod~955001_baseline.stm")
# TOTAL FOR ALL SNAKE RIVER WATER DISTRICT WELLS (KEYSTONE MUNICIPAL)
SetDiversionDemandTSMonthly(ID="955002",TSID="955002..DivTotal.MONTH~StateMod~955002_baseline.stm")
# VAIL VALLEY CONSOLIDATED WATER DISTRICT - nonirr. season
SetDiversionDemandTSMonthly(ID="955003",TSID="955003..DivTotal.MONTH~StateMod~955003_baseline.stm")
#
# RESERVOIR STRUCTURES
#
# GRN MTN HYDRO-ELECTRIC
SetDiversionDemandTSMonthly(ID="360881",TSID="360881..DivTotal.MONTH~StateMod~360881_baseline.stm")

```

```

# WILLIAMS FORK POWER COND
SetDiversionDemandTSMonthly(ID="511237",TSID="511237..DivTotal.MONTH~StateMod~511237_baseline.stm")
# Green Mountain Contract water
SetDiversionDemandTSMonthly(ID="950060",TSID="950060..DivTotal.MONTH~StateMod~950060_baseline.stm")
# FRASER BASIN demands out of Woford Mountain Reservoir
SetDiversionDemandTSMonthly(ID="953101",TSID="953101..DivTotal.MONTH~StateMod~953101_baseline.stm")
# MIDDLE PARK demands out of Woford Mountain Reservoir
SetDiversionDemandTSMonthly(ID="953102",TSID="953102..DivTotal.MONTH~StateMod~953102_baseline.stm")
# Green Mtn Annual Rep Est
SetDiversionDemandTSMonthly(ID="950061",TSID="950061..DivTotal.MONTH~StateMod~950061_baseline.stm")
# Ruedi Rnd 1-Muni Demand
SetDiversionDemandTSMonthly(ID="953001",TSID="953001..DivTotal.MONTH~StateMod~953001_baseline.stm")
# Ruedi Rnd 1-Ind Demand
SetDiversionDemandTSMonthly(ID="953002",TSID="953002..DivTotal.MONTH~StateMod~953002_baseline.stm")
# Ruedi Rnd 2-Muni Demand
SetDiversionDemandTSMonthly(ID="953003",TSID="953003..DivTotal.MONTH~StateMod~953003_baseline.stm")
# Ruedi Rnd 2-Ind Demand
SetDiversionDemandTSMonthly(ID="953004",TSID="953004..DivTotal.MONTH~StateMod~953004_baseline.stm")
#
# CAMEO DEMAND / GRAND VALLEY AREA
#
# GRAND VALLEY PROJECT IRRIGATION (fill in years from 1909 to 1916)
SetDiversionDemandTSMonthly(ID="950001",TSID="950001..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
# Orchard Mesa Irrigation District (OMID power) demand
SetDiversionDemandTSMonthlyConstant(ID="950002",Constant=0)
# ORCHARD_MESA_CHECK
SetDiversionDemandTSMonthly(ID="950003",TSID="950003..DivTotal.MONTH~StateMod~950003_baseline.stm")
# Orchard Mesa Irrigation District (OMID pump) demand
SetDiversionDemandTSMonthly(ID="950004",TSID="950004..DivTotal.MONTH~StateMod~950004_baseline.stm")
# OMID Bypass (950005) time series for baseline data set
SetDiversionDemandTSMonthlyConstant(ID="950005",Constant=0)
# OMID Bypass (950006) time series for baseline data set
SetDiversionDemandTSMonthly(ID="950006",TSID="950006..DivTotal.MONTH~StateMod~950006_baseline.stm")
# USA PP-Winter-OM Stip
SetDiversionDemandTSMonthly(ID="950007",TSID="950007..DivTotal.MONTH~StateMod~950007_baseline.stm")
# USA PP-Summer-OM Stip
SetDiversionDemandTSMonthly(ID="950008",TSID="950008..DivTotal.MONTH~StateMod~950008_baseline.stm")
#
# FISH DEMAND (in Baseline it is located at instream flow node 952002)
#
# 15 Mile Reach area for endangered fish
SetDiversionDemandTSMonthlyConstant(ID="952001",Constant=0)
#
# Step 4 - set calculated demand to current demand for structures whose historical acreage is
# different from current
#
SetDiversionDemandTSMonthly(ID="360687",TSID="360687..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360725",TSID="360725..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360728",TSID="360728..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360729",TSID="360729..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360765",TSID="360765..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360780",TSID="360780..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="360800",TSID="360800..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="370519",TSID="370519..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="370571",TSID="370571..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="370723",TSID="370723..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="370848",TSID="370848..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="380528",TSID="380528..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")

```

```

SetDiversionDemandTSMonthly(ID="380572",TSID="380572..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="380663",TSID="380663..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="380939",TSID="380939..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="380996",TSID="380996..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="381062",TSID="381062..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
SetDiversionDemandTSMonthly(ID="381078",TSID="381078..DivTotal.MONTH~StateMod~..\StateMod\cm2005C-
AcreageChange.ddm")
#
#   Step 5 - create baseline demand file
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\statemod\cm2005B.ddm")
#
# Check the results
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Bddm.commands.StateDMI.check.html")

```

#### 5.4.6 Diversion Demand Time Series Override (Monthly)

Demand override time series, if specified, will be used instead of the time series in the primary demand file. StateDMI does not process demand time series override (monthly). If needed, TSTool, a spreadsheet, or other software can be used to create the data.

#### 5.4.7 Diversion Demand Time Series (Average Monthly)

StateDMI does not process demand time series (average monthly). If needed, TSTool, a spreadsheet, or other software can be used to create the data.

#### 5.4.8 Diversion Demand Time Series (Daily)

StateDMI does not process daily diversion demand time series. TSTool, a spreadsheet, or other software can be used to create the data. More commonly, the monthly historical data can be distributed to daily time series internally by StateMod by specifying the appropriate daily station identifier.

#### 5.4.9 Irrigation Practice Time Series (Yearly)

The irrigation practice time series (yearly) file is created by the StateCU commands in StateDMI, for use with a StateCU data set. The StateMod data set can reference the StateCU file or a copy of the file. This file provides maximum efficiency, ground water acres, sprinkler acres, by year, to be used with variable efficiency calculations. The consumptive water requirement file from the StateCU model (see next section) is also used as input.

An example of variable efficiency is as follows: if the diversion is 100 and the CWR (IWR) is 25, the efficiency is 25%; if the diversion is 25 and the CWR (IWR) is 20, the efficiency is 80%. If variable efficiency is used and the irrigation practice time series file is provided, the efficiencies in the diversion station file are ignored. If variable efficiency is used by the irrigation practice time series is not used, the average efficiency in the structure station file is used.

#### 5.4.10 Consumptive Water Requirement (Monthly, Daily)

StateDMI does not process consumptive water requirement time series. The consumptive water requirement file is typically the same as the StateMod format IWR (irrigation water requirement) time



series file from StateCU output, for agricultural structures, but can contain consumptive water requirement time series for municipal and industrial locations. Therefore, unlike the demand time series, these data represent on-site requirements and do not reflect a delivery loss (as do diversion headgate demands). See the StateMod documentation for more information about specifying the demand type.

#### 5.4.11 Soil Moisture

The soil moisture file allows both StateCU and StateMod to consider soil moisture for supply. StateDMI does not process the soil moisture file. Previously this file was the same as the StateCU parameter (\*.par) file, which supplied available water content to start each year; however, this file is not used in the new version of StateCU.

### 5.5 Precipitation Data

Precipitation data consist of:

- Precipitation Time Series (Monthly)

Precipitation data are used to estimate net evaporation from reservoirs. Reservoir stations can reference both precipitation and evaporation data, or may include only net evaporation data (evaporation - precipitation). StateMod data sets do not include a file for precipitation stations. Therefore, the precipitation time series referenced in reservoir stations (if net evaporation is not used) must use the same identifiers found in the precipitation time series file.

#### 5.5.1 Precipitation Time Series (Monthly)

StateDMI does not process precipitation time series. Instead, use TSTool, a spreadsheet or other software to prepare the time series file. See **Chapter 4 – Creating StateCU Data Set Files** for an example TSTool commands file for monthly precipitation data. Often, precipitation time series are not provided and instead net evaporation time series (evaporation minus precipitation) are provided (see **Section 5.6 Evaporation Data**). The StateMod control file indicates whether the precipitation time series contain monthly or average monthly values. Precipitation time series identifiers typically match the precipitation station identifiers from HydroBase or other data source.

### 5.6 Evaporation Data

Evaporation data consist of:

- Evaporation Time Series (Monthly)

Evaporation data are used to estimate net evaporation from reservoirs. Reservoir stations can reference both precipitation and evaporation data (in which case the net evaporation is computed by StateMod), or may include only net evaporation data (evaporation - precipitation) in the evaporation time series. StateMod data sets do not include a file for evaporation stations. Therefore, the evaporation time series referenced in reservoir stations must use the same identifiers found in the evaporation time series file.

#### 5.6.1 Evaporation Time Series (Monthly)

StateDMI does not process evaporation time series. Instead, use TSTool, a spreadsheet or other software to prepare the time series file. The StateMod control file indicates whether the precipitation time series contain monthly or average monthly values. For example, use TSTool to review the average monthly

values for key evaporation and precipitation stations and manually create an average monthly net evaporation time series file. Evaporation time series identifiers typically match the evaporation station identifiers from HydroBase or other data source.

## 5.7 Reservoir Data

Reservoir data consists of:

- Reservoir stations
- Reservoir rights
- Historical content time series (monthly, daily)
- Target time series (monthly, daily)

Each of the above data types is stored in a separate file, using the diversion station identifier as the primary identifier.

The processing of each data file is discussed below.

### 5.7.1 Reservoir Stations

Each reservoir station used with StateMod can be one of two types:

1. Explicit reservoir, where no aggregation occurs – this type is used for key structures that need to be explicitly modeled. The reservoir station identifier is usually a 7-character water district identifier (6-character for old data sets) or fabricated identifier that starts with the water district number.
2. Reservoir aggregate, in which reservoir characteristics (maximum volume) are summed and water rights are aggregated into classes. Currently, aggregation of the water rights occurs when the `ReadReservoirRightsFromHydroBase()` command is executed. The naming convention for modeling in CDSS is to use an identifier similar to 20\_ARCNNN, where the leading 20 indicates the water district, ARC indicates aggregate reservoir, and NNN is a number to allow multiple reservoir aggregates in a water district. This convention allows summary of storage for basins. Aggregates should be defined using the `SetReservoirAggregate*()` commands and need to be defined when processing all reservoir station files (if aggregates are used).

Currently, StateDMI does not support Reservoir Systems (which would be similar to Diversion Systems), in which reservoir physical characteristics are combined but all water rights are explicitly represented.

The determination of the reservoir station type for each reservoir station is usually made by reviewing available data (e.g., water rights), and discussing administrative data with knowledgeable persons (e.g., water commissioners). Typically, key reservoirs have large capacities, and/or have important water rights and administrative roles. Minor reservoirs, or groups of reservoirs for which independent data are difficult to determine, may be lumped together in an aggregate or system. Grouping reservoirs into aggregates reduces the overall number of model nodes, size of output, and model run time.

The **Commands...Reservoir Data...Reservoir Stations** menus insert commands to process reservoir station data:

Reservoir Stations - Commands
ReadReservoirStationsFromList() ...
ReadReservoirStationsFromNetwork() ...
ReadReservoirStationsFromStateMod() ...
SetReservoirAggregate() ...
SetReservoirAggregateFromList() ...
SetReservoirStation() ...
SortReservoirStations() ...
FillReservoirStationsFromHydroBase() ...
FillReservoirStationsFromNetwork() ...
FillReservoirStation() ...
WriteReservoirStationsToList() ...
WriteReservoirStationsToStateMod() ...
CheckReservoirStations() ...
WriteCheckFile() ...

MenuCommands\_ReservoirStations

### Commands...Reservoir Data...Reservoir Stations Menu

The following table summarizes the use of each command:

#### Reservoir Stations Commands

Command	Description
ReadReservoirStationsFromList()	Read from a delimited list file the list of reservoir stations to be included in the data set.
ReadReservoirStationsFromNetwork()	Read from a StateMod network file a list of reservoir stations to be included in the data set.
ReadReservoirStationsFromStateMod()	Read from a StateMod reservoir stations file the list of reservoir stations to be included in the data set.
SetReservoirAggregate()	Specify that a reservoir is an aggregate and define its parts.
SetReservoirAggregateFromList()	Specify that one or more reservoirs are aggregates and define their parts, using a delimited list file.
SetReservoirStation()	Set the data for, and optionally add, reservoir stations.
SortReservoirStations()	Sort the reservoir stations. This is useful to force consistency between files.
FillReservoirStationsFromHydroBase()	Fill missing data for defined reservoir stations, using data from HydroBase. For example, retrieve the station names, and maximum

Command	Description
	volumes.
FillReservoirStationsFromNetwork()	Fill missing data for defined reservoir stations, using data from the network. For example, retrieve the station names.
FillReservoirStation()	Fill missing data for defined reservoir stations, user user-supplied values.
WriteReservoirStationsToList()	Write defined reservoir stations to a delimited file.
WriteReservoirStationsToStateMod()	Write defined reservoir stations to a StateMod file.
CheckReservoirStations()	Check reservoir stations data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the reservoir station file is shown below (from Colorado cm2005 data set):

```

StartLog(LogFile="res.commands.StateDMI.log")
# res.commands.StateDMI
#
# Creates the reservoir station file for the Upper Colorado River monthly models
# Initial reservoir contents are set to 9/1908 estimated contents
#
# Phase IIIB modifications
#   to reflect reservoir storage as of October 1908 - zero out account owners' current
#   storage capacity if the reservoir came on-line during the study period.
#   No changes made to reservoirs that were on-line in 10/1908 (including aggregate storage).
#
#   Turned on Wolford Mountain and added Wolford Mountain accounts and storage rights per CWCB
#
#   Eliminated Unallocated Pool from Vega Reservoir; it was getting filled but not booked over
#   to the Power Exchange pool, and could not get released for use
#
# commands used in this file establish reservoir capacity, fill date,
# reservoir account ownership, area-capacity tables and representative
# evaporation stations (see StateMod documentation)
#
# Step 1 - read reservoirs from network file and sort alphabetically
#
ReadReservoirStationsFromNetwork(InputFile="..\network\cm2005.net")
SortReservoirStations(Order=Ascending)
#
# Step 2 - read reservoir information from HydroBase
#
FillReservoirStationsFromHydroBase(ID="*")
#
# Step 3 - set reservoir information not available in HydroBase including min/max
#         content, starting content, and account information
#
# GREEN MOUNTAIN RESERVIOR Characteristics
SetReservoirStation(ID="363543",OnOff=3,OneFillRule=4,DailyID="5",ContentMin=0,ContentMax=154645,
  ReleaseMax=4010,DeadStorage=0,AccountID=1,
  AccountName="Hist_Users",AccountMax=66000,AccountInitial=0,AccountEvap=0,AccountOneFill=1,
  EvapStations="10008,100",IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=2,AccountName="CBT_Pool",AccountMax=52000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=3,AccountName="Contract",AccountMax=20000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=4,AccountName="Silt_Proj",AccountMax=5000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=5,AccountName="Inactive",AccountMax=11645,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=6,AccountName="SurplusFish",AccountMax=66000,
  AccountInitial=0,AccountEvap=0,IfNotFound=Warn)

```

```

... similar commands for other reservoirs omitted...
#
# District 50 Aggregated Reservoirs
SetReservoirStation(ID="50_ARC006",Name="50_ARC006",OnOff=1,OneFillRule=-1,
  DailyID="5",ContentMin=0,ContentMax=11481,ReleaseMax=999999,DeadStorage=0,
  AccountID=1,AccountName="50_ARC006",AccountMax=11481,AccountInitial=11481,
  AccountEvap=0,AccountOneFill=1,EvapStations="10008,100",
  ContentAreaSeepage="0,0,0;11481,1148.1,0;9999999,1148.1,0",IfNotFound=Warn)
...similar commands for other reservoirs omitted...
#
WriteReservoirStationsToStateMod(OutputFile="..\statemod\cm2005.res")
#
# Check the results
CheckReservoirStations(ID="*")
WriteCheckFile(OutputFile="res.commands.StateDMI.check.html")

```

### 5.7.2 Reservoir Rights

Reservoir rights correspond to the reservoir stations, using the reservoir station identifier to relate the data. Reservoir right identifiers are typically the reservoir station identifier followed by .NN, where NN is a sequential number starting with 01. Reservoir aggregate stations have rights corresponding to water right classes.

The **Commands...Reservoir Data...Reservoir Rights** menu items insert commands to process reservoir rights data:

Reservoir Rights - Commands
ReadReservoirStationsFromList() ...
ReadReservoirStationsFromStateMod() ...
SetReservoirAggregate() ...
SetReservoirAggregateFromList() ...
ReadReservoirRightsFromHydroBase() ...
ReadReservoirRightsFromStateMod() ...
SetReservoirRight() ...
SortReservoirRights() ...
FillReservoirRight() ...
WriteReservoirRightsToList() ...
WriteReservoirRightsToStateMod() ...
CheckReservoirRights() ...
WriteCheckFile() ...

MenuCommands\_ReservoirRights

### Commands...Reservoir Data...Reservoir Rights Menu

The following table summarizes the use of each command:

### Reservoir Rights Commands

Command	Description
ReadReservoirStationsFromList()	Read from a delimited file the list of reservoir stations to be included in the data set – the list indicates the stations for which to process rights.
ReadReservoirStationsFromStateMod()	Read from a StateMod reservoir stations file the list of reservoir stations to be included in the data set – the list indicates the stations for which to process rights.
SetReservoirAggregate()	Specify that a reservoir is an aggregate and define its parts.
SetReservoirAggregateFromList()	Specify that one or more reservoirs are aggregates and define their parts, using a delimited list file.
ReadReservoirRightsFromHydroBase()	For each reservoir station, read the corresponding reservoir rights from HydroBase.
ReadReservoirRightsFromStateMod()	Read reservoir rights from a StateMod reservoir rights file.
SetReservoirRight()	Set the data for, and optionally add, reservoir rights.
SortReservoirRights()	Sort the reservoir rights. This is useful to force consistency between files.
FillReservoirRight()	Fill missing data for defined reservoir rights, using user-supplied values.
WriteReservoirRightsToList()	Write defined reservoir rights to a delimited file.
WriteReservoirRightsToStateMod()	Write defined reservoir rights to a StateMod file.
CheckReservoirRights()	Check reservoir rights data for problems.
WriteCheckFile()	Write the results of data checks to a file.

The following example command file (from the Colorado cm2005 data set) illustrates how to create the reservoir rights file:

```

StartLog(LogFile="rer.commands.StateDMI.log")
# rer.commands.StateDMI
#
# Creates the reservoir rights file for the Upper Colorado River model
#
# Step 1 - read reservoirs from reservoir station file
#
ReadReservoirStationsFromStateMod(InputFile="..\StateMod\cm2005.res")
#
# Step 2 - read reservoir rights from HydroBase
#
ReadReservoirRightsFromHydroBase(ID="*",OnOffDefault=1)
#
# Step 3 - assign rights to specific accounts, if required
#
# assign rights not in hydrobase and free-river rights
SetReservoirRight(ID="363543.01",Name="GREEN_MOUNTAIN_RESERVOIR",StationID="ID",
AdministrationNumber=31258.00000,Decree=154645,OnOff=1943,AccountDist="-5",
RightType=1,FillType=1,IfNotFound=Warn,IfFound=Set)
# Set Green Mountain's senior refill right to be junior to the Con-Hoosier and
# Dillon/Roberts Tunnel projects and the Blue River Decree Exchange
# this is based on agreements with the USBR and Denver.
SetReservoirRight(ID="363543.02",Name="GREEN_MOUNTAIN_RESERVOIR",StationID="ID",
AdministrationNumber=38628.00001,Decree=6316,OnOff=1943,AccountDist="-5",

```

```

RightType=1,FillType=2,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="363543.03",Name="GREEN_MOUNTAIN-refill",StationID="ID",
  AdministrationNumber=50403.49309,Decree=154645,OnOff=1943,AccountDist="-5",
  FillType=2,IfNotFound=Add,IfFound=Set)
# 363543.04 right is used by Type 41 Rule in accordance with the Blue River Decree and the
Interim Policy
SetReservoirRight(ID="363543.04",Name="GREEN_MOUNTAIN_RES_Exch",StationID="ID",
  AdministrationNumber=38628.00000,Decree=154645,OnOff=1,AccountDist="-5",
  FillType=2,IfNotFound=Add,IfFound=Warn)
# Con-Hoosier Res (aka Upper Blue Lakes) set 0.00001 junior to Con-Hoosier tunnel diversion
SetReservoirRight(ID="363570.01",Name="CON_HOOSIER_RES-orig",StationID="ID",
  AdministrationNumber=35927.00001,Decree=10000,OnOff=1,AccountDist="1",
  FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="363570.02",Name="CON_HOOSIER_RES-free",StationID="ID",
  AdministrationNumber=99999.99999,Decree=10000,OnOff=1,
  AccountDist="1",RightType=1,FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="363575.01",Name="Clinton Gulch Original Modified",Decree=600,
  AccountDist="-9",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="363575.02",Name="CLINTON_GULCH-refill",StationID="ID",OnOff=1,
  AccountDist="-9",RightType=1,FillType=2,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="363575.03",Name="Clinton Gulch Modified Sr to Dillon",StationID="ID",
  AdministrationNumber=31257.99999,Decree=3650,OnOff=1,
  AccountDist="-9",RightType=1,FillType=1,IfNotFound=Add,IfFound=Warn)
# Denver's Dillon Reservoir set junior to Colorado Springs' Continental Hoosier Project
SetReservoirRight(ID="364512.01",Name="DILLON_RESERVOIR-modify",StationID="ID",
  AdministrationNumber=35927.00005,Decree=252678,OnOff=1,AccountDist="-3",RightType=1,
  FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="364512.02",Name="DILLON_RESERVOIR-refill",StationID="ID",
  AdministrationNumber=50038.49309,Decree=252678,OnOff=1,AccountDist="-3",RightType=1,
  FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="373639.01",Name="Wolcott_Reservoir",StationID="ID",
  AdministrationNumber=42485.00000,Decree=65975,OnOff=1,AccountDist="-1",RightType=1,
  FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="373699.01",AccountDist="-4",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="374516.01",AccountDist="-2",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="374516.02",Name="HOMESTAKE_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=43505,OnOff=1,AccountDist="-2",RightType=1,
  FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="383713.01",Name="RUEDI_RESERVOIR",StationID="ID",OnOff=1,
  AccountDist="-6",RightType=1,FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="383713.02",Name="RUEDI_RESERVOIR-refill",StationID="ID",
  Decree=101280,OnOff=1,AccountDist="-3",RightType=1,FillType=2,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="393505.01",AccountDist="1",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="393505.02",AccountDist="1",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="393505.03",Name="GRASS_VALLEY_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=5920,OnOff=1,
  AccountDist="1",RightType=1,FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="393508.01",AccountDist="-2",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="393508.02",Name="RIFLE_GAP_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=13601,OnOff=1,AccountDist="-2",RightType=1,
  FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="503668.01",Name="WOLFORD_MOUNTAIN_RES",StationID="ID",OnOff=1,
  AccountDist="-2",RightType=1,FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="503668.02",Name="WOLFORD_MOUNTAIN_RES",StationID="ID",OnOff=1,
  AccountDist="3",RightType=1,FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="503668.03",Name="WOLFORD_MOUNTAIN-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=30000,OnOff=1,AccountDist="-2",RightType=1,
  FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="513686.01",AccountDist="-3",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="513686.02",Name="MEADOW_CREEK_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=5100,OnOff=1,AccountDist="-3",RightType=1,
  FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="513695.01",AccountDist="-2",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="513695.02",Name="SHADOW_MTN_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=19669,OnOff=1,AccountDist="-2",RightType=1,
  FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight(ID="513709.01",AccountDist="-2",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="513709.02",AccountDist="-1",FillType=2,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="513710.01",AccountDist="-2",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight(ID="513710.02",Name="WILLOW_CREEK_RES-refill",StationID="ID",
  AdministrationNumber=99999.99999,Decree=10553,OnOff=1,

```

```

AccountDist="-2",RightType=1,FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="514620.01",Name="GRANBY_RESERVOIR",StationID="ID",
AdministrationNumber=31258.00000,Decree=543758,OnOff=1,AccountDist="-2",RightType=1,
FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="514620.02",Name="GRANBY_RESERVOIR-refill",StationID="ID",
AdministrationNumber=99999.99999,Decree=543758,OnOff=1,AccountDist="-2",RightType=1,
FillType=2,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="723844.01",AccountDist="-3",FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight (ID="723844.02",Name="VEGA_RESERVOIR_refill",StationID="ID",
AdministrationNumber=99999.99999,Decree=33500,OnOff=1,AccountDist="-2",RightType=1,
FillType=2,IfNotFound=Add,IfFound=Warn)
#
#   set rights for reservoirs and stock pond to capacity with senior water right
#
SetReservoirRight (ID="36_ARC001.01",Name="36_ARC001",StationID="ID",
AdministrationNumber=1.00000,Decree=8702,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="37_ARC002.01",Name="37_ARC002",StationID="ID",
AdministrationNumber=1.00000,Decree=6671,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="38_ARC003.01",Name="38_ARC003",StationID="ID",
AdministrationNumber=1.00000,Decree=13074,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="39_ARC004.01",Name="39_ARC004",StationID="ID",
AdministrationNumber=1.00000,Decree=2236,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="45_ARC005.01",Name="45_ARC005",StationID="ID",
AdministrationNumber=1.00000,Decree=2054,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="50_ARC006.01",Name="50_ARC006",StationID="ID",
AdministrationNumber=1.00000,Decree=11481,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="51_ARC007.01",Name="51_ARC007",StationID="ID",
AdministrationNumber=1.00000,Decree=8480,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="52_ARC008.01",Name="52_ARC008",StationID="ID",
AdministrationNumber=1.00000,Decree=821,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="53_ARC009.01",Name="53_ARC009",StationID="ID",
AdministrationNumber=1.00000,Decree=2261,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="72_ARC010.01",Name="72_ARC010",StationID="ID",
AdministrationNumber=1.00000,Decree=25664,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="72_ASC001.01",Name="72_ASC001",StationID="ID",
AdministrationNumber=1.00000,Decree=2261,OnOff=1,FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="953802.01",Name="LEON_CREEK_AGGREG_RES",StationID="ID",
AdministrationNumber=1.00000,Decree=4933,OnOff=1,AccountDist="1",RightType=1,
FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="953800.01",Name="BONHAM_AGGREGATED_RES",StationID="ID",
AdministrationNumber=1.00000,Decree=6778,OnOff=1,AccountDist="1",RightType=1,
FillType=1,IfNotFound=Add,IfFound=Warn)
SetReservoirRight (ID="953801.01",Name="COTTONWOOD_AGGREG_RES",StationID="ID",
AdministrationNumber=1.00000,Decree=3812,OnOff=1,AccountDist="1",RightType=1,
FillType=1,IfNotFound=Add,IfFound=Warn)
#
#   Step 4 - create output for Historic and Calculated datasets
#
WriteReservoirRightsToStateMod(OutputFile="..\StateMod\cm2005.rer")
#
#   Step 5 - Reset Green Mountain Rights' Start Dates for Baseline dataset
#
SetReservoirRight (ID="363543.01",Name="GREEN_MOUNTAIN_RESERVOIR",StationID="ID",
AdministrationNumber=31258.00000,Decree=154645,OnOff=1,AccountDist="-5",RightType=1,
FillType=1,IfNotFound=Warn,IfFound=Set)
SetReservoirRight (ID="363543.02",Name="GREEN_MOUNTAIN_RESERVOIR",StationID="ID",
AdministrationNumber=31258.00000,Decree=6316,OnOff=1,AccountDist="-5",RightType=1,
FillType=2,IfNotFound=Warn,IfFound=Set)
SetReservoirRight (ID="363543.03",Name="GREEN_MOUNTAIN-refill",StationID="ID",
AdministrationNumber=50403.49309,Decree=154645,OnOff=1,AccountDist="-5",
FillType=2,IfNotFound=Add,IfFound=Set)
#
#   Step 6 - create output for Baseline dataset
#
WriteReservoirRightsToStateMod(OutputFile="..\StateMod\cm2005B.rer")
#
#   Check the results
CheckReservoirRights(ID="*")
WriteCheckFile(OutputFile="rer.commands.StateDMI.check.html")

```



### 5.7.3 Reservoir Content, Target Time Series (Monthly, Daily)

StateDMI does not process reservoir time series. Instead, use TSTool, a spreadsheet or other software to prepare the time series file. For example, use TSTool's `CreateFromList()` command to specify a list of reservoir station identifiers and create time series identifiers for HydroBase time series.

The following example TSTool command file (from the Colorado cm2005 data set) illustrates how end of month content time series can be created:

```
# eom.commands.TSTool
#
# commands in this file either pull historical EOM contents from the CRDSS database
# (i.e. Rifle Gap) or from user-defined *.stm files
#
# rrb 98/09/29; Revised aggregated reservoir and stockpond ID's (e.g. 36_ADC_001 = 36_ADC001)
#
# Phase IIIb modifications
#   Include extended replacement files from Task 11.1 and Cont. Auth. #5
#   Add Wolford Mtn EOM Data from River District
#   Fill missing data using water district indicator gages determined in demandts runs
#   Fill with historical monthly average if no wetness pattern average available
#   Set start dates for reservoirs in March of year listed in Ray A fax (9/8/98)
#
# James Heath, LRE (heath@lrcwe.com) updated the previous version of the file to reflect changes
#   in the TSTool commands and formatting. Data has also been updated through 2005. Some
#   underlying engineering estimates have changed and are reflected in this command file.
#
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
ReadPatternFile(PatternFile="..\Diversion\fill2005.pat")
#
# GREEN MOUNTAIN RESERVOIR
363543...MONTH~StateMod~363543.stm
#
# UPPER BLUE RESERVOIR (ConHoosier)
# Data from HydroBase is used to better represent actual operations of the reservoir in the cm2005
# update rather than setting the contents to its maximum as in previous model versions.
363570.DWR.ResMeasStorage.Day~HydroBase
TS ConHoosier363570 = NewEndOfMonthTSFromDayTS(DayTSID="363570.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="363570.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="ConHoosier363570",PatternID="09037500")
SetConstant(TSList=LastMatchingTSID,TSID="ConHoosier363570",ConstantValue=0,SetEnd="03/1962")
FillInterpolate(TSList=LastMatchingTSID,TSID="ConHoosier363570",MaxIntervals=0,Transformation=None)
#
# CLINTON GULCH RESERVOIR
# Data from HydroBase is used to better represent actual operations of the reservoir in the cm2005
# update rather than setting the contents to its maximum as in previous model versions.
363575.DWR.ResMeasStorage.Day~HydroBase
TS ClintonGulch363575 = NewEndOfMonthTSFromDayTS(DayTSID="363575.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="363575.DWR.ResMeasStorage.Day")
FillInterpolate(TSList=LastMatchingTSID,TSID="ClintonGulch363575",FillStart="10/1992",FillEnd="09/2004")
FillPattern(TSList=LastMatchingTSID,TSID="ClintonGulch363575",PatternID="09037500")
SetConstant(TSList=LastMatchingTSID,TSID="ClintonGulch363575",ConstantValue=0,SetEnd="03/1977")
FillInterpolate(TSList=LastMatchingTSID,TSID="ClintonGulch363575",MaxIntervals=0,Transformation=None)
#
# DILLON RESERVOIR
364512...MONTH~StateMod~364512.stm
#
36_ARC001...MONTH~StateMod~36_ARC001.stm
FillPattern(TSList=LastMatchingTSID,TSID="36_ARC001...MONTH",PatternID="09037500")
#
# WOLCOTT RESERVOIR
373639...MONTH~StateMod~zero.stm
#
# EAGLE PARK RESERVOIR
373699...MONTH~StateMod~zero.stm
```

```

# Data is available in HydroBase for Eagle Park Reservoir but currently the reservoir is only a
# placeholder for future updates to fill in the details at a later date.
#373699.DWR.ResMeasStorage.Day~HydroBase
#TS EaglePark373699 = NewEndOfMonthTSFromDayTS(373699.DWR.ResMeasStorage.Day,16)
#Free(TSID="373699.DWR.ResMeasStorage.Day")
#FillPattern(EaglePark373699,09085000)
#SetConstant(TSID="EaglePark373699",ConstantValue=0,SetEnd="04/1997")
#FillInterpolate(EaglePark373699,0,Linear)
#
# HOMESTAKE PROJ RESERVOIR
# Data from HydroBase is used exclusively as it was representative of what was previously in the .stm
# file as used in previous model versions. This allows for easier updating in the future.
374516.DWR.ResMeasStorage.Day~HydroBase
TS Homestake374516 = NewEndOfMonthTSFromDayTS(DayTSID="374516.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="374516.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="Homestake374516",PatternID="09085000")
SetConstant(TSList=LastMatchingTSID,TSID="Homestake374516",ConstantValue=0,SetEnd="03/1967")
FillInterpolate(TSList=LastMatchingTSID,TSID="Homestake374516",MaxIntervals=0,Transformation=None)
#
37_ARC002...MONTH~StateMod~37_ARC002.stm
FillPattern(TSList=LastMatchingTSID,TSID="37_ARC002...MONTH",PatternID="09085000")
#
# RUEDI RESERVOIR
383713...MONTH~StateMod~383713.stm
#
38_ARC003...MONTH~StateMod~38_ARC003.stm
FillPattern(TSList=LastMatchingTSID,TSID="38_ARC003...MONTH",PatternID="09085000")
#
# GRASS VALLEY RESERVOIR
# Data from HydroBase is used exclusively as it was representative of what was previously in the .stm
# file as used in previous model versions. This allows for easier updating in the future.
# There was one data point, in April 1981, that was replaced with 5989 af (mis-key).
393505.DWR.ResMeasStorage.Day~HydroBase
TS GrassValley393505 = NewEndOfMonthTSFromDayTS(DayTSID="393505.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="393505.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="GrassValley393505",PatternID="09095500")
FillInterpolate(TSList=LastMatchingTSID,TSID="GrassValley393505",MaxIntervals=0,Transformation=None)
SetConstant(TSList=LastMatchingTSID,TSID="GrassValley393505",ConstantValue=5989,
SetStart="04/1981",SetEnd="04/1981")
#
# RIFLE GAP RESERVOIR
# Data from HydroBase is used exclusively as it was previously in past model versions.
# August of 2004 appeared to be a typo and has been corrected below to what appeared to be the
# correct value.
393508.DWR.ResMeasStorage.Day~HydroBase
TS RifleGap393508 = NewEndOfMonthTSFromDayTS(DayTSID="393508.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="393508.DWR.ResMeasStorage.Day")
SetConstant(TSList=LastMatchingTSID,TSID="RifleGap393508",ConstantValue=700.16,
SetStart="08/2004",SetEnd="08/2004")
FillPattern(TSList=LastMatchingTSID,TSID="RifleGap393508",PatternID="09095500")
SetConstant(TSList=LastMatchingTSID,TSID="RifleGap393508",ConstantValue=0,SetEnd="03/1967")
FillInterpolate(TSList=LastMatchingTSID,TSID="RifleGap393508",MaxIntervals=0,Transformation=None)
#
39_ARC004...MONTH~StateMod~39_ARC004.stm
FillPattern(TSList=LastMatchingTSID,TSID="39_ARC004...MONTH",PatternID="09095500")
#
45_ARC005...MONTH~StateMod~45_ARC005.stm
FillPattern(TSList=LastMatchingTSID,TSID="45_ARC005...MONTH",PatternID="09095500")
#
# WOLFORD MOUNTAIN RES
503668...MONTH~StateMod~503668.stm
SetConstant(TSList=LastMatchingTSID,TSID="503668...MONTH",ConstantValue=0,SetEnd="03/1995")
#
50_ARC006...MONTH~StateMod~50_ARC006.stm
FillPattern(TSList=LastMatchingTSID,TSID="50_ARC006...MONTH",PatternID="09034500")
#
# MEADOW CREEK RESERVOIR
# Data from HydroBase is used exclusively as it was representative of what was previously in the .stm
# file as used in previous model versions. This allows for easier updating in the future.
# Additionally a shift has been added as it represents 300 af additional dead storage not represented
# in the HydroBase records (as stated in the previous model version's .stm file).

```

```

513686.DWR.ResMeasStorage.Day~HydroBase
TS MeadowCreek513686 = NewEndOfMonthTSFromDayTS(DayTSID="513686.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="513686.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="MeadowCreek513686",PatternID="09034500")
AddConstant(TSList=LastMatchingTSID,TSID="MeadowCreek513686",ConstantValue=300)
SetConstant(TSList=LastMatchingTSID,TSID="MeadowCreek513686",ConstantValue=0,SetEnd="03/1956")
FillInterpolate(TSList=LastMatchingTSID,TSID="MeadowCreek513686",MaxIntervals=0,Transformation=None)
#
# CBT SHADOW MTN GRAND L
# Data from HydroBase is used exclusively as it was previously in past model versions.
513695.DWR.ResMeasStorage.Day~HydroBase
TS ShadowMountainGrandLake513695 =
NewEndOfMonthTSFromDayTS(DayTSID="513695.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="513695.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="ShadowMountainGrandLake513695",PatternID="09034500")
SetConstant(TSList=LastMatchingTSID,TSID="ShadowMountainGrandLake513695",
ConstantValue=0,SetEnd="03/1946")
FillInterpolate(TSList=LastMatchingTSID,TSID="ShadowMountainGrandLake513695",
MaxIntervals=0,Transformation=None)
#
# WILLIAMS FORK RESERVOIR
513709...MONTH~StateMod~513709.stm
#
# CBT WILLOW CREEK RES
# Data from HydroBase is used exclusively as it was previously in past model versions.
513710.DWR.ResMeasStorage.Day~HydroBase
TS WillowCreek513710 = NewEndOfMonthTSFromDayTS(DayTSID="513710.DWR.ResMeasStorage.Day",Bracket=16)
Free(TSList=LastMatchingTSID,TSID="513710.DWR.ResMeasStorage.Day")
FillPattern(TSList=LastMatchingTSID,TSID="WillowCreek513710",PatternID="09034500")
SetConstant(TSList=LastMatchingTSID,TSID="WillowCreek513710",ConstantValue=0,SetEnd="03/1953")
FillInterpolate(TSList=LastMatchingTSID,TSID="WillowCreek513710",MaxIntervals=0,Transformation=None)
#
# CBT GRANBY RESERVOIR
514620...MONTH~StateMod~514620.stm
# Setting specific discrepancies that Meg Frantz and Heather Thompson found
# during the Windy Gap Firing Project modeling by Boyle Engineering
SetDataValue(TSList=LastMatchingTSID,TSID="514620...MONTH",SetDateTime="03/1954",NewValue=372900)
SetDataValue(TSList=LastMatchingTSID,TSID="514620...MONTH",SetDateTime="10/1960",NewValue=411100)
SetDataValue(TSList=LastMatchingTSID,TSID="514620...MONTH",SetDateTime="10/1961",NewValue=478100)
SetDataValue(TSList=LastMatchingTSID,TSID="514620...MONTH",SetDateTime="06/1967",NewValue=263400)
#
51_ARC007...MONTH~StateMod~51_ARC007.stm
FillPattern(TSList=LastMatchingTSID,TSID="51_ARC007...MONTH",PatternID="09034500")
#
52_ARC008...MONTH~StateMod~52_ARC008.stm
FillPattern(TSList=LastMatchingTSID,TSID="52_ARC008...MONTH",PatternID="09085000")
#
53_ARC009...MONTH~StateMod~53_ARC009.stm
FillPattern(TSList=LastMatchingTSID,TSID="53_ARC009...MONTH",PatternID="09085000")
#
# VEGA RESERVOIR
723844...MONTH~StateMod~723844.stm
SetConstant(TSList=LastMatchingTSID,TSID="723844...MONTH",ConstantValue=0,SetEnd="03/1960")
#
72_ARC010...MONTH~StateMod~72_ARC010.stm
FillPattern(TSList=LastMatchingTSID,TSID="72_ARC010...MONTH",PatternID="09095500")
#
72_ASC001...MONTH~StateMod~72_ASC001.stm
FillPattern(TSList=LastMatchingTSID,TSID="72_ASC001...MONTH",PatternID="09095500")
#
# BONHAM AGGREGATED RES
953800...MONTH~StateMod~953800.stm
FillPattern(TSList=LastMatchingTSID,TSID="953800...MONTH",PatternID="09095500")
#
# COTTONWOOD AGGREG RES
953801...MONTH~StateMod~953801.stm
FillPattern(TSList=LastMatchingTSID,TSID="953801...MONTH",PatternID="09095500")
#
# LEON CREEK AGGRES RES
953802...MONTH~StateMod~953802.stm
FillPattern(TSList=LastMatchingTSID,TSID="953802...MONTH",PatternID="09095500")

```

```

FillHistMonthAverage(TSList=AllTS)
#
WriteStateMod(TSList=AllTS,OutputFile="..\statemod\cm2005.eom",Precision=0)
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="eom.commands.TSTool.check.html")

```

Reservoir targets can be created similarly; however, each reservoir have a minimum target time series (often zero) and a maximum target. StateMod will also allow the minimum target time series to be omitted. The following command file (from the Colorado cm2005 data set) illustrates how to create the historical case reservoir target file:

```

# Htar.commands.TSTOOL
#
# Targets for Step 1 calibration (release to target)
# Minimum targets set to "0", Maximum targets same as eom file
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Green Mountain Reservoir
363543...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="363543...MONTH",ConstantValue=0)
363543...MONTH~StateMod~..\statemod\cm2005.eom
#
# UPPER BLUE RESERVOIR
363570...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="363570...MONTH",ConstantValue=0)
363570...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="363570...MONTH",
    MonthValues="0,0,0,2113,2113,2113,2113,1850,2113,2113,0,0",SetStart="04/1962")
#
# CLINTON GULCH RESERVOIR
363575...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="363575...MONTH",ConstantValue=0)
363575...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="363575...MONTH",ConstantValue=4300,SetStart="04/1977")
#
# DILLON RESERVOIR
364512...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="364512...MONTH",ConstantValue=0)
364512...MONTH~StateMod~..\statemod\cm2005.eom
#
36_ARC001...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="36_ARC001...MONTH",ConstantValue=0)
36_ARC001...MONTH~StateMod~..\statemod\cm2005.eom
#
# WOLCOTT RESERVOIR
373639...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="373639...MONTH",ConstantValue=0)
373639...MONTH~StateMod~..\statemod\cm2005.eom
#
# EAGLE PARK RESERVOIR
373699...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="373699...MONTH",ConstantValue=0)
373699...MONTH~StateMod~..\statemod\cm2005.eom
#
# HOMESTAKE PROJ RESERVOIR
374516...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="374516...MONTH",ConstantValue=0)
374516...MONTH~StateMod~..\statemod\cm2005.eom
#
37_ARC002...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="37_ARC002...MONTH",ConstantValue=0)
37_ARC002...MONTH~StateMod~..\statemod\cm2005.eom
#
# RUEDI RESERVOIR
383713...MONTH~StateMod~..\statemod\cm2005.eom

```

```
SetConstant(TSList=LastMatchingTSID,TSID="383713...MONTH",ConstantValue=0)
383713...MONTH~StateMod~..\statemod\cm2005.eom
#
38_ARC003...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="38_ARC003...MONTH",ConstantValue=0)
38_ARC003...MONTH~StateMod~..\statemod\cm2005.eom
#
# GRASS VALLEY RESERVOIR
393505...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="393505...MONTH",ConstantValue=0)
393505...MONTH~StateMod~..\statemod\cm2005.eom
#
# RIFLE GAP RESERVOIR
393508...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="393508...MONTH",ConstantValue=0)
393508...MONTH~StateMod~..\statemod\cm2005.eom
#
39_ARC004...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="39_ARC004...MONTH",ConstantValue=0)
39_ARC004...MONTH~StateMod~..\statemod\cm2005.eom
#
45_ARC005...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="45_ARC005...MONTH",ConstantValue=0)
45_ARC005...MONTH~StateMod~..\statemod\cm2005.eom
#
# WOLFORD MOUNTAIN RES
503668...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="503668...MONTH",ConstantValue=0)
503668...MONTH~StateMod~..\statemod\cm2005.eom
#
50_ARC006...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="50_ARC006...MONTH",ConstantValue=0)
50_ARC006...MONTH~StateMod~..\statemod\cm2005.eom
#
# MEADOW CREEK RESERVOIR
513686...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="513686...MONTH",ConstantValue=0)
513686...MONTH~StateMod~..\statemod\cm2005.eom
#
# CBT SHADOW MTN GRAND L
513695...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="513695...MONTH",ConstantValue=0)
513695...MONTH~StateMod~..\statemod\cm2005.eom
#
# WILLIAMS FORK RESERVOIR
513709...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="513709...MONTH",ConstantValue=0)
513709...MONTH~StateMod~..\statemod\cm2005.eom
#
# CBT WILLOW CREEK RES
513710...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="513710...MONTH",ConstantValue=0)
513710...MONTH~StateMod~..\statemod\cm2005.eom
#
# CBT GRANBY RESERVOIR
514620...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="514620...MONTH",ConstantValue=0)
514620...MONTH~StateMod~..\statemod\cm2005.eom
#
51_ARC007...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="51_ARC007...MONTH",ConstantValue=0)
51_ARC007...MONTH~StateMod~..\statemod\cm2005.eom
#
52_ARC008...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="52_ARC008...MONTH",ConstantValue=0)
52_ARC008...MONTH~StateMod~..\statemod\cm2005.eom
#
53_ARC009...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="53_ARC009...MONTH",ConstantValue=0)
53_ARC009...MONTH~StateMod~..\statemod\cm2005.eom
#
```

```

# VEGA RESERVOIR
723844...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="723844...MONTH",ConstantValue=0)
723844...MONTH~StateMod~..\statemod\cm2005.eom
#
72_ARC010...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="72_ARC010...MONTH",ConstantValue=0)
72_ARC010...MONTH~StateMod~..\statemod\cm2005.eom
#
72_ASC001...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="72_ASC001...MONTH",ConstantValue=0)
72_ASC001...MONTH~StateMod~..\statemod\cm2005.eom
#
# BONHAM AGGREGATED RES
953800...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="953800...MONTH",ConstantValue=0)
953800...MONTH~StateMod~..\statemod\cm2005.eom
#
# COTTONWOOD AGGREG RES
953801...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="953801...MONTH",ConstantValue=0)
953801...MONTH~StateMod~..\statemod\cm2005.eom
#
# LEON CREEK AGGREG RES
953802...MONTH~StateMod~..\statemod\cm2005.eom
SetConstant(TSList=LastMatchingTSID,TSID="953802...MONTH",ConstantValue=0)
953802...MONTH~StateMod~..\statemod\cm2005.eom
#
#
WriteStateMod(TSList=AllTS,OutputFile="..\StateMod\cm2005H.tar",Precision=0)
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="Htar.commands.TSTool.check.html")

```

## 5.8 Instream Flow Data

Instream flow data consist of:

- Instream flow stations
- Instream flow rights
- Instream flow demand time series (average monthly)
- Instream flow demand time series (monthly, daily)

Each of the above data types is stored in a separate file, using the instream flow station identifier as the primary identifier. StateMod represents the instream flow as a stream reach, with upstream and downstream termini. The processing of each data file is discussed below.

### 5.8.1 Instream Flow Stations

Instream flow stations used with StateMod are typically specified based on water rights for a stream reach.

Key instream flow stations to include in a model are typically determined by reviewing available data, including HydroBase water rights and the CWCB instream flow database, for streams that are included in the model. The streams are those that are associated with stream gage, diversion, reservoir, and well stations included in the data set. The upstream instream flow station identifier is usually a 7-character water district identifier (6-character for old data sets) or fabricated identifier that starts with the water district number. The downstream node is typically inserted into the network as an “other” node having the same identifier as the upstream terminus followed by “\_Dwn”.

The **Commands...Instream Flow Data...Instream Flow Stations** menu items insert commands to process instream flow station data:

Instream Flow Stations - Commands
ReadInstreamFlowStationsFromList() ...
ReadInstreamFlowStationsFromNetwork() ...
ReadInstreamFlowStationsFromStateMod() ...
SetInstreamFlowStation() ...
SortInstreamFlowStations() ...
FillInstreamFlowStationsFromHydroBase() ...
FillInstreamFlowStationsFromNetwork() ...
FillInstreamFlowStation() ...
WriteInstreamFlowStationsToList() ...
WriteInstreamFlowStationsToStateMod() ...
CheckInstreamFlowStations() ...
WriteCheckFile() ...

MenuCommands\_InstreamFlowStations

### Commands...Instream Flow Data...Instream Flow Stations Menu

The following table summarizes the use of each command:

#### Instream Flow Station Commands

Command	Description
ReadInstreamFlowStationsFromList()	Read from a delimited list file the list of instream flow stations to be included in the data set.
ReadInstreamFlowStationsFromNetwork()	Read from a StateMod network file a list of instream flow stations to be included in the data set.
ReadInstreamFlowStationsFromStateMod()	Read from a StateMod instream flow stations file the list of instream flow stations to be included in the data set.
SetInstreamFlowStation()	Set the data for, and optionally add, instream flow stations.
SortInstreamFlowStations()	Sort the instream flow stations. This is useful to force consistency between files.
FillInstreamFlowStationsFromHydroBase()	Fill missing data for defined instream flow stations, using data from HydroBase. For example, retrieve the station names.
FillInstreamFlowStationsFromNetwork()	Fill missing data for defined instream flow stations, using data from a StateMod network file. This is useful when the station names are not found in HydroBase and numerous SetInstreamFlowStation() commands would otherwise be required.

Command	Description
FillInstreamFlowStation()	Fill missing data for defined instream flow stations, user user-supplied values.
WriteInstreamFlowStationsToList()	Write defined instream flow stations to a delimited file.
WriteInstreamFlowStationsToStateMod()	Write defined instream flow stations to a StateMod file.
CheckInstreamFlowStations()	Check instream flow stations data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file (from the Colorado cm2005 data set) to create the instream flow station file is shown below:

```

StartLog(LogFile="ifs.commands.StateDMI.log")
#
# Create the Colorad Instream Flow Stations file
#
# Step 1 - read instream flow structures from network file, sort alphabetically.
#
ReadInstreamFlowStationsFromNetwork(InputFile="..\Network\cm2005.net")
SortInstreamFlowStations(Order=Ascending)
#
# Step 2 - create file and set daily flags
#
SetInstreamFlowStation(ID="",DailyID="0",DemandType=2)
#
# Step 3 - set instream flow information for non-HB structures
#
# Following insf are reservoir bypasses
SetInstreamFlowStation(ID="953508",Name="Rifle_Gap_Res_Bypass",UpstreamRiverNodeID="953508",
    DownstreamRiverNodeID="953508_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953543",Name="Green_Mtn_Res_Bypass",UpstreamRiverNodeID="953543",
    DownstreamRiverNodeID="953543_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953668",Name="Wolford_Res_Bypass",UpstreamRiverNodeID="953668",
    DownstreamRiverNodeID="953668_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953695",Name="Shadow_Mtn_Res_Bypass",UpstreamRiverNodeID="953695",
    DownstreamRiverNodeID="953695_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953709",Name="Williams_Fork_Res_Bypass",UpstreamRiverNodeID="953709",
    DownstreamRiverNodeID="953709_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953710",Name="Willow_Crk_Res_Bypass",UpstreamRiverNodeID="953710",
    DownstreamRiverNodeID="953710_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="953713",Name="Ruedi_Res_Bypass",UpstreamRiverNodeID="953713",
    DownstreamRiverNodeID="953713_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="951146",Name="Wolcott_PP_Bypass",UpstreamRiverNodeID="951146",
    OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Following insf are minimum reservoir release requirements (operating rules control)
SetInstreamFlowStation(ID="954512",Name="Dillon_Res_Min_Rel",UpstreamRiverNodeID="954512",
    DownstreamRiverNodeID="954512_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="954620",Name="Granby_Res_Min_Rel",UpstreamRiverNodeID="954620",
    DownstreamRiverNodeID="954620_Dwn",OnOff=1,DailyID="0",DemandType=1,IfNotFound=Warn)
# Following insf are Fraser collection system bypass requirements (Denver's Moffat)
SetInstreamFlowStation(ID="950639",Name="Jim_Creek_Bypass",UpstreamRiverNodeID="950639",
    DownstreamRiverNodeID="950639_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="951269",Name="Den_Ranch_Crk_Bypass",UpstreamRiverNodeID="951269",
    DownstreamRiverNodeID="951269_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="951309",Name="St_Louis_Crk_Bypass",UpstreamRiverNodeID="951309",
    DownstreamRiverNodeID="951309_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="951310",Name="Vasquez_Crk_Bypass",UpstreamRiverNodeID="951310",
    DownstreamRiverNodeID="951310_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Following insf are minimum bypass for Williams Fork Diversion Project (Denver)
SetInstreamFlowStation(ID="954603",Name="Gumlick_Tunnel_Bypass",UpstreamRiverNodeID="954603",
    DownstreamRiverNodeID="954603_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Following insf are minimum bypass for Fry-Ark Project
SetInstreamFlowStation(ID="950786",Name="Thomasville_Gage_Bypass",UpstreamRiverNodeID="950786",
    DownstreamRiverNodeID="950786_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)

```



```

SetInstreamFlowStation(ID="951594",Name="Hunter_Crk_Bypass",UpstreamRiverNodeID="951594",
  DownstreamRiverNodeID="951594_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
SetInstreamFlowStation(ID="954625",Name="Boustead_Tunnel_Bypass",UpstreamRiverNodeID="954625",
  DownstreamRiverNodeID="954625_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Following insf is minimum bypass below Homestake Tunnel (Col. Springs)
SetInstreamFlowStation(ID="954516",Name="Gold_Park_Gage_Min_Flow",UpstreamRiverNodeID="954516",
  DownstreamRiverNodeID="954516_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Following insf is minimum release for the Clinton Res. agreement
SetInstreamFlowStation(ID="954655",Name="Winter_Park_Ski_Min_Flow",UpstreamRiverNodeID="954655",
  DownstreamRiverNodeID="954655_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
# Insf node added above the Shoshone Power Plant to allow simulation of Green Mtn. Res.
# operations prior to 1985
SetInstreamFlowStation(ID="950500",Name="Shoshone_Call_Flows",UpstreamRiverNodeID="950500",
  DownstreamRiverNodeID="950500_Dwn",OnOff=1,DailyID="0",DemandType=1,IfNotFound=Warn)
# CWCB insf in 15-mile reach
SetInstreamFlowStation(ID="952002",Name="USFWS_Recomm._Fish_Flow",UpstreamRiverNodeID="952002",
  DownstreamRiverNodeID="952002_Dwn",OnOff=1,DailyID="0",DemandType=1,IfNotFound=Warn)
# GVWM Bypass
SetInstreamFlowStation(ID="950099",Name="GVWM_Bypass",UpstreamRiverNodeID="950099",OnOff=1,
  DailyID="0",DemandType=2,IfNotFound=Warn)
# Eagle River Minimum Flow Second Reach
SetInstreamFlowStation(ID="372059_2",Name="MIN_FLOW_EAGLE_RIVER_2",UpstreamRiverNodeID="372059_2",
  DownstreamRiverNodeID="372059_2_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Warn)
#
# Step 4 - fill remaining instream flow information from HB and output file
#
FillInstreamFlowStationsFromHydroBase(ID="*")
WriteInstreamFlowStationsToStateMod(OutputFile="..\StateMod\cm2005.ifs",WriteHow=OverwriteFile)
#
# Check the results
CheckInstreamFlowStations(ID="*")
WriteCheckFile(OutputFile="ifs.commands.StateDMI.check.html")

```

## 5.8.2 Instream Flow Rights

Instream flow rights correspond to the instream flow stations, using the instream flow station identifier to relate the data. Instream flow right identifiers are typically the reservoir right identifier followed by .NN, where NN is a sequential number starting with 01. The **Commands...Instream Flow Data...Instream Flow Rights** menu items insert commands to process instream flow rights data:

Instream Flow Rights - Commands
ReadInstreamFlowStationsFromList() ... ReadInstreamFlowStationsFromStateMod() ...
ReadInstreamFlowRightsFromHydroBase() ... ReadInstreamFlowRightsFromStateMod() ...
SetInstreamFlowRight() ...
SortInstreamFlowRights() ...
FillInstreamFlowRight() ...
WriteInstreamFlowRightsToList() ... WriteInstreamFlowRightsToStateMod() ...
CheckInstreamFlowRights() ... WriteCheckFile() ...

MenuCommands\_InstreamFlowRights

### Commands...Instream Flow Data...Instream Flow Rights Menu

The following table summarizes the use of each command:

### Instream Flow Rights Commands

Command	Description
ReadInstreamFlowStationsFromList()	Read from a delimited file the list of instream flow stations to be included in the data set – the list indicates the stations for which to process rights.
ReadInstreamFlowStationsFromStateMod()	Read from a StateMod instream flow stations file the list of instream flow stations to be included in the data set – the list indicates the stations for which to process rights.
ReadInstreamFlowRightsFromHydroBase()	For each instream flow station, read the corresponding instream flow rights from HydroBase.
ReadInstreamFlowRightsFromStateMod()	Read instream flow rights from a StateMod instream flow rights file.
SetInstreamFlowRight()	Set the data for, and optionally add, instream flow rights.
SortInstreamFlowRights()	Sort the instream flow stations. This is useful to force consistency between files.
FillInstreamFlowRight()	Fill missing data for defined instream flow rights, using user-supplied values.
WriteInstreamFlowRightsToList()	Write instream flow rights to a delimited list file.
WriteInstreamFlowRightsToStateMod()	Write instream flow rights to a StateMod file.
CheckInstreamFlowRights()	Check instream flow rights data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the instream flow rights file is shown below (from the Colorado cm2005 data set):

```

StartLog(LogFile="ifr.commands.StateDMI.log")
# ifr.commands.StateDMI
#
# StateDMI command file to create the annual instream flow rights file for the
# Colorado model Historical and Calibrated models
#
# Step 1 - read instream flow structures from instream flow structure file
#
ReadInstreamFlowStationsFromStateMod(InputFile="..\STATEMOD\cm2005.ifs")
#
# Step 2 - read instream flow rights from HydroBase
#
ReadInstreamFlowRightsFromHydroBase(ID="*",OnOffDefault=1)
#
# Step 3 - set instream flow rights for non-HydroBase structures
#
# Following insf are reservoir bypasses
SetInstreamFlowRight(ID="953508.01",Name="Rifle_Gap_Res_Bypass",StationID="ID",
AdministrationNumber=37503.36898,Decree=5.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953543.01",Name="Green_Mtn_Res_Bypass",StationID="ID",
AdministrationNumber=31257.99994,Decree=85.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953668.01",Name="Wolford_Res_Bypass",StationID="ID",
AdministrationNumber=50385.99999,Decree=13.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953695.01",Name="Shadow_Mtn_Res_Bypass",StationID="ID",
AdministrationNumber=31257.99999,Decree=50.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953709.01",Name="Williams_Fork_Res_Bypass",StationID="ID",
AdministrationNumber=31358.99999,Decree=15.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953710.01",Name="Willow_Crk_Res_Bypass",StationID="ID",

```

```

AdministrationNumber=31257.99999,Decree=7.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="953713.01",Name="Ruedi_Res_Bypass",StationID="ID",
AdministrationNumber=39290.99999,Decree=110.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="951146.01",Name="Wolcott_PP_Bypass",StationID="ID",
AdministrationNumber=42484.99999,Decree=110.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf are minimum reservoir release requirements (operating rules control)
SetInstreamFlowRight(ID="954512.01",Name="Dillon_Res_Min_Rel",StationID="ID",
AdministrationNumber=31257.99997,Decree=50.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="954620.01",Name="Granby_Res_Min_Rel",StationID="ID",
AdministrationNumber=31257.99999,Decree=75.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf are Fraser collection system bypass requirements (Denver's Moffat)
SetInstreamFlowRight(ID="950639.01",Name="Jim_Creek_Bypass",StationID="ID",
AdministrationNumber=30870.26116,Decree=10.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="951269.01",Name="Den_Ranch_Crk_Bypass",StationID="ID",
AdministrationNumber=30870.26116,Decree=4.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="951309.01",Name="St_Louis_Crk_Bypass",StationID="ID",
AdministrationNumber=30870.26116,Decree=10.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="951310.01",Name="Vasquez_Crk_Bypass",StationID="ID",
AdministrationNumber=30870.26116,Decree=8.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf are minimum bypass for Williams Fork Diversion Project (Denver)
SetInstreamFlowRight(ID="954603.01",Name="Gumlick_Tunnel_Bypass",StationID="ID",
AdministrationNumber=30870.26116,Decree=1.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf are minimum bypass for Fry-Ark Project
SetInstreamFlowRight(ID="950786.01",Name="Thomasville_Gage_Bypass",StationID="ID",
AdministrationNumber=39290.99999,Decree=200.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="951594.01",Name="Hunter_Crk_Bypass",StationID="ID",
AdministrationNumber=39290.99999,Decree=21.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
SetInstreamFlowRight(ID="954625.01",Name="Boustead_Tunnel_Bypass",StationID="ID",
AdministrationNumber=39290.99999,Decree=30.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf is minimum bypass below Homestake Tunnel (Col. Springs)
SetInstreamFlowRight(ID="954516.01",Name="Gold_Park_Gage_Min_Flow",StationID="ID",
AdministrationNumber=39650.37519,Decree=24.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Following insf is minimum release for the Clinton Res. agreement
SetInstreamFlowRight(ID="954655.01",Name="Winter_Park_Ski_Min_Flow",StationID="ID",
AdministrationNumber=30870.26116,Decree=3.90,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Insf node added above the Shoshone Power Plant to allow simulation of Green Mtn. Res.
operations prior to 1985
SetInstreamFlowRight(ID="950500.01",Name="Shoshone_Call_Flows",StationID="ID",
AdministrationNumber=99999.80000,Decree=1250.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# CWCB insf in 15-mile reach
SetInstreamFlowRight(ID="952002.01",Name="USFWS_Recomm._Fish_Flow",StationID="ID",
AdministrationNumber=99999.92000,Decree=16000.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# GVWM Bypass
SetInstreamFlowRight(ID="950099.01",Name="GVWM_Bypass",StationID="ID",
AdministrationNumber=99999.00000,Decree=0.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
# Eagle River Minimum Flow Second Reach
SetInstreamFlowRight(ID="372059_2.01",Name="MIN_FLOW_EAGLE_RIVER_2",StationID="ID",
AdministrationNumber=47558.00000,Decree=155.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
#
# Step 4 - create output file
#
WriteInstreamFlowRightsToStateMod(OutputFile="..\STATEMOD\cm2005.ifr")
#
# Check the results
CheckInstreamFlowRights(ID="*")
WriteCheckFile(OutputFile="ifr.commands.StateDMI.check.html")

```

### 5.8.3 Instream Flow Demand Time Series (Average Monthly)

Instream flow demand time series correspond to the instream flow stations, using the instream flow station identifier as a key. Instream flow demand time series (average monthly) are typically generated from instream flow water rights. When read from HydroBase, these time series currently have the same value for each month of the year (although future enhancements may support seasonal right values in HydroBase).

The **Commands...Instream Flow Data...Instream Flow Demands (Average Monthly)** menu items insert commands to process instream flow demand time series (average monthly):

<b>Instream Flow Demand TS (Average Monthly) - Commands</b>
SetOutputYearType() ...
ReadInstreamFlowDemandTSAverageMonthlyFromStateMod() ...
1: ReadInstreamFlowRightsFromStateMod() ... 2: SetInstreamFlowDemandTSAverageMonthlyFromRights() ...
SetInstreamFlowDemandTSAverageMonthlyConstant() ...
WriteInstreamFlowDemandTSAverageMonthlyToStateMod() ...
CheckInstreamFlowDemandTSAverageMonthly() ...
WriteCheckFile() ...

MenuCommands\_InstreamFlowDemandTSAverageMonthly

### **Commands...Instream Flow Data...Instream Flow Demand TS (Average Monthly) Menu**

The following table summarizes the use of each command:

#### **Instream Flow Demands (Average Monthly) Commands**

<b>Command</b>	<b>Description</b>
SetOutputYearType()	Set the output year type for time series. This should correspond to the model data set year type and ensures that time series data are in the proper order. Omitting this information may result in missing data in the output.
ReadInstreamFlowDemandTSAverageMonthlyFromStateMod()	Read the instream flow demand average monthly time series from a StateMod file (if reading and manipulating).
ReadInstreamFlowRightsFromStateMod()	Read instream flow rights from a StateMod instream flow rights file.
SetInstreamFlowDemandTSAverageMonthlyFromRights()	For the specified instream flow water right(s), create a demand time series (average monthly).
SetInstreamFlowDemandTSAverageMonthlyConstant()	For the specified instream flow location, create a demand time series (average monthly) that is a constant value monthly pattern (twelve values).
WriteInstreamFlowDemandTSAverageMonthlyToStateMod()	Write defined instream flow demand time series (average monthly) to a StateMod file.
CheckInstreamFlowDemandTSAverageMonthly()	Check instream flow demand time series (average monthly) data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the instream flow demand time series (average monthly) file is shown below (from the Colorado cm2005 data set):

```
StartLog(LogFile="ifa.commands.StateDMI.log")
# ifa.commands.StateDMI
#
# StateDMI command file to create the annual instream flow demand file for the Colorado model
#
SetOutputYearType(OutputYearType=Water)
#
# Structures and total demands (rights) are defined in the instream flow rights file
#
ReadInstreamFlowRightsFromStateMod(InputFile="..\StateMod\cm2005.ifr")
#
# Step 1 - Set monthly instream flow demand to water rights for structures that are of
#         DemandType = 2 (*.dds)
#
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="3*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="5*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="7*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9500*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9506*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9507*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="951*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9535*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9536*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9537*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="9545*",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="954603",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="954625",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyFromRights(ID="954655",IfNotFound=Add)
#
#
# StateDMI expects monthly values to be entered in Calendar Year.
#
# Step 2 - Set monthly instream flow demands that vary by month
#
SetInstreamFlowDemandTSAverageMonthlyConstant(ID="362000",
    MonthValues="3.00,3.00,3.00,3.00,6.00,6.00,6.00,6.00,6.00,3.00,3.00,3.00",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyConstant(ID="362012",
    MonthValues="3.00,3.00,3.00,3.00,7.00,7.00,7.00,7.00,7.00,3.00,3.00,3.00",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyConstant(ID="362030",
    MonthValues="10.00,10.00,10.00,10.00,20.00,20.00,20.00,20.00,20.00,10.00,10.00",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyConstant(ID="362033",
    MonthValues="6.00,6.00,6.00,6.00,12.00,12.00,12.00,12.00,12.00,6.00,6.00,6.00",IfNotFound=Add)
SetInstreamFlowDemandTSAverageMonthlyConstant(ID="362037",
    MonthValues="16.00,16.00,16.00,16.00,32.00,32.00,32.00,32.00,32.00,16.00,16.00",IfNotFound=Add)
...similar commands omitted
#
# Step 3 - Create StateMod file
#
WriteInstreamFlowDemandTSAverageMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ifa")
#
# Check the results
CheckInstreamFlowDemandTSAverageMonthly(ID="*")
WriteCheckFile(OutputFile="ifa.commands.StateDMI.check.html")
```

### 5.8.4 Instream Flow Demand Time Series (Monthly, Daily)

StateDMI does not process monthly or daily instream flow demand time series. In most cases, the average monthly time series described in the previous section are sufficient. To create complete monthly or daily time series, use TSTool, a spreadsheet, or other software to prepare the time series file.

## 5.9 Well Data

Wells can be used to supply water to irrigated lands and municipal/industrial (M&I) demands (similar to diversions). However, in most cases, StateMod modeling and StateDMI focuses on agricultural wells. For agriculture, wells can be the only source of supply or can supplement surface water supply from diversion stations. Well features were added to the StateMod model after diversions; consequently, much of the processing for wells is similar to diversions.

Well stations that supplement diversion stations are often determined through GIS, where the service area for the diversion station is intersected with well locations. Well stations that fall within a service area, or are within a reasonable distance, are associated with the ditch service area. However, at a more fundamental level, diversion and well stations in CDSS are associated with irrigated parcels. The parcel data and its supply relationship from diversions and wells are then stored in HydroBase and can be processed by StateDMI. Because a service area will typically contain multiple wells, the wells in the StateMod well station file are typically aggregated and given an identifier that matches the diversion station. The diversion station is then indicated as a D&W (diversion and well) node in the model network. In general a “well station” in the StateMod well station file is not actually a single hole in the ground, but is a group of physical wells that serve an area.

Well station data consists of:

- Well stations (will be associated with a diversion station if the well supplements the diversion station)
- Well rights
- Historical pumping time series (monthly, daily)
- Demand time series (monthly, daily)
- Irrigation practice time series (yearly)
- Consumptive water requirement time series (monthly, daily)
- Soil moisture time series (yearly)

Each of the above data types is stored in a separate file, using the well station identifier as the primary identifier. The processing of each data file is discussed below, with background on specific issues.

### 5.9.1 Well Stations

Each well station used with StateMod can be one of the following types:

1. Explicit well, where the no aggregation occurs – this type is used for key structures that need to be explicitly modeled. For example, this type of well station may be appropriate for a large municipal supply well. This type of well does not supplement a diversion station and therefore will have a unique identifier that is represented in the model network. The well station identifier is usually a 7-character water district identifier or fabricated identifier that starts with the water district number.
2. Well system, where the characteristics (capacity, historical diversion, demand) of multiple wells are summed at one location and water rights are modeled explicitly – this type is used when related well structures operate as a system (e.g., a well field). Only the well system identifier is included in the model network and this identifier should be different from the parts in the collection. Well systems should be defined using the `SetWellSystem* ( )` commands and need to be defined when processing all well station files (if well systems are used). Well systems can be one of the following types:

- Well-only supply (does not supplement diversions). The naming convention for modeling in RGDSS is to use groundwater unit response function zones (URF); however, aggregating wells by basin or some other logical grouping as appropriate.
  - Well systems that supplement a diversion station's supply. In this case the identifier for the well should be the same as the diversion station, the well station should indicate the diversion station identifier in the well station file, and the diversion station should be represented in the network as a D&W node. Because relationships between wells and diversion stations occur via parcels in HydroBase, the diversion station systems should be defined (and the wells associated with each diversion station will consequently be treated as a system).
3. Well aggregate, which is the same as a well system except that water rights are aggregated into classes. Aggregation of the water rights typically occurs at the end of the command file with an `AggregateWellRights()` command. Aggregates should be defined using the `SetWellAggregate*()` commands.

Because the number of wells can be very large, well stations often are grouped by whether they supplement surface water supply (in which case the well is associated with a diversion via its service area) or are the only source of supply for irrigated lands (in which case the well is associated with one or more parcels). Processing the data then involves interpreting relationships between parcels, wells (holes in the ground), and diversion stations, in order to lump wells into a model station that represents the total groundwater supply in an area.

The well stations file may be updated several times, as follows:

1. Initial creation (see this section).
2. Adjust well station capacities based on historical well pumping (see **Section 5.9.3**).
3. Adjust well monthly efficiencies based on estimates from consumptive water requirement (see **Section 5.9.5**).

If a list of well stations is determined initially, the secondary files can be processed first and then the well stations file can be fully created with one command file.

The **Commands...Well Data...Well Stations** menus insert commands to process well station data:

Well Stations - Commands
ReadWellStationsFromList() ... ReadWellStationsFromNetwork() ... ReadWellStationsFromStateMod() ...
SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
SetWellStation() ... SetWellStationsFromList() ... 1: ReadCropPatternTSFromStateCU() ... 2: SetWellStationAreaToCropPatternTS() ... 1: ReadWellRightsFromStateMod() ... 2: SetWellStationCapacityToWellRights() ...
SortWellStations() ...
1: ReadDiversionStationsFromStateMod() ... 2: FillWellStationsFromDiversionStations() ... FillWellStationsFromNetwork() ... FillWellStation() ...
SetWellStationDelayTablesFromNetwork() ... SetWellStationDelayTablesFromRTN() ... SetWellStationDepletionTablesFromRTN() ...
WriteWellStationsToList() ... WriteWellStationsToStateMod() ...
CheckWellStations() ... WriteCheckFile() ...

MenuCommands\_WellStations

### Commands...Well Data...Well Stations Menu



The following table summarizes the use of each command:

### Well Stations Commands

Command	Description
<code>ReadWellStationsFromList()</code>	Read from a delimited list file the list of well stations to be included in the data set.
<code>ReadWellStationsFromNetwork()</code>	Read from a StateMod network file a list of well stations to be included in the data set.
<code>ReadWellStationsFromStateMod()</code>	Read from a StateMod diversion stations file the list of well stations to be included in the data set.
<code>SetWellAggregate()</code>	Specify that a well station is an aggregate and define its parts.
<code>SetWellAggregateFromList()</code>	Specify that one or more well stations are aggregates and define their parts, using a delimited list file.
<code>SetWellSystem()</code>	Specify that a well station is a system and define its parts.
<code>SetWellSystemFromList()</code>	Specify that one or more well stations are systems and define their parts, using a delimited list file.
<code>SetWellStation()</code>	Set the data for, and optionally add, well stations.
<code>SetWellStationsFromList()</code>	Set well station data from a list file.
<code>ReadCropPatternTSFromStateCU()</code>	Read the crop pattern time series file, for use by the <code>SetWellStationAreaToCropPatternTS()</code> command.
<code>SetWellStationAreaToCropPatternTS()</code>	Set the well station area data to the maximum area value from the crop pattern time series (see previous command to read the rights).
<code>ReadWellRightsFromStateMod()</code>	Read the well rights file, for use by the <code>SetWellStationCapacityToWellRights()</code> command.
<code>SetWellStationCapacityToWellRights()</code>	Set the well station capacity to the sum of the well rights for the station (see the previous command to read the rights).
<code>SortWellStations()</code>	Sort the well stations. This is useful to force consistency between files.
<code>ReadDiversionStationsFromStateMod()</code>	Read the diversion stations data, to fill well station data using <code>FillWellStationsFromDiversionStations()</code> .
<code>FillWellStationsFromDiversionStations()</code>	Fill well stations from diversion stations (see the previous command to read the diversion stations).
<code>FillWellStationsFromNetwork()</code>	Fill missing data for defined well stations, using data from the network. For example, retrieve the well names, and capacities.
<code>FillWellStation()</code>	Fill missing data for defined well stations, using user-supplied values.
<code>SetWellStationDelayTablesFromNetwork()</code>	Set default delay table information using network relationships.
<code>SetWellStationDelayTablesFromRTN()</code>	Set delay table information using information in a return flow format file.

Command	Description
SetWellStationDepletionTablesFromRTN()	Set depletion table information using information in a return flow format file.
WriteWellStationsToList()	Write the well stations to a list file.
WriteWellStationsToStateMod()	Write defined well stations to a StateMod file.
CheckWellStations()	Check well station data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An initial well station list is typically created from the network. The list is then used to create other files, including water rights and time series. Finally, the completed files can be read and summarized in the well station file, to update the following:

- capacity
- default monthly efficiencies
- acreage
- use and demand type
- delay and depletion tables

In the future, some data are expected to be split out of the well station file, to minimize updating the station file.

An example command file to create the initial well station file is shown below (from the preliminary South Platte Sp2008L data set).

```
#
#
# Wells_WES.StateDMI
#
# -----
StartLog(LogFile="Sp2008L_WES.log")
#
# -----
# Step 1 - Set the output period, used to compute averages...
SetOutputPeriod(OutputStart="1950-01",OutputEnd="2006-12")
#
# -----
# Step 2 - Read the list of well stations (all diversions + all well only)
ReadWellStationsFromList(ListFile="..\Network\sp2008L_Wells.csv",IDCol=1,NameCol=2,DiversionIDCol=8)
#
# -----
# Step 2b - Read Aug and recharge well list (currently not in network, assigned to aug station ID)
#
readWellStationsFromList(ListFile="sp2008L_AugRchWells.csv",IDCol="1",NameCol="2",DiversionIDCol="8")
#
# -----
# Step 3 - Read diversion station information. This allows some diversion data to
# be transferred to wells (e.g., demand source) and provides memory for
# aggregate/system information.
ReadDiversionStationsFromList(ListFile="..\Network\Sp2008L_Diversion.csv",IDCol=1,NameCol=2)
#
# -----
# Step 4 - Set Well aggregates (GW Only lands)
SetWellSystemFromList(ListFile="1956_01_GW.csv",Year=1956,Div=1,
  PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="1976_01_GW.csv",Year=1976,Div=1,
  PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="1987_01_GW.csv",Year=1987,Div=1,
  PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="2001_01_GW.csv",Year=2001,Div=1,
  PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
```

```

# -----
# Step 5 - Set Diversion _IRR aggregates...
SetWellAggregateFromList(ListFile="..\Sp2008L_SwAgg.csv",Year=2001,Div=1,
  PartType=Ditch,IDCol=1,PartIDsCol=3,PartsListedHow=InRow)
# -----
# Step 6 - Set Diversion Systems
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",Year=2001,Div=1,
  PartType=Ditch,IDCol=1,PartIDsCol=3,PartsListedHow=InRow)
# -----
# Step 7 - Set Diversion ID For D&W wells
# -----
# Step 8 -**** Get capacity from well right file
FillWellStation(ID="*")
# -----
# Step 9 - rrb 2007/10/10; Added commands to set well area to data in *.cds
ReadCropPatternTSFromStateCU(InputFile="..\Crops\Sp2008L.cds")
SetWellStationAreaToCropPatternTS(ID="*")
# -----
# Step 10 - Fill remaining missing data in well stations...
FillWellStation(ID="*",RiverNodeID="ID",Capacity=999,DailyID="4",AdminNumShift=0,
  DemandType=1,UseType=1,DemandSource=1,EffAnnual=60)
# -----
# Step 11 - Set delay and depletion data
SetWellStationDelayTablesFromRTN(InputFile="..\DelaySW\sp2008L_Sw.rtn",SetEfficiency=False)
SetWellStationDepletionTablesFromRTN(InputFile="..\DelaySW\sp2008L_Gw.rtn")
# -----
# Include Aug & Recharge wells
# -----
# Step 12 - rrb 2007/11/16 Read Well rights from a StateMod well right file
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
# -----
# Step 13 - rrb 2007/10/03 Set capacity to total of water rights
SetWellStationCapacityToWellRights(ID="*")
# -----
#
#      SmOpr
#      State of Colorado
#      Version: 1.00
#      Last revision date: 2006/10/27
# -----
SetWellStation(ID="0102522_AuW ",Name="RIVERSIDE Aug Well      ",
  RiverNodeID="0102522_AuW ",Capacity=999.,DailyID="4",
  AdminNumShift=0,DiversionID="NA",DemandType=1,IrrigatedAcres=0.0,
  UseType=5,DemandSource=8,EffAnnual=100.0>Returns="06759910,100.0,1",
  Depletions="06759910,100.0,2",IfNotFound=Add)
SetWellStation(ID="0102528_AuW ",Name="FT Aug Well          ",
  RiverNodeID="0102528_AuW ",Capacity=999.,DailyID="4",
  AdminNumShift=0,DiversionID="NA",DemandType=1,IrrigatedAcres=0.0,
  UseType=5,DemandSource=8,EffAnnual=100.0>Returns="06759910,100.0,1",
  Depletions="06759910,100.0,2",IfNotFound=Add)
SetWellStation(ID="0102529_AuW ",Name="UPPER Aug Well        ",
  RiverNodeID="0102529_AuW ",Capacity=999.,DailyID="4",
  AdminNumShift=0,DiversionID="NA",DemandType=1,IrrigatedAcres=0.0,
  UseType=5,DemandSource=8,EffAnnual=100.0>Returns="06759910,100.0,1",
  Depletions="06759910,100.0,2",IfNotFound=Add)
...similar commands omitted...
# -----
#
# rrb add Alternate Point wells SmAltP
# -----
SetWellStation(ID="0102520_AlP ",Name="Alternate Point      ",
  Capacity= 999.,EffAnnual=100.0,IfNotFound=Add)
SetWellStation(ID="0102524_AlP ",Name="Alternate Point      ",
  Capacity= 999.,EffAnnual=100.0,IfNotFound=Add)
...similar commands omitted...
# -----
SortWellStations(Order=Ascending)

```

```
# Step 14 - Write the updated stations with estimated efficiencies to the StateMod file...
WriteWellStationsToStateMod(OutputFile="Sp2008L.wes")
WriteWellStationsToStateMod(OutputFile="..\StateMod\Historic\Sp2008L.wes")
#
# Check well stations
CheckWellStations(ID="*")
WriteCheckFile(OutputFile="Sp2008L_WES.StateDMI.check.html")
```

## 5.9.2 Well Rights

Well rights correspond to the well stations, using the well station identifier to relate the data. Well right identifiers are typically the HydroBase identifier if modeling all rights explicitly. For Rio Grande modeling, right identifiers used the convention of well station identifier followed by W.NN, where W indicates well right (to avoid conflict with diversion rights that would otherwise have the same identifier), and NN is a sequential number starting with 01. Rights for well aggregate stations have rights corresponding to water right classes.

The **Commands...Well Data...Well Rights** menu items insert commands to process well rights data:

Well Rights - Commands
ReadWellStationsFromList() ...
ReadWellStationsFromNetwork() ...
ReadWellStationsFromStateMod() ...
SetWellAggregate() ...
SetWellAggregateFromList() ...
SetWellSystem() ...
SetWellSystemFromList() ...
ReadWellRightsFromHydroBase() ...
ReadWellRightsFromStateMod() ...
SetWellRight() ...
FillWellRight() ...
MergeWellRights() ...
AggregateWellRights() ...
SortWellRights() ...
WriteWellRightsToList() ...
WriteWellRightsToStateMod() ...
CheckWellRights() ...
WriteCheckFile() ...

MenuCommands\_WellRights

### Commands...Well Data...Well Rights Menu

The following table summarizes the use of each command. Note that well right aggregation (if aggregate well stations are used) occurs after other processing.

### Well Rights Commands

Command	Description
ReadWellStationsFromList()	Read from a delimited file the list of well stations to be included in the data set – the list indicates the stations for which to process rights.
ReadWellStationsFromNetwork()	Read from the network the list of well stations to be included in the data set – the list indicates the stations for which to process rights.
ReadWellStationsFromStateMod()	Read from a StateMod well stations file the list of well stations to be included in the data set – the list indicates the stations for which to process rights.
SetWellAggregate()	Specify that a well station is an aggregate and define its parts.
SetWellAggregateFromList()	Specify that one or more well stations are aggregates and define their parts, using a delimited list file.
SetWellSystem()	Specify that a well station is a system and define its parts.
SetWellSystemFromList()	Specify that one or more well stations are systems and define their parts, using a delimited list file.
ReadWellRightsFromHydroBase()	For each well station, read the corresponding well rights from HydroBase.
ReadWellRightsFromStateMod()	Read well rights from a StateMod well rights file.
SetWellRight()	Set the data for, and optionally add, well rights.
FillWellRight()	Fill missing data for defined well rights, using user-supplied values.
MergeWellRights()	Merge well rights determined from multiple years of irrigated lands parcel data in HydroBase. This is necessary to avoid double-counting rights. Well/parcel matching data are unique to each year of parcel data.
AggregateWellRights()	Aggregate well rights. This is used in some data sets to reduce the number of well rights, which decreases model run time and simplifies output.
SortWellRights()	Sort the well rights. This is useful to force consistency between files.
WriteWellRightsToList()	Write well rights to a list file.
WriteWellRightsToStateMod()	Write well rights to a StateMod file.
CheckWellRights()	Check well right data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the well rights file is shown below (from preliminary South Platte Sp2008L data set):

```
#
# Sp2008L_WER.StateDMI
#
# rrb 2009/06/09; Revised to read 2005 data and recognize Aug and Recharge wells are in the network
#
# Well Rights File (*.wer)
#
StartLog(LogFile="Sp2008L_WER.log")
#
# -----
# Step 1 - Read all structures
#
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
SortWellStations()
#
# -----
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,
    IDCol=1,NameCol=2,PartIDCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,
    IDCol=1,NameCol=2,PartIDCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",
    PartType=Well,IDCol=1,PartIDCol=2,PartsListedHow=InRow)
#
# -----
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,
    Div=1,PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,
    Div=1,PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,
    Div=1,PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,
    Div=1,PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,
    Div=1,PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
#
# -----
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",
    PartType=Well,IDCol=1,PartIDCol=2,PartsListedHow=InRow,PartIDColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",
    PartType=Well,IDCol=1,PartIDCol=2,PartsListedHow=InRow,PartIDColMax=1,IfNotFound=Ignore)
#
# -----
# Step 5 - Read rights from HydroBase
#
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",
    Year="1956,1976,1987,2001,2005",Div="1",DefaultAppropriationDate="1950-01-01",
    DefineRightHow=RightIfAvailable,ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# -----
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds and ipy acreage filling
# Write Data Comments="False" provides merged file used for seting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
# -----
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
# -----
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",WriteDataComments=False,WriteHow=OverwriteFile)
```

```
WriteWellRightsToStateMod(OutputFile="..\StateCU\Historic\Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
WriteWellRightsToStateMod(OutputFile="..\StateMod\Historic\Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
```

### 5.9.3 Well Historical Pumping Time Series (Monthly)

Well historical pumping time series (monthly) are estimated by the StateCU software. StateDMI does provide commands to process well pumping, as documented below. However, it is typical to use TSTool or other software to process the StateCU output, as shown in the following example (adapted from preliminary South Platte Sp2008L data set):

```
#
# Sp2008L_WEH.TsTool
#
SetOutputPeriod(OutputStart="1950-01",OutputEnd="2006-12")
#
# Read a list of recharge wells and set to zero.
# Note that the following says to read a StateMod file with name "x".
# An input type is needed but use the above and HandleMissingTShow to trick it:
# Specify the HandleMissingTShow parameter to default to all missing values.
# Then fill with zero below.
CreateFromList(ListFile="Sp2008L_AugRchWells.csv",IDCol=1,DataType="WellPumping",Interval=Month,
    InputType=StateMod,InputName="x",HandleMissingTShow=DefaultMissingTS)
FillConstant(TSList=AllTS,ConstantValue=0)
#
# Read a list of Alternate Point Structures (ID col = 2) and set to zero using same approach as above
CreateFromList(ListFile="Sp2008L_AlternatePoint.csv",IDCol=2,DataType="WellPumping",Interval=Month,
    InputType=StateMod,InputName="x",HandleMissingTShow=DefaultMissingTS)
FillConstant(TSList=AllTS,ConstantValue=0)
#
# Now read time series from the historical well pumping file produced by StateCU.
# Don't read the time series file directly because it contains diversions and
# wells. Instead, read the list of well stations with CreateFromList() and
# specify the well pumping time series file to read. This takes a little
# longer to run because the time series file is opened and read for each ID
# in the list, but at least only the wells are added as time series.
#
CreateFromList(ListFile="..\Network\Sp2008L_Wells.csv",IDCol=1,DataType="WellPumping",Interval=Month,
    InputType=StateMod,InputName="..\StateCU\Historic\Sp2008L.gwp",HandleMissingTShow=DefaultMissingTS)
#
# Now output all of the time series
#
SortTimeSeries()
WriteStateMod(TSList=AllTS,OutputFile="Sp2008L.weh")
WriteStateMod(TSList=AllTS,OutputFile="..\StateMod\Historic\Sp2008L.weh")
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="Sp2008L_WEH.TSTool.check.html")
```

Commands are available in StateDMI to support alternative approaches. The **Commands...Well Data...Well Historical Pumping TS (Monthly)** menu items insert commands to process well historical pumping time series (monthly) data:

<b>Well Historical Pumping TS (Monthly) - Commands</b>
SetOutputPeriod() ... SetOutputYearType() ...
ReadWellStationsFromList() ... ReadWellStationsFromStateMod() ...
SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
ReadWellHistoricalPumpingTSMonthlyFromStateCU() ...
SetWellHistoricalPumpingTSMonthly() ... SetWellHistoricalPumpingTSMonthlyConstant() ...
FillWellHistoricalPumpingTSMonthlyAverage() ... FillWellHistoricalPumpingTSMonthlyConstant() ... 1: ReadPatternFile() ... 2: FillWellHistoricalPumpingTSMonthlyPattern() ...
1: ReadWellRightsFromStateMod() ... 2: LimitWellHistoricalPumpingTSMonthlyToRights() ...
SortWellHistoricalPumpingTSMonthly() ... WriteWellHistoricalPumpingTSMonthlyToStateMod() ...
SetWellStationCapacitiesFromTS() ... SetWellStation() ... SetWellStationsFromList() ...
WriteWellStationsToStateMod() ...
CheckWellHistoricalPumpingTSMonthly() ... WriteCheckFile() ...

MenuCommands\_WellHistoricalPumpingTSMonthly

### **Commands...Well Data...Well Historical Pumping TS (Monthly) Menu**



The following table summarizes the use of each command.

### Well Historical Pumping TS (Monthly) Commands

Command	Description
SetOutputPeriod()	Set the output period. Time series are automatically extended to this period if necessary.
SetOutputYearType()	Set the output year type, which is used when writing the files and for determining the monthly efficiency order in station data.
ReadWellStationsFromList()	Read from a delimited file the list of well stations to be included in the data set.
ReadWellStationsFromStateMod()	Read from a StateMod well stations file the list of well stations to be included in the data set.
SetWellAggregate()	Specify that a well station is an aggregate and define its parts.
SetWellAggregateFromList()	Specify that one or more well stations are aggregates and define their parts, using a delimited list file.
SetWellSystem()	Specify that a well station is a system and define its parts.
SetWellSystemFromList()	Specify that one or more well stations are systems and define their parts, using a delimited list file.
ReadWellHistoricalPumpingTS MonthlyFromStateCU()	Read well historical pumping time series (monthly) from a StateCU results file (StateMod time series format), when directly manipulating an existing file.
SetWellHistoricalPumpingTSMonthly()	Set the data for a well historical pumping time series (monthly). This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
SetWellHistoricalPumpingTS MonthlyConstant()	Set the data for a well historical pumping time series (monthly) to a constant value. This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
FillWellHistoricalPumpingTS MonthlyAverage()	Fill missing data in well historical pumping time series (monthly) to the historical monthly average values. If an aggregate/system, the historical average is computed from the total.
FillWellHistoricalPumpingTS MonthlyConstant()	Fill missing data in well historical pumping time series (monthly) to a constant value.
ReadPatternFile()	Read the pattern file used with FillWellHistoricalPumpingTS MonthlyPattern() commands.
FillWellHistoricalPumpingTS MonthlyPattern()	Fill missing data in well historical pumping time series (monthly) to the monthly average values, using wet/dry/average values.

Command	Description
ReadWellRightsFromStateMod()	Read well rights from a StateMod file, used to limit the time series to rights.
LimitWellHistoricalPumpingTSMonthlyToRights()	Limit the well historical pumping time series (monthly) to the water rights that were available at each point in time.
SortWellHistoricalPumpingTSMonthly()	Sort the well historical pumping time series (monthly). This is useful to force consistency between files.
WriteWellHistoricalPumpingTSMonthlyToStateMod()	Write well historical pumping time series (monthly) to a StateMod file.
SetWellStationCapacitiesFromTS()	Set well station capacities from historical pumping time series maximum values, to update the well station data.
SetWellStation()	Set well station data (for example to override capacities from time series).
SetWellStationsFromList()	Set well station data (for example to override capacities from time series).
WriteWellStationsToStateMod()	Write well stations to a StateMod file (if the stations have been updated).
CheckWellHistoricalPumpingTSMonthly()	Check well historical pumping time series data for problems.
WriteCheckFile()	Write the results of data checks to a file.

#### 5.9.4 Well Historical Pumping Time Series (Daily)

StateDMI does not process daily well pumping time series. Instead, use StateCU output, TSTool, a spreadsheet, or other software to prepare the time series file.

#### 5.9.5 Well Demand Time Series (Monthly)

Well demand time series (monthly) correspond to each well station, using the station identifier to relate the data. Demands for well stations that supplement diversion stations are typically associated with the diversion stations and are processed by commands described in **Section 5.4.5 – Diversion Demand Time Series (Monthly)**.

The following example TSTool file illustrates how historical well pumping time series can be used for the historical demand case (from the preliminary South Platte Sp2008L data set):

```
#
# Wells_Wem.TsTool; command used to create a historic well demand file
# from a historic pumpinng file
SetOutputPeriod(OutputStart="1950-01",OutputEnd="2006-12")
ReadStateMod( InputFile="Sp2008L.wem" )
#
SortTimeSeries()
WriteStateMod(TSList=AllTS,OutputFile="Sp2008L.wem")
WriteStateMod(TSList=AllTS,OutputFile="..\StateMod\Historic\Sp2008L.wem")
CheckTimeSeries(CheckCriteria="Missing")
WriteCheckFile(OutputFile="Sp2008L_WEH.TSTool.check.html")
```

StateDMI also provides commands to process the well demand time series, should an approach different from the above be required. The **Commands...Well Data...Well Demand TS (Monthly)** menus insert commands to process well demand time series (monthly) data (and optionally the well stations file, to save estimated efficiencies):

<b>Well Demand TS (Monthly) - Commands</b>
SetOutputPeriod() ... SetOutputYearType() ...
ReadWellStationsFromList() ... ReadWellStationsFromStateMod() ...
SetWellAggregate() ... SetWellAggregateFromList() ... SetWellSystem() ... SetWellSystemFromList() ...
ReadWellDemandTSMonthlyFromStateMod() ...
1: ReadIrrigationWaterRequirementTSMonthlyFromStateCU() ... 2: ReadWellHistoricalPumpingTSMonthlyFromStateMod() ... 3: CalculateWellStationEfficiencies() ... SetWellStation() ... SetWellStationsFromList() ... WriteWellStationsToStateMod() ...
CalculateWellDemandTSMonthly() ... CalculateWellDemandTSMonthlyAsMax() ...
SetWellDemandTSMonthly() ... SetWellDemandTSMonthlyConstant() ...
FillWellDemandTSMonthlyAverage() ... FillWellDemandTSMonthlyConstant() ... 1: ReadPatternFile() ... 2: FillWellDemandTSMonthlyPattern() ...
ReadWellRightsFromStateMod() ... LimitWellDemandTSMonthlyToRights() ...
SortWellDemandTSMonthly() ... WriteWellDemandTSMonthlyToStateMod() ...
CheckWellDemandTSMonthly() ... WriteCheckFile() ...

MenuCommands\_WellDemandTSMonthly

### Commands...Well Data...Well Demand TS (Monthly) Menu

The following table summarizes the use of each command:

### Well Demand Time Series (Monthly) Commands

Command	Description
SetOutputPeriod()	Set the output period. Time series are automatically extended to this period if necessary.
SetOutputYearType()	Set the output year type, which is used when writing the files.
ReadWellStationsFromList()	Read from a delimited file the list of well stations to be included in the data set.
ReadWellStationsFromStateMod()	Read from a StateMod well stations file the list of well stations to be included in the data set.
SetWellAggregate()	Specify that a well station is an aggregate and define its parts.
SetWellAggregateFromList()	Specify that one or more well stations are aggregates and define their parts, using a delimited list file.
SetWellSystem()	Specify that a well station is a system and define its parts.
SetWellSystemFromList()	Specify that one or more well stations are systems and define their parts, using a delimited list file.
ReadWellDemandTSMonthlyFromStateMod()	Read the well demand time series from a StateMod file (if manipulating an existing file).
ReadIrrigationWaterRequirementTSMonthlyFromStateCU()	Read irrigation water requirement (IWR) time series generated by the StateCU model.
ReadWellHistoricalPumpingTSMonthlyFromStateMod()	Read well historical pumping time series (monthly) from a StateMod file (can also read a StateCU file).
CalculateWellStationEfficiencies()	Calculate well station average efficiencies as IWR/Diversions.
SetWellStation()	Set well station data, in particular efficiency data, to override the result from the previous command.
SetWellStationsFromList()	Set well station data from a delimited file, in particular efficiency data, to override the result from the previous command.
WriteWellStationsToStateMod()	Write well stations to StateMod – the data will include updated average efficiencies.
CalculateWellDemandTSMonthly()	Calculate the well demand time series (monthly) using IWR/ $Eff_{ave}$ and historical pumping time series.
CalculateWellDemandTSMonthlyAsMax()	Calculate the well demand time series (monthly) as the maximum of the demand (see previous command) and the well historical pumping time series.

Command	Description
SetWellDemandTSMonthly()	Set the data for a well demand time series (monthly). This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
SetWellDemandTSMonthlyConstant()	Set the data for a well demand time series (monthly) to monthly constant values. This cannot be used to set the data for an aggregate/system part (only the aggregate/system total can be set).
FillWellDemandTSMonthlyAverage()	Fill missing data in well demand time series (monthly) to the monthly average values. If an aggregate/system, the average is computed from the total.
FillWellDemandTSMonthlyConstant()	Fill missing data in well demand time series (monthly) to a constant value.
ReadPatternFile()	Read the pattern file used with FillWellDemandTSMonthlyPattern() commands.
FillWellDemandTSMonthlyPattern()	Fill missing data in well demand time series (monthly) to the monthly average values, using wet/dry/average values.
ReadWellRightsFromStateMod()	Read well rights from a StateMod file, used to limit the time series to rights.
LimitWellDemandTSMonthlyToRights()	Limit the well demand time series (monthly) to the water rights that were available at each point in time.
SortWellDemandTSMonthly()	Sort the well demand time series (monthly). This is useful to force consistency between files.
WriteWellDemandTSMonthlyToStateMod()	Write well demand time series (monthly) to a StateMod file.
CheckWellDemandTSMonthly()	Check well demand time series data for problems.
WriteCheckFile()	Write the results of data checks to a file.

### 5.9.6 Irrigation Practice Time Series (Yearly)

The irrigation practice time series for well and diversion stations is typically copied from the StateCU data files or the StateCU file is directly referenced.

### 5.9.7 Consumptive Water Requirement (Monthly, Daily)

The consumptive water requirement for well and diversion stations is typically copied from the StateCU output or the StateCU file is directly referenced.

### 5.9.8 Soil Moisture Time Series (Yearly)

The soil moisture time series are stored in the same file as for diversion stations. See the StateMod documentation for more information.

## 5.10 Stream Estimate Data

Stream estimate data consists of:

- Stream estimate stations
- Stream estimate coefficients
- Natural flow time series (monthly, daily)

Each of the above data types is stored in a separate file, using the stream estimate station identifier as the primary identifier. Stream estimate stations correspond to locations where historical data are not available, and instead streamflow is estimated by prorating gaged flows from other locations. StateMod now supports separate stream gage (see **Section 5.2 – Stream Gage Data**) and stream estimate data; however, stream estimate stations are often still mixed with stream gage stations in one station file (\*.ris), and stream estimate stations are indicated by stations that have data in the stream estimate coefficients file. Stream estimate stations can correspond to existing diversion, reservoir, or well nodes, or can correspond to the “other” node type, which will only have data in the network file and the stream estimate files. In other words, a stream estimate “station” is a modeling term but does not correspond to a typical station at which measurements are recorded. It is possible that a stream gage station has insufficient historical data to serve as a true stream gage but the location of the gage is important in the model. In this situation, the stream gage should not be identified as a flow node in the network but should instead be identified as an “other” node type that is also a natural flow node. To support previous modeling conventions, StateDMI allows combined stream gage/estimate stations lists in the stream stations file (\*.ris).

### 5.10.1 Stream Estimate Stations

The **Commands...Stream Estimate Data...Stream Estimate Stations** menus insert commands to process stream estimate station data (in general, the features are very similar to the stream gage stations):

Stream Estimate Stations - Commands
ReadStreamEstimateStationsFromList() ... ReadStreamEstimateStationsFromNetwork() ... ReadStreamEstimateStationsFromStateMod() ...
SetStreamEstimateStation() ...
SortStreamEstimateStations() ...
FillStreamEstimateStationsFromHydroBase() ... 1: ReadNetworkFromStateMod() ... 2: FillStreamEstimateStationsFromNetwork() ... FillStreamEstimateStation() ...
WriteStreamEstimateStationsToList() ... WriteStreamEstimateStationsToStateMod() ...
CheckStreamEstimateStations() ... WriteCheckFile() ...

MenuCommands\_StreamEstimateStations

### Commands...Stream Estimate Data...Stream Estimate Stations Menu

The following table summarizes the use of each command:

### Stream Estimate Station Commands

Command	Description
<code>ReadStreamEstimateStationsFromList()</code>	Read from a delimited list file the list of stream estimate stations to be included in the data set.
<code>ReadStreamEstimateStationsFromNetwork()</code>	Read from a StateMod network file a list of stream estimate stations to be included in the data set.
<code>ReadStreamEstimateStationsFromStateMod()</code>	Read from a StateMod stream estimate stations file the list of stream estimate stations to be included in the data set.
<code>SetStreamEstimateStation()</code>	Set the data for, and optionally add, stream estimate stations.
<code>SortStreamEstimateStations()</code>	Sort the stream estimate stations. This is useful to force consistency between files.
<code>FillStreamEstimateStationsFromHydroBase()</code>	Fill missing data for defined stream estimate stations, using data from HydroBase. For example, retrieve the station names.
<code>ReadNetworkFromStateMod()</code>	Read the network file, providing data for the <code>FillStreamEstimateStationsFromNetwork()</code> command.
<code>FillStreamEstimateStationsFromNetwork()</code>	Fill missing data for defined stream estimate stations, using data from a StateMod network file. This is useful when the station names are not found in HydroBase and numerous <code>SetStreamEstimateStation()</code> commands would otherwise be required.
<code>FillStreamEstimateStation()</code>	Fill missing data for defined stream estimate stations, user user-supplied values.
<code>WriteStreamEstimateStationsToList()</code>	Write defined stream estimate stations to a delimited file.
<code>WriteStreamEstimateStationsToStateMod()</code>	Write defined stream estimate stations to a StateMod file.
<code>CheckStreamEstimateStations()</code>	Check stream estimate station data for problems.
<code>WriteCheckFile()</code>	Write the results of data checks to a file.

An example command file to create the stream estimate station file is shown below (adapted from Rio Grande data set but not implemented in production because a combined gage/estimate *\*.ris* file was used, note that some comments indicate the stations that are present in the stream gage station file):

```
# Create the Rio Grande Stream Estimate Stations File
#
# Note some stations that were in the original RIS file are now in the SES file.
#
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\StateMod\rgTW.net")
#
# Fill in the name from HydroBase...
#
FillStreamEstimateStationsFromHydroBase(ID="*",NameFormat="StationName_NodeType")
# Set specific data, including name and daily ID overrides.
#
# Set key gages to include actual daily observations
#
/* Stream gages are in the RIS
```

```

SetStreamGageStation(ID="08213500",Name="RG:THIRTYMILEBRG",DailyID="08213500",IfNotFound=Warn)
... similar commands omitted...
*/
#
# Set these key gages to use higher gages (less depletion) in the estimation
# of baseflows
#
/* Stream gages are in the RIS
SetStreamGageStation(ID="08221500",Name="RG:MONTEVISTA",DailyID="08220000",IfNotFound=Warn)
...similar commands omitted...
*/
#
# Set all diversion point flow stations to key gages in their water districts
#
SetStreamEstimateStation(ID="20*",DailyID="08220000",IfNotFound=Warn)
...similar commands omitted...
# Set gage flow stations to key gages in their general area
/* Stream gages are in the RIS
SetStreamGageStation(ID="08214500",Name="NCLEAR:BLWCONTRES",DailyID="08220000",IfNotFound=Warn)
...similar commands omitted...
*/
SetStreamEstimateStation(ID="VenAbvSan",Name="VenteroFlowAboveS",DailyID="08242500",
    IfNotFound=Warn)
/* Stream gages are in the RIS
SetStreamGageStation(ID="08250000",Name="CULEBRA:SANLUIS",DailyID="08242500",IfNotFound=Warn)
...similar commands omitted...
#
SetStreamGageStation(ID="McIntyreSpr",Name="MCINTYRESRINGS",DailyID="0",IfNotFound=Warn)
...similar commands omitted...
*/
SetStreamEstimateStation(ID="BrkCrBFL",Name="BROOKCREEK",DailyID="08227500",IfNotFound=Warn)
...similar commands omitted...
# Set all no flow nodes to use zero daily ID
/* Stream gages are in the RIS
SetStreamGageStation(ID="ROCKCRNF",Name="ROCKCRK-NF",DailyID=0,IfNotFound=Warn)
...similar commands omitted...
*/
# Fill in the name from the network (anything not filled or set above)...
#
FillStreamEstimateStationsFromNetwork(ID="*",NameFormat="StationName_NodeType")
# Write the output...
WriteStreamEstimateStationsToStateMod(OutputFile="..\StateMod\rgTW.ses",
    WriteHow="OverwriteFile")
CheckStreamEstimateStations(ID="*")
WriteCheckFile(OutputFile="ses.commands.StateDMI.check.html")

```

An example command file to create a combined stream gage/estimate stations file is shown in **Section 5.2 – Stream Gage Data**.

### 5.10.2 Stream Estimate Coefficients

The stream estimate coefficients data indicate information to estimate natural streamflow at ungaged locations, by prorating flows at stream gages. Each stream estimate station has data in the stream estimate coefficients file. Proration factors may be any coefficient but are generally developed from drainage area and precipitation data. During model calibration, the proration factors may be modified using `SetStreamEstimateCoefficients()` commands.



The **Commands...Stream Estimate Data...Stream Estimate Coefficients** menus insert commands to process stream estimate coefficient data:

Stream Estimate Coefficients - Commands
ReadStreamEstimateCoefficientsFromStateMod() ...
ReadStreamEstimateStationsFromList() ...
ReadStreamEstimateStationsFromNetwork() ...
ReadStreamEstimateStationsFromStateMod() ...
SortStreamEstimateStations() ...
SetStreamEstimateCoefficientsPFGage() ...
CalculateStreamEstimateCoefficients() ...
SetStreamEstimateCoefficients() ...
WriteStreamEstimateCoefficientsToList() ...
WriteStreamEstimateCoefficientsToStateMod() ...
CheckStreamEstimateCoefficients() ...
WriteCheckFile() ...

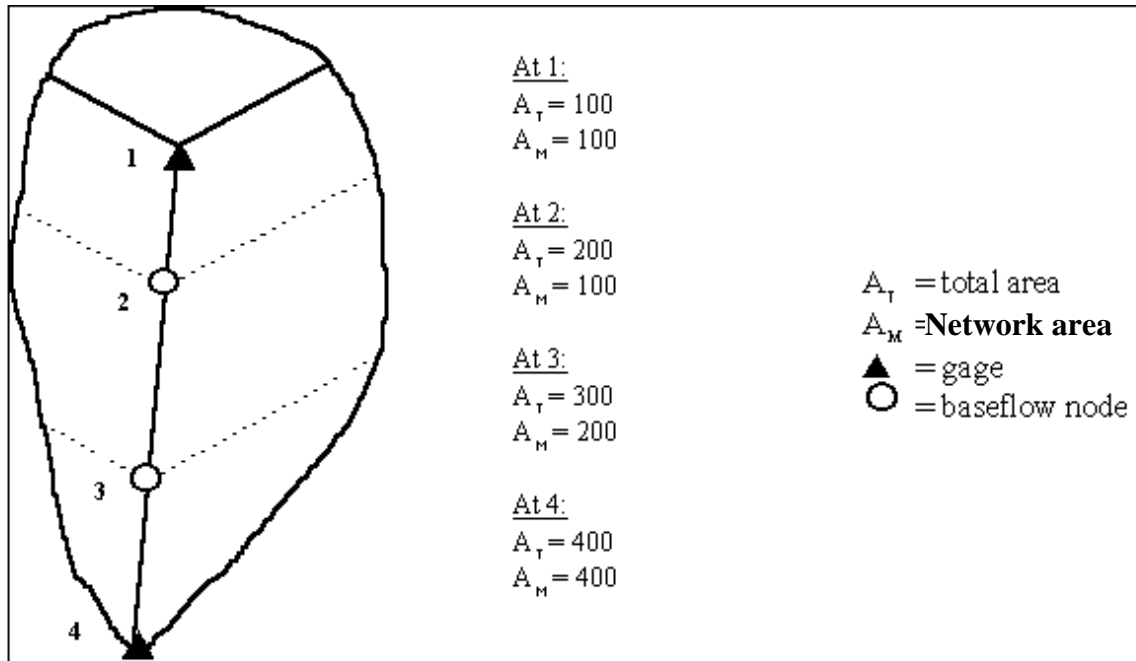
MenuCommands\_StreamEstimateCoefficients

#### **Commands...Stream Estimate Data...Stream Estimate Coefficients Menu**

Note that although the stream estimate stations are the primary data component related to stream estimate coefficients, the list of stream estimate stations is typically read from the network. This is because the network file includes data necessary to process the coefficients, including upstream/downstream relationships and the area\*precipitation information. Area and precipitation data are typically developed using GIS tools – currently StateDMI does not estimate these values.

The area and precipitation data provided at a stream gage station represents the total area and average precipitation for that drainage area. The area and precipitation provided at a stream estimate station represent the incremented area between that station and the next upstream stream gage station. In the following figure, the incremented area between each node is 100 units. As presented, the total area (and the area specified in the generalized network file) is equal for the stream gage station. At stream estimate stations (i.e., natural flow nodes, also previously called baseflow nodes), the area is the incremental area from the upstream stream gage (or gages) to the stream estimate station (or the headwater area if there is no upstream gage in the basin): 100 units at point 2 and 200 units at point 3.

EXAMPLE:



The default method used to provide natural flow data to StateMod is the Gain Approach, which includes two terms: the upstream gage term and the gain term. The **StateMod Users' Manual** discusses this equation in detail. The following summarizes the method StateDMI's

CalculateStreamEstimateCoefficients() command uses to provide natural flow information to StateMod, using the Gain Approach.

- To estimate the gaged flow term, StateDMI determines all of the stream gage stations that have area-precipitation data upstream of the stream estimate station. This results in a list of upstream stream gage stations with a coefficient of +1 being entered on the first line of the stream estimate coefficients file.
- To estimate the gaged component of the gain term, StateDMI identifies the downstream stream gage stations and all upstream stream gage stations. This results in the downstream stream gage station being assigned a coefficient of +1 and all the upstream stream gage stations being assigned a coefficient of -1 on the second line of the stream estimate coefficients file.
- To estimate the proration factor component of the gain term, StateDMI calculates the ratio of the incremental area-precipitation at the stream estimate station divided by the ungaged area-precipitation (the downstream stream gage station area-precipitation minus the upstream stream gage station area-precipitation).

- In the example above, the Gain approach results in the following at point 2:

Gage term = stream gage station flow at point 1

Gain term =  $100/(400-100) * (\text{flow (gage) at point 4} - \text{flow (gage) at point 1})$ .

The Neighboring Gage Approach is a second method that may be used to provide baseflow data to headwater nodes only. Like the Gain Approach, this technique includes two terms: the upstream gage term and the gain term. Implement this approach by using

`SetStreamEstimateCoefficientsPFGage(ID,GageID)` commands. The Neighboring Gage approach is commonly used at headwater stream estimate stations to provide an ungaged hydrograph that is more representative than the hydrograph from the Gain Approach. The following summarizes the method used by StateDMI to provide natural flow information to StateMod using the Neighboring Gage Approach:

- Because the gaged flow term is, by definition, null for a headwater node, no upstream stream gage station values are entered on the first line of the stream estimate coefficients file.
- To estimate the gaged component of the gain term, StateDMI provides the neighboring gage assigned by the user a coefficient of +1 on the second line of the stream estimate coefficients file.
- To estimate the proration factor component of the gain term, StateDMI calculates the ratio of the area precipitation data provided at the natural flow node divided by the area-precipitation at stream gage station assigned by the user.
- Assuming point 1 of the example is not a gage but instead is stream estimate station using the Neighboring Gage Approach, then a typical application to a headwater node at point 1 would result in the following (if the neighboring gage selected was at point 4):

Gage term = none

Gain term =  $100/400 * (\text{flow (gage) at point 4})$

Note that when the Neighboring Gage approach is applied to a headwater node, StateDMI treats that stream estimate station as if it were a gage. Therefore when the Neighboring Gage approach is used, the area and precipitation data at any other stream estimate stations in the reach must be adjusted to reflect that the aforementioned stream estimate station has taken on the characteristics of a gage. For the example, where point 1 was a stream estimate station that uses the Neighboring Gage approach, the area and precipitation assigned to points 2 and 3 would be altered as if a gage existed at point 1.

For example, to tie the characteristics of stream estimate station 360345 to stream gage station 09053000, use the following command before the `CalculateStreamEstimateCoefficients()` command:

```
SetStreamEstimateCoefficientsPFGage(360345,09053000)
```

Use `SetStreamEstimateCoefficients()` commands, as appropriate, to edit the calculated proration factor and coefficients. This is particularly useful if the calculated proration factor does not accurately represent the hydrology at that baseflow node. For example, to adjust the proration factor for stream estimate station 384625 to 0.6:

```
SetStreamEstimateCoefficients(ID="384625",ProrationFactor=0.6)
```

The following table summarizes the use of each command:

### Stream Estimate Coefficients Commands

Command	Description
<code>ReadStreamEstimateCoefficientsFromStateMod()</code>	Read stream estimate coefficients from a StateMod file, which is useful if simple manipulation is occurring.
<code>ReadStreamEstimateStationsFromList()</code>	Read from a delimited file a list of stream estimate stations to be included in the data set.
<code>ReadStreamEstimateStationsFromNetwork()</code>	Read from a StateMod network file a list of stream estimate stations to be included in the data set.
<code>ReadStreamEstimateStationsFromStateMod()</code>	Read from a StateMod stream estimate (or gage) stations file a list of stream estimate stations to be included in the data set.
<code>SortStreamEstimateStations()</code>	Sort the stream estimate stations. The default when read from the network is top to bottom; however this order may be difficult to interpret or compare with other files and therefore sorting is useful.
<code>SetStreamEstimateCoefficientsPFGage()</code>	Specify that the proration factor for a stream estimate station should be calculated using only the area*precipitation data for a specific gage, rather than the downstream node. The station is then treated as if were a stream gage node for other base flow calculations.
<code>CalculateStreamEstimateCoefficients()</code>	Calculate stream estimate coefficients from the network relationships and the area*precipitation data stored in the network for stream estimate stations.
<code>SetStreamEstimateCoefficients()</code>	Set stream estimate coefficients, for example, if the values that were calculated need to be adjusted.
<code>WriteStreamEstimateCoefficientssToList()</code>	Write defined stream estimate coefficients to a delimited file.
<code>WriteStreamEstimateCoefficientssToStateMod()</code>	Write defined stream estimate coefficients to a StateMod file.
<code>CheckStreamEstimateCoefficients()</code>	Check stream estimate coefficients data for problems.
<code>WriteCheckFile()</code>	Write the results of data checks to a file.

An example command file to create the stream estimate coefficients file is shown below (from the Colorado cm2005 data set):

```
StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#           this baseflow nodes are generally on smaller non-gaged tribs and have
#           different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="360801",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="362002",GageID="09054000")
SetStreamEstimateCoefficientsPFGage(ID="360829",GageID="09047500")
SetStreamEstimateCoefficientsPFGage(ID="381441",GageID="09075700")
SetStreamEstimateCoefficientsPFGage(ID="382013",GageID="09075700")
SetStreamEstimateCoefficientsPFGage(ID="380959",GageID="09075700")
SetStreamEstimateCoefficientsPFGage(ID="381104",GageID="09075700")
SetStreamEstimateCoefficientsPFGage(ID="BaseFlow",GageID="09091500")
SetStreamEstimateCoefficientsPFGage(ID="450632",GageID="09092600")
SetStreamEstimateCoefficientsPFGage(ID="450685",GageID="09089500")
SetStreamEstimateCoefficientsPFGage(ID="450810",GageID="09089500")
SetStreamEstimateCoefficientsPFGage(ID="450788",GageID="09089500")
SetStreamEstimateCoefficientsPFGage(ID="500601",GageID="09041200")
SetStreamEstimateCoefficientsPFGage(ID="500627",GageID="09041200")
SetStreamEstimateCoefficientsPFGage(ID="510594",GageID="09026500")
SetStreamEstimateCoefficientsPFGage(ID="510728",GageID="09032000")
SetStreamEstimateCoefficientsPFGage(ID="510941",GageID="09033500")
SetStreamEstimateCoefficientsPFGage(ID="512061",GageID="09039000")
SetStreamEstimateCoefficientsPFGage(ID="520658",GageID="09060500")
SetStreamEstimateCoefficientsPFGage(ID="522006",GageID="09060500")
SetStreamEstimateCoefficientsPFGage(ID="530883",GageID="09060500")
SetStreamEstimateCoefficientsPFGage(ID="530632",GageID="09071300")
SetStreamEstimateCoefficientsPFGage(ID="530585",GageID="09085200")
SetStreamEstimateCoefficientsPFGage(ID="531051",GageID="09085200")
SetStreamEstimateCoefficientsPFGage(ID="720649",GageID="09097500")
SetStreamEstimateCoefficientsPFGage(ID="720580",GageID="09097500")
SetStreamEstimateCoefficientsPFGage(ID="720557",GageID="09104500")
SetStreamEstimateCoefficientsPFGage(ID="09104000",GageID="09104500")
SetStreamEstimateCoefficientsPFGage(ID="09101500",GageID="09104500")
SetStreamEstimateCoefficientsPFGage(ID="953800",GageID="09097500")
SetStreamEstimateCoefficientsPFGage(ID="720816",GageID="09104500")
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374641",ProrationFactor=0.200,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374648",ProrationFactor=0.350,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="380880",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="381594",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="384617",ProrationFactor=0.700,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510639",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514603",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514620",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510728",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530555",ProrationFactor=0.180,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530678",ProrationFactor=0.230,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="531082",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="954683",ProrationFactor=0.400,IfNotFound=Warn)
#
# Step 5 - create streamflow estimate coefficient file
```

```
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")
```

### 5.10.3 Stream Estimate Natural Flow Time Series (Monthly, Daily)

Stream estimate natural flow time series for stream estimate stations are not processed by StateDMI. Instead, use StateMod's baseflow module, TSTool, or other software to create monthly and daily base flow time series files.

Refer to **Section 5.2.3 – Stream Natural Flow Time Series (Monthly, Daily)** for more information.

## 5.11 River Network Data

River network data consists of:

- Network (used by StateDMI, StateMod GUI)
- River network (used by StateMod)

The river network file used by StateMod is a relatively simple file, containing river node identifiers, river node name, and downstream river node. The purpose of this file is to indicate the upstream to downstream connectivity of model nodes. StateMod is designed to allow the river node identifiers to be different from station identifiers. This allows, for example, multiple diversion stations to be located at the same river node. Much of the detailed StateMod output is by river node, using river node identifiers. However, StateDMI and the StateMod GUI enforce the convention that the river node identifiers are the same as station identifiers in other files and allow only a single station at a river node. This tends to minimize data errors and confusion with identifiers. To be consistent with StateMod documentation and nomenclature, the term “river node” is used in some parts of StateDMI where a river node identifier is needed in addition to a station identifier.

The network data are listed after other data components in this documentation and StateDMI menus, mainly because network data (and the operational data described in **Section 5.12**) are related to higher-level modeling concepts and do not need to be completely addressed until other data files are created.

There are two main ways that network data will be created:

1. An existing data set and model network are available and only minor changes to the network need to occur. In this case, the StateDMI interface can be used to insert, delete, or modify nodes to make minor edits to the network. Lists of stations can then be read from the network in order to process all data files. This is the approach taken throughout the current StateDMI documentation.
2. Data files are created starting with lists of identifiers. For example, for a new data set, rather than developing a network file and reading station identifiers from the network, simple delimited files with lists of identifiers are used. These lists of station identifiers could be created by StateView or other software. StateDMI supports this approach by allowing list files to be provided when creating station files. Subsequent processing (e.g., for water rights and time series files), rely on the StateMod format station files. This approach has not been fully implemented due to limited resources and because of a number of technical issues. For example, some data file processing requires that the network file be available (e.g., assigning default return flow locations). Also, StateDMI does not currently provide the capability to display lists of stations to allow upstream to downstream relationships to be defined. This capability is planned for the future. Consequently,

the only alternative at this time is to create a model network using the StateDMI interface. Because efforts have focused on developing StateDMI to support existing modeling efforts, option 1 is fully supported. When the list-based approach is fully supported, a network file would be initialized using the list data and then data set maintenance would revert to option 1.

The generalized network file contains more data than the simple StateMod river network file:

- node locations, label position, and other information for the network diagram
- area and precipitation information used to calculate proration factors in the stream estimate coefficients file (see **Section 5.10.2 – Stream Estimate Coefficients**)
- indicators for natural flow (stream estimate) nodes

The generalized network file was previously hand-edited and used as input to the Makenet program. StateDMI generally allows reading the old Makenet file. However, the new version is in XML format and allows for greater flexibility. See **Section 3.6.1 – Updating an Old Makenet Format to New Format** for information about how to interactively convert an old network file to the new format. These steps can also be performed using commands, as described below. StateMod only reads the river network file. However, the generalized network file is needed by StateDMI and StateMod GUI. Therefore, effort should be taken to keep these files synchronized. In general, all network editing should use the generalized network and the StateMod river network file should be created from this file.

The processing of each data file is discussed below.

#### 5.11.1 Network (used by StateDMI, StateMod GUI)

**These features are under development but are made available for evaluation.**

The **Commands...River Network Data...Network** menus insert commands to process the generalized network file:

Network (used by StateDMI, StateMod GUI) - Commands
ReadNetworkFromStateMod() ...
1: ReadRiverNetworkFromStateMod() ...
2: ReadStreamGageStationsFromStateMod() ...
3: ReadDiversionStationsFromStateMod() ...
4: ReadReservoirStationsFromStateMod() ...
5: ReadInstreamFlowStationsFromStateMod() ...
6: ReadWellStationsFromStateMod() ...
7: ReadStreamEstimateStationsFromStateMod() ...
8: CreateNetworkFromRiverNetwork() ...
FillNetworkFromHydroBase() ...
WriteNetworkToStateMod() ...
PrintNetwork() ...

MenuCommands\_Network

#### Commands...River Network Data...Network Menu

These commands can be used to convert a StateMod river network file to a generalized network file – this may be necessary when StateDMI or Makenet was not used to create the StateMod river network file. The StateMod river network file contains limited information about nodes and therefore information must be read from other StateMod files. For most data sets, river network commands can be used instead (see the next section).

The following table summarizes the use of each command:

**Network Commands**

Command	Description
<code>ReadNetworkFromStateMod()</code>	Read the generalized network from an XML network file (for manipulation).
<code>ReadRiverNetworkFromStateMod()</code>	Read the river network from a StateMod river network file.
<code>ReadStreamGageStationsFromStateMod()</code>	Read stream gage stations from a StateMod stream gage stations file.
<code>ReadDiversionStationsFromStateMod()</code>	Read diversion stations from a StateMod diversion stations file.
<code>ReadReservoirStationsFromStateMod()</code>	Read reservoir stations from a StateMod reservoir stations file.
<code>ReadInstreamFlowStationsFromStateMod()</code>	Read instream flow stations from a StateMod instream flow stations file.
<code>ReadWellStationsFromStateMod()</code>	Read well stations from a StateMod well stations file.
<code>ReadStreamEstimateStationsFromStateMod()</code>	Read stream estimate stations from a StateMod stream estimate stations file.
<code>CreateNetworkFromRiverNetwork()</code>	Create the generalized network from the StateMod river network and information from stations files.
<code>FillNetworkFromHydroBase()</code>	Fill missing network information using HydroBase.
<code>WriteNetworkToStateMod()</code>	Write the generalized network to a StateMod XML network file.
<code>PrintNetwork()</code>	Print all or a subset of the network diagram.

An example command file to create the generalized network file is shown below (contrived example not validated in production):

```
# This commands file creates a generalized network file from an existing
# StateMod network file (RIN).
#
ReadRiverNetworkFromStateMod("../StateMod\rgTW_orig.rin")
ReadStreamGageStationsFromStateMod("../StateMod\rgTW.ris")
ReadDiversionStationsFromStateMod("../StateMod\rgTW.dds")
ReadReservoirStationsFromStateMod("../StateMod\rgTW.res")
ReadInstreamFlowStationsFromStateMod("../StateMod\rgTW.ifs")
ReadWellStationsFromStateMod("../StateMod\rgTW.wes")
ReadStreamEstimateStationsFromStateMod("../StateMod\rgTW.ses")
CreateNetworkFromRiverNetwork()
FillNetworkFromHydroBase(LocationEstimate="Interpolate")
WriteNetworkToStateMod(OutputFile="../StateMod\rgtw.net",WriteHow="OverwriteFile")
```



After the above commands are run, the resulting network file can be edited with the network editor to add stream labels, adjust node positions as necessary, and specify area/precipitation data that can be used when processing stream estimate coefficients.

### 5.11.2 River Network (used by StateMod)

The **Commands...River Network Data...River Network** menus insert commands to process the StateMod river network file:

<b>River Network (used by StateMod) - Commands</b>
ReadNetworkFromStateMod() ...
CreateRiverNetworkFromNetwork() ...
SetRiverNetworkNode() ...
FillRiverNetworkFromHydroBase() ...
FillRiverNetworkFromNetwork() ...
FillRiverNetworkNode() ...
WriteRiverNetworkToList() ...
WriteRiverNetworkToStateMod() ...
CheckRiverNetwork() ...
WriteCheckFile() ...

MenuCommands\_RiverNetwork

#### Commands...River Network Data...River Network Menu

These commands are used to convert the generalized network file, which StateDMI uses for the network diagram, into the StateMod river network, which contains a subset of the data and is used by StateMod. The following table summarizes the use of each command:

#### River Network Commands

<b>Command</b>	<b>Description</b>
ReadNetworkFromStateMod()	Read the generalized network from a StateMod XML (or old Makenet) network file.
CreateRiverNetworkFromNetwork()	Create the StateMod river network from the generalized network data. Node names are not, by default taken from the network because they may have been adjusted from database names to facilitate labeling or presentation.
FillRiverNetworkFromHydroBase()	Fill river network data (e.g., station names) from HydroBase. This allows “official” names to be used in the river network file, rather than those used in the generalized network file.
FillRiverNetworkFromNetwork()	Fill river network data (e.g., station names) from the generalized network. This allows names that are not in HydroBase (e.g., from aggregate nodes) to be taken from the network file.
FillRiverNetworkNode()	Fill river network data (e.g., station names) from user supplied values.

Command	Description
WriteRiverNetworkToList()	Write the river network to a delimited file.
WriteRiverNetworkToStateMod()	Write the river network to a StateMod river network file.
CheckRiverNetwork ()	Check river network data for problems.
WriteCheckFile()	Write the results of data checks to a file.

An example command file to create the StateMod river network file is shown below (from the Colorado cm2005 data set):

```

StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#           names from from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")

```

The river network file that is created by the above commands should not in general be edited further. Instead, if changes to the model network are needed, edit the generalized network file, using the StateDMI network editor, and regenerate the StateMod river network file.

## 5.12 Operational Data

Operational data consist of the operational rights file (\*.opr), which contains unique operation criteria used within a river basin. Operational rights are priority based and control operations such as transfers between structures, reservoir releases, etc. If a data set cannot be configured to simulate a known behavior in a basin, then a new operational right type may need to be developed. The operational right file is generally copied from the test data or base data and hand-edited according to the format described in the StateMod Users' Manual.

StateDMI currently does not prepare the operational rights data file. Commands may be added in the future, in particular to allow warnings to be generated when operational rights data do not match other data.

## 5.13 San Juan Sediment Recovery Plan Data

StateDMI currently does not prepare the San Juan Sediment Recover Plan data file. Refer to the StateMod documentation for more information.

## 5.14 Spatial Data

StateDMI currently does not prepare spatial data for the StateMod. StateMod itself does not use spatial data files – the response file includes data for GIS to support the StateMod GUI. Standard spatial data layers from CDSS can be used with a GeoView project file (\*.gvp) to provide displays in the StateMod GUI. See the **GeoView Mapping Tools Appendix** in the **StateMod GUI Documentation** for more information.

Standard ESRI shapefiles as created by GIS software can be specified in the \*.gvp file. For performance and display reasons, the spatial data files associated with a data set are typically filtered by the StateMod GUI to only contain the information related to the StateMod data set. Otherwise, displays may be crowded with unnecessary information and performance will suffer. Shapefiles for stations and structures in HydroBase are available on the CDSS web site. These shapefiles typically contain all data for a division and therefore may need to be filtered for best performance, although it is usually possible to simply use layers generated from HydroBase.

In order to take advantage of all StateMod GUI features, it may be necessary to create new spatial data layers for points not represented in HydroBase, including aggregate and system nodes and also “other” nodes. StateMod stations that are not included in standard spatial data files (e.g., aggregate structures) can be digitized into shapefiles using standard GIS tools. See the StateMod GUI documentation for information about configuring the layers for use in the StateMod GUI.

This page is intentionally blank.

---

## 6 Troubleshooting

Version 3.09.01, 2010-02-11

This chapter discusses how to troubleshoot StateDMI problems.

The StateDMI log file is created in the logs directory under the main installation directory (e.g., *C:\CDSS\StateDMI-03.09.01\logs\StateDMI\_USER.log*), where the version will agree with the software version.

The most common problems are program configuration (see the **Installation and Configuration Appendix**), user input error (see the **Command Reference** for a summary of commands), and database errors (more below). Other problems should be reported to the StateDMI developers (see **Acknowledgements** for support contacts). When contacting support, provide as much information as possible, including system information and the command file that is being run. For example, use the **Tools...Diagnostics** tool to turn on the debugging checkbox and then get the system information from **Help...About**. Send this information to support.

Checks have been implemented to detect common errors and use of the `Check*()` and `WriteCheckFile()` commands is recommended. However, to fully diagnose a problem you may need to refer to the log file. The log file is accessible from the **Tools...Diagnostics – View Log File** dialog.

In general, when running StateDMI, you will be warned about problems with yellow and red markers displayed next to commands in the command list.

Due to the complexity of the State of Colorado's HydroBase database and other input sources and the complexity of some commands, user and database errors can occur for a number of reasons. The following table summarizes common errors and their fixes.

**StateDMI Errors and Possible Solutions**

Error	Possible solutions
StateDMI does not run (error at start-up).	<ol style="list-style-type: none"><li>1. StateDMI uses a startup program to run and all files related to StateDMI are stored in the installation folder. If the software does not run, files may have been moved, removed, or modified, or there may be an incompatibility with the computer. Try running the <i>\CDSS\StateDMI-Version\bin\StateDMI.exe</i> program from a command shell window to see if messages are printed.</li><li>2. Review problems reported in the <i>\CDSS\StateDMI-Version\logs\StateDMI_USER.log</i> file.</li><li>3. Report the problem to support.</li></ol>
StateDMI fails on large queries.	StateDMI may run out of memory on large queries. Increase the memory by changing the value of the <code>-mxNNm</code> option in the <i>\CDSS\StateDMI-Version\bin\StateDMI.l4j.ini</i> file. There is a limit of approximately 1440 MB on 32-bit operating systems.
Commands appear to be split into pieces when processed and errors occur for the	Command files are simple text files and each command must exist on one line. Comments are indicated by lines that start with a <code>#</code> character. When editing commands files with a text editor, or when pasting commands into the comment editor dialog, <code>Ctrl-M</code> characters (carriage return) may be inserted by some software. These characters will display as <code>^M</code> in some software or a box

Error	Possible solutions
partial commands.	in Notepad.  To correct the problem, remove the Ctrl-M characters with an editor that is able to display the characters.
Unable to find files correctly.	The working directory is assumed to be the same as the location of the most recently opened or saved command file. The current working directory is generally displayed by editor dialogs that read or write files. If files are not being found, verify that the path to the file is correct, whether specified as an absolute path or relative to the command file. Confirm that the command file is saved to a location relative to the files that are being referenced.
Unexpected failure.	If there was a serious error in input, StateDMI may quit processing input. See the log file for details. If the log file does not offer insight, contact support.

---

# 7 Quality Control

Version 03.09.01, 2010-02-11

This chapter discusses how StateDMI software is quality controlled and how to use StateDMI to perform quality control of data, processes, and other software. Similar capabilities are built into the TSTool software.

## 7.1 Quality Control for StateDMI Software

StateDMI software provides many data processing commands. Each command typically provides multiple parameters. The combination of commands and parameters coupled with potential data changes and user errors can make it difficult to confirm that StateDMI software is performing as expected. In particular, it would be very time consuming and expensive to manually check software functionality every time a change is made. These are the same challenges faced by any software tool, including spreadsheets, and models. To address this quality control concern, a testing framework has been built into StateDMI to allow the software to test itself. Test cases can be defined for each command, with test cases for various combinations of parameters. The suite of all the test cases can then be run to confirm that the version of StateDMI does generate expected results. This approach performs regression testing using the test framework and utilizes StateDMI's error-handling features to provide visual feedback during testing.

Test cases are developed by software developers as new features are implemented, according to the following documentation. However, users can also develop test cases and this is encouraged to ensure that all combinations of parameters and input data are tested. Users who provide verified test data and results prior to new development can facilitate the new development.

### 7.1.1 Writing a Single Test Case

The following example illustrates a single test case (indented lines indicate commands that are too long to fit on one line in the documentation).

```
# Test setting diversion stations with a couple of generated stations
StartLog(LogFile="Results/Test_SetDiversionStation.StateDMI.log")
RemoveFile(InputFile="Results\Test_SetDiversionStation_out.csv")
RemoveFile(InputFile="Results\Test_SetDiversionStation_out_ReturnFlows.csv")
RemoveFile(InputFile="Results\Test_SetDiversionStation_out_Collections.csv")
SetDiversionStation(ID="2000505",Name="Diversion 1",RiverNodeID="ID",OnOff=1,
    Capacity=101,ReplaceResOption=0,DailyID="ID",UserName="User1",DemandType=1,
    IrrigatedAcres=1001,UseType=1,DemandSource=1,EffMonthly="60,61,62,63,64,65,66,67,68,69,70,71",
    Returns="ret11,75,101;ret12,25,102",IfNotFound=Add)
SetDiversionStation(ID="2000631",Name="Diversion 2",RiverNodeID="ID",OnOff=1,
    Capacity=102,ReplaceResOption=1,DailyID="ID",UserName="user2",DemandType=1,
    IrrigatedAcres=1002,UseType=1,DemandSource=1,EffMonthly="70,71,72,73,74,75,76,77,78,79,80,81",
    Returns="ret21,75,21;ret22,25,22",IfNotFound=Add)
# Uncomment the following command to regenerate the expected results.
# WriteDiversionStationsToList(OutputFile="ExpectedResults/Test_SetDiversionStation_out.csv")
WriteDiversionStationsToList(OutputFile="Results/Test_SetDiversionStation_out.csv")
CompareFiles(InputFile1="ExpectedResults/Test_SetDiversionStation_out.csv",
    InputFile2="Results/Test_SetDiversionStation_out.csv",WarnIfDifferent=True)
CompareFiles(InputFile1="ExpectedResults/Test_SetDiversionStation_out_ReturnFlows.csv",
    InputFile2="Results/Test_SetDiversionStation_out_ReturnFlows.csv",WarnIfDifferent=True)
CompareFiles(InputFile1="ExpectedResults/Test_SetDiversionStation_out_Collections.csv",
    InputFile2="Results/Test_SetDiversionStation_out_Collections.csv",WarnIfDifferent=True)
```

**Example Test Case Command File**

The purpose of the test case command file is to regenerate results and then compare the results to previously generated and verified expected results. The example illustrates the basic steps that should be included in any test case:

1. **Start a log file to store the results of the specific test case.** The previous log file will be closed and the new log file will be used until it is closed. The log file is not crucial to the test but helps with troubleshooting if necessary (for example if evaluating the test case output when run in a test suite, as explained later in this chapter).
2. **Remove the results that are to be generated by the test.** This is necessary because if the software fails and old results match expected results, it may appear that the command was successful. The `IfNotFound=Ignore` parameter is useful because someone who is running the tests for the first time may not have previous results to remove. Test developers should use `IfNotFound=Warn` when setting up the test to confirm that the results being removed match the name that is actually generated in a later command, and then switch to `IfNotFound=Ignore`.
3. **Generate or read test data.** The `SetDiversionStation()` command is used in the example to create a diversion station. This is a useful technique because it allows full control over the initial data and minimizes the number of files associated with the test. Synthetic data are often appropriate for simple tests. If the test requires more complicated data, then files can be read.
4. **Process the data using the command being tested.** In the example, the `SetDiversionStation()` command itself is being tested. In many cases, a single command can be used in this step. However, in some cases, it is necessary to use multiple commands (e.g., define diversion stations and then test a fill command). Using more than one command is OK as long as each command is sufficiently tested with appropriate test cases to ensure that a false pass does not occur.
5. **Write the results.** The resulting data objects are written to a standard format. Comma-separated value files are useful for general testing because they are simple and the format will not change over time. Note that two write commands are used in the example – one writes the expected results and the other writes the results from the current test. The expected results should only be written when the creator of the test has confirmed that it contains verified values. In the example, the command to write expected results is commented out because the results were previously generated. Commands to test writing a specific file format (e.g., StateMod time series file) might read an original file, write a new file, and compare the two files (see next step).
6. **Compare the expected results and the current results.** The example uses the `CompareFiles()` command to compare the CSV files generated for the expected and current results. This command omits comment lines in the comparison because file headers often change due to dynamic comments with date/time. If the software is functioning as expected, the data lines in the file will exactly match. The example illustrates that if the files are different, a warning will be generated because of the `WarnIfDifferent=True` parameter. Options for comparing results include:
  - a. **Use the `CompareTimeSeries()` command.** This command is not implemented in StateDMI but is available in TSTool – it may be implemented in StateDMI in the future.
  - b. **If testing a read/write command, compare the results with the original data file.** For example, if the test case is to verify that a certain file format is properly read, then there will generally also be a corresponding write command. The test case can then consist of a command to read the file, a command to write the results, and a comparison command to compare the two files. This may not work if the header of the file uses comment lines that are not recognized by the `CompareFiles()` command. Another example where the comparison may fail is the “total” column in StateMod time series files, which is the sum of the other columns. This is typically generated with in-memory values that may



round off when printed, rather than being the total of the numbers as printed (this issue may be corrected but unfortunately it will cause slight changes in many files and tests).

If the test case example command file is opened and run in StateDMI, it will produce diversion station results, the log file, and the output files. If the expected and current results are the same, no errors will be indicated. However, if the files are different, a warning indicator will be shown in the command list area of the main window next to the **CompareFiles( )** command.

General guidelines for defining test cases are as follows. Following these conventions will allow the test cases to be incorporated into the full test suite used by software developers.

- Define the test case in a folder matching the command name (e.g., *SetDiversionStation*).
- Name the command file with prefix *Test\_*, extension *.StateDMI*, and use the following guidelines, in combination if appropriate:
  - for the default case (default command parameter values) use the filename pattern *Test\_CommandName.StateDMI*
  - If there is a reason to define a test for a specific data set or input, add additional information to the filename, for example: *Test\_CommandName\_cm2006.StateDMI*
  - If defining a test for legacy syntax (meaning that the current software will support running old commands), name the command file as follows (and use the `#@readOnly` comment tag described in **Section 7.1.3**): *Test\_CommandName\_Legacy.StateDMI*
  - If defining a test for parameter values other than the default values, use a command file name similar to the following, where the parameters are listed at the end of the file name body: *Test\_CommandName\_Param1=Value1,Param2=Value2.StateDMI*  
Although this can result in very long names, the explicit naming clarifies the purpose of the test. If this becomes cumbersome, just indicate that a parameter is being tested, for example: *Test\_CommandName\_OutputPeriod.StateDMI*
- Add a short comment to the top of the test case explaining the test.
- If many test cases are being defined for a command, consider including a spreadsheet or document in the test folder to describe the tests in more detail. Additional tests that cannot initially be implemented due to lack of resources can be documented as placeholders for future implementation.
- Use as little data as possible to perform the test – long time series or big input files cause tests to run longer and take up more space in the repository that is used for revision control. Even though hundreds or thousands of tests may ultimately be defined, it is important to be able to run them in a short time to facilitate testing.
- If possible, test only one command in the test – more complicated testing is described in **Section 7.1.4**. If multiple commands are needed, make absolutely sure that prerequisite commands are functioning properly (make sure that they have tests).
- If an input file is needed, place it in a folder named *Data*, if necessary copying the same input from another command – this may require additional disk space but ensures that each command can stand alone. An exception to this is if the input data are very large, in which case data should be stored with one command and be used by other commands.
- Write the expected results to a folder named *ExpectedResults*.
- Write the generated results and other dynamic content, including log file, to a folder named *Results*. When using a revision control system, the files in this folder should be excluded from the repository because they are dynamic.
- (Recommended) When creating output files, use *\_out* in the filename before the extension and use an extension that is appropriate for the file content – this helps identify final output products in cases where intermediate files might be produced.

### 7.1.2 Creating and Running a Test Suite

The previous section described how to define a single test case. However, opening and running each test case command file would be very tedious and inefficient. Therefore, StateDMI provides a way to generate and run test suites, which is the approach taken to perform a full regression test prior to a software release. The following example command file

(*test\regression\commands\TestSuites\commands\create\Create\_RunTestSuite\_commands.StateDMI*) illustrates how to create a test suite:

```
#
# Create the regression test runner for the
# StateDMI/test/regression/TestSuites/commands files.
#
# Only command files that match Test_*.StateDMI are included in the output.
# Don't append the generated commands, in order to force the old file to be
# overwritten.
#
CreateRegressionTestCommandFile(SearchFolder="..\..\..\commands",
    OutputFile="..\run\RunRegressionTest_commands.StateDMI",
    Append=False, IncludeTestSuite="*", IncludeOS="*")
```

When the command file is run, it searches the indicated search folder for files matching the pattern *Test\_\*.StateDMI*. It then uses this list to create a command file with contents similar to the following excerpted example. This file will be listed as an output file after running the above command file. The IncludeTestSuite and IncludeOS parameters are described in **Section 7.1.3**.

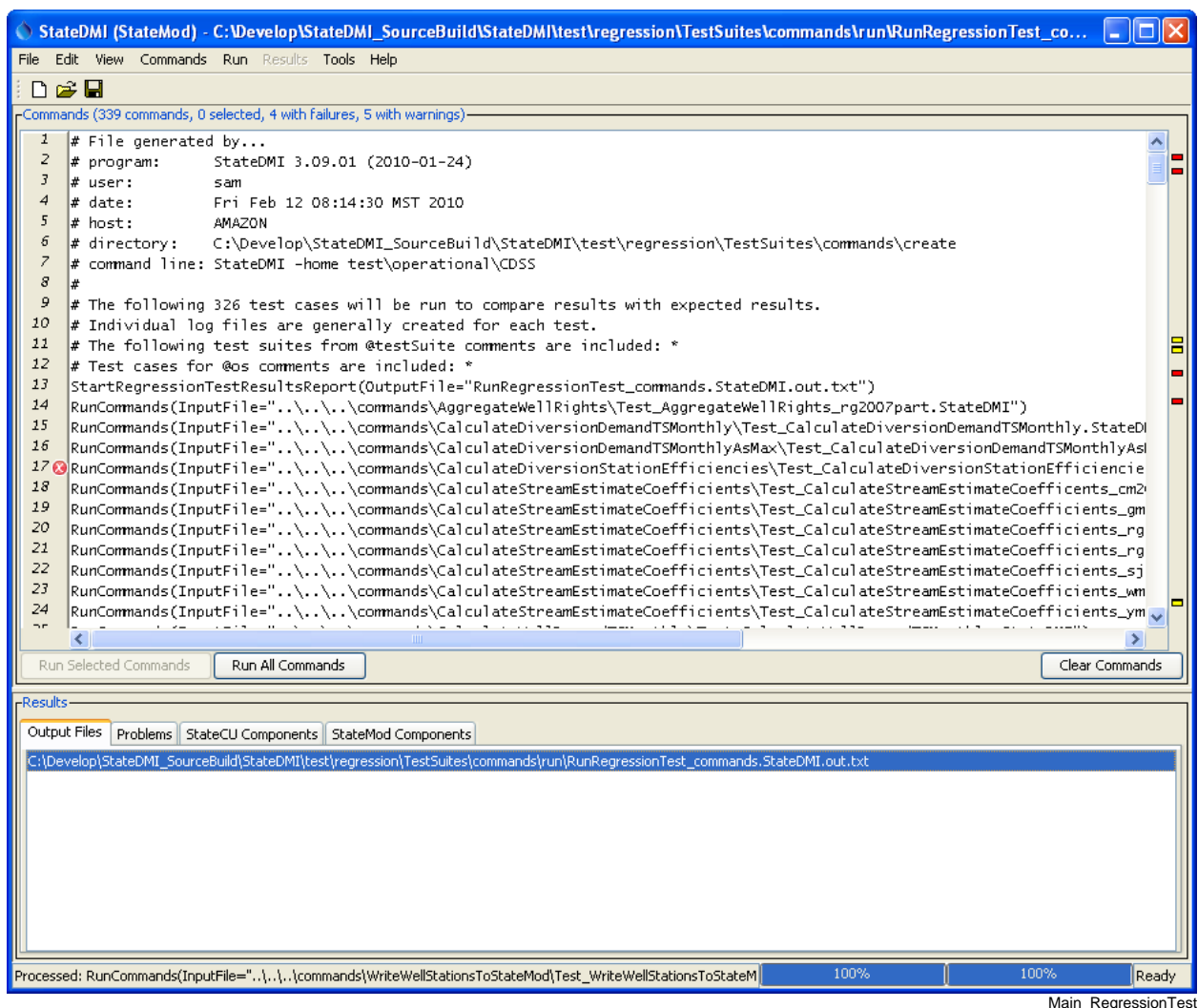
```
# File generated by...
# program:      StateDMI 3.08.02 (2009-09-29)
# user:         sam
# date:         Wed Sep 30 11:21:51 MDT 2009
# host:         SOPRIS
# directory:    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\TestSuites\commands\create
# command line: StateDMI -home test\operational\CDSS
#
# The following 321 test cases will be run to compare results with expected results.
# Individual log files are generally created for each test.
# The following test suites from @testSuite comments are included: *
# Test cases for @os comments are included: *
StartRegressionTestResultsReport(OutputFile="RunRegressionTest_commands.StateDMI.out.txt")
RunCommands(InputFile="..\..\..\commands\AggregateWellRights\Test_AggregateWellRights_rg2007part.StateDMI")
RunCommands(InputFile="..\..\..\commands\CalculateDiversionDemandTSMonthly\
    Test_CalculateDiversionDemandTSMonthly.StateDMI")
RunCommands(InputFile="..\..\..\commands\CalculateDiversionDemandTSMonthlyAsMax\
    Test_CalculateDiversionDemandTSMonthlyAsMax.StateDMI")
RunCommands(InputFile="..\..\..\commands\CalculateDiversionStationEfficiencies\
    Test_CalculateDiversionStationEfficiencies.StateDMI")
RunCommands(InputFile="..\..\..\commands\CalculateStreamEstimateCoefficients\
    Test_CalculateStreamEstimateCoefficients_cm2005.StateDMI",
    ExpectedStatus=warning)
RunCommands(InputFile="..\..\..\commands\CalculateStreamEstimateCoefficients\
    Test_CalculateStreamEstimateCoefficients_gm2004.StateDMI", ExpectedStatus=warning)
RunCommands(InputFile="..\..\..\commands\CalculateStreamEstimateCoefficients\
    Test_CalculateStreamEstimateCoefficients_rg2007.StateDMI", ExpectedStatus=warning)
RunCommands(InputFile="..\..\..\commands\CalculateStreamEstimateCoefficients\
    Test_CalculateStreamEstimateCoefficients_rg2007b.StateDMI")
RunCommands(InputFile="..\..\..\commands\CalculateStreamEstimateCoefficients\
    Test_CalculateStreamEstimateCoefficients_sj2004.StateDMI", ExpectedStatus=warning)
R
RunCommands(InputFile="..\..\..\commands\FillDiversionStationsFromNetwork\
    Test_FillDiversionStationsFromNetwork.StateDMI")
```

```

RunCommands(InputFile="..\..\..\commands\FillInstreamFlowRight\Test_FillInstreamFlowRight.StateDMI")
RunCommands(InputFile="..\..\..\commands\FillInstreamFlowStation\Test_FillInstreamFlowStation.StateDMI")
RunCommands(InputFile="..\..\..\commands\FillInstreamFlowStationsFromHydroBase\
Test_FillInstreamFlowStationsFromHydroBase.StateDMI")
RunCommands(InputFile="..\..\..\commands\FillInstreamFlowStationsFromNetwork\
Test_FillStreamGageStationsFromNetwork.StateDMI")

```

The above command file can then be opened and run. Each `RunCommands()` command will run a single test case command file. Warning and failure statuses from each test case command file are propagated to the test suite `RunCommands()` command. The output from running the test suite will be all of the output from individual test cases (in the appropriate *Results* folders) plus the regression test report provided in the StateDMI **Results...Output Files** tab in the main window. An example of the StateDMI main window after running the test suite is shown in the following figure. Note the warnings and errors, which should be addressed before releasing the software (in some cases commands are difficult to test and more development on the test framework is needed).



StateDMI Main Interface Showing Regression Test Results

An excerpt from the output file is shown below:.

```
# File generated by...
# program:      StateDMI 3.09.01 (2010-01-24)
# user:         sam
# date:         Fri Feb 12 08:15:00 MST 2010
# host:         AMAZON
# directory:    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\TestSuites\commands\run
# command line: StateDMI -home test\operational\CDSS
#
# The test status below may be PASS or FAIL.
# A test can pass even if the command file actual status is FAILURE, if failure is expected.
#
#   Test   Commands   Commands
#   Pass/  Expected   Actual
#   Num Fail   Status     Status     Command File
#-----
#   1 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   2 PASS    SUCCESS    SUCCESS    AggregateWellRights\Test_AggregateWellRights_rg2007part.StateDMI
#   3 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   CalculateDiversi...Test_CalculateDiversi...
#   4 *FAIL*  SUCCESS    FAILURE    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   CalculateDiversi...Test_CalculateDiversi...
#   5 PASS    warning    WARNING    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   CalculateStream...Test_CalculateStream...
#   6 PASS    warning    WARNING    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   CalculateStream...Test_CalculateStream...
# ..many tests omitted...
# 323 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   WriteWellRights...Test_WriteWellRights...
# 324 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   WriteWellRights...Test_WriteWellRights...
# 325 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   WriteWellStations...Test_WriteWellStations...
# 326 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\
#   WriteWellStations...Test_WriteWellStations...
#-----
# FAIL count = 9
# PASS count = 317
```

A test passes if its expected status (SUCCESS) matches the actual status, and the test fails otherwise. Note that there are cases where a test case is actually intended to fail, in order to test that StateDMI is properly detecting and handling the failure (rather than ignoring it or crashing). In these cases, the expected status (WARNING or FAILURE) must match the actual status to pass the test.

The features built into StateDMI can therefore be used to efficiently test the software, contributing to increased software quality and efficient software releases. New development results in additional tests. See the next section for more information on controlling the test process.

### 7.1.3 Controlling Tests with Special Comments

The previous two sections described how to define individual test cases and how to automatically create and run a test suite comprised of test cases. However, there are special conditions that will cause the normal testing procedures to fail, in particular:

- tests depend on a database that is not available
- tests depend on a database version that is not available (data in the “default” database have changed)
- tests can only be run on a certain operating system
- tests depend on a specific environment configuration that is not easily reproduced for all users

Any of these conditions can cause a test case to fail, leading to inappropriate errors and wasted time tracking down problems that do not exist. To address this issue, StateDMI recognizes special comments that can be included in test case command files. The following table lists tags that can be placed in # comments in command files to provide information for to the `CreateRegressionTestCommandFile()` command and command processor. The syntax of the special comments is illustrated by the following example:

```
#@expectedStatus Failure
```

### Special #-comment Tags

Parameter	Description
@expectedStatus Failure @expectedStatus Warning	The <code>RunCommands()</code> command <code>ExpectedStatus</code> parameter is by default <code>Success</code> . However, a different status can be specified if it is expected that a command file will result in <code>Warning</code> or <code>Failure</code> and still be a successful test. For example, if a command is obsolete and should generate a failure, the expected status can be specified as <code>Failure</code> and the test will pass. Another example is to test that the software properly treats a missing file as a failure.
@os Windows @os UNIX	Using this tag indicates that the test is designed to work only on the specified platform and will be included in the test suite by the <code>CreateRegressionTestCommandFile()</code> command only if the <code>IncludeOS</code> parameter includes the corresponding operating system (OS) type. This is primarily used to test specific features of the OS and similar but separate test cases should be implemented for both OS types. If the OS type is not specified as a tag in a command file, the test is always included.
@readOnly	Use this tag to indicate that a command file is read-only. This is useful when legacy command files are being tested because StateDMI will automatically update old syntax to new. Consequently, saving the command file will overwrite the legacy syntax and void the test. If this tag is included, the StateDMI interface will warn the user that the file is read-only and will only save if the user indicates to do so.
@testSuite ABC	Indicate that the command file should be considered part of the specified test suite, as specified with the <code>IncludeTestSuite</code> parameter of the <code>CreateRegressionTestCommandFile()</code> command. Do not specify a test suite tag for general tests. This tag is useful if a group of tests require special setup, for example connecting to a database. The suite names should be decided upon by the test developer.

Using the above special comment tags, it is possible to create test suites that are appropriate for specific environments. For example, using `@testSuite HydroBase` indicates that a test case should be included in the `HydroBase` test suite, presumably run in an environment where a connection to `HydroBase` has been opened. Consequently, multiple test suites can be created and run as appropriate depending on the system environment.

### 7.1.4 Verifying StateDMI Software Using a Full Dataset

The previous sections described how to test StateDMI software using a suite of test cases. This approach can be utilized when performing general tests, for example prior to a normal software release. However, there may be cases where StateDMI has been used to produce a large data set and it is desirable to confirm that a software release will still create the full dataset without differences. For example, for the State of Colorado's Decision Support Systems, large basin model data sets are created and are subject to significant scrutiny. Approaches previously described in this chapter can be utilized to verify that StateDMI is functioning properly and creates the dataset files. The following procedure is recommended and uses CDSS as an example:

1. If not already installed, install the data set in its default location (e.g., *C:\CDSS\data\colorado\_1\_2007*) – these files **will not be modified** during testing.
2. Create a parallel folder with a name indicating that it is being used for verification (e.g., *C:\CDSS\data\colorado\_1\_2007\_verify20090216*).
3. Copy the data set files from step 1 to the folder created in step 2 (e.g., copy to *C:\CDSS\data\colorado\_1\_2007\_verify20090216\colorado\_1\_2007*) – these files **will be modified** during testing.
4. Create a StateDMI command file in the folder created in step 2 that will run the tests (e.g., *VerifyStateDMI.StateDMI*). It is often easier to edit this command file with a text editor rather than with StateDMI itself. The contents of the file are illustrated in the example below. Some guidelines for this step are as follows:
  - a. Organize the command file by data set folder, in the order that data need to be created.
  - b. Process every \*.*StateDMI* command to verify that it runs and generates the same results.
  - c. If command files do not produce the same results, copy the command file to a name with “-updated” or similar in the filename and then change the file until it creates the expected results. This may be required due to changes in the command, for example implementing stricture error handling. These command files can then be shared with maintainers of the data set so that future releases can be updated.
  - d. As tests are formalized, it may be beneficial to save a copy of this file with the original data set so future tests can simply copy the verification command file rather than recreating it (e.g., save in a *QualityControl* folder in the master data set). This effort will allow the creator of the data set to quality control their work as well as helping to quality control the software.
5. Run the command file – any warnings or failures should be evaluated to determine if they are due to software or data changes. Software differences should be evaluated by software developers. It may be necessary to use command parameters such as *Version*, available for some commands, to recreate legacy data formats.

The following example command file (developed for the Colorado cm2005 data set) illustrates how StateDMI software is verified using the full data set (indented lines indicate commands that are too long to fit on one line in the documentation). Note that intermediate input files that would normally be modified by other software (e.g., TSTool for CDSS data sets) could impact StateDMI verification. However, a similar quality control procedure can be implemented for TSTool.

Guidelines for setting up the each test in the command file are as follows:

1. Remove output files that are generated from each individual command file that is run using *RemoveFile( )* commands. This will ensure that a test does not use old results for its output comparison.
2. Run each individual command file using the *RunCommands( )* command.

### 3. Compare the results of the run with the original data set file using the CompareFiles() command.

```

StartLog(LogFile="VerifyStateDMI.StateDMI.log")
# This command file verifies the StateDMI functionality by recreating a released
# StateMod/StateCU data set. The general process is as follows:
# 1) Copy the entire original data set to this folder (e.g., do manually).
# 2) Commands below will remove output files from product and StateMod/StateCU
# folders. This is done in case regeneration stops - don't want any confusion
# with original output and what should be created here.
# 3) Commands below will run the command files used to generate the model files.
# 4) Commands below will use CompareFile() commands to compare results. Comment
# lines are ignored so only data differences (processing output) will be
# flagged.
# If run interactively from StateDMI, indicators will show where results are
# different. Differences must then be evaluated to determine if input data,
# process, or software have changed. Differences may be valid.
#
#####
#####
# Diversions
#####
#####
#
# Stations...
RemoveFile(InputFile="colorado_1_2007\Diversions\cm2005_dds.dds")
#RunCommands(InputFile="colorado_1_2007\Diversions\dds.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\dds.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\Diversions\cm2005_dds.dds",
  InputFile2="..\colorado_1_2007\Diversions\cm2005_dds.dds",WarnIfDifferent=True)
#
# Rights...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ddy")
#RunCommands(InputFile="colorado_1_2007\Diversions\ddy.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\ddy.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ddy",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ddy",WarnIfDifferent=True)
#
# DDH (and final DDS)...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ddh")
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.dds")
#RunCommands(InputFile="colorado_1_2007\Diversions\ddh.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\ddh.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ddh",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ddh",WarnIfDifferent=True)
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.dds",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.dds",WarnIfDifferent=True)
#
# IWR...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.iwr")
#RunCommands(InputFile="colorado_1_2007\Diversions\iwr.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\iwr.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.iwr",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.iwr",WarnIfDifferent=True)
#
# IWRB...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005B.iwr")
#RunCommands(InputFile="colorado_1_2007\Diversions\iwrB.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\iwrB.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005B.iwr",
  InputFile2="..\colorado_1_2007\StateMod\cm2005B.iwr",WarnIfDifferent=True)
#
# Hddm...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005H.ddm")
#RunCommands(InputFile="colorado_1_2007\Diversions\Hddm.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\Hddm.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005H.ddm",
  InputFile2="..\colorado_1_2007\StateMod\cm2005H.ddm",WarnIfDifferent=True)
#
# Cddm...

```

```
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005C.ddm")
#RunCommands(InputFile="colorado_1_2007\Diversions\Cddm.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\Cddm.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005C.ddm",
  InputFile2="..\colorado_1_2007\StateMod\cm2005C.ddm",WarnIfDifferent=True)
#
# Cddm-AcreageChange...
/* Output file does not exist in master
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005C-AcreageChange.ddm")
#RunCommands(InputFile="colorado_1_2007\Diversions\Cddm-AcreageChange.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\Cddm-AcreageChange.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005C-AcreageChange.ddm",
  InputFile2="..\colorado_1_2007\StateMod\cm2005C-AcreageChange.ddm",WarnIfDifferent=True)
*/
#
# Bddm...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005B.ddm")
#RunCommands(InputFile="colorado_1_2007\Diversions\Bddm.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\Diversions\Bddm.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005B.ddm",
  InputFile2="..\colorado_1_2007\StateMod\cm2005B.ddm",WarnIfDifferent=True)
#
#####
#####
# instream
#####
#####
#
# ifs...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ifs")
#RunCommands(InputFile="colorado_1_2007\instream\ifs.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\instream\ifs.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ifs",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ifs",WarnIfDifferent=True)
#
# ifr...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ifr")
#RunCommands(InputFile="colorado_1_2007\instream\ifr.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\instream\ifr.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ifr",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ifr",WarnIfDifferent=True)
#
# ifa...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ifa")
#RunCommands(InputFile="colorado_1_2007\instream\ifa.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\instream\ifa.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ifa",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ifa",WarnIfDifferent=True)
#
#####
#####
# network
#####
#####
#
# rin...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.rin")
#RunCommands(InputFile="colorado_1_2007\network\rin.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\network\rin.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.rin",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.rin",WarnIfDifferent=True)
#
#####
#####
# reservoirs
#####
#####
#
# res...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.res")
#RunCommands(InputFile="colorado_1_2007\reservoirs\res.commands.StateDMI")
```



```

RunCommands(InputFile="colorado_1_2007\reservoirs\res.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.res",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.res",WarnIfDifferent=True)
#
# rer...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.rer")
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005B.rer")
#RunCommands(InputFile="colorado_1_2007\reservoirs\rer.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\reservoirs\rer.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.res",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.res",WarnIfDifferent=True)
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005B.rer",
  InputFile2="..\colorado_1_2007\StateMod\cm2005B.rer",WarnIfDifferent=True)
#
# cmdly...
RemoveFile(InputFile="colorado_1_2007\StateMod\cmdly.res")
#RunCommands(InputFile="colorado_1_2007\reservoirs\cmdly.res.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\reservoirs\cmdly.res.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cmdly.res",
  InputFile2="..\colorado_1_2007\StateMod\cmdly.res",WarnIfDifferent=True)
#
# B.res...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005B.res")
#RunCommands(InputFile="colorado_1_2007\reservoirs\cm2005B.res.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\reservoirs\cm2005B.res.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005B.res",
  InputFile2="..\colorado_1_2007\StateMod\cm2005B.res",WarnIfDifferent=True)
#
#####
#####
# streamSW
#####
#####
#
# ris...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.ris")
#RunCommands(InputFile="colorado_1_2007\streamSW\ris.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\streamSW\ris.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.ris",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.ris",WarnIfDifferent=True)
#
# rib...
RemoveFile(InputFile="colorado_1_2007\StateMod\cm2005.rib")
#RunCommands(InputFile="colorado_1_2007\streamSW\rib.commands.StateDMI")
RunCommands(InputFile="colorado_1_2007\streamSW\rib.commands-updated.StateDMI")
CompareFiles(InputFile1="colorado_1_2007\StateMod\cm2005.rib",
  InputFile2="..\colorado_1_2007\StateMod\cm2005.rib",WarnIfDifferent=True)
#

```

### Command File to Verify Data Set Creation

## 7.2 Using StateDMI and TSTool to Quality Control Data and Processes

The testing concepts discussed in this chapter can be utilized similarly to perform quality control on data or processes. In many cases, these tests must be defined by software users because the data and processes are only accessible and familiar to the users. However, once implemented, the tests can become a part of standard software test suites to further increase software quality control. The following are examples of tests that may be useful:

- Historical Data.** For a database such as HydroBase, implement a sequence of tests to ensure that data continue to exist, are accessible, and are in expected ranges. For example, for historical data, set the period to query so that it is not likely to be impacted by new data being added to the database, then create test command files to read many or all of the data types (use the `Read*HydroBase()` commands). It is expected that these data should not change over time.

A failed test will indicate that the database contents have changed and impacts on users may need to be evaluated (or at a minimum documented).

- **Real-Time Data.** Similarly, for recent historical data and real-time data, tests can be defined to ensure that data are available, in this case, it may not be important that data match expected results but only that data are returned. For example, TSTool capabilities to read time series and then compute statistics on time series, such as the number of missing, can be used to check for valid data. Additional test capabilities may need to be developed to fully implement these types of tests. The purpose of this type of testing is to ensure that an operational system continues to function as expected.
- **Standard Processes.** An organization's staff typically defines and executes standard processes to perform the business functions of the organizations. Related to water resources engineering, these processes may involve data collection, processing, analysis, and modeling, using a variety of tools. Standard tests, as described in this chapter, can confirm that a process is working as intended by verifying logic and data processing. For example, a standard process can be run on test data to confirm that it is still working. StateDMI and TSTool (or other tools) can be used to automate file comparisons, or perhaps to run several programs and then compare files, in order to demonstrate that a standard process is working.
- **Models.** Models can be complex and are often referred to as "black boxes" because it may not be obvious what occurs inside a model. Tests can be implemented in a number of ways:
  - **Unit Tests.** Similar to the small command files described in this chapter, small model data sets can be defined and run to confirm that basic model functionality is correct.
  - **Full Tests.** Full "accepted" data sets can be run, and tools can be used to verify that the results are consistent with expected results. This is similar to testing StateDMI and TSTool on full data sets.
  - **Results Evaluation.** Assuming that the model software has tested out, tests can be performed on any data set to confirm that results are reasonable. This recognizes that the synergy of a variety of model inputs and the results from a simulation may indicate unanticipated conditions (e.g., an impact is exponential rather than simply additive). The test process can therefore check for missing or zero values, out of range, or combinations of critical values.

The use of automated testing frameworks, such as the features available in StateDMI and TSTool, can streamline quality control checks and lead to a more robust quality assurance program. If StateDMI or TSTool are not used, then other test frameworks are highly desirable.

---

# Command Glossary

Version 2.14.00, 2007-07-11, Acrobat Distiller

The following parameter names and terms are used throughout StateDMI commands. A term indicated in **bold** font is a definition. A term indicated in **bold courier** font is a parameter name. Parameters specific to one or a few commands are cross-referenced with the commands. Common parameters are defined but long lists of corresponding commands are not provided. Possible values for parameters used in modeling (e.g., numerical options) are described in StateCU and StateMod model documentation.

**AccountDist** – The account distribution option for reservoir rights. See the `fillReservoirRight()`, and `setReservoirRight()` commands.

**AccountEvap** – Indicate how to distribute evaporation for a reservoir account. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AccountID** – The account identifier for a reservoir account. A reservoir can have multiple accounts. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AccountInitial** – The account initial content for a reservoir account. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AccountMax** – The account maximum content for a reservoir account. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AccountName** – The account name for a reservoir account. A reservoir can have multiple accounts. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AccountOneFill** – Indicate how to handle one fill rule calculations for a reservoir account. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**AcresGW** – The groundwater acres for a CU Location. See the `setIrrigationPracticeTS()` command.

**AcresGWCol** – The column number (or name) for groundwater acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresGWFloodCol** – The column number (or name) for groundwater flood acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresGWSprinklerCol** – The column number (or name) for groundwater sprinkler acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresSprinkler** – The sprinkler acres for a CU Location. See the `setIrrigationPracticeTS()` command.

**AcresSprinklerCol** – The column number (or name) for sprinkler acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresSWFloodCol** – The column number (or name) for surface water flood acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresSWSprinklerCol** – The column number (or name) for surface water sprinkler acres. See the `setIrrigationPracticeTSFromList()` command.

**AcresTotal** – The total acres for a CU Location. See the `setIrrigationPracticeTS()` command.

**AcresTotalCol** – The column number (or name) for total acres. See the `setIrrigationPracticeTSFromList()` command.

**AdministrationNumber** – The administration number (numerical priority) for a water right. See the `fillDiversionRight()`, `fillInstreamFlowRight()`, `fillReservoirRight()`, `fillWellRight()`, `setDiversionRight()`, `setInstreamFlowRight()`, `setReservoirRight()`, and `setWellRight()` commands.

**AdminNumClasses** – The administration number classes for water rights, used to define aggregates. See the `readDiversionRightsFromHydroBase()`, `readReservoirRightsFromHydroBase()`, `readWellRightsFromHydroBase()`, and `setIrrigationPracticeTSFromHydroBase()` commands.

**AdminNumShift** – The administration number shift for a well station. See the `fillWellStation()` and `setWellStation()` commands.

**AdminNumShiftCol** – The column number (or name) to be read from a delimited file for AdminNumShift data. See the `setWellStationsFromList()` command.

**Aggregate** – See Collection.

**Alias** – A (generally) short identifier for a time series, used in place of the TSID, which simplifies commands. The Alias and TSID values are interchangeable when used as parameters to commands and may both be referred to as TSID in command editors. See also TSID.

**Alias** – A (generally) short identifier for a time series, used in place of the TSID, which simplifies commands. When used to create/read a time series, the syntax of a command is typically similar to: `TS Alias = command(...)`. See also TSID.

**AnalysisEnd** – A DateTime that indicates the end of an analysis.

**AnalysisStart** – A DateTime that indicates the start of an analysis.

**Append** – Indicates whether data from a read should be appended to in-memory data. The default in most cases is True, but in some cases in-memory data are to be discarded before the read. See the `readWellRightsFromStateMod()` and `readWellStationsFromStateMod()` commands.

**AreaCol** – The column number (or name) to be read from a delimited file for area data. See the `setCropPatternTSFromList()` command.

**AutoAdjust** – Indicate that automatic adjustments should be made to data, typically in cases where some type of version compatibility issue is being addressed. See the `writeCropCharacteristicsToStateCU()` command.

**AWC** – The available water content (AWC) fraction, for a CU Location. See the `setCULocation()` command.

**AWCCol** – The column number (or name) to be read from a delimited file for AWC data. See the `fillCULocationsFromList()`, and `setCULocationsFromList()` commands.

**BaseData** – The base flow coefficient and station data for stream estimate stations. See the `setStreamEstimateCoefficients()` command.

**BlaneyCriddleMethod** – The Blaney-Criddle method in HydroBase for Blaney-Criddle data. Regional variations are provided. See the `readBlaneyCriddleFromHydroBase()` command.

**Capacity** – The capacity for a diversion or well. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**CapacityCol** – The column number (or name) to be read from a delimited file for Capacity data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**CheckStructures** – Used when filling stream gage stations from HydroBase. See the `fillStreamGageStationsFromHydroBase()` command.

**Coefficients** – Crop growth coefficients. See the `setBlaneyCriddle()` command.

**Collection** – A group of parts that modeled as a single item. StateMod diversions can be one of the following:

- **Aggregate** – the physical characteristics of the diversion stations are combined, and the water rights are aggregated into classes
- **MultiStruct** – multiple diversions are grouped but are each represented in the model network; for historical modeling the time series at each point are used; for calculated demands the demands are totaled at a key structure and set to zero for the others. The definition of a MultiStruct is only necessary when processing demands.
- **System** – the physical characteristics of the diversion are combined, but water rights are retained in their individual form.

**CommentFormat** – The format to use when setting the comment for a station. Various data can be combined into the name. See the `fillRiverNetworkFromNetwork()` command.

**Constant** – A constant value used to fill or set time series. See the `fillCropPatternTSConstant()`, `fillDiversionDemandTSMonthlyConstant()`, `fillDiversionHistoricalTSMonthlyConstant()`, `fillWellDemandTSMonthlyConstant()`, and `setDiversionDemandTSMonthlyConstant()` commands.

**ContentAreaSeepage** – Content/area/seepage table values for a reservoir station. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**ContentMax** – The maximum content for a reservoir. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**ContentMin** – The minimum content for a reservoir. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**CropPattern** – A crop pattern (crop type and area values). See the `setCropPatternTS()` command.

**CropType** – A crop type/name (e.g., ALFALFA), which in some cases may be a pattern (e.g., ALFALFA\*). See the `fillCropPatternTSConstant()`, `removeCropPatternTS()`, and `setBlaneyCriddle()`, and `setCropCharacteristics()` commands.

**CropTypeCol** – The column number (or name) to be read from a delimited file for CropType data. See the `setCropPatternTSFromList()` commands.

**CULocType** – Consumptive use location (CU Location) type. StateDMI currently processes data for structures but can be extended to process data for climate station locations. The location type can therefore be used to control which database tables are queried for information. See the `fillCULocationFromHydroBase()` command.

**CUMethod** – The CU method in HydroBase for crop type and characteristics. See the `readCropCharacteristicsFromHydroBase()` command.

**CurveType** – Indicate whether crop growth data are for annual or perennial crops. See the `setBlaneyCriddle()` command.

**DailyID** – The station identifier used to specify daily data for a station. See the `fillDiversionStation()`, `fillInstreamFlowStation()`, `fillReservoirStation()`, `fillStreamEstimateStation()`, `fillStreamGageStation()`, `fillWellStation()`, `setDiversionStation()`, `setInstreamFlowStation()`, `setReservoirStation()`, `setStreamEstimateStation()`, and `setWellStation()` commands.

**DailyIDCol** – The column number (or name) to be read from a delimited file for DailyID data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**DatabaseName** – The name of a database, when making a database connection. See the `openHydroBase()` command.

**DatabaseServer** – The name of a database server, when making a database connection. See the `openHydroBase()` command.

**DataType** – The data type used when processing time series, necessary when there are more than one time series data types available. See the `fillIrrigationPracticeTSInterpolate()`, and `fillIrrigationPracticeTSRepeat()` command.

**DateTime** – A date/time value, typically represented as a string, which indicates a point in time. Date/time strings have a precision that is interpreted by the software. For example, the date/time string 1990 has a precision of year, whereas the string 1990-01-12 has a precision of day.

**DaysToFullCover** – The days to full cover for a crop. See the `setCropCharacteristics()` command.

**DaysTo2ndCut** – The days to second cut for a crop. See the `setCropCharacteristics()` command.

**DaysTo3rdCut** – The days to third cut for a crop. See the `setCropCharacteristics()` command.

**DeadStorage** – The dead storage for a reservoir. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**Decree** – The decree amount for a water right. See the `fillDiversionRight()`, `fillInstreamFlowRight()`, `fillReservoirRight()`, `fillWellRight()`, `setDiversionRight()`, `setInstreamFlowRight()`, `setReservoirRight()`, and `setWellRight()` commands.

**DecreeMin** – The minimum decree to accept as a valid right (others are ignored). See the `readDiversionRightsFromHydroBase()`, and `readReservoirRightsFromHydroBase()` commands.

**DefaultAppropriationDate** – The default appropriation date to use with well right/permit data, if a date is not available. See the `readWellRightsFromHydroBase()`, and `setIrrigationPracticeTSFromHydroBase()` commands.

**DefaultTable** – The default delay table to use when setting returns from the river network. See the `setDiversionStationDelayTablesFromNetwork()` and `setWellStationDelayTablesFromNetwork()` commands.

**DefineRightHow** – Indicate how well rights should be defined from water right/permit data (e.g., earliest date, latest date, right if available). See the `readWellRightsFromHydroBase()`, `setIrrigationPracticeTSFromHydroBase()`, and `fillWellStationsFromHydroBase()` commands.

**Delim** – The delimiter character(s) used when processing delimited files. See the `read*FromList()` and `write*ToList()` commands.

**DemandSource** – The demand source, indicating whether demands are estimated from geographic information system acreage, total acreage estimate, etc., for a diversion station. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**DemandSourceCol** – The column number (or name) to be read from a delimited file for DemandSource data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**DemandType** – The demand type for a diversion station. See the `fillDiversionStation()` and `fillInstreamFlowStation()`, `fillWellStation()`, `setDiversionStation()`, `setInstreamFlowStation()`, and `setWellStation()` commands.

**DemandTypeCol** – The column number (or name) to be read from a delimited file for DemandType data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**Depletions** – The depletion locations, percentages, and delay table, for a well station. See the `fillWellStation()` and `setWellStation()` commands.

**Div** – The water division associates with data. See the `fillWellStationsFromHydroBase()`, `readWellRightsFromHydroBase()`, `setIrrigationPracticeTSFromHydroBase()`, `setIrrigationPracticeTSSprinklerAreaFromList()`, `setWellAggregate()`, `setWellAggregateFromList()`, `setWellSystem()`, and `setWellSystemFromList()` commands.

**DivAndWellGWAcreage** – Indicate how to adjust the groundwater acreage for locations that have surface diversion and groundwater supply. See the `synchronizeIrrigationPracticeAndCropPatternTS()` command.

**DiversionID** – The diversion station identifier associated with a well station. See the `fillWellStation()` and `setWellStation()` commands.

**DiversionIDCol** – The column number (or name) to be read from a delimited file for DiversionID data. See the `readWellStationsFromFromList()` command.

**DiversionIDCol** – The column number (or name) to be read from a delimited file for DiversionID data. See the `setWellStationsFromList()` command.

**DownstreamRiverNodeID** – The river node identifier for the downstream node in an instream flow reach, for instream flow stations. It is also used to indicate the node downstream from a river node, to indicate network connectivity. See the `fillInstreamFlowStation()`, `setInstreamFlowStation()`, and `setRiverNetworkNode()` commands.

**EarliestMoistureUseTemp** – The earliest moisture use temperature for a crop. See the `setCropCharacteristics()` command.

**EffAnnual** – The annual efficiency (% , 0-100) for a diversion station. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**EffAnnualCol** – The column number (or name) to be read from a delimited file for EffAnnual data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**EffCalcEnd** – A DateTime that indicates the end of an efficiency calculation analysis. See the `calculateDiversionStationEfficiencies()` and `calculateWellStationEfficiencies()` commands.

**EffCalcStart** – A DateTime that indicates the start of an efficiency calculation analysis. See the `calculateDiversionStationEfficiencies()` and `calculateWellStationEfficiencies()` commands.



**Effmin** – The minimum efficiency. See the `calculateDiversiionStationEfficiencies()` and `calculateWellStationEfficiencies()` commands.

**EffMonthly** – The monthly efficiency (% , 0-100) for a diversion station. The order of efficiencies in the model data file depends on the model and control information. However, StateDMI requires that efficiencies be entered in the order January through December. See the `fillDiversiionStation()`, `fillWellStation()`, `setDiversiionStation()`, and `setWellStation()` commands.

**EffMonthlyCol** – The column number (or name) to be read from a delimited file for `EffMonthly` data. See the `setDiversiionStationsFromList()` and `setWellStationsFromList()` commands.

**Effmax** – The maximum efficiency. See the `calculateDiversiionStationEfficiencies()` and `calculateWellStationEfficiencies()` commands.

**EffReportFile** – The name of the report file containing the results of efficiency calculations. See the `calculateDiversiionStationEfficiencies()` and `calculateWellStationEfficiencies()` commands.

**Elevation** – Elevation. See the `fillClimateStation()`, `setClimateStation()`, and `setCULocation()` commands.

**ElevationCol** – The column number (or name) to be read from a delimited file for `Elevation` data. See the `fillCULocationsFromList()`, `readCULocationsFromList()`, and `setCULocationsFromList()` commands.

**EvapStations** – The list of evaporation stations and weights for a reservoir station. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**FallFrostFlag** – The fall frost flag for a crop. See the `setCropCharacteristics()` command.

**FillAverageOrder** – When multiple fill techniques are used within one command, indicate the order for filling using historical average. See the `fillDiversiionHistoricalTSMonthlyFromHydroBase()` command.

**FillDirection** – Indicate which direction (Forward or Backward) that filling should occur. This is important because statistics computed to perform filling can be different depending on the processing direction. See the `fillCropPatternTSProrateAgStats()`, `fillCropPatternTSRepeat()`, and `fillIrrigationPracticeTSRepeat()` commands.

**FillEnd** – A `DateTime` that indicates the end of a fill process.

**FillFlag** – A character flag used to indicate when time series values are filled. See the `fillDiversiionDemandTSAverage()`, `fillDiversiionDemandTSConstant()`, `fillDiversiionDemandTSPattern()`, `fillDiversiionHistoricalTSAverage()`, `fillDiversiionHistoricalTSConstant()`, `fillDiversiionHistoricalTSMonthlyPattern()`, and `fillWellDemandTSMonthlyAverage()`,

`fillWellDemandTSMonthlyConstant()`, and  
`fillWellDemandTSMonthlyPattern()` commands.

**FillPatternOrder** – When multiple fill techniques are used within one command, indicate the order for filling using historical average patterns. See the `fillDiversionHistoricalTSMonthlyFromHydroBase()` command.

**FillStart** – A `DateTime` that indicates the start of fill process.

**FillType** – The reservoir right fill type. See the `fillReservoirRight()` and `setReservoirRight()` commands.

**FillUsingCIU** – Fill diversion records with additional zeros using the “currently in use” (CIU) data from HydroBase. See the `readDiversionHistoricalTSMonthlyFromHydroBase()` command.

**FillUsingCIUFlag** – Indicate how to flag filled data values when using “currently in use” (CIU) data from HydroBase. See the `readDiversionHistoricalTSMonthlyFromHydroBase()` command. The flags can be displayed on graphs.

**FloodAppEffMax** – The flood application efficiency maximum for a CU Location. See the `setIrrigationPracticeTS()` command.

**FloodAppEffMaxCol** – The column number (or name) to be read from a delimited file for FloodAppEffMax data. See the `setIrrigationPracticeTSFromList()` command.

**FreeWaterAdministrationNumber** – Indicate the administration number  $\geq$  to which a right is considered a free water right. See the `setIrrigationPracticeTSPumpingMaxUsingWellRights()` command.

**FreeWaterAppropriationDate** – A date to be used for free water rights. See the `limitDiversionDemandTSMonthlyToRights()`, `limitDiversionHistoricalTSMonthlyToRights()`, `setIrrigationPracticeTSMaxPumpingToRights()`, and `setIrrigationPracticeTSPumpingMaxUsingWellRights()` commands.

**FreeWaterMethod** – Indicate how to handle processing of free water rights. See the `setIrrigationPracticeTSPumpingMaxUsingWellRights()` command.

**GageID** – The stream gage station identifier to use instead of the downstream gage. See the `setStreamEstimateCoefficients()` command.

**GainData** – The base flow coefficient and station data for stream estimate stations. See the `setStreamEstimateCoefficients()` command.

**GWMode** – The groundwater mode for a CU Location. See the `setIrrigationPracticeTS()` command.

**GWModeCol** – The column number (or name) for groundwater mode for a CU Location. See the `setIrrigationPracticeTSFromList()` command.

**GWOnlyGWAcreeage** – Indicate how to adjust the groundwater acreage for locations that have only groundwater supply. See the `synchronizeIrrigationPracticeAndCropPatternTS()` command.

**HandleMissingHow** – Indicate how to handle missing data values when processing time series. For example, when adding time series, missing values can be ignored or can result in a missing value in the result. See the `add()`, `cumulate()`, and `subtract()` commands.

**HarvestMonth** – The harvest month for a crop. See the `setCropCharacteristics()` command.

**HarvestDay** – The harvest day for a crop. See the `setCropCharacteristics()` command.

**ID** – The identifier to match in a file. Typically this is a location (e.g., station, structure identifier) and can be specified using a wildcard pattern (e.g., 20\*). This parameter is used by many commands as the primary key to associate data.

**IDCol** – The column number (or name) to be read from a delimited file for identifier data. See the `read*FromList()` command.

**IfFound** – Indicate the action to be taken if a matching data item (usually by ID) is found. For example, the action typically includes warning the user or continuing with a data edit. See the `set*()` command.

**IfNotFound** – Indicate the action to be taken if a matching data item (usually by ID) is not found. For example, the action typically includes warning the user or continuing with a data edit. See the `set*()` commands.

**IDFormat** – The format to use for identifiers, used when default formatting is not appropriate. See the `readWellRightsFromHydroBase()` command.

**IgnoreDiversions** – Indicate whether diversion nodes should be ignored by a command.

**IgnoreDws** – Indicate whether D&W (diversion + well) nodes should be ignored by a command. See the `readWellStationsFromStateMod()` command.

**IgnoreID** – A list of identifiers to ignore when processing a command. See the `limitDiversionDemandTSMonthlyToRights()`, and `limitDiversionHistoricalTSMonthlyToRights()` commands.

**IgnoreLEZero** – Indicate whether values less than or equal to zero should be ignored when computing historical averages for time series. See the `setIgnoreLEZero()` command.

**IgnoreWells** – Indicate whether well nodes should be ignored by a command. See the `readWellStationsFromStateMod()` command.

**IncludeCollections** – Indicate whether locations that are collections (aggregates and systems) should be processed by a command. In particular, when processing time series, filling can be controlled to occur for individual collection parts or on total time series. See the `fillDiversionHistoricalTSMonthlyAverage()`,

`fillDiversionHistoricalTSMonthlyPattern()`, and  
`fillDiversionHistoricalTSMonthlyFromHydroBase()` commands.

**IncludeExplicit** – Indicate whether locations that are explicit (key) locations should be processed by a command. In particular, when processing time series, filling can be controlled to occur for explicit locations or collections (aggregates and systems). See the `fillDiversionHistoricalTSMonthlyFromHydroBase()` command.

**IncludeGroundwaterOnlySupply** – Indicate whether locations that have only groundwater supply should be processed by a command. See the `fillIrrigationPracticeTSAcreageUsingWellRights()` and `setIrrigationPracticeTSPumpingMaxUsingWellRights()` commands.

**IncludeStreamEstimateStations** – Indicate whether stream estimate stations should be processed by a command. In particular, this is used when processing stream gage/estimate station data. See the `fillIrrigationPracticeTSAcreageUsingWellRights()` and `readStreamGageStationsFromNetwork()` command.

**IncludeSurfaceWaterSupply** – Indicate whether locations that have surface water supply should be processed by a command. See the `setIrrigationPracticeTSPumpingMaxUsingWellRights()` command.

**InputEnd** – A `DateTime` that indicates the end of a file read or a database query.

**InputFile** – The name/path for a file that is used as input to a command. See the `limitDiversionDemandTSMonthlyToRights()`, `limitDiversionHistoricalTSMonthlyToRights()`, and `ReadAgStatsTSFromDateValue()` commands.

**InputStart** – A `DateTime` that indicates the start of file read or a database query.

**Interval** – The data interval (day or month) for delay tables. See the `writeDelayTablesToStateMod()` command.

**IrrigatedAcres** – The irrigated acres for a diversion station. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**IrrigatedAcresCol** – The column number (or name) to be read from a delimited file for `IrrigatedAcres` data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**IrrigationMethodCol** – The column number (or name) for irrigation method (e.g., `SPRINKLER`, `FLOOD`). See the `setCropPatternTSFromList()` and `setIrrigationPracticeTSFromList()` commands.

**LatestMoistureUseTemp** – The latest moisture use temperature for a crop. See the `setCropCharacteristics()` command.

**Latitude** – Latitude in decimal degrees. See the `fillClimateStation()`, `fillCULocation()`, `setClimateStation()`, and `setCULocation()` commands.

**LatitudeCol** – The column number (or name) to be read from a delimited file for Latitude data. See the `fillCULocationsFromList()`, `readCULocationsFromList()`, and `setCULocationsFromList()` commands.

**LengthOfSeason** – The length of the growing season for a crop. See the `setCropCharacteristics()` command.

**LEZeroInAverage** – Indicate whether historical averages should consider values less than or equal to zero. See the `calculateDiversionStationEfficiencies()`, `calculateWellStationEfficiencies()`, `fillDiversionDemandTSMonthlyPattern()`, `fillDiversionHistoricalTSMonthlyPattern()`, `fillWellDemandTSMonthlyPattern()`, `fillWellHistoricalTSMonthlyFromHydroBase()`, `setDiversionDemandTSMonthly()`, `setDiversionHistoricalTSMonthly()`, and `setWellDemandTSMonthly()` commands.

**LimitToCurrent** – Indicate whether only the most recent water rights conditions should be used when limiting time series to rights (use a single value and not a step function). See the `limitDiversionDemandTSMonthlyToRights()` command.

**ListFile** – The name of an input or output list (delimited) file to be written or read, specified using a relative or absolute path. See the `read*FromList()` and `write*toList()` commands.

**LocationEstimate** – Indicate how to estimate missing coordinates for nodes, when used with network diagram features. See the `fillNetworkFromHydroBase()` command.

**LogFile** – The name of the log file, specified using a relative or absolute path. See the `setLogFile()` command.

**LogFileLevel** – The level for messages printed to the log file. See the `setDebugLevel()` and `setWarningLevel()` commands.

**MaxAppDepth** – The maximum irrigation application depth for a crop. See the `setCropCharacteristics()` command.

**MaxIntervals** – The maximum number of intervals to process when processing time series. For example, indicate the widest gap of missing data to fill. See the `fillCropPatternTSInterpolate()`, `fillCropPatternTSProrateAgStats()`, `fillIrrigationPracticeTSInterpolate()`, and `fillIrrigationPracticeTSRepeat()` commands.

**MaxRechargeLimit** – The maximum recharge limit (CFS) when modeling groundwater. See the `setRiverNetworkNode()` command.

**MaxRootZoneDepth** – The maximum root zone depth for a crop. See the `setCropCharacteristics()` command.

**MergeDelim** – Indicates whether adjacent delimiters should be treated as one when processing delimited files. See the `read*FromList()` and `write*ToList()` commands.

**MonthValues** – Monthly values used to set time series data. See the `setInstreamFlowdemandTSAverageMonthlyConstant()` command.

**MultiStruct** – See Collection.

**Name** – The name associated with a data item (e.g., station, structure, water right name). This parameter is used by many commands.

**NameCol** – The column number (or name) to be read from a delimited file for Name data. See the `fill*FromList()` and `set*FromList()` commands.

**NameFormat** – The format to use when setting the name for a station from HydroBase. Various data can be combined into the name. See the `fillRiverNetworkFromHydroBase()`, `fillRiverNetworkFromNetwork()`, `fillStreamEstimateStation()`, `fillStreamEstimateStationsFromNetwork()`, `fillStreamGageStation()`, `fillStreamGageStationsFromHydroBase()`, and `fillStreamGageStationsFromNetwork()` commands.

**NewCropType** – The new crop type. See the `translateBlaneyCriddle()`, `translateCropCharacteristics()`, and `translateCropPatternTS()` commands.

**NumberOfDaysInMonth** – The number of days in each month, used when an approximation is used rather than exact values. See the `setIrrigationPracticeTSMaxPumpingToRights()` command.

**OldCropType** – The old crop type. See the `translateBlaneyCriddle()`, `translateCropCharacteristics()`, and `translateCropPatternTS()` commands.

**OneFillRule** – The date for one fill rule administration for a reservoir. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**OnOff** – The on/off switch used to indicate if a station, right, or other information is active for a data set. See `fill*()` and `set*()` commands for StateMod data files.

**OnOffCol** – The column number (or name) to be read from a delimited file for OnOff data. See the `setDiversionStationsFromList()` command.

**OnOffDefault** – The default value of the OnOff parameter for water rights (e.g., 1, or as determined from a water right appropriation date). See the `readDiversionRightsFromHydroBase()`, `readInstreamFlowRightsFromHydroBase()`, and `readReservoirRightsFromHydroBase()`, and `readWellRightsFromHydroBase()` commands.

**OpRightID** – The operational right identifier associated with a reservoir right. See the `fillReservoirRight()` and `setReservoirRight()` commands.

**Order** – The primary order to sort data. See the `sort*()` commands.

**Order2** – The secondary order to sort data. See the `sort*( )` commands.

**OrographicPrecAdjCol** – The column number (or name) to be read from a delimited file for the orographic precipitation adjustment factor. See the `setCULocationClimateStationWeightsFromList( )` command.

**OrographicTempAdjCol** – The column number (or name) to be read from a delimited file for the orographic temperature adjustment factor. See the `setCULocationClimateStationWeightsFromList( )` command.

**OutputEnd** – A `DateTime` that indicates the end of output.

**OutputFile** – The name of an output file to be written, specified using a relative or absolute path.

**OutputStart** – A `DateTime` that indicates the start of output.

**OutputYearType** – Indicate the type of year (e.g., calendar year, water year) for output. See the `setOutputYearType( )` command.

**ParcelAreaCol** – The column number (or name) to be read from a delimited file for parcel area data (used when overriding `HydroBase` data during development). See the `setIrrigationPracticeTSSprinklerAreaFromList( )` command.

**ParcelIDCol** – The column number (or name) to be read from a delimited file for `ParcelID` data. See the `setIrrigationPracticeTSSprinklerAreaFromList( )` command.

**ParcelIDYear** – The year to use for parcel identifiers (which can vary by year). See the `setIrrigationPracticeTSSprinklerAreaFromList( )` command.

**ParcelYear** – A specific year for irrigated lands parcel data. See the `fillIrrigationPracticeTSAcreageUsingWellRights( )`, and `setIrrigationPracticeTSPumpingMaxUsingWellRights( )` command.

**PartIDs** – The identifiers for parts of a collection (aggregates and systems). See the `setDiversionAggregate( )`, `setDiversionMultiStruct( )`, `setDiversionSystem( )`, `setReservoirAggregate( )`, `setWellAggregate( )`, and `setWellSystem( )` commands.

**PartIDsCol** – The column number (or name) to be read from a delimited file for `PartID` data (an identifier for part of a collection). See the `setDiversionAggregatesFromList( )`, `setDiversionMultiStructFromList( )`, `setDiversionSystemFromList( )`, `setReservoirAggregateFromList( )`, `setWellAggregateFromList( )`, and `setWellSystemFromList( )` commands.

**PartIDsColMax** – The maximum column number (or name) to be read from a delimited file for `PartID` data (an identifier for part of a collection). This is useful when ignoring additional columns on the right side of a delimited file. See the `setDiversionAggregatesFromListFromList( )`, `setDiversionMultiStructFromList( )`, `setDiversionSystemFromList( )`, `setReservoirAggregateFromList( )`, `setWellAggregateFromList( )`, and `setWellSystemFromList( )` commands.

**PartIDsListedHow** – Indicate whether part identifiers in a collection are listed in columns (one record per collection) or rows (multiple rows per collection). See the `setDiversionAggregatesFromListFromList()`, `setDiversionMultiStructFromList()`, `setDiversionSystemFromList()`, `setReservoirAggregateFromList()`, `setWellAggregateFromList()`, and `setWellSystemFromList()` commands.

**PatternFile** – The file name for a pattern file. See the `setPatternFile()` command.

**PatternID** – An identifier for a pattern (e.g., WET, DRY, AVG). See the `fillDiversionDemandTSMonthlyPattern()`, `fillDiversionHistoricalTSMonthlyPattern()`, `fillWellDemandTSMonthlyPattern()`, and `readDiversionHistoricalTSMonthlyFromHydroBase()` commands.

**PlantingMonth** – The planting month for a crop. See the `setCropCharacteristics()` command.

**PlantingDay** – The planting day for a crop. See the `setCropCharacteristics()` command.

**Precision** – The precision (digits after the decimal) for output. See the `writeBlaneyCriddleToStateCU()` command.

**PrecipStations** – The list of precipitation stations and weights for a reservoir station. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**PrecWtCol** – The column number (or name) to be read from a delimited file for PrecWt (precipitation weight) data. See the `setCULocationsFromList()` command.

**ProcessData** – Indicates whether crop pattern data should be processed or used only to define relationships between data (which will then be used by another command). See the `readCropPatternTSFromHydroBase()` command.

**ProcessWhen** – Indicates when crop pattern data should be processed. Data can be processed with the command (immediate set) or when HydroBase data are read. The latter allows more sophisticated processing that may be required. See the `setCropPatternTS()`, `setCropPatternTSFromList()`, `setIrrigationPracticeTS()`, and `setIrrigationPracticeTSFromList()` commands.

**ProrationFactor** – The proration factor for stream estimate stations. See the `setStreamEstimateCoefficients()` command.

**PumpingMax** – The maximum monthly pumping rate for a CU Location. See the `setIrrigationPracticeTS()` command.

**PumpingMaxCol** – The column number (or name) for pumping maximum. See the `setIrrigationPracticeTSFromList()` command.

**ReadWellRights** – Indicates whether well rights should be read, rather than relying on summed “pseudo rights”. The default is now to read individual well rights; however, this parameter can be



used to match data processing for earlier versions of the software. See the `fillWellStationsFromHydroBase()`, `readWellRightsFromHydroBase()`, and `setIrrigationPracticeTSFromHydroBase()` commands. See also `UseApex`.

**Region1** – Traditionally, the StateCU model used County/HUC identifiers to indicate the bounds of an area of interest, for calculations/reporting. StateDMI uses generalized Region1/Region2 identifiers, to allow more flexibility. See the `fillClimateStation()`, `fillCULocation()`, `setClimateStation()`, and `setCULocation()` commands. See also `Region1Type`.

**Region1Col** – The column number (or name) to be read from a delimited file for Region1 data. See the `fillCULocationsFromList()`, `readCULocationsFromList()`, `setCULocationsFromList()`, and `setCULocationsFromList()` commands.

**Region1Type** – Traditionally, the StateCU model used County/HUC identifiers to indicate the bounds of an area of interest, for calculations/reporting. StateDMI uses generalized Region1/Region2 identifiers, to allow more flexibility and some commands use this parameter to indicate that Region1 is County or another value. See the `fillCULocationsFromHydroBase()` command. See also `Region1`.

**Region2** – Traditionally, the StateCU model used County/HUC identifiers to indicate the bounds of an area of interest, for calculations/reporting. StateDMI uses generalized Region1/Region2 identifiers, to allow more flexibility. See the `fillClimateStation()`, `fillCULocation()`, `setClimateStation()`, and `setCULocation()` commands. See also `Region2Type`.

**Region2Col** – The column number (or name) to be read from a delimited file for Region2 data. See the `fillCULocationsFromList()`, `readCULocationsFromList()`, `setCULocationsFromList()`, and `setCULocationsFromList()` commands.

**Region2Type** – Traditionally, the StateCU model used County/HUC identifiers to indicate the bounds of an area of interest, for calculations/reporting. StateDMI uses generalized Region1/Region2 identifiers, to allow more flexibility and some commands use this parameter to indicate that Region2 is HUC or another value. See the `fillCULocationsFromHydroBase()` command. See also `Region2`.

**ReleaseMax** – The maximum release for a reservoir. See the `fillReservoirStation()` and `setReservoirStation()` commands.

**ReplaceResOption** – The replacement reservoir option for a diversion station. See the `fillDiversionStation()`, and `setDiversionStation()` commands.

**ReplaceResOptionCol** – The column number (or name) to be read from a delimited file for `ReplaceResOption` data. See the `setDiversionStationsFromList()` command.

**Returns** – The return flow locations, percentages, and delay table, for a diversion or well station. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**RightType** – The reservoir right type. See the `fillReservoirRight()` and `setReservoirRight()` commands.

**RiverNodeID** – The river node identifier associated with a station. See the `fillDiversionStation()`, `fillReservoirStation()`, `fillStreamEstimateStation()`, `fillStreamGageStation()`, `fillWellStation()`, `setDiversionStation()`, `setReservoirStation()`, and `setStreamEstimateStation()` commands.

**RiverNodeIDCol** – The column number (or name) to be read from a delimited file for RiverNodeID data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**Scale** – A scale factor to apply to data. See the `readDelayTablesFromStateMod()` command.

**SetEfficiency** – Indicate whether to set the efficiency when setting delay table information. See the `setDiversionStationDelayTablesFromRTN()` and `setWellStationDelayTablesFromRTN()` commands.

**SetEnd** – A DateTime that indicates the end of a data set process.

**SetFlag** – A character flag used to indicate when time series values are set. See the `limitDiversionDemandTSMonthlyToRights()`, and `limitDiversionHistoricalTSMonthlyToRights()` commands.

**SetStart** – A DateTime that indicates the start of a data set process.

**SetToMissing** – Indicate whether a set command should result in missing data, rather than supplying actual data values. This is sometimes necessary to undo previous processing. See the `setCropPatternTS()` commands.

**SpringFrostFlag** – The spring frost flag for a crop. See the `setCropCharacteristics()` command.

**SprinklerAcreage** – Indicate how to adjust the sprinkler acreage for locations that are irrigated by sprinklers. See the `synchronizeIrrigationPracticeAndCropPatternTS()` command.

**SprinklerAppEffMax** – The sprinkler application efficiency maximum for a CU Location. See the `setIrrigationPracticeTS()` command.

**SprinklerAppEffMaxCol** – The column number (or name) to be read from a delimited file for SprinklerAppEffMax data. See the `setIrrigationPracticeTSFromList()` command.

**StationID** – The station identifier associated with a data item (e.g., the station ID associated with a water right). See the `fillDiversionRight()`, `fillInstreamFlowRight()`, `fillReservoirRight()`, `fillWellRight()`, `setDiversionRight()`, `setInstreamFlowRight()`, and `setWellRight()` commands.

**SupplyTypeCol** – The column number (or name) for supply type (e.g., Surface or Ground indicator). See the `setIrrigationPracticeTSFromList()` and `setCropPatternTSFromList()` commands.

**SurfaceDelEffMax** – The surface water delivery efficiency maximum for a CU Location. See the `setIrrigationPracticeTS()` command.

**SurfaceDelEffMaxCol** – The column number (or name) to be read from a delimited file for SurfaceDelEffMax data. See the `setIrrigationPracticeTSFromList()` command.

**System** – See Collection.

**TempWtCol** – The column number (or name) to be read from a delimited file for TempWt (temperature weight) data. See the `setCULocationsFromList()` command.

**TSID** – Time series identifier, which is used to uniquely identify a time series. In full notation, this consists of a string similar to the following:  
Location.DataSource.DataType.Interval.Scenario~InputType~InputName. In abbreviated form, the InputType and InputName are often omitted. The InputType and InputName are typically used only by read and write commands. Because a TSID may be long (especially when file names are used for the InputName), an Alias may be assigned to the time series. The TSID parameter is typically used in commands for the time series that is being processed. See also Alias.

**TSID** – When used as a command parameter the time series identifier indicates the time series to be processed. The TSID or alias can typically be specified. See the `setDiversionDemandTSMonthly()` and `setWellDemandTSMonthly()` commands.

**Units** – Units associated with a data, often time series. See the `createCropPatternTSForCULocations()` and `createIrrigationPracticeTSForCULocations()` commands.

**UpstreamRiverNodeID** – The river node identifier for the upstream node in an instream flow reach, for instream flow stations. See the `fillInstreamFlowStation()`, and `setInstreamFlowStation()` commands.

**UseApex** – Indicates whether well rights APEX (alternate point and exchange) data should be added to water rights when they are read. See the `fillWellStationsFromHydroBase()`, `readWellRightsFromHydroBase()`, and `setIrrigationPracticeTSFromHydroBase()` commands. See also `ReadWellRights`.

**UseDiversionComments** – Indicate whether diversion comments in HydroBase should be used to provide additional zero diversion values for diversion time series. See the `readDiversionHistoricalTSMonthlyFromHydroBase()` command.

**UseOnOffDate** – Indicate whether the OnOff switch value for water rights should be used to determine the appropriation date for water rights. See the `limitDiversionDemandTSMonthlyToRights()`, `limitDiversionHistoricalTSMonthlyToRights()`, and `setIrrigationPracticeTSMaxPumpingToRights()` commands.

**UserName** – The user name for a diversion station. See the `fillDiversionStation()` and `setDiversionStation()` commands.

**UserNameCol** – The column number (or name) to be read from a delimited file for `UserName` data. See the `setDiversionStationsFromList()` command.

**UseStoredProcedures** – Indicates whether stored procedures should be used (versus straight SQL calls). This is being used to transition `HydroBase` queries to stored procedures. See the `openHydroBase()` command.

**UseType** – The water use type (e.g., to indicate agriculture) for a diversion station. See the `fillDiversionStation()`, `fillWellStation()`, `setDiversionStation()`, and `setWellStation()` commands.

**UseTypeCol** – The column number (or name) to be read from a delimited file for `UseType` data. See the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.

**Version** – Indicates the file version, to allow the software to handle different data formats. See the `readCropPatternTSTFromStateCU()`, `readIrrigationPracticeTSTFromStateCU()`, `readStateModB()`, `writeBlaneyCriddleToStateCU()`, `writeCropCharacteristicsToStateCU()`, `writeCULocationsToStateCU()`, and `writeIrrigationPracticeTSTToStateCU()` commands.

**Weights** – Station weights. See the `fillCULocationClimateStationWeights()` and `setCULocation()` commands.

**WorkingDir** – The working directory for the software, which can be used with relative paths to form absolute paths to files. See the `setWorkingDir()` command.

**WriteCropArea** – Indicate whether to write the crop area in addition to the percent, for the crop pattern time series file. See the `writeCropPatternTSTToStateCU()` commands.

**WriteOnlyTotal** – Indicate whether to write only the total crop area for the crop pattern time series file. See the `writeCropPatternTSTToStateCU()` commands.

**WriteHow** – Indicate how to write an output file (update or overwrite). See the `write*()` commands.

**Year** – Specify year(s) of interest. For example, when processing data related to wells, the year is used to indicate the year for parcel data. See the `fillWellStationsFromHydroBase()`, `readIrrigationPracticeTSTFromHydroBase()`, `readWellRightsFromHydroBase()`, `setIrrigationPracticeTSTFromHydroBase()`, `setIrrigationPracticeTSSprinklerAreaFromList()`, `setWellAggregate()`, `setWellSystem()`, and `setWellSystemFromList()` commands.

**YearCol** – The column number (or name) to be read from a delimited file for `Year` data. See the `setIrrigationPracticeTSTFromList()` command.

---

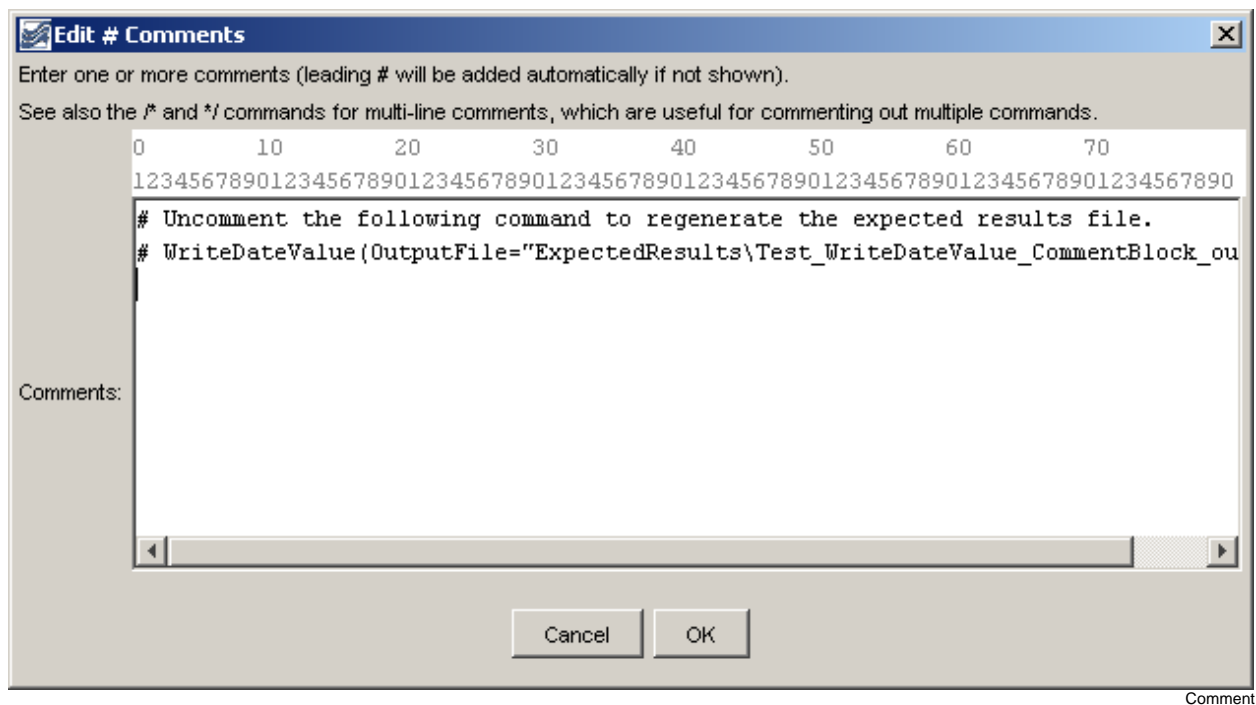
# Command Reference: #

## Comment line

**General Command**  
Version 3.08.02, 2010-01-06

The # command indicates single-line comments. Commands can be converted to and from # comments. See also the /\* and \*/ comment block commands, which are to comment multiple commands.

The following dialog is used to edit the command and illustrates the command syntax:



**# Command Editor**

The command syntax is as follows:

```
# Some text
```

A sample command file is as follows:

```
#  
# Some comments...  
#
```

This page is intentionally blank.

---

# Command Reference: \*/

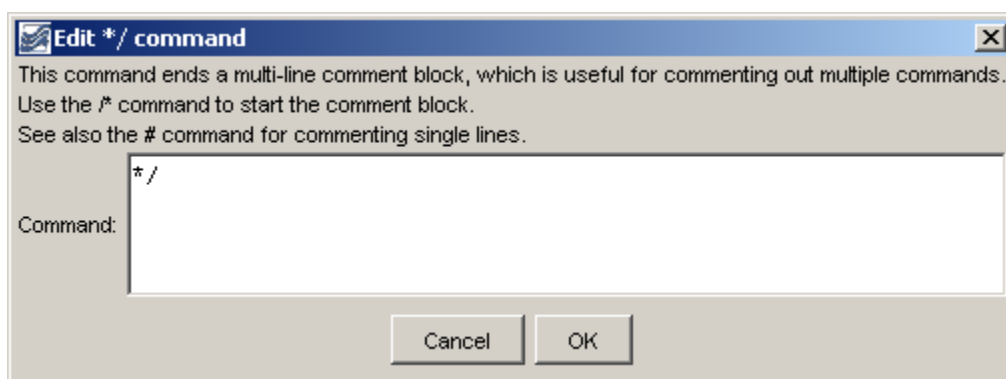
## Comment block end

### General Command

Version 3.08.02, 2010-01-06

The \*/ command ends a multi-line comment block and is useful for inserting long comments or temporarily commenting out blocks of commands. See also the /\* and # commands. Commands between the /\* and \*/ are not converted to comments but are skipped during processing.

The following dialog is used to edit the command and illustrates the command syntax:



CommentBlockEnd

### \*/ Command Editor

The command syntax is as follows:

\*/

A sample command file is as follows:

```
/*  
SomeCommentedOutCommands()...  
*/
```

This page is intentionally blank.



---

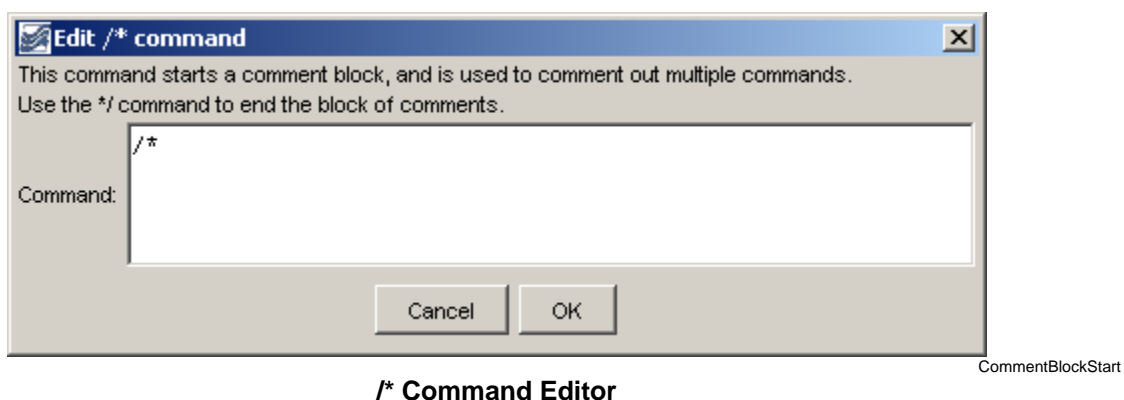
# Command Reference: /\*

## Comment block start

**General Command**  
Version 3.08.02, 2010-01-06

The /\* command starts a multi-line comment block and is useful for inserting long comments or temporarily commenting out blocks of commands. See also the \*/ and # commands. Commands between the /\* and \*/ are not converted to comments but are skipped during processing.

The following dialog is used to edit the command and illustrates the command syntax:



**/\* Command Editor**

The command syntax is as follows:

```
/*
```

A sample command file is as follows:

```
/*  
SomeCommentedOutCommands ( ) ...  
*/
```

This page is intentionally blank.

---

# Command Reference: AggregateWellRights ()

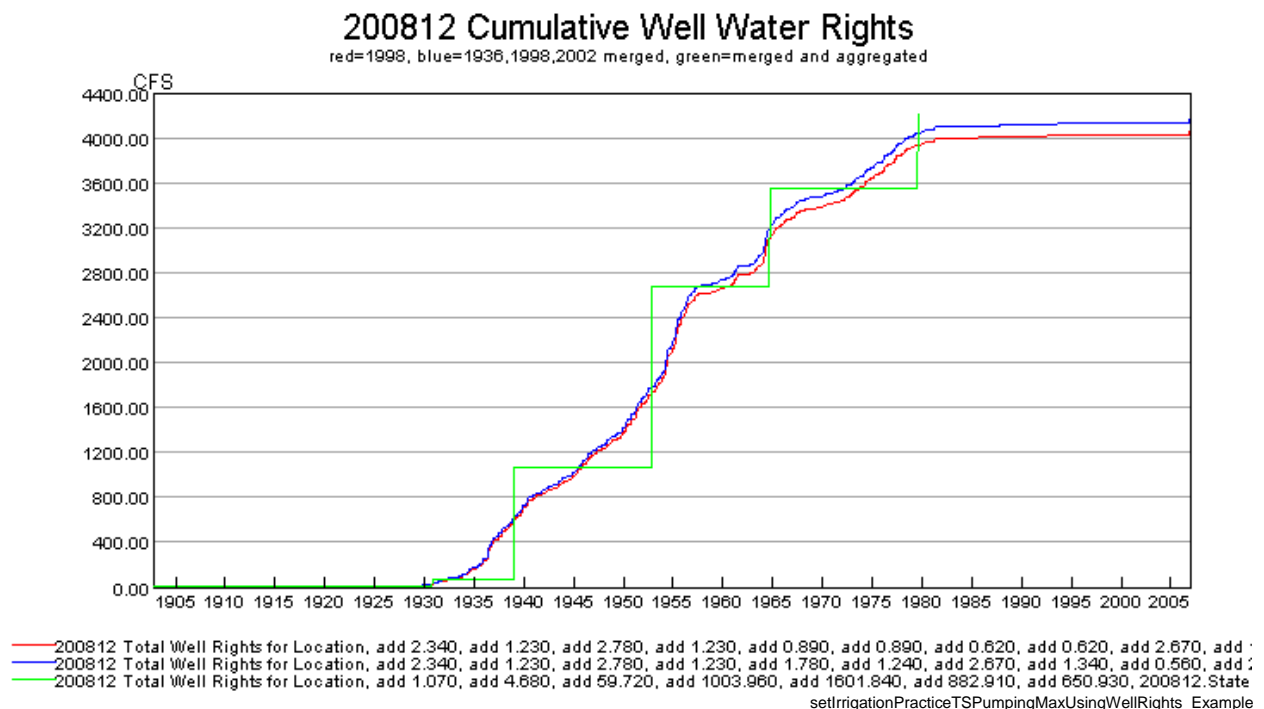
Aggregate well right data from by weighting the decree by administration number

## StateMod Command

Version 3.09.00, 2010-01-26

The `AggregateWellRights()` command aggregates well rights by weighting the decree by administration number (in simple terms the number of days since 1849). It is typical to aggregate water rights in basins where individual rights do not need to be modeled (modeling individual rights increases the run time and amount of model output). For example, Río Grande well rights are typically aggregated; however, South Platte rights are not, due to detailed modeling of augmentation plans. Aggregating well rights is typically the last step in well right processing before writing the well rights file. The `ReadWellRightsFromHydroBase()` command prior to StateDMI 2.14 performed aggregation in one step; however, this is no longer desirable because unaggregated rights are needed for data processing, such as limiting groundwater-only supply parcels back in time, and setting the pumping maximum in the irrigation practice time series.

The following figure illustrates the difference between raw, merged, and aggregated rights. Raw rights contain output for multiple years of irrigated lands parcel data. Merged rights consider all years of irrigated lands data but avoid double-counting rights that result from more than one year of parcel data processing (see the `MergeWellRights()` command).



The end result of aggregation is well rights that have an identifier matching the location, with a number suffix. The suffix “.01” corresponds to rights with an administration number  $\leq$  to the first administration number class. The last administration number class should therefore be larger than any administration number that is expected (e.g., use 99999.99999). Example output in StateMod format is as follows:

#>	ID	Name	Struct	Admin #	Decree	On/Off
#>	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----e
	200511W.08		200511	57343.00000	3.01	2006
	200812W.03		200812	21307.00000	4.68	1908
	200812W.04		200812	29515.00000	45.48	1930
	200812W.05		200812	32589.00000	954.42	1939
	200812W.06		200812	37671.00000	1608.05	1953
	200812W.07		200812	41917.00000	911.48	1964
	200812W.08		200812	47211.00000	659.28	1979

The following steps occur to aggregate well water rights at each location where aggregate/systems are specified with parcels or a well station has an associated diversion ID:

1. Initialize aggregate water rights. Aggregate water rights for each water class are initialized to zero. If at the end of processing the value is still zero, a right will NOT be added for the class. Aggregate rights for a groundwater-only location have an identifier that starts with the location. Other locations that have supplemental supply use the location identifier, followed by a "W". All rights then have a .NN ending, corresponding to the water right class.
2. For each class, the following sums are calculated:  $\text{sum}(\text{decree} * \text{AdminNum})$  and  $\text{sum}(\text{decree})$ , where the administration number is determined from the appropriation date derived from the original HydroBase administration number (it will not have a remainder).
3. After processing all rights for the location, the final administration number for the class is determined (it will not have a remainder) as:  $\text{int}(\text{sum}(\text{decree} * \text{AdminNum}) / \text{sum}(\text{decree}))$ .
4. For each non-zero aggregate, a well right is added for the location. Only the whole number part of the administration will be set (the remainder will be zero).
5. The well rights are added to the overall list for output. All previous rights for the location are replaced by the aggregate rights.

If the output does not show aggregation as expected, verify that the location is properly being specified as a groundwater only location with aggregate/system parcel list, and that the associated diversion ID is specified in well station or list file used as input.

The following dialog is used to edit the command and illustrates the syntax of the command:

**Edit AggregateWellRights() Command**

This command aggregates well water rights, resulting in fewer water rights. This increases model performance. Aggregation occurs by weighting decree and administration numbers at a location. Water right classes must be supplied as administration number (NNNNN.NNNNN) breaks. The resulting aggregate rights replace the original rights.

Admin. number classes: 5000.00000,30000.00000,35000.00000,40000.00000,45000.00000,99999.99999|

OnOff default: AppropriationDate  Optional - default OnOff switch (default=AppropriationDate).

Command: AggregateWellRights (AdminNumClasses="10000.00000,20000.00000,25000.00000,30000.00000,35000.00000,40000.00000,45000.00000,99999.99999",OnOffDefault="AppropriationDate")

OK Cancel

AggregateWellRights

### AggregateWellRights() Command Editor

The command syntax is as follows:

```
AggregateWellRights(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
AdminNumClasses	A list of administration numbers, separated by spaces or commas, to define the breaks for aggregate water rights, for well aggregates. For example, if the class breaks are 10000.000, 20000.00000, and 99999.99999, the first group will contain water rights with administration numbers $\leq 10000.00000$ , the second will contain water rights with administration number $> 10000.00000$ and $\leq 20000.00000$ , and the third will contain water rights with administration number $> 20000.00000$ and $\leq 99999.99999$ . The last administration number should be larger than any data value that is expected to occur.	If not specified, diversion aggregates will be treated as diversion systems, with all water rights explicitly included in output.
OnOffDefault	Indicates how to set the on/off switch for resulting water rights. A value of 1 indicates that the right is on for the whole period. If the value is AppropriationDate, the switch is set to the year corresponding to the appropriation date, indicating that the right will be turned on starting in the year. The appropriation date for aggregate rights is taken from the whole number part of the administration number because the remainder is a result of the weighting and does not have meaning.	Appropriation Date

The following example illustrates the full process for creating well rights in the Rio Grande basin, including well right aggregation (this is an abbreviated command file with repetitive steps removed):

```

StartLog(LogFile="Wells_wer.StateDMI.log")
# Wells_WER.StateDMI
#
# Step 1 - open a log file for this run
#
StartLog(LogFile="Wells_WER.StateDMI.log")
#
# Step 2 - read stations
# readWellStationsFromStateMod(InputFile="rg2007.wes")
ReadWellStationsFromStateMod(InputFile="rg2007.wes")
#
# Step 3 - define aggregates and systems
# Diversions are collections using a list of WDIDs, and the list of IDs is
# constant through the model period.
# Aggregates will result in well rights being aggregated.
# Systems will be modeled with all well rights (no aggregation).
# Well-only lands are collections using a list of parcel identifiers, and
# the lists are specified for each year where data are available because the
# parcel identifiers change from year to year.
#
# Diversions with and without groundwater supply...
SetWellAggregateFromList(ListFile="..\Diversions\rgTW_divaggregates.csv",
  IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartType=Ditch)
SetDiversionsSystemFromList(ListFile="..\Diversions\rgTW_divsystems.csv",
  IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartType=Ditch)
# Wells with only groundwater supply...#
SetWellSystemFromList(ListFile="..\Wells\1998_GWonly_agg.csv",Year=1998,Div=3,
  PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - read rights from HydroBase (NO AGGREGATION)
# Include Appropriation Date for on/off
# 1936 is included for more rights and because used in later data filling.
# APEX is NOT used.
#
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1998",Div="3",
  DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
  ReadWellRights=True,UseApex=False,OnOffDefault=AppropriationDate)
#
# Step 5 - set data not in HydroBase
# M&I are not tied to an irrigated parcel and therefore may not be in
# HydroBase.
# Also, StateDMI does not currently read well rights/permits for explicit
# non-irrigation well locations.
#
# 5a; Set Alamosa Refuge
# Mumm Well and estimated small wells (4 cfs) (refine only with additional information from USFWS)
SetWellRight(ID="20MS06W.98",Name="Small_ANWR_Wells",StationID="20MS06",
  AdministrationNumber=90000.00000,Decree=4.00,OnOff=1,IfNotFound=Add,IfFound=Warn)
...many omitted
#
# Step 6 - write rights from multiple years of irrigated lands
# Note since not aggregating, the ID's assigned will be
# true Well IDs, not structure id.01, etc.
# The *wer file is written containing all parcel years and
# "data comments" on the right side of the file are written to
# facilitate use when filling the *cds and *ipy files.
# The following is used to fill the CDS and IPY acreage prior to 1998,
# using the rights resulting from 1998 parcels.
#
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
WriteWellRightsToStateMod(OutputFile="rg2007_NotMerged.wer",WriteDataComments=True)
#
# Step 7 - merge multiple years (but do not aggregate)
# The water rights resulting from multiple years of parcel data (above) are
# merged. Blocks of rights with the same right ID and location ID are
# checked. If all are the same in two years, then all are kept in the
# result. Otherwise, the rights from the year resulting in the highest

```

```
# decree sum are kept in the result. The process compares two years at a
# time, going through all years where data are available.
# The following version of the file is used to set IPY max pumping.
#
MergeWellRights()
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
WriteWellRightsToStateMod(OutputFile="rg2007.wer")
#
WriteWellRightsToStateMod(OutputFile="rg2007.wer")
WriteWellRightsToStateMod(OutputFile="..\StateMod\Historic\rg2007.wer")
#
# Step 8 - aggregate into water rights classes
# This step is needed in the Rio Grande but not in the South Platte.
# Rights are aggregated by weighting by decree and administration number.
# The right identifiers are set to LocationID.##, where ## is the class.
#
AggregateWellRights(AdminNumClasses="10000.00000,20000.00000,25000.00000,
30000.00000,35000.00000,40000.00000,45000.00000,99999.99999",OnOffDefault="AppropriationDate")
WriteWellRightsToStateMod(OutputFile="rg2007_Agg.wer")
WriteWellRightsToStateMod(OutputFile="..\StateMod\Historic\rg2007_Agg.wer")
#
# Check the results
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Wells_wer.StateDMI.check.html")
```

This page is intentionally blank.



---

# Command Reference:

## CalculateDiversiionDemandTSMonthly()

**Calculate diversion demand time series (monthly) using irrigation water requirement and average monthly efficiencies**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CalculateDiversiionDemandTSMonthly()` command calculates diversion demand time series (monthly) by dividing the irrigation water requirement (IWR) time series (monthly) by average monthly efficiencies. The diversion stations should first be read with another command (e.g., `ReadDivsionStationsFromStateMod()`) and provide the list of diversion stations to be processed – every diversion station will have a demand time series in the result. The IWR time series should have been read by a previous command. The diversion station efficiencies should also have been calculated previously. The output year type must be specified correctly because efficiencies are stored in diversion stations according to the year type for the StateMod data set. The following rules apply:

- If a diversion station is defined as a `MultiStruct`, the demand for the primary station (the first one listed in the `MultiStruct`) is the sum of the demands for all of its parts and the average efficiency for the total will be used (as set in previous commands). The demands for the secondary stations will be set to zero.
- If required time series data are not available for calculations (i.e., no IWR time series is found), a demand time series with zero values is created. This demand time series can be replaced with `SetDiversiionDemandTSMonthly()` commands, if necessary.
- If an IWR value for a month is zero, then the demand value for the month is set to zero (whether there was a historical diversion or not). In this case the demand can later be adjusted to a larger value using the `CalculateDiversiionDemandTSMonthlyAsMax()` command.
- If the efficiency for a month is zero: if the IWR is zero, then the demand is set to zero; otherwise the demand is set to missing.

The following dialog is used to edit the command and illustrates the syntax of the command.

**CalculateDiversionDemandTSMonthly() Command Editor**

The command syntax is as follows:

`CalculateDiversionDemandTSMonthly (Parameter=Value,...)`

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following abbreviated command file illustrates how irrigation water requirement time series can be processed into average demand time series:

```

StartLog(LogFile="Cddm.commands.StateDMI.log")
# Cddm.commands.StateDMI
#
# StateDMI command file to create the Calculated demand file
#
# Step 1 - set the output period, used to compute averages...
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read historical diversion file -defines structures for *.ddm file
#           plus read *.ddh file
#
ReadDiversionStationsFromStateMod(InputFile="..\StateMod\cm2005.dds")
ReadDiversionHistoricalTSMonthlyFromStateMod(InputFile="..\StateMod\cm2005.ddh")
#
# Step 3 - read StateCU *.iwr and *.def files (irrigation requirements and average efficiencies)
#
ReadIrrigationWaterRequirementTSMonthlyFromStateCU(InputFile="..\StateMod\cm2005.iwr")
# calculateDiversionStationEfficiencies(ID="*",EffMin=0,EffMax=60,
#   EffCalcStart=10/1974,EffCalcEnd=9/2004,LEZeroInAverage=False)
SetDiversionStationsFromList(ListFile="cm2005.def",IDCol="1",EffMonthlyCol="2",
  Delim="Space",MergeDelim=True)
#
# Step 4 - determine calculated demand = iwr/efficiency
#           - take max of calculated demand and historical diversion
#
CalculateDiversionDemandTSMonthly(ID="*")
CalculateDiversionDemandTSMonthlyAsMax(ID="*")
#
# Step 5 - set carriers nodes demand to 0, set full demand and summary demand nodes
#
# set carrier "transbasin" diversion to Divide Creek to "0", use operating rules to satisfy demand
SetDiversionDemandTSMonthlyConstant(ID="724721",Constant=0)
# place summary demand at the Moffat Tunnel, zero out collection points
SetDiversionDemandTSMonthly(ID="514655",TSID="514655..DivTotal.Month~StateMod~514655.stm")
... similar commands omitted...
#
# Step 6 - set calculated demand to historic for structures whose historical acreage is
#           different from current
#
SetDiversionDemandTSMonthly(ID="360687",TSID="360687..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="360725",TSID="360725..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
...similar commands omitted...
#
# Set Ute WCD demand node structure and set other structures to zero
SetDiversionDemandTSMonthly(ID="950020",TSID="950020..DivTotal.Month~StateMod~950020.stm")
SetDiversionDemandTSMonthlyConstant(ID="950030",Constant=0)
... similar commands omitted...
#
# Set Orchard Mesa Check
SetDiversionDemandTSMonthly(ID="950003",TSID="950003..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
#
# Set Excess HUP node demands for Homestake, Dillon, Williams Fork, and Wolford Reservoirs
SetDiversionDemandTSMonthlyConstant(ID="954516D",Constant=999999)
...similar commands omitted...
# Step 7 - write out calculated demand file
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005C.ddm")
#
# Check the results
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Cddm.commands.StateDMI.check.html")

```

This page is intentionally blank.

---

# Command Reference:

## CalculateDiversiionDemandTSMonthlyAsMax()

**Calculate diversion demand time series (monthly) as the maximum of the existing demands and the historical time series**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `CalculateDiversiionDemandTSMonthlyAsMax()` command calculates diversion demand time series (monthly) as the maximum of the existing demands and the historical diversion time series. This command is typically used after the `CalculateDiversiionDemandTSMonthly()` command.

If a diversion is defined as a `MultiStruct`, the primary diversion station will be checked using the sum of the historical time series and a sum of the demand time series. Secondary diversion stations will not be checked (the demand will likely have been set to zero in a previous `CalculateDiversiionDemandTSMonthly()` command).

If necessary, use set commands after this command to force demand time series values (e.g., zeros).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CalculateDiversiionDemandTSMonthlyAsMax() Command**

This command calculates the diversion demand time series as the maximum of the demand time series and historical diversion time series. An initial estimate of the demand time series must have been made (e.g., from IWR/EffAve). For diversion `MultiStruct` locations, the comparison for the primary (first) diversion station is made using the primary station demand and total historical time series for the `MultiStruct` parts. Demands for secondary locations are not checked (they should be zero from previous commands). The diversion station identifier is used to match the time series that is read. The output period must be specified with a previous command.

Diversion station ID:  Required - stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CalculateDiversiionDemandTSMonthlyAsMax ( ID="*" )
```

**CalculateDiversiionDemandTSMonthlyAsMax() Command Editor**

The command syntax is as follows:

```
CalculateDiversiionDemandTSMonthlyAsMax ( Parameter=value , ... )
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNot Found	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following abbreviated command file illustrates how irrigation water requirement time series can be processed into average demand time series:

```

StartLog(LogFile="Cddm.commands.StateDMI.log")
# Cddm.commands.StateDMI
#
# StateDMI command file to create the Calculated demand file
#
# Step 1 - set the output period, used to compute averages...
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read historical diversion file -defines structures for *.ddm file
#           plus read *.ddh file
#
ReadDiversionStationsFromStateMod(InputFile="..\StateMod\cm2005.dds")
ReadDiversionHistoricalTSMonthlyFromStateMod(InputFile="..\StateMod\cm2005.ddh")
#
# Step 3 - read StateCU *.iwr and *.def files (irrigation requirements and average efficiencies)
#
ReadIrrigationWaterRequirementTSMonthlyFromStateCU(InputFile="..\StateMod\cm2005.iwr")
# calculateDiversionStationEfficiencies(ID="*",EffMin=0,EffMax=60,
#   EffCalcStart=10/1974,EffCalcEnd=9/2004,LEZeroInAverage=False)
SetDiversionStationsFromList(ListFile="cm2005.def",IDCol="1",EffMonthlyCol="2",
  Delim="Space",MergeDelim=True)
#
# Step 4 - determine calculated demand = iwr/efficiency
#           - take max of calculated demand and historical diversion
#
CalculateDiversionDemandTSMonthly(ID="*")
CalculateDiversionDemandTSMonthlyAsMax(ID="*")
#
# Step 5 - set carriers nodes demand to 0, set full demand and summary demand nodes
#
# set carrier "transbasin" diversion to Divide Creek to "0", use operating rules to satisfy
demand
SetDiversionDemandTSMonthlyConstant(ID="724721",Constant=0)
...similar commands omitted...
# Step 7 - write out calculated demand file
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005C.ddm")
#
# Check the results
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Cddm.commands.StateDMI.check.html")

```

---

# Command Reference: CalculateDiversiOnStationEfficiencies()

Calculate diversion station average efficiencies using historical and irrigation water requirement time series

StateMod Command  
Version 3.09.01, 2010-02-01

**This command is generally not used with current modeling procedures. Instead, a variable efficiency approach is used where monthly average efficiencies are computed in StateCU and are set in diversion stations using a `SetDiversiOnStationsFromList(..., EffMonthlyCol=...)` command. This command is retained to duplicate previous work.**

The `CalculateDiversiOnStationEfficiencies()` command calculates average monthly efficiencies for diversion stations and updates the diversion station information in memory. Efficiencies are calculated as irrigation water requirement divided by historical diversion time series. The detailed results of calculations can optionally be printed to a report file. The diversion historical time series (monthly) and irrigation water requirement time series (monthly) should be read or created with other commands, and should be filled before calculations, if appropriate. Only StateMod diversion stations with demand source for agricultural irrigation will be processed. The output year type must be specified correctly because efficiencies are stored in diversion stations according to the year type for the StateMod data set. Diversion MultiStruct stations are processed by using the total irrigation water requirement and historical diversions for all stations in the MultiStruct. A `WriteDiversiOnStationsToStateMod()` command must be executed to actually write the updated efficiency data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CalculateDiversiOnStationEfficiencies() Command**

This command calculates monthly efficiencies for each diversion station that is defined. Efficiencies are computed as the ratio of irrigation (consumptive) water requirement divided by historical diversions. It is expected that both sets of time series have been filled appropriately. If the efficiency report file is provided, details of the efficiency calculations will be printed to the file.

Diversion station ID:	<input type="text" value="*"/>	Required - stations to process (use * for wildcard).
Efficiency min. (%):	<input type="text" value="0"/>	Optional - minimum efficiency (default=no constraint).
Efficiency max. (%):	<input type="text" value="60"/>	Optional - maximum efficiency (default=no constraint).
Calculation start date:	<input type="text" value="10/1974"/>	Optional - start date for efficiency calculations (blank=all).
Calculation end date:	<input type="text" value="9/2004"/>	Optional - end date for efficiency calculations (blank=all).
<= zero values in average?:	<input type="checkbox"/> False	Optional - are values <= zero used in averages (used later in filling)? (default=True.)
Efficiency report file:	<input type="text" value=""/>	<input type="button" value="Browse"/>
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
CalculateDiversiOnStationEfficiencies (ID="*", EffMin=0, EffMax=60, EffCalcStart=10/1974, EffCalcEnd=9/2004, LEZeroInAverage=False)
```

CalculateDiversiOnStationEfficiencies() Command Editor

The command syntax is as follows:

```
CalculateDiversionStationEfficiencies(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
EffMin	Minimum efficiency to allow, percent. Calculated efficiencies less than this value will be set to the minimum.	Do not constrain the efficiency.
EffMax	Maximum efficiency to allow, percent. Calculated efficiencies greater than this value will be set to the maximum.	Do not constrain the efficiency.
EffCalcStart	The start date (e.g., YYYY-MM) for efficiency calculations. Use this to limit the period for data considered in calculations.	Use the full period.
EffCalcEnd	The end date (e.g., YYYY-MM) for efficiency calculations. Use this to limit the period for data considered in calculations.	Use the full period.
LEZeroInAverage	If true, values less than or equal to zero will be considered when computing monthly time series averages. If false, values less than or equal to zero will be excluded from the averages.	True
EffReportFile	If specified, a high-detail report will be created, listing for each diversion station the irrigation water requirement, historical diversion, and resulting efficiency values. Creating the report slows processing slightly.	If blank, no report is generated.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

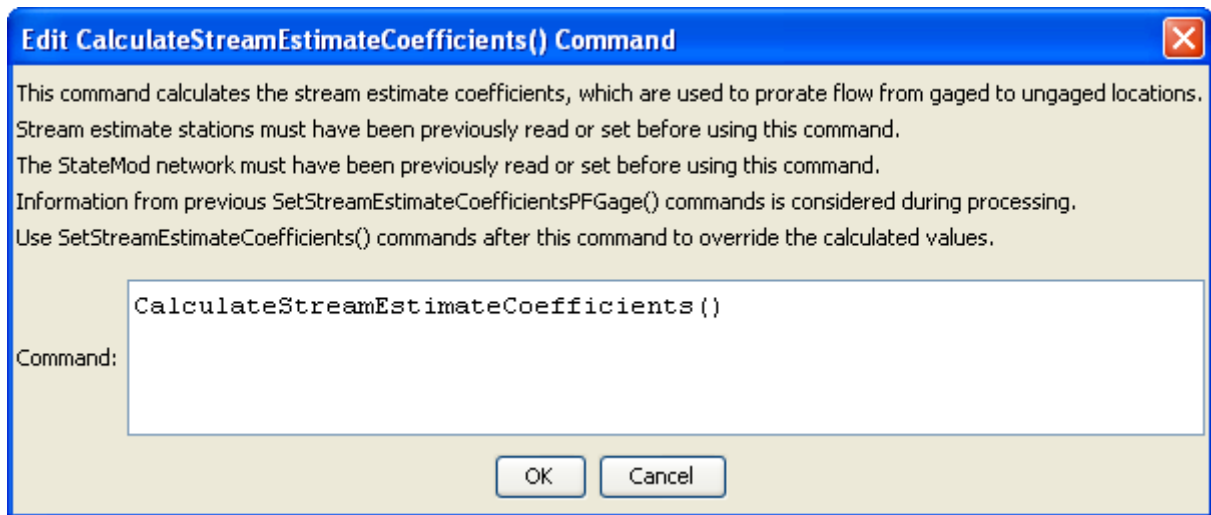
# Command Reference: CalculateStreamEstimateCoefficients()

Calculate stream estimate coefficients data

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CalculateStreamEstimateCoefficients()` command calculates stream estimate coefficients for each stream estimate station that is in memory – the previous values will be overwritten. If `SetStreamEstimateCoefficientsPFGage()` commands are used, they should be specified before this command. Conversely, `SetStreamEstimateCoefficients()` commands, if used, should be provided after this command. The following dialog is used to edit the command and illustrates the syntax of the command.



CalculateStreamEstimateCoefficients

## CalculateStreamEstimateCoefficients() Command Editor

The command syntax is as follows:

```
CalculateStreamEstimateCoefficients(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
	Currently, this command has no parameters.	

The following command file illustrates how a StateMod stream estimate coefficients file can be created:

```

StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#           this baseflow nodes are generally on smaller non-gaged tribs and have
#           different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="360801",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="362002",GageID="09054000")
SetStreamEstimateCoefficientsPFGage(ID="360829",GageID="09047500")
..similar commands omitted...
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374641",ProrationFactor=0.200,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374648",ProrationFactor=0.350,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="380880",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="381594",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="384617",ProrationFactor=0.700,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510639",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514603",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514620",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510728",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530555",ProrationFactor=0.180,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530678",ProrationFactor=0.230,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="531082",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="954683",ProrationFactor=0.400,IfNotFound=Warn)
#
# Step 5 - create streamflow estimate coefficient file
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")

```

---

# Command Reference: CalculateWellDemandTSMonthly()

**Calculate well demand time series (monthly) using irrigation water requirement and average monthly efficiencies**

## StateMod Command

Version 3.09.01, 2010-02-01

The `CalculateWellDemandTSMonthly()` command calculates well demand time series (monthly) by dividing the irrigation water requirement (IWR) time series (monthly) by average monthly efficiencies. The IWR time series should have been read by a previous command. The well station efficiencies should also have been calculated, set, or read using previous commands. The output year type must be specified correctly because efficiencies are stored in well stations according to the year type for the StateMod data set. If time series data are not available, a demand time series with zero values is created – this time series can be replaced with `SetWellDemandTSMonthly()` commands, if necessary. Only well stations that have a demand type (StateMod well station `idvcomw`) equal to one are processed. For “diversion + well” well stations, the demand is typically calculated using only the diversion station IWR and historical diversion time series and is written to the diversion demand time series file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CalculateWellDemandTSMonthly() Command**

This command calculates the well demand time series (monthly) by dividing the IWR/CWR time series by the average monthly efficiencies. The well station identifier is used to match the time series that is read. The output period must be specified with a previous command.

Well station ID:  Required - stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CalculateWellDemandTSMonthly (ID="*")
```

CalculateWellDemandTSMonthly

### CalculateWellDemandTSMonthly() Command Editor

The command syntax is as follows:

```
CalculateWellDemandTSMonthly(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference:

## CalculateWellDemandTSMonthlyAsMax()

**Calculate well demand time series (monthly) as the maximum of the existing demands and the historical pumping time series**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CalculateWellDemandTSMonthlyAsMax()` command calculates well demand time series (monthly) as the maximum of the existing demands and the historical pumping time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CalculateWellDemandTSMonthlyAsMax() Command**

This command calculates the well demand time series as the maximum of the demand time series and historical pumping time series. An initial estimate of the demand time series must have been made (e.g., from IWR/EffAve). The well station identifier is used to match the time series that is read. The output period must be specified with a previous command.

Well station ID:  Required - stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CalculateWellDemandTSMonthlyAsMax ( ID="*" )
```

CalculateWellDemandTSMonthlyAsMax

**CalculateWellDemandTSMonthlyAsMax() Command Editor**

The command syntax is as follows:

```
CalculateWellDemandTSMonthlyAsMax( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNot Found	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Add – add the time series if the ID is not matched and is not a wildcard</li><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: CalculateWellStationEfficiencies()

**Calculate well station average efficiencies using historical pumping and irrigation water requirement time series**

**StateMod Command**  
Version 3.09.01, 2010-02-01

**This command is generally not used with current modeling procedures. Instead, a variable efficiency approach is used where monthly average efficiencies are computed in StateCU and are set in well stations using a `SetWellStationsFromList(...,EffMonthlyCol=...)` command. This command is retained to duplicate previous work.**

The `CalculateWellStationEfficiencies()` command calculates average monthly efficiencies for well stations and updates the well station information in memory. Efficiencies are calculated as irrigation water requirement divided by historical well pumping time series. The detailed results of calculations can optionally be printed to a report file. The well historical pumping time series (monthly) and irrigation water requirement time series (monthly) should be read or created with other commands, and should be filled before efficiency calculations, if appropriate. Only StateMod well stations with demand type of 1 (monthly total demand) will be processed. The output year type must be specified correctly because efficiencies are stored in diversion stations according to the year type for the StateMod data set. A `WriteWellStationsToStateMod()` command must be executed to actually write the updated efficiency data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CalculateWellStationEfficiencies() Command**

This command calculates monthly efficiencies for each well station that is defined.  
Efficiencies are computed as the ratio of irrigation (consumptive) water requirement divided by historical pumping.  
It is expected that both sets of time series have been filled appropriately.  
If the efficiency report file is provided, details of the efficiency calculations will be printed to the file.

Well station ID:	*	Required - stations to process (use * for wildcard).
Efficiency min. (%):	10	Optional - minimum efficiency (default=no constraint).
Efficiency max. (%):	70	Optional - maximum efficiency (default=no constraint).
Calculation start date:		Optional - start date for efficiency calculations (blank=all).
Calculation end date:		Optional - end date for efficiency calculations (blank=all).
<= zero values in average?:	<input type="button" value="v"/>	Optional - are values <= zero used in averages (used later in filling)? (default=True.)
Efficiency report file:		<input type="button" value="Browse"/>
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
CalculateWellStationEfficiencies (ID="*", EffMin=10, EffMax=70)
```

CalculateWellStationEfficiencies

**CalculateWellStationEfficiencies() Command Editor**

The command syntax is as follows:

```
CalculateWellStationEfficiencies(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
EffMin	Minimum efficiency to allow, percent. Calculated efficiencies less than this value will be set to the minimum.	Do not constrain the efficiency.
EffMax	Maximum efficiency to allow, percent. Calculated efficiencies greater than this value will be set to the maximum.	Do not constrain the efficiency.
EffCalcStart	The start date (e.g., YYYY-MM) for efficiency calculations. Use this to limit the period for data considered in calculations.	Use the full period.
EffCalcEnd	The end date (e.g., YYYY-MM) for efficiency calculations. Use this to limit the period for data considered in calculations.	Use the full period.
LEZeroInAverage	If true, values less than or equal to zero will be considered when computing monthly time series averages. If false, values less than or equal to zero will be excluded from the averages.	true
EffReportFile	If specified, a high-detail report will be created, listing for each well station the irrigation water requirement, historical well pumping, and resulting efficiency values. Creating the report slows processing.	If blank, no report is generated.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference: CheckBlaneyCriddle()

## Check Blaney-Criddle data for problems

### StateCU Command

Version 3.08.02, 2010-01-05

The `CheckBlaneyCriddle()` command checks the Blaney-Criddle crop coefficient data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckBlaneyCriddle() Command**

This command checks StateCU Blaney-Criddle crop coefficients.  
Currently no cross-checks are done with other StateCU components.  
Warnings are generated for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid numerical values (e.g., day > 365).

Crop type (name):  Required - specify the crops to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`CheckBlaneyCriddle ( ID = "*" )`

CheckBlaneyCriddle

### CheckBlaneyCriddle() Command Editor

The command syntax is as follows:

```
CheckBlaneyCriddle(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The name of the crop(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

The following example command file illustrates how Blaney-Criddle coefficients can be defined, checked, and written to a StateCU file:

```
StartLog(LogFile="Crops_KBC.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Blaney-Criddle coefficients File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-23 SAM, RTi               Update for Rio Grande Phase 5.
#
# Step 1 - read data from HydroBase
#
# Read the general Blaney-Criddle coefficients first and then override with Rio Grande
# data.
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 3 - write the file
#
SortBlaneyCriddle(Order=Ascending)
WriteBlaneyCriddleToStateCU(OutputFile="rg2007.kbc")
#
# Check the results
#
CheckBlaneyCriddle(ID="*")
WriteCheckFile(OutputFile="rg2007.kbc.check.html")
```

---

# Command Reference: CheckClimateStations()

Check climate station data for problems

## StateCU Command

Version 3.08.02, 2010-01-05

The `CheckClimateStations()` command checks the climate stations for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckClimateStations() Command**

This command checks StateCU climate station data.  
Currently no cross-checks are done with other StateCU components.  
Warnings are generated for the follow conditions:  
1) Missing (undefined) required values.  
2) Invalid numerical values (e.g., latitude > 90 degrees).

Climate station identifier:  Required - specify the climate stations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`CheckClimateStations (ID="*")`

OK Cancel

CheckClimateStations

**CheckClimateStations() Command Editor**

The command syntax is as follows:

```
CheckClimateStations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The identifier for the station(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the climate station identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the climate station identifier is not matched</li> <li>Warn – generate a warning message if the climate station identifier is not matched</li> </ul>	Warn

The following example command file illustrates how climate stations can be defined, sorted, checked, and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
FillClimateStationsFromHydroBase(ID="*")
SetClimateStation(ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation(ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation(ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation(ID="0484",Region1="MOFFAT",IfNotFound=Add)
SortClimateStations()
WriteClimateStationsToStateCU(OutputFile="COclim2006.cli")
#
# Check the results
#
CheckClimateStations(ID="*")
WriteCheckFile(OutputFile="COclim2006.cli.check.html")
```

---

# Command Reference: CheckCropCharacteristics()

Check crop characteristics data for problems

**StateCU Command**  
Version 3.08.02, 2010-01-05

The `CheckCropCharacteristics()` command checks the crop characteristics data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckCropCharacteristics() Command**

This command checks StateCU crop characteristics.  
Currently no cross-checks are done with other StateCU components.  
Warnings are generated for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid numerical values (e.g., month > 12).

Crop type (name):  Required - specify the crops to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

**CheckCropCharacteristics() Command Editor**

CheckCropCharacteristics

The command syntax is as follows:

```
CheckCropCharacteristics( Parameter=Value, ... )
```

### Command Parameters

Parameter	Description	Default
ID	The name of the crop(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

The following example command file illustrates how crop characteristics can be defined, checked, and written to a StateCU file:

```
StartLog(LogFile="Crops_CCH.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Crop Characteristics File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-22 SAM, RTi               Use new directory structure, current
#                                   software and HydroBase.
#
# Step 1 - read data from HydroBase
#
# Read the general TR-21 characteristics first and then override with Rio Grande
# data.
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 2 - adjust crop characteristics if needed
#   No resets are needed.
#
# Step 3 - write the file
#
WriteCropCharacteristicsToStateCU(OutputFile="rg2007.cch")
#
# Check the results
#
CheckCropCharacteristics(ID="*")
WriteCheckFile(OutputFile="rg2007.cch.check.html")
```

# Command Reference: CheckCropPatternTS()

## Check crop pattern time series data for problems

### StateCU Command

Version 3.08.02, 2010-01-05

The `CheckCropPatternTS()` command checks the crop pattern time series data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckCropPatternTS() Command**

This command checks StateCU crop pattern time series at CU locations.  
Currently no cross-checks are done with other StateCU components.  
Crop acreage in a year is each used to calculate the total acreage and fraction for crop - only the crop acreage and total are checked.  
Warnings are generated for the follow conditions:  
1) Missing (undefined) required values.  
2) Invalid numerical values (e.g., negative acreage).

CU location identifier:  Required - specify the CU locations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
CheckCropPatternTS (ID= "\"*\")
```

CheckCropPatternTS

### CheckCropPatternTS() Command Editor

The command syntax is as follows:

```
CheckCropPatternTS (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The name of the crop(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the identifier is not matched</li><li>Ignore – ignore (don't generate a message) if the identifier is not matched</li><li>Warn – generate a warning message if the identifier is not matched</li></ul>	Warn

The following example command file illustrates how crop pattern time series can be defined, checked, and written to a StateCU file:

```
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
# Step 4 - Need to translate crops out of HB to include TR21 suffix
# Translate all crops from HB to include .TR21 suffix
TranslateCropPatternTS(ID="*",OldCropType="GRASS_PASTURE",NewCropType="GRASS_PASTURE.TR21")
TranslateCropPatternTS(ID="*",OldCropType="CORN_GRAIN",NewCropType="CORN_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SMALL_GRAINS",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="VEGETABLES",NewCropType="VEGETABLES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WO_COVER",NewCropType="ORCHARD_WO_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WITH_COVER",NewCropType="ORCHARD_WITH_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="DRY_BEANS",NewCropType="DRY_BEANS.TR21")
TranslateCropPatternTS(ID="*",OldCropType="GRAPES",NewCropType="GRAPES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="WHEAT",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SUNFLOWER",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SOD_FARM",NewCropType="GRASS_PASTURE.TR21")
# Step 5 - Translate crop names
# use high-altitude coefficients for structures with more than 50% of
# irrigated acreage above 6500 feet
TranslateCropPatternTS(ListFile="cm2005_HA.lst",IDCol=1,
    OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE.DWHA")
# Step 6 - Fill Acreage
#     Fill SW structure acreage backward from 1999 to 1950
#     Fill acreage forward for all structures from 2000 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1993,FillDirection=Backward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1993,FillEnd=1999,FillDirection=Forward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2000,FillEnd=2006,FillDirection=Forward)
# Step 7 - Write final *.cds file
WriteCropPatternTSToStateCU(OutputFile="..\StateCU\cm2006.cds",
    WriteCropArea=True,WriteHow=OverwriteFile)
# Check the results
CheckCropPatternTS(ID="*")
WriteCheckFile(OutputFile="cm2006.cds.StatedMI.check.html")
```



# Command Reference: CheckCULocations()

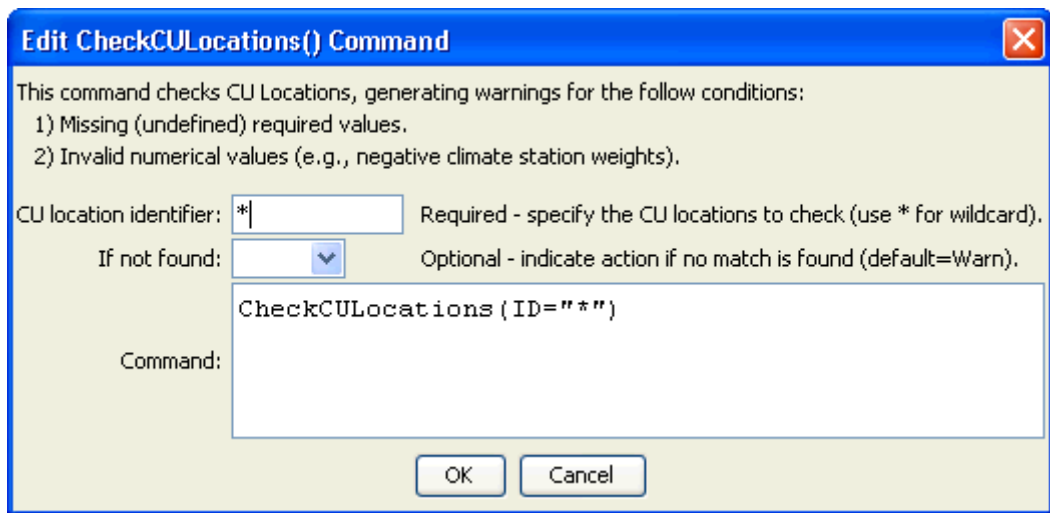
## Check CU location data for problems

### StateCU Command

Version 3.09.00, 2010-01-10

The `CheckCULocations()` command checks the CU Location data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.



**CheckCULocations() Command Editor**

CheckCULocations

The command syntax is as follows:

```
CheckCULocations(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the location identifier is not matched</li><li>Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following example command file illustrates how CU locations can be defined, sorted, checked, and written to a StateCU file (this is an abbreviated command file):

```
# Sp2008L_STR.StateDMI
# South Platte Decision Support System
# Historic Consumptive Use Model
# Structure File (*.str)
#
# Step 1 - Read Structure List File (WDID, Name)
#
# Structure List includes Key Structures from Task 3, Aggregate GW, and Aggregate SW
ReadCULocationsFromList(ListFile="Sp2008L_StructList.csv",IDCol=1,NameCol=3)
#
# Step 2 - Read structure information from HydroBase (Latitude, County, HUC)
FillCULocationsFromHydroBase(ID="*",CULocType=Structure,Region1Type=County,Region2Type=HUC)
#
# Step 3 - Assign AWC values based on Task 57, generate using the CDSS Toolbox
#
# # Key Structure AWC Values
SetCULocationsFromList(ListFile="AWC_2001.csv",IDCol=1,AWCCol=2)
#
# # GW AGG Structure AWC Values
SetCULocationsFromList(ListFile="AWC_Agg_GW.csv",IDCol=1,AWCCol=2)
#
# # SW AGG Structure AWC Values
SetCULocationsFromList(ListFile="AWC_Agg_SW.csv",IDCol=1,AWCCol=2)
#
# Step 4 - Assign Elevation
FillCULocationsFromList(ListFile="Key_Elev.csv",IDCol=1,ElevationCol=3)
#
# Step 5 - Set Demand Structure Information based on Demand Carrier
SetCULocation(ID="0100503_I",Latitude=40.38,Elevation=4533.00,Region1="WELD",
  Region2="10190003",AWC=0.1375,IfNotFound=Warn)
SetCULocation(I
#
SetCULocation(ID="6400526",AWC=0.1393,IfNotFound=Warn)
#
# Missing values assigned to Diversion Systems
SetCULocation(ID="0100503_D",Latitude=40.28567,Region1="MORGAN",IfNotFound=Warn)
# DivSys and Aggregate use weighted latitude from climate station assignments
# County and HUC information not assigned to DivSys or Aggregate Structures
#
# Step 6 - Read structure climate weights from list created from the CDSS Toolbox Climate Tool
SetCULocationClimateStationWeightsFromList(ListFile="Climate_2001.csv",IDCol=1,
  StationIDCol=2,TempWtCol=3,PrecWtCol=3)
SetCULocationClimateS
# Set Climate Stations above 6500
SetCULocationClimateStationWeightsFromList(ListFile="SP2008_DWHA_OroAdj.csv",IDCol=1,
  StationIDCol=2,TempWtCol=3,PrecWtCol=4,OrographicTempAdjCol=6,OrographicPrecAdjCol=5)
#
# Step 8 - Fill Key Climate Station
#
FillCULocationClimateStationWeights(ID="01*",IncludeOrographicTempAdj=False,
  IncludeOrographicPrecAdj=False,Weights="0945,1.0,1.0")
#
# Step 7 - Write Structure File
SortCULocations()
WriteCULocationsToStateCU(OutputFile="SP2008L.str")
# Check the results
CheckCULocations(ID="*")
WriteCheckFile(OutputFile="SP2008L.str.check.html")
```

# Command Reference: CheckDiversionDemandTSMonthly()

Check diversion demand time series (monthly) data for problems

**StateMod Command**  
Version 3.09.01, 2010-02-05

The `CheckDiversionDemandTSMonthly()` command checks diversion demand monthly time series for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckDiversionDemandTSMonthly() Command**

This command checks diversion demand time series (monthly), generating warnings for the follow conditions:

- 1) Missing values.
- 2) Invalid values (negatives).
- 3) Diversion station ID is not found (if diversion station list is available).

Diversion station identifier: \* Required - specify the diversion demand time series (monthly) to check (use \* for wildcard).

If not found: Warn Optional - indicate action if no match is found (default=Warn).

Command: `CheckDiversionDemandTSMonthly ( ID="*" )`

OK Cancel

CheckDiversionDemandTSMonthly

## CheckDiversionDemandTSMonthly() Command Editor

The command syntax is as follows:

`CheckDiversionDemandTSMonthly (Parameter=Value,...)`

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how diversion demand time series can be checked and written to a StateMod file:

```
#  
# Create diversion demand monthly time series file  
#  
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\STATEMOD\rg2007C.ddm")  
#  
# Check the results  
CheckDiversionDemandTSMonthly(ID="*")  
WriteCheckFile(OutputFile="Cddm.commands.StateDMI.check.html")
```

---

# Command Reference: CheckDiversiOnHistoricalTSMonthly()

**Check diversion historical time series (monthly) data for problems**

**StateMod Command**  
Version 3.09.01, 2010-02-05

The `CheckDiversiOnHistoricalTSMonthly()` command checks diversion historical monthly time series for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit `CheckDiversiOnHistoricalTSMonthly()` Command**

This command checks diversion historical time series (monthly), generating warnings for the follow conditions:

- 1) Missing values.
- 2) Invalid values (negatives).
- 3) Diversion station ID is not found (if diversion station list is available).

Diversion station identifier:  Required - specify the diversion historical time series (monthly) to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CheckDiversiOnHistoricalTSMonthly(ID="*")
```

OK Cancel

CheckDiversiOnHistoricalTSMonthly

**CheckDiversiOnHistoricalTSMonthly() Command Editor**

The command syntax is as follows:

`CheckDiversiOnHistoricalTSMonthly(Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how diversion historical time series can be checked and written to a StateMod file:

```
#  
# Create diversion historical monthly time series file  
#  
WriteDiversionsHistoricalTSMonthlyToStateMod(OutputFile="..\STATEMOD\rg2007.ddh")  
#  
# Check the results  
CheckDiversionsHistoricalTSMonthly(ID="*")  
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")
```

# Command Reference: CheckDiversiionRights()

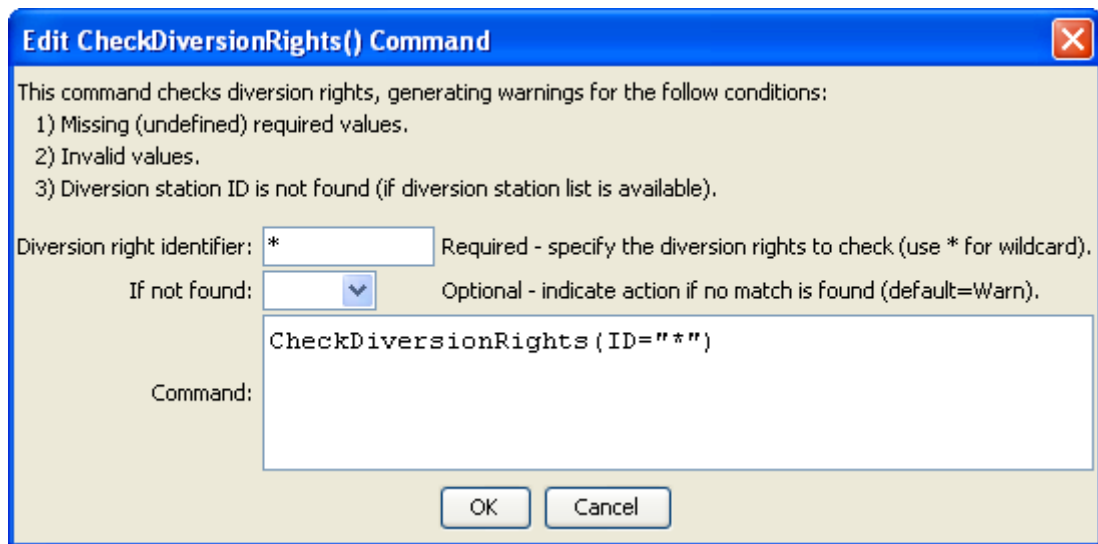
Check diversion rights data for problems

StateCU and StateMod Command

Version 3.09.00, 2010-01-24

The CheckDiversiionRights() command checks diversion rights data for problems. The command should usually be used with a WriteCheckFile() command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.



CheckDiversiionRights() Command Editor

CheckDiversiionRights

The command syntax is as follows:

CheckDiversiionRights( Parameter=Value,...)

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the location identifier is not matched</li><li>Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how diversion rights can be checked and written to a StateMod file:

```
#
# Create direct diversion rights file
#
WriteDiversionRightsToStateMod(OutputFile="..\STATEMOD\cm2005.ddy")
#
# Check the results
CheckDiversionRights(ID="*")
WriteCheckFile(OutputFile="ddy.commands.StateDMI.check.html")
```



# Command Reference: CheckDiversionsStations()

## Check diversion stations data for problems

### StateMod Command

Version 3.09.01, 2010-02-01

The `CheckDiversionsStations()` command checks diversion stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckDiversionsStations() Command**

This command checks diversion stations, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) River network ID is not found in the network (if network node list is available).
- 4) Daily ID is not a diversion (if diversion station list is available).

Diversion station identifier:  Required - specify the diversion stations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

CheckDiversionsStations

### CheckDiversionsStations() Command Editor

The command syntax is as follows:

```
CheckDiversionsStations (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how diversion stations can be checked and written to a StateMod file:

```
#  
# Create direct diversion stations file  
#  
WriteDiversionStationsToStateMod(OutputFile="..\STATEMOD\cm2005.dds")  
#  
# Check the results  
CheckDiversionStations (ID="*")  
WriteCheckFile(OutputFile="dds.commands.StateDMI.check.html")
```

---

# Command Reference: CheckInstreamFlowDemandTSAverageMonthly()

**Check instream flow demand time series (average monthly) data for problems**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CheckInstreamFlowDemandTSAverageMonthly()` command checks instream flow demand time series (average monthly) data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit `CheckInstreamFlowDemandTSAverageMonthly()` Command**

This command checks instream flow demand time series (average monthly), generating warnings for the follow conditions:

- 1) Missing values.
- 2) Invalid values (negatives).
- 3) Instream flow station ID is not found (if instream flow station list is available).

Diversion station identifier:  Required - specify the diversion historical time series (monthly) to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

CheckInstreamFlowDemandTSAverageMonthly

## **CheckInstreamFlowDemandTSAverageMonthly() Command Editor**

The command syntax is as follows:

`CheckInstreamFlowDemandTSAverageMonthly (Parameter=Value,...)`

### **Command Parameters**

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how instream flow demand time series can be checked and written to a StateMod file:

```
#  
# Create instream demand time series file  
#  
WriteInstreamFlowDemandTSAverageMonthlyToStateMod(OutputFile="..\STATEMOD\cm2005.ifa")  
#  
# Check the results  
CheckInstreamFlowDemandTSAverageMonthly(ID="*")  
WriteCheckFile(OutputFile="ifa.commands.StateDMI.check.html")
```

---

# Command Reference: CheckInstreamFlowRights()

Check instream flow rights data for problems

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `CheckInstreamFlowRights()` command checks instream flow rights data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckInstreamFlowRights() Command**

This command checks instream flow rights, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) Instream flow station ID is not found (if instream flow station list is available).

Instream flow right identifier:  Required - specify the instream flow rights to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `CheckInstreamFlowRights ( ID="*" )`

CheckInstreamFlowRights

**CheckInstreamFlowRights() Command Editor**

The command syntax is as follows:

```
CheckInstreamFlowRights (Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how instream flow rights can be checked and written to a StateMod file:

```
#  
# Create instream flow rights file  
#  
WriteInstreamFlowRightsToStateMod(OutputFile="..\STATEMOD\cm2005.ifr")  
#  
# Check the results  
CheckInstreamFlowRights(ID="*")  
WriteCheckFile(OutputFile="ifr.commands.StateDMI.check.html")
```

---

# Command Reference: CheckInstreamFlowStations()

Check instream flow stations data for problems

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `CheckInstreamFlowStations()` command checks instream flow stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckInstreamFlowStations() Command**

This command checks instream flow stations, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) River network ID is not found in the network (if network node list is available).
- 4) Downstream river network ID is not found in the network (if network node list is available).
- 5) Daily ID is not an instream flow station (if station list is available).

Instream flow station identifier:  Required - specify the instream flow stations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`CheckInstreamFlowStations ( ID = " * " )`

OK Cancel

**CheckInstreamFlowStations() Command Editor**

CheckInstreamFlowStations

The command syntax is as follows:

```
CheckInstreamFlowStations( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the location identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the location identifier is not matched</li> <li>Warn – generate a warning message if the location identifier is not matched</li> </ul>	Warn

The following excerpt from a command file illustrates how instream flow stations can be checked and written to a StateMod file:

```
#
# Create instream flow stations file
#
WriteInstreamFlowStationsToStateMod( outputFile="..\STATEMOD\cm2005.ifs" )
#
# Check the results
CheckInstreamFlowStations ( ID="*" )
WriteCheckFile( outputFile="ifs.commands.StateDMI.check.html" )
```



---

# Command Reference: CheckIrrigationPracticeTS()

Check irrigation practice time series data for problems

**StateCU Command**  
Version 3.08.02, 2010-01-05

The `CheckIrrigationPracticeTS()` command checks the irrigation practice time series data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckIrrigationPracticeTS() Command**

This command checks StateCU irrigation practice time series at CU locations.  
The total acreage is cross-checked with the crop pattern time series total if available.  
Warnings are generated for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid numerical values (e.g., negative acreage).
- 3) Acreage parts do not add to the total.
- 4) Irrigation practice total acreage and crop pattern total acreage do not agree.

CU location identifier: \* Required - specify the CU locations to check (use \* for wildcard).

If not found: [v] Optional - indicate action if no match is found (default=Warn).

Command: CheckIrrigationPracticeTS (ID="\*")

OK Cancel

CheckIrrigationPracticeTS

**CheckIrrigationPracticeTS() Command Editor**

The command syntax is as follows:

```
CheckIrrigationPracticeTS(Parameter=Value,...)
```

**Command Parameters**

Parameter	Description	Default
ID	The location identifiers for the time series to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

The following excerpt from a command file illustrates how irrigation practice time series can be checked and written to a StateCU file:

```
#
# Create irrigation practice file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="cm2006.ipy")
#
# Check the results
CheckIrrigationPracticeTS(ID="*")
WriteCheckFile(OutputFile="ipy.commands.StateDMI.check.html")
```

---

# Command Reference: CheckPenmanMonteith()

## Check Penman-Monteith data for problems

### StateCU Command

Version 3.10.00, 2010-04-02

The `CheckPenmanMonteith()` command checks the Penman-Monteith crop coefficient data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckPenmanMonteith() Command**

This command checks StateCU Penman-Monteith crop coefficients.  
Currently no cross-checks are done with other StateCU components.  
Warnings are generated for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid numerical values (e.g., percent > 100).

Crop type (name):  Required - specify the crops to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
CheckPenmanMonteith ( ID = "*" )
```

OK Cancel

CheckPenmanMonteith

### CheckPenmanMonteith() Command Editor

The command syntax is as follows:

```
CheckPenmanMonteith(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The name of the crop(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

The following example command file illustrates how Penman-Monteith coefficients can be defined, checked, and written to a StateCU file:

```
StartLog(LogFile="Crops_KPM.StateDMI.log")
#
# StateDMI commands to create the Penman-Monteith crop coefficients file
#
# Step 1 - read data from HydroBase
#
# Read the general ASCE standardized coefficients
ReadPenmanMonteithFromHydroBase(PenmanMonteithMethod="PENMAN-MONTEITH_ALFALFA")
#
# Step 3 - write the file
#
SortPenmanMonteith()
WritePenmanMonteithToStateCU(OutputFile="rg2007.kpm")
#
# Check the results
#
CheckPenmanMonteith(ID="*")
WriteCheckFile(OutputFile="Crops_KPM.StateDMI.check.html")
```

# Command Reference: CheckReservoirRights()

## Check reservoir rights data for problems

### StateMod Command

Version 3.09.01, 2010-02-01

The `CheckReservoirRights()` command checks reservoir rights data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckReservoirRights() Command**

This command checks reservoir rights, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) Reservoir station ID is not found (if reservoir station list is available).

Reservoir right identifier:  Required - specify the reservoir rights to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CheckReservoirRights ( ID="*" )
```

CheckReservoirRights

### CheckReservoirRights() Command Editor

The command syntax is as follows:

```
CheckReservoirRights (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how reservoir rights can be checked and written to a StateMod file:

```
#  
# Create reservoir rights file  
#  
WriteReservoirRightsToStateMod(OutputFile="..\STATEMOD\cm2005.rer")  
#  
# Check the results  
CheckReservoirRights(ID="*")  
WriteCheckFile(OutputFile="ddr.commands.StateDMI.check.html")
```

# Command Reference: CheckReservoirStations()

Check reservoir stations data for problems

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CheckReservoirStations()` command checks reservoir stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckReservoirStations() Command**

This command checks reservoir stations, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) River network ID is not found in the network (if network node list is available).
- 4) Daily ID is not a reservoir (if reservoir station list is available).

Reservoir station identifier:  Required - specify the reservoir stations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

CheckReservoirStations

**CheckReservoirStations() Command Editor**

The command syntax is as follows:

```
CheckReservoirStations ( Parameter=Value , ... )
```

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how reservoir stations can be checked and written to a StateMod file:

```
#  
# Create reservoir stations file  
#  
WriteReservoirStationsToStateMod(OutputFile="..\STATEMOD\cm2005.res")  
#  
# Check the results  
CheckReservoirStations (ID="*")  
WriteCheckFile(OutputFile="res.commands.StateDMI.check.html")
```



# Command Reference: CheckRiverNetwork()

## Check river network data for problems

### StateMod Command

Version 3.09.01, 2010-02-03

The `CheckRiverNetwork()` command checks river network data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckRiverNetwork() Command**

This command checks river network stations (nodes), generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) Downstream river network ID is not found in the network.

River network node identifier:  Required - specify the river network nodes to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CheckRiverNetwork ( ID = "*" )
```

OK Cancel

CheckRiverNetwork

**CheckRiverNetwork() Command Editor**

The command syntax is as follows:

```
CheckRiverNetwork ( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following command file illustrates how a StateMod river network file can be created from the generalized network file:

```
StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#           names from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",
    CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")
```

---

# Command Reference: CheckStreamEstimateCoefficients()

Check stream estimate coefficients data for problems

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CheckStreamEstimateCoefficients()` command checks stream estimate coefficients data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckStreamEstimateCoefficients() Command**

This command checks stream estimate coefficients, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) Invalid values.
- 3) River node ID is not found in the network (if network node list is available).

Stream estimate station identifier: \* Required - specify the stream estimate coefficients to check (use \* for wildcard).

If not found: Warn Optional - indicate action if no match is found (default=Warn).

Command: CheckStreamEstimateCoefficients (ID="\*")

OK Cancel

CheckStreamEstimateCoefficients

**CheckStreamEstimateCoefficients() Command Editor**

The command syntax is as follows:

`CheckStreamEstimateCoefficients (Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following command file illustrates how a StateMod stream estimate coefficients file can be created:

```

StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#           this baseflow nodes are generally on smaller non-gaged tribs and have
#           different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="360801",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="362002",GageID="09054000")
SetStreamEstimateCoefficientsPFGage(ID="360829",GageID="09047500")
..similar commands omitted...
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374641",ProrationFactor=0.200,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374648",ProrationFactor=0.350,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="380880",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="381594",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="384617",ProrationFactor=0.700,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510639",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514603",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514620",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510728",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530555",ProrationFactor=0.180,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530678",ProrationFactor=0.230,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="531082",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="954683",ProrationFactor=0.400,IfNotFound=Warn)
#
# Step 5 - create streamflow estimate coefficient file
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")

```

# Command Reference: CheckStreamEstimateStations()

Check stream estimate stations data for problems

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CheckStreamEstimateStations()` command checks stream estimate stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckStreamEstimateStations() Command**

This command checks stream estimate stations, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) River network ID is not found in the network (if network node list is available).
- 3) Daily ID is not a stream gage station (if the stream gage list is available).

Stream estimate station identifier: \* Required - specify the stream estimate stations to check (use \* for wildcard).

If not found: Warn Optional - indicate action if no match is found (default=Warn).

Command: CheckStreamEstimateStations ( ID="\*" )

OK Cancel

CheckStreamEstimateStations

**CheckStreamEstimateStations() Command Editor**

The command syntax is as follows:

`CheckStreamEstimateStations (Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how stream estimate stations can be checked and written to a StateMod file:

```
#  
# Create stream gage stations file  
#  
WriteStreamEstimateStationsToStateMod(OutputFile="..\STATEMOD\cm2005.ses")  
#  
# Check the results  
CheckStreamEstimateStations (ID="*")  
WriteCheckFile(OutputFile="ses.commands.StateDMI.check.html")
```

# Command Reference: CheckStreamGageStations()

Check stream gage stations data for problems

**StateMod Command**

Version 3.09.01, 2010-02-01

The `CheckStreamGageStations()` command checks stream gage stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckStreamGageStations() Command**

This command checks stream gage stations, generating warnings for the follow conditions:

- 1) Missing (undefined) required values.
- 2) River network ID is not found in the network (if network node list is available).
- 3) Daily ID is not a stream gage.

Stream gage station identifier:  Required - specify the stream gage stations to check (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `CheckStreamGageStations ( ID = "*" )`

CheckStreamGageStations

**CheckStreamGageStations() Command Editor**

The command syntax is as follows:

```
CheckStreamGageStations (Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how stream gage stations can be checked and written to a StateMod file:

```
#  
# Create stream gage stations file  
#  
WriteStreamGageStationsToStateMod(OutputFile="..\STATEMOD\cm2005.ris")  
#  
# Check the results  
CheckStreamGageStations (ID="*")  
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")
```



# Command Reference: CheckWellDemandTSMonthly()

Check well demand time series (monthly) data for problems

**StateMod Command**  
Version 3.09.00, 2010-01-24

The `CheckWellDemandTSMonthly()` command checks well demand monthly time series for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckWellDemandTSMonthly() Command**

This command checks well demand time series (monthly), generating warnings for the follow conditions:

- 1) Missing values.
- 2) Invalid values (negatives).
- 3) Well station ID is not found (if well station list is available).

Well station identifier: \* Required - specify the well demand time series (monthly) to check (use \* for wildcard).

If not found: [dropdown] Optional - indicate action if no match is found (default=Warn).

Command: CheckWellDemandTSMonthly ( ID = "\*" )

OK Cancel

**CheckWellDemandTSMonthly() Command Editor**

CheckWellDemandTSMonthly

The command syntax is as follows:

`CheckWellDemandTSMonthly( Parameter=Value, ... )`

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how well demand time series can be checked and written to a StateMod file:

```
#  
# Create well demand file  
#  
WriteWellDemandTSMonthlyToStateMod(OutputFile="..\STATEMOD\rg2007.wem")  
#  
# Check the results  
CheckWellDemandTSMonthly(ID="*")  
WriteCheckFile(OutputFile="wem.commands.StateDMI.check.html")
```

# Command Reference: CheckWellHistoricalPumpingTSMonthly()

Check well historical pumping (monthly) data for problems

StateCU and StateMod Command

Version 3.09.00, 2010-01-24

The CheckWellHistoricalPumpingTSMonthly( ) command checks well historical pumping monthly time series for problems. The command should usually be used with a WriteCheckFile( ) command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CheckWellHistoricalPumpingTSMonthly() Command**

This command checks well historical pumping time series (monthly), generating warnings for the follow conditions:

- 1) Missing values.
- 2) Invalid values (negatives).
- 3) Well station ID is not found (if well station list is available).

Well station identifier: \* Required - specify the well historical pumping time series (monthly) to check (use \* for wildcard).

If not found: Warn Optional - indicate action if no match is found (default=Warn).

Command: CheckWellHistoricalPumpingTSMonthly ( ID="\*" )

OK Cancel

CheckWellHistoricalPumpingTSMonthly

## CheckWellHistoricalPumpingTSMonthly() Command Editor

The command syntax is as follows:

CheckWellHistoricalPumpingTSMonthly(Parameter=Value,...)

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the location identifier is not matched</li><li>• Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>• Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how well historical pumping time series can be checked and written to a StateMod file:

```
#  
# Create well pumping file  
#  
WriteWellHistoricalPumpingTSMonthlyToStateMod(OutputFile="..\STATEMOD\rg2007.weh")  
#  
# Check the results  
CheckWellHistoricalPumpingTSMonthly(ID="*")  
WriteCheckFile(OutputFile="weh.commands.StateDMI.check.html")
```

# Command Reference: CheckWellRights()

Check well rights data for problems

StateCU and StateMod Command

Version 3.09.00, 2010-01-24

The CheckWellRights() command checks well rights data for problems. The command should usually be used with a WriteCheckFile() command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.



CheckWellRights() Command Editor

CheckWellRights

The command syntax is as follows:

```
CheckWellRights(Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the location identifier is not matched</li><li>Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following example command file illustrates how well rights can be defined, sorted, checked, and written to a StateMod file:

```
# Well Rights File (*.wer)
#
StartLog(LogFile="Sp2008L_WER.log")
#
# Step 1 - Read all structures
#
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
SortWellStations()
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=1,IfNotFound=Ignore)
#
# Step 5 - Read rights from HydroBase
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1956,1976,1987,2001,2005",
    Div="1",DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
    ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds & ipy acreage filling
# Write Data Comments="False" provides merged file used for seting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Sp2008L.wer.check.html",Title="Well Rights Check File")
```

# Command Reference: CheckWellStations()

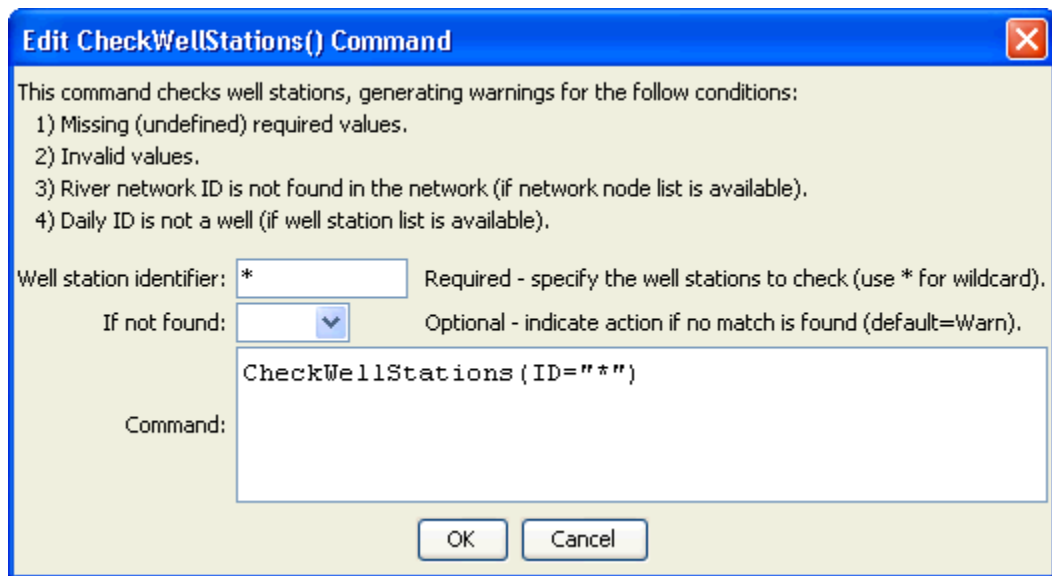
## Check well stations data for problems

### StateMod Command

Version 3.09.01, 2010-02-01

The `CheckWellStations()` command checks well stations data for problems. The command should usually be used with a `WriteCheckFile()` command at the end of a command file.

The following dialog is used to edit the command and illustrates the syntax of the command.



CheckWellStations

**CheckWellStations() Command Editor**

The command syntax is as follows:

```
CheckWellStations (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The identifier for the location(s) to check. Use * to match a pattern.	None – must be specified.
IfNotFound	One of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the location identifier is not matched</li><li>Ignore – ignore (don't generate a message) if the location identifier is not matched</li><li>Warn – generate a warning message if the location identifier is not matched</li></ul>	Warn

The following excerpt from a command file illustrates how diversion stations can be checked and written to a StateMod file:

```
#  
# Create well stations file  
#  
WriteWellStationsToStateMod(OutputFile="..\STATEMOD\rg2007.wes")  
#  
# Check the results  
CheckWellStations (ID="*")  
WriteCheckFile(OutputFile="wes.commands.StateDMI.check.html")
```



---

# Command Reference: CompareFiles()

## Compare text files to determine whether they are different

### General Command

Version 03.08.02, 2010-01-06

The `CompareFiles()` command compares text files to determine data differences. For example, the command can be used to compare old and new files produced by a software process.

Each line in the file is compared. By default, lines beginning with # are treated as comment lines and are ignored (see `CommentLineChar` to specify the comment indicator). Therefore, only non-comment lines are compared. Differences and simple statistics are printed to the log file. A warning can be generated if a difference is detected or if no differences are detected.

The following dialog is used to edit the command and illustrates the syntax for the command.

**Edit CompareFiles() command**

This command compares text files. Comment lines starting with # are ignored.  
A line by line comparison is made.  
It is recommended that file names be relative to the working directory, which is:  
C:\Develop\TSTool\_SourceBuild\TSTool\test\regression\UserManualExamples\TestCases\CommandReference\CompareFiles

First file to compare:

Second file to compare:

Comment line character:  Optional (default=#)

Warn if different?:  Generate a warning if different? (default=false)

Warn if same?:  Generate a warning if same? (default=false)

Command:  

```
CompareFiles(InputFile1="Data/A1.txt",InputFile2="Data/B1.txt",WarnIfDifferent=True)
```

CompareFiles

**CompareFiles() Command Editor**

The command syntax is as follows:

```
CompareFiles( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
InputFile1	The name of the first file to read. Enclose the name in double quotes to protect whitespace and special characters.	None – the file name is required.
InputFile2	The name of the second file to read. Enclose the name in double quotes to protect whitespace and special characters.	None – the file name is required.
CommentLineChar	The character(s) that if found at the start of a line indicate comment lines. Comment lines are ignored in the comparison because they typically may include information such as date/time that changes even if the remainder of the file contents are the same.	#
WarnIfDifferent	If <b>True</b> and at least one difference is detected, a warning will be generated by the command, which will result in software like TSTool displaying a warning. If <b>False</b> , only status messages are written to the log file. The warning is useful if it is critical to detect any difference in the files.	Do not generate a warning if the files are different. Differences are printed to the log file.
WarnIfSame	If <b>True</b> and no differences are detected, a warning will be generated by the command, which will result in software like TSTool displaying a warning. If <b>False</b> , only status messages are written to the log file. The warning is useful if it is critical to detect files that are the same.	Do not generate a warning if the files are the same.

The following example illustrates how two files can be compared. For example, use similar commands to compare results from two model runs, two database queries, or when testing software:

```
CompareFiles( InputFile1="Data/A1.txt" , InputFile2="Data/B1.txt" ,
  WarnIfDifferent=True )
```

# Command Reference: CreateCropPatternTSForCULocations()

Create empty crop pattern time series for each CU Location

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `CreateCropPatternTSForCULocations()` command creates empty crop pattern time series for each CU Location. This is necessary to ensure that the crop pattern time series are in the same order as the CU locations and that lists of crop pattern time series are initialized for each location. The following dialog is used to edit the command and illustrates the syntax of the command.

**CreateCropPatternTSForCULocations() Command Editor**

The command syntax is as follows:

```
CreateCropPatternTSForCULocations ( Parameter=Value , ... )
```

## Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified as *
Units	The units for crop area time series.	ACRE
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>Fail – generate a failure message if the ID is not matched</li><li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>Warn – generate a warning message if the ID is not matched</li></ul>	Warn

The following command file illustrates how to create a crop pattern time series file:

```
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
# Step 4 - Need to translate crops out of HB to include TR21 suffix
# Translate all crops from HB to include .TR21 suffix
TranslateCropPatternTS(ID="*",OldCropType="GRASS_PASTURE",NewCropType="GRASS_PASTURE.TR21")
TranslateCropPatternTS(ID="*",OldCropType="CORN_GRAIN",NewCropType="CORN_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SMALL_GRAINS",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="VEGETABLES",NewCropType="VEGETABLES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WO_COVER",NewCropType="ORCHARD_WO_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WITH_COVER",NewCropType="ORCHARD_WITH_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="DRY_BEANS",NewCropType="DRY_BEANS.TR21")
TranslateCropPatternTS(ID="*",OldCropType="GRAPES",NewCropType="GRAPES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="WHEAT",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SUNFLOWER",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SOD_FARM",NewCropType="GRASS_PASTURE.TR21")
# Step 5 - Translate crop names
# use high-altitude coefficients for structures with more than 50% of
# irrigated acreage above 6500 feet
TranslateCropPatternTS(ListFile="cm2005_HA.lst",IDCol=1,
    OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE.DWHA")
# Step 6 - Fill Acreage
#     Fill SW structure acreage backward from 1999 to 1950
#     Fill acreage forward for all structures from 2000 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1993,FillDirection=Backward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1993,FillEnd=1999,FillDirection=Forward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2000,FillEnd=2006,FillDirection=Forward)
# Step 7 - Write final *.cds file
WriteCropPatternTSToStateCU(OutputFile="..\StateCU\cm2006.cds",
    WriteCropArea=True,WriteHow=OverwriteFile)
# Check the results
CheckCropPatternTS(ID="*")
WriteCheckFile(OutputFile="cm2006.cds.StateDMI.check.html")
```

---

# Command Reference: CreateIrrigationPracticeTSForCULocations()

Create empty irrigation practice time series for each CU Location

**StateCU Command**

Version 3.09.01, 2010-02-01

The `CreateIrrigationPracticeTSForCULocations()` command creates empty irrigation practice time series for each CU Location. This ensures that the irrigation practice time series will be in the same order as the CU locations initializes the time series with missing data values that can be filled with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit CreateIrrigationPracticeTSForCULocations() Command**

This command creates irrigation practice time series for each CU Location that is defined. Time series are initialized to default values (missing values). This command should be used after CU Locations and their aggregates/systems are defined. After this command, use other commands to read and fill the irrigation practice time series data.

CU Location ID:  Required - locations for data (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
CreateIrrigationPracticeTSForCULocations ( ID="*" )
```

CreateIrrigationPracticeTSForCULocations

**CreateIrrigationPracticeTSForCULocations() Command Editor**

The command syntax is as follows:

```
CreateIrrigationPracticeTSForCULocations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified as *
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

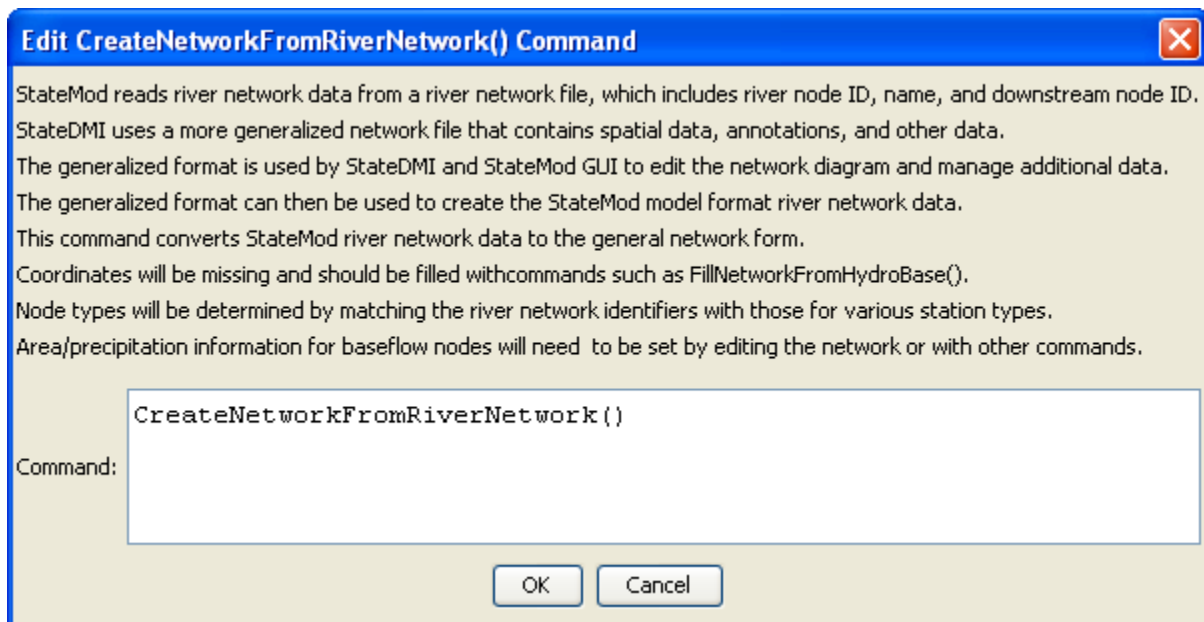
---

# Command Reference: CreateNetworkFromRiverNetwork()

Create a generalized network from a StateMod river network

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `CreateNetworkFromRiverNetwork()` command creates a generalized network from a StateMod river network. This is used, for example, when a StateMod data set has been developed without StateDMI and a generalized network file is now needed for full StateDMI use. The generalized network file can be edited using the interactive model schematic editor. The following dialog is used to edit the command and illustrates the syntax of the command.



**CreateNetworkFromRiverNetwork() Command Editor**

CreateNetworkFromRiverNetwork

The command syntax is as follows:

```
CreateNetworkFromRiverNetwork()
```

### Command Parameters

Parameter	Description	Default
	There are no parameters for this command.	

The following example command file illustrates how the command might be used:

```
# Create a generalized XML network from individual StateMod files
# Read the network, which contains upstream to downstream connectivity but does
# not indicate node types
ReadRiverNetworkFromStateMod(InputFile=cm2005.rin)
# Read the stations, which imply the node types
ReadRiverStreamGageStationsFromStateMod(InputFile=cm2005.ris)
ReadRiverDiversionStationsFromStateMod(InputFile=cm2005.dds)
ReadRiverReservoirStationsFromStateMod(InputFile=cm2005.res)
ReadRiverInstreamFlowStationsFromStateMod(InputFile=cm2005.ifs)
ReadRiverWellStationsFromStateMod(InputFile=cm2005.wes)
# To be developed...
#ReadRiverPlanStationsFromStateMod()
ReadRiverStreamEstimateStationsFromStateMod(InputFile=cm2005.ris)
# Now create the generalized network, using the connectivity and node types
CreateNetworkFromRiverNetwork()
# Fill in node names and locations from HydroBase, if any is still missing
FillNetworkFromHydroBase()
# Write the generalized network
WriteNetworkToStateMod(OutputFile="cm2005.net")
# Check for errors (the following is not yet implemented)
#CheckNetwork()
WriteCheckFile(OutputFile="cm2005.net.check.html")
```



---

# Command Reference:

## CreateRegressionTestCommandFile()

Create a command file to run software regression tests

General Command  
Version 3.08.02, 2010-01-06

The `CreateRegressionTestCommandFile()` command is used for software testing (or certification of processes used in operations) and creates a command file that includes a `StartRegressionTestResultsReport()` and multiple `RunCommands()` commands. A starting search folder is provided and all files that match the given pattern (by convention `Test_*.StateDMI`) are assumed to be command files that can be run to test the software. The resulting command file is a test suite comprised of all the individual tests and can be used to verify software before release. The goal is to have all tests pass before software release.

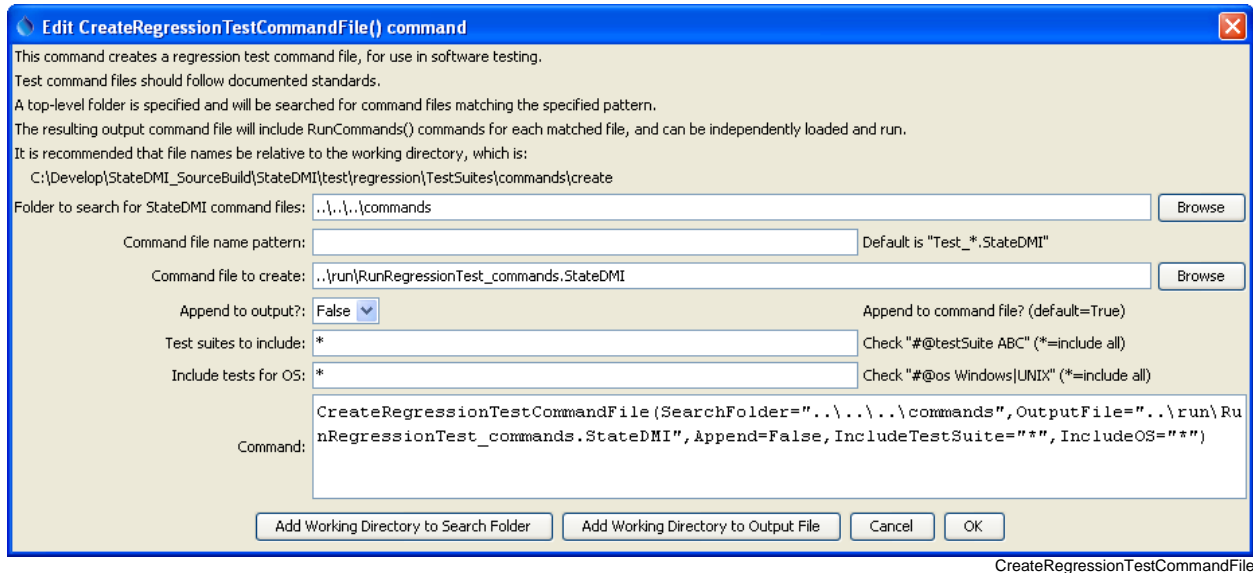
The following table lists tags that can be placed in # comments in command files to provide information for testing, for example:

```
#@expectedStatus Failure
```

Command # Comment Tags

Parameter	Description
@expectedStatus Failure @expectedStatus Warning	The <code>RunCommands()</code> command <code>ExpectedStatus</code> parameter is by default <code>Success</code> . However, a different status can be specified if it is expected that a command file will result in <code>Warning</code> or <code>Failure</code> and still be a successful test. For example, if a command is obsolete and should generate a failure, the expected status can be specified as <code>Failure</code> and the test will pass. Another example is to test that the software properly treats a missing file as a failure.
@os Windows @os UNIX	The test is designed to work only on the specified platform and will be included in the test suite only if the <code>IncludeOS</code> parameter includes the corresponding operating system (OS) type. This is primarily used to test specific features of the OS and similar but separate test cases should be implemented for both OS types. If the OS type is not specified as a tag in a command file, the test is always included (see also the handling of included test suites).
@testSuite ABC	Indicate that the command file should be considered part of the specified test suite, as specified with the <code>IncludeTestSuite</code> parameter. The test is included in all test collections if the tag is not specified; therefore, for general tests, do not specify a test suite. This tag is useful if a group of tests require special setup, for example connecting to a database. The suite names should be decided upon by the test developer.

The following dialog is used to edit the command and illustrates the syntax for the command.



### CreateRegressionTestCommandFile() Command Editor

The command syntax is as follows:

```
CreateRegressionTestCommandFile (Parameter=Value, ...)
```

### Command Parameters

Parameter	Description	Default
SearchFolder	The folder to search for regression test command files. All subfolders will also be searched.	None – must be specified.
OutputFile	The name of the command file to create, enclosed in double quotes if the file contains spaces or other special characters. A path relative to the command file containing this command can be specified.	None – must be specified.
SetupCommandFile	The name of a StateDMI command file that supplies setup commands, and which will be prepended to output. Use such a file to open database connections and set other global settings that apply to the entire test run.	Do not include setup commands.
FilenamePattern	Pattern for StateDMI command files, using wildcards.	Test_*.StateDMI
Append	Indicate whether to append to the output file (True) or overwrite (False). This allows multiple directory trees to be searched for tests, where the first command typically specifies False and additional commands specify True.	True
IncludeTestSuite	If *, all tests that match FilenamePattern and IncludeOS are included. If a test suite is specified, only include tests that have @testSuite tag values that match a value in IncludeTestSuite. One or more tags can be specified, separated by commas.	* – include all test cases.
IncludeOS	If *, all tests that match FilenamePattern and	* – include all test

Parameter	Description	Default
	IncludeTestSuite are included. If an OS is specified, only include tests that have @os tag values that match a value in IncludeTestSuite. This tag is typically specified once or not at all.	cases.

See the RunCommands ( ) documentation for how to set up a regression test. The following command file illustrates how to create a regression test suite.

```
CreateRegressionTestCommandFile(SearchFolder="..\..\..\commands\general",
    OutputFile="..\run\RunRegressionTest_commands_general.StateDMI",
    Append=False)
```

An example of the output file from running the tests is:

```
# File generated by...
# program:      StateDMI 3.08.02 (2009-09-29)
# user:         sam
# date:         Wed Sep 30 13:26:41 MDT 2009
# host:         SOPRIS
# directory:    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\TestSuites\commands\run
# command line: StateDMI
#
# The test status below may be PASS or FAIL.
# A test can pass even if the command file actual status is FAILURE, if failure is expected.
#   Test   Commands   Commands
#   Pass/  Expected   Actual
#   Num Fail Status   Status   Command File
#-----
1 PASS    SUCCESS    SUCCESS
C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\AggregateWellRights\Test_AggregateWellRights_rg2007part.StateDMI
2 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateDiversionDemandTSMonthly\
Test_CalculateDiversionDemandTSMonthly.StateDMI
3 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateDiversionDemandTSMonthlyAsMax\
Test_CalculateDiversionDemandTSMonthlyAsMax.StateDMI
4 PASS    SUCCESS    SUCCESS    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateDiversionStationEfficiencies\
Test_CalculateDiversionStationEfficiencies.StateDMI
5 PASS    warning    WARNING    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateStreamEstimateCoefficients\
Test_CalculateStreamEstimateCoefficients_cm2005.StateDMI
6 PASS    warning    WARNING    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateStreamEstimateCoefficients\
Test_CalculateStreamEstimateCoefficients_gm2004.StateDMI
7 PASS    warning    WARNING    C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\commands\CalculateStreamEstimateCoefficients\
Test_CalculateStreamEstimateCoefficients_rg2007.StateDMI
```

This page is intentionally blank.

---

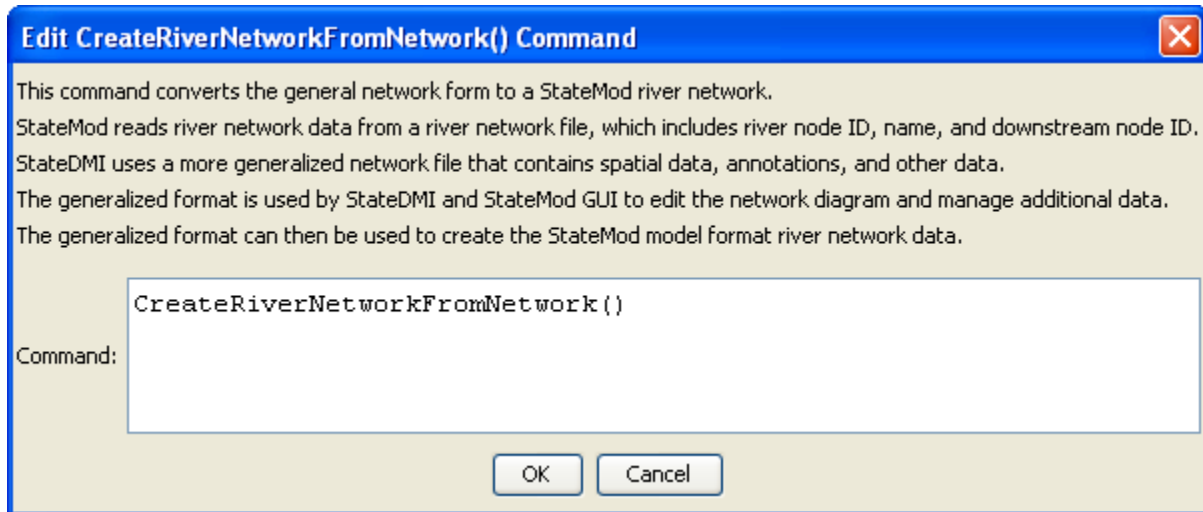
# Command Reference: CreateRiverNetworkFromNetwork()

Create a StateMod river network from a generalized network

## StateMod Command

Version 3.09.01, 2010-02-03

The `CreateRiverNetworkFromNetwork()` command creates a StateMod river network from a generalized network. This is used, for example, when a change has been made to the generalized network (e.g., by editing interactively in StateDMI) and a consistent StateMod river network file must be created.



CreateRiverNetworkFromRiverNetwork

## CreateRiverNetworkFromRiverNetwork() Command Editor

The command syntax is as follows:

```
CreateRiverNetworkFromRiverNetwork()
```

### Command Parameters

Parameter	Description	Default
	There are no parameters for this command.	

The following command file illustrates how a StateMod river network file can be created from the generalized network file:

```
StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#          names from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",
    CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")
```

---

# Command Reference: Exit()

## Stop processing commands

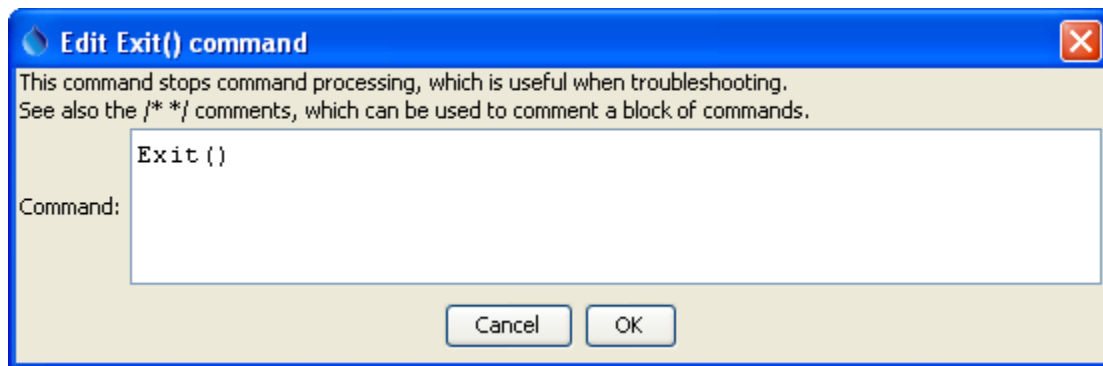
### General Command

Version 3.08.02, 2010-01-07

The `Exit()` command can be inserted anywhere in a command file and causes the processing of commands to stop at that line. This is useful for temporarily processing a subset of a long list of commands. Multi-line comments (`/* */`) can also be used to temporarily disable one or more commands. It may also useful to add an `Exit()` command at the end of the file so that it is easy to insert commands above this command when the end line is selected (rather than having to deselect all commands when editing).

In the future the command may be enhanced to have parameters that more explicitly control processing shut-down.

The following dialog is used to edit the command and illustrates the command syntax:



Exit

**Exit() Command Editor**

The command syntax is as follows:

```
Exit(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
	There are currently no command parameters.	

A sample command file is as follows:

```
Exit()
```

This page is intentionally blank.



# Command Reference: FillClimateStation()

## Fill climate station data

### StateCU Command

Version 3.08.02, 2010-01-05

The `FillClimateStation()` command fills missing data in existing climate stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillClimateStation() Command**

This command fills missing data in climate station(s), using the climate station ID to look up the station. The climate station ID can contain a \* wildcard pattern to match one or more stations. Use blanks in the any field to indicate no change to the existing value.

Climate station ID:  Required - climate station(s) to fill (use \* for wildcard).

Latitude:  Optional - decimal degrees.

Elevation:  Optional - feet.

Region 1:  Optional - primary region for the climate station (typically county).

Region 2:  Optional - secondary region for the climate station (traditionally HUC or blank).

Name:  Optional - up to 28 characters for StateCU.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillClimateStation(ID="20*", Latitude=37.5, Elevation=6000, Region1="RIO GRANDE")
```

OK Cancel

FillClimateStation

**FillClimateStation() Command Editor**

The command syntax is as follows:

```
FillClimateStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single climate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Latitude	The climate station latitude to be assigned for all matching climate stations with missing latitude.	If not specified, the original value will remain.
Elevation	The climate station elevation to be assigned for all matching climate stations with missing elevation.	If not specified, the original value will remain.
Region1	The climate station Region1 (typically county) to be assigned for all matching climate stations with missing Region1.	If not specified, the original value will remain.
Region2	The climate station Region2 (traditionally HUC but can be blank) to be assigned for all matching climate stations with missing Region2.	If not specified, the original value will remain.
Name	The climate station name to be assigned for all matching climate stations with missing name.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID pattern is not matched</li> <li>• Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>• Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

---

# Command Reference: FillClimateStationsFromHydroBase()

Fill climate station data from HydroBase

**StateCU Command**

Version 3.08.02, 2010-01-05

The `FillClimateStationsFromHydroBase()` command fills missing data in existing climate stations, using HydroBase for data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillClimateStationsFromHydroBase() Command**

This command fills missing data in climate stations using data from HydroBase.  
The following values from HydroBase are set if missing in a climate station:

- Name
- Latitude
- Elevation (feet)
- Region1 (currently can only be a county)
- Region2 (currently can only be a Hydrologic Unit Code, HUC)

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

```
FillClimateStationsFromHydroBase ( ID="*" )
```

OK Cancel

FillClimateStationsFromHydroBase

**FillClimateStationsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillClimateStationsFromHydroBase( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single climate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the climate station is not found</li> <li>Ignore – ignore (don't generate a message) if the climate station is not found</li> <li>Warn – generate a warning message if the climate station is not found</li> </ul>	Warn

The following example command file illustrates how climate stations can be defined, filled from HydroBase, and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
FillClimateStationsFromHydroBase( ID="*" )
SetClimateStation( ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation( ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation( ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation( ID="0484",Region1="MOFFAT",IfNotFound=Add)
WriteClimateStationsToStateCU(OutputFile="COclim2006.cli")
```

# Command Reference: FillCropPatternTSConstant()

Fill crop pattern time series values using a constant value

**StateCU Command**

Version 3.09.01, 2010-02-01

The `FillCropPatternTSConstant()` command fills crop pattern time series data for a CU Location, using a constant value. Only data for matching locations, years, and crop type are filled. A common use of this command is to ensure that there are no missing data values for years when an irrigated lands assessment has occurred. In this case it is assumed that if other data sources for the year of study have not identified crops (e.g., GIS and user-supplied values), then remaining missing values should be set to zero, indicating that no irrigation occurred. For example: if in one year a structure has irrigated acreage but in another year is has no acreage, the time series for the crop acreage will have a missing value in the second case. Using this command with a zero constant value will ensure that zero is used for the second year. Subsequent data filling by repeating values or interpolation will be impacted by the constant values.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCropPatternTSConstant() Command**

This command fills missing data in crop pattern time series, using the CU Location ID, crop type, and year to uniquely identify time series. Missing values are replaced with a constant value.  
The CU Location ID and crop type can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only years in the output period can be filled.

CU location ID:  Required - CU location(s) to fill (use \* for wildcard)

Crop type:  Required - specify the crops to fill (use \* for wildcard, or separate by commas).

Constant:  Required - constant value to use for filling.

Include surface water supply?:  Optional - include locations with surface water supply? (default=True).

Include groundwater only supply?:  Optional - include locations with only groundwater supply? (default=True).

Fill start (year):  Optional - start year as 4-digits (default=fill all).

Fill end (year):  Optional - end year as 4-digits or (default=fill all).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

FillCropPatternTSConstant

**FillCropPatternTSConstant() Command Editor**

The command syntax is as follows:

```
FillCropPatternTSConstant (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
CropType	A single crop type to match or a pattern using wildcards (e.g., *).	None – must be specified.
Constant	The constant value to be used to fill missing data.	None – must be specified.
IncludeSurfaceWaterSupply	Indicate whether locations with surface water supply should be processed (those other than groundwater-only locations).	True
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply (collections where PartType=Parcel) should be processed. Typically this is specified as True.	True
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: FillCropPatternTSInterpolate()

Fill crop pattern time series values using interpolation

**StateCU Command**

Version 3.09.01, 2010-02-01

The `FillCropPatternTSInterpolate()` command fills crop pattern time series data for a CU Location, using interpolation. Data will not be extrapolated past the end-points and therefore another fill method (e.g., `FillCropPatternTSRepeat()`) may be required after the interpolation command. Filling is currently always in a forward direction. If the data set contains groundwater, it is typical to fill groundwater-only crop pattern time series prior to the first year of HydroBase data using `FillCropPatternTSUsingWellRights()` and use `FillCropPatternTSRepeat()` and/or `FillCropPatternTSInterpolate()` for all other time series and parts of the period.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCropPatternTSInterpolate() Command**

This command fills missing data in crop pattern time series, using the CU Location ID, crop type, and year to uniquely identify time series. Missing values are replaced by interpolating between known crop area values. The CU Location ID and crop type can contain a \* wildcard pattern to match one or more time series. The fill period can optionally be specified. Only years in the output period can be filled.

CU location ID:  Required - CU location(s) to fill (use \* for wildcard)

Crop type:  Required - specify the crops to fill (use \* for wildcard, or separate by commas).

Include surface water supply?:  Optional - include locations with surface water supply? (default=True).

Include groundwater only supply?:  Optional - include locations with only groundwater supply? (default=True).

Fill start (year):  Optional - start year as 4-digits (default=fill all).

Fill end (year):  Optional - end year as 4-digits or (default=fill all).

Max Intervals:  Optional - specify limit on intervals to fill (default=no limit).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `FillCropPatternTSInterpolate ( ID="*", IncludeSurfaceWaterSupply=True, CropType="*")`

OK Cancel

FillCropPatternTSInterpolate

**FillCropPatternTSInterpolate() Command Editor**

The command syntax is as follows:

```
FillCropPatternTSInterpolate(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
CropType	A single crop type to match or a pattern using wildcards (e.g., *).	None – must be specified.
IncludeSurfaceWaterSupply	Indicate whether locations with surface water supply should be processed (those other than groundwater-only locations).	True
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply (collections where PartType=Parcel) should be processed. Typically this is specified as true unless FillCropPatternTSUsingWellRights() has been applied for the fill period.	True
FillStart	The first year to fill. This should be a year with observations to allow interpolation.	If not specified, fill the full period.
FillEnd	The last year to fill. This should be a year with observations to allow interpolation.	If not specified, fill the full period.
MaxIntervals	The maximum number of intervals to fill in any gap.	If not specified, fill the entire gap.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



# Command Reference: FillCropPatternTSRepeat()

Fill crop pattern time series values by repeating values

## StateCU Command

Version 3.09.01, 2010-02-01

The `FillCropPatternTSRepeat()` command fills crop pattern time series data for a CU Location by repeating known values. Filling can occur forward or backward in time, but not both. Therefore, it may be necessary to use two similar commands, one filling forward, and one filling backward, in order to completely fill the ends of time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCropPatternTSRepeat() Command**

This command fills missing data in crop pattern time series, using the CU Location ID, crop type, and year to uniquely identify time series. Missing values are replaced by repeating known values. The CU Location ID and crop type can contain a \* wildcard pattern to match one or more time series. The fill period can optionally be specified. Only years in the output period can be filled.

CU location ID:  Required - CU location(s) to fill (use \* for wildcard)

Crop type:  Required - specify the crops to fill (use \* for wildcard, or separate by commas).

Include surface water supply?:  Optional - include locations with surface water supply? (default=True).

Include groundwater only supply?:  Optional - include locations with only groundwater supply? (default=True).

Fill start (year):  Optional - start year as 4-digits (default=fill all).

Fill end (year):  Optional - end year as 4-digits or (default=fill all).

Fill Direction:  Optional - direction to process data (default=Forward).

Max Intervals:  Optional - specify limit on intervals to fill (default=no limit).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillCropPatternTSRepeat (ID="*", CropType="*", FillStart=1950, FillEnd=1993, FillDirection=Backward)
```

OK Cancel

FillCropPatternTSRepeat

### FillCropPatternTSRepeat() Command Editor

The command syntax is as follows:

```
FillCropPatternTSRepeat (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
CropType	A single crop type to match or a pattern using wildcards (e.g., *).	None – must be specified.
IncludeSurfaceWaterSupply	Indicate whether locations with surface water supply should be processed (those other than groundwater-only locations).	True
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply (collections where PartType=Parcel) should be processed. Typically this is specified as true.	True
FillStart	The first year to fill, typically a year with observations if filling forward.	If not specified, fill the full period.
FillEnd	The last year to fill, typically a year with observations if filling backward.	If not specified, fill the full period.
FillDirection	The direction to fill, either Forward or Backward.	Forward
MaxIntervals	The maximum number of intervals to fill in any gap.	If not specified, fill the entire gap.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

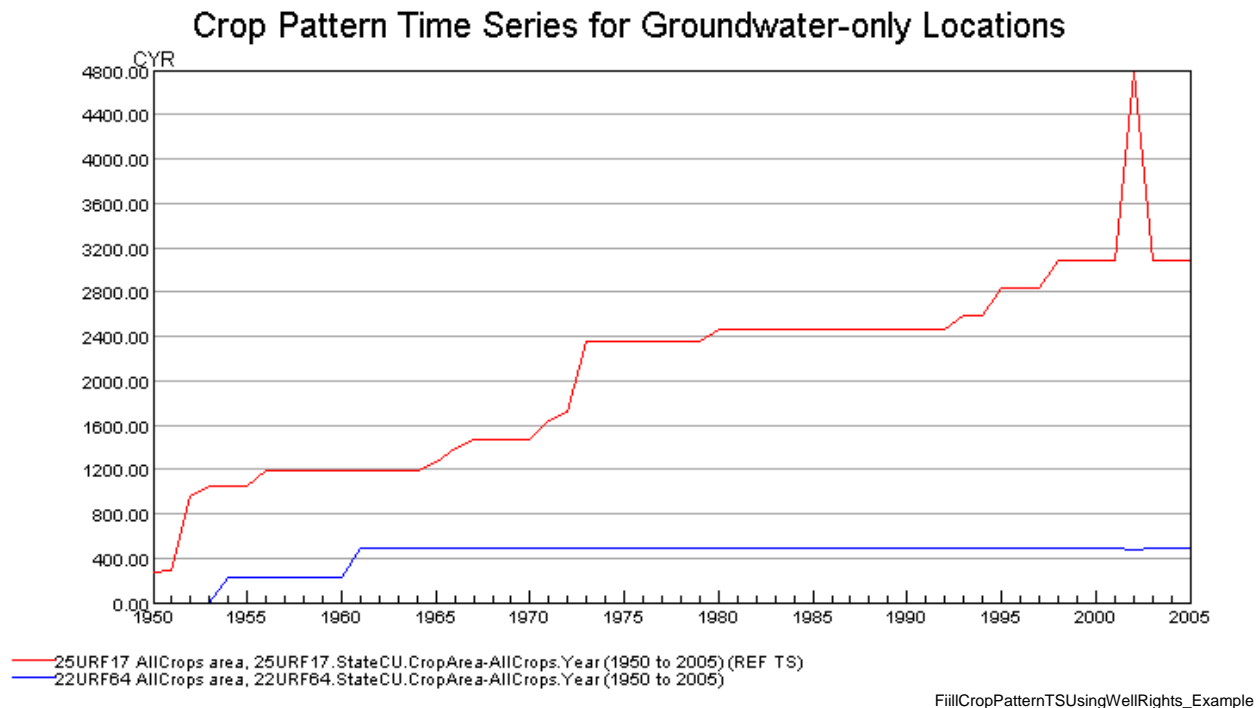
# Command Reference: FillCropPatternUsingWellRights()

Fill missing crop pattern time series (yearly) acreage values using well rights

StateCU Command  
Version 3.09.01, 2010-02-01

**This is a legacy command that should not be used in current work. It is included to help migrate legacy command files.**

The `FillCropPatternTSUsingWellRights()` fills missing crop pattern time series (yearly) information for CU locations using well rights. This command should typically only be used to fill data in the period before the earliest modeling year for which data are available in HydroBase and helps initialize the acreage data in the early period. For example, in the Río Grande, 1998 parcel data and associated rights are used to fill the earlier period. The parcels associated with groundwater are turned off earlier in time, in years when no well water rights are associated with parcels. This results in the crop pattern acreage decreasing back in time. It is typical that only the groundwater-only locations are filled with this command, given that the parcel's supply can be related directly to well water rights. Crop pattern time series for locations having surface water supply are often then interpolated or repeated back in time. The following figure shows groundwater acreage filled using well water rights.



Prerequisites:

1. This command should be executed after the crop pattern time series are read from HydroBase (see `ReadCropPatternTSFromHydroBase()`), which saves a list of parcels associated with

each location during processing). Data for lands that are not in HydroBase should have been specified with `SetCropPatternTSFromList()` commands.

2. A non-merged, non-aggregated well water right file should have been read using the `ReadWellRightsFromStateMod()` or similar command. A StateMod well rights file with comments including parcel year and parcel identifier are needed to ensure that rights matching the parcels for `ParcelYear` are available (see parameter description below).

The steps executed by the command are described below. Note that “CU location” refers to the StateCU model identifier.

1. For each parcel found in the water rights data, create a yearly time series of decree. The resulting time series indicates for a parcel the decreed water rights (y-axis) associated with the parcel over time (x-axis).
2. Loop through each CU location that matches the ID pattern and perform the following:
  - a. Get the list of parcels associated with the location for `ParcelYear`, taken from the crop pattern time series. The list of parcels will have been saved when the `ReadCropPatternTSFromHydroBase()` command was processed.
  - b. For each year being processed, if acreage time series are missing, loop over the list of parcels for the location (note that the parcel area will be multiplied by the ditch coverage percent irrigated if the parcel is for a D&W node):
    - i. If no parcels were found for the location in the `ParcelYear`, set all crop pattern time series to zero. Consequently, an estimate of zero acreage will occur.
    - ii. Otherwise, set the crop pattern time series values as follows:
      - A. If the decree time series for the parcel is zero in a year, set the acreage for all crops and the total to zero for the year.
      - B. If the parcel has groundwater supply (one or more wells in `ParcelYear`): increment the acreage for the crop grown on the parcel. Recompute the total acreage.
      - C. If the result is missing, set the acreage for all crops and the total to zero.

The following dialog is used to edit the command and illustrates the syntax of the command:

**Edit FillCropPatternTSUsingWellRights() Command**
✖

This command fills missing data in crop pattern time series, using the CU Location ID, crop type, and year to uniquely identify time series. Missing values are replaced by using crop data at parcels for the specified parcel year only if at least one water right is available for the parcel.

The resulting parcels are added to give a total by crop for the year.

The CU Location ID and crop type can contain a \* wildcard pattern to match one or more time series.

The fill period can optionally be specified. Only years in the output period can be filled.

CU location ID:

\*

Required - CU location(s) to fill (use \* for wildcard)

Crop type:

\*

Required - specify the crops to fill (use \* for wildcard, or separate by commas).

Parcel data year:

1998

Required - 4-digit year for parcel data to turn acreage on/off.

Include surface water supply?:

False

Optional - include locations with surface water supply? (default=True).

Include groundwater only supply?:

True

Optional - include locations with only groundwater supply? (default=True).

Fill start (year):

1936

Optional - start year as 4-digits (default=fill all).

Fill end (year):

1997

Optional - end year as 4-digits or (default=fill all).

If not found:

▼

Optional - indicate action if no match is found (default=Warn).

Command:

```
FillCropPatternTSUsingWellRights ( ID="*", IncludeSurfaceWaterSupply=False, CropType="*", FillStart=1936, FillEnd=1997, ParcelYear=1998)
```

OK

Cancel

FillCropPatternTSUsingWellRights

### FillCropPatternTSUsingWellRights() Command Editor

The command syntax is as follows:

```
FillCropPatternTSUsingWellRights( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IncludeSurfaceWaterSupply	Indicate whether locations with surface water supply should be processed (those other than groundwater-only locations). Locations will only be processed if they also have groundwater supply. <b>Currently this must always be specified as False – interpolation or repeat commands are typically used for surface water supply lands. Additional capability may be enabled in the future.</b>	True
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply should be processed. Typically this is specified as true.	True
CropType	Crop type(s) to fill or blank to fill all. If more than one specific crop, separate with commas.	Fill all.
FillStart	A starting year to fill data, normally the start of the output period.	The output period start.
FillEnd	An ending year to fill data, normally one year prior to the ParcelYear.	The output period end.
ParcelYear	A calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems. Only the water rights generated from parcels in this year will be used to limit groundwater acreage.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillCULocation()

## Fill CU Location data

### StateCU Command

Version 3.09.00, 2010-01-24

The `FillCULocation()` command fills missing data in existing CU Locations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCULocation() Command**

This command fills missing data in CU Location(s), using the CU Location ID to look up the location.  
The CU Location ID can contain a \* wildcard pattern to match one or more locations.  
Use blanks in the any field to indicate no change to the existing value.  
See also the FillCULocationClimateStationWeights() command.

CU Location ID:  Required - specify the CU Location(s) to fill (use \* for wildcard)

Name:  Optional - up to 28 characters for StateCU.

Latitude:  Optional - decimal degrees.

Elevation:  Optional - feet.

Region 1:  Optional - primary region for the CU location (typically county).

Region 2:  Optional - secondary region for the CU location (traditionally HUC or blank).

AWC:  Optional - Available Water Content, fraction (0-1).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillCULocation(ID="21*",Latitude=37.1667,Region1="CONEJOS",Region2="13010002")
```

OK Cancel

FillCULocation

### FillCULocation() Command Editor

The command syntax is as follows:

```
FillCULocation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for matching locations.	If not specified, the original value will remain.
Latitude	The latitude to be assigned for all matching CU Locations with missing latitude.	If not specified, the original value will remain.
Elevation	The elevation to be assigned for matching locations.	If not specified, the original value will remain.
Region1	The Region1 to be assigned for all matching CU Locations with missing Region1.	If not specified, the original value will remain.
Region2	The Region2 to be assigned for all matching CU Locations with missing Region2.	If not specified, the original value will remain.
AWC	The available water content (AWC) to be assigned for all matching CU Locations with missing AWC.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn



# Command Reference: FillCULocationClimateStationWeights ( )

Fill CU Location climate station weights data

**StateCU Command**

Version 3.09.00, 2010-01-24

The `FillCULocationClimateStationWeights ( )` command fills climate station weights data in existing CU Locations. Only locations that have no climate stations assigned will be modified. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit FillCULocationClimateStationWeights() Command

This command fills the climate station identifiers and weights associated with a CU location.  
Only locations with no previous station weights are changed.  
Each CU location must be associated with one or more precipitation and temperature stations.  
The data from each station is weighted, and the weights should add to 1.0.  
The climate station weights should be specified using the format (, and ; are equivalent):  
    StationID,TempWt,PrecWt;StationID,TempWt,PrecWt,...  
For example:  
    2184,.7,.7,3951,.3,.3  
The weights are specified as a fraction 0.0 to 1.0.  
If orographic adjustment factors are included, insert the factors after the weights in the order of temperature adjustment, precipitation adjustment, where the temperature adjustment is degrees F/1000 ft. of elevation and the precipitation adjustment is a fraction (0.0 to 1.0).

CU location ID:  Required - specify the CU Location(s) to process (use \* for wildcard)

Include orographic temperature adjustment?:  Optional - include orographic temperature adjustment factor in data (default=False).

Include orographic precipitation adjustment?:  Optional - include orographic precipitation adjustment factor in data (default=False).

Climate station weights:

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

FillCULocationClimateStationWeights (ID="36\*",Weights="4664,1.0,0,3592,0,1.0")

OK

Cancel

FillCULocationClimateStationWeights

**FillCULocationClimateStationWeights ( ) Command Editor**

The command syntax is as follows:

```
FillCULocationClimateStationWeights(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Include Orographic TempAdj	If True, include the orographic temperature adjustment factor, after the Weights described below, specified as degrees/1000 feet.	False
Include Orographic PrecAdj	If True, include the orographic precipitation adjustment factor, after the Weights described below, specified as a fraction 0.0 to 1.0. Place after the orographic temperature adjustment factor if it is specified.	False
Weights	A repeating pattern of StationID, TempWt, PrecWt, where the station identifiers match climate station identifiers and the weights are specified as fractions in the range 0.0 to 1.0. Also include the orographic temperature and/or orographic precipitation adjustment factors if the above parameters are True.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

An example command file is shown below:

```
ReadCULocationsFromList(ListFile="cmstrlist.csv",IDCol=1,NameCol=6)
FillCULocationsFromHydroBase(ID="*",CULocType=Structure,Region1Type=County,Region2Type=HUC)
SetCULocationsFromList(ListFile="cmstrlist.csv",IDCol=1,LatitudeCol=2,AWCCol=11)
SetCULocationsFromList(ListFile="plateau.csv",IDCol=1,Region1Col=2)
SetCULocationClimateStationWeightsFromList(ListFile="cowts.csv",StationIDCol=1,
    Region1Col=2,Region2Col=3,TempWtCol=4,PrecWtCol=5)
FillCULocationClimateStationWeights(ID="72_ADC065",Weights="3146,0.68,0.68,3489,0.32,0.32")
FillCULocationClimateStationWeights(ID="36*",Weights="4664,1.0,0.3592,0,1.0")
FillCULocationClimateStationWeights(ID="37*",Weights="2454,1.0,1.0")
FillCULocationClimateStationWeights(ID="38*",Weights="3359,1.0,1.0")
FillCULocationClimateStationWeights(ID="39*",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="45*",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="50*",Weights="3500,0.5,0.5,4664,0.5,0.5")
FillCULocationClimateStationWeights(ID="51*",Weights="3500,0.5,0.5,4664,0.5,0.5")
FillCULocationClimateStationWeights(ID="52*",Weights="9265,1.0,1.0")
FillCULocationClimateStationWeights(ID="53*",Weights="9265,1.0,1.0")
FillCULocationClimateStationWeights(ID="70*",Weights="0214,1.0,1.0")
FillCULocationClimateStationWeights(ID="72*",Weights="1741,1.0,1.0")
FillCULocationClimateStationWeights(ID="950001",Weights="3146,0.68,0.68,3489,0.32,0.32")
FillCULocationClimateStationWeights(ID="950010",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="950011",Weights="7031,1.0,1.0")
FillCULocationClimateStationWeights(ID="950050",Weights="3146,0.68,0.68,3489,0.32,0.32")
WriteCULocationsToStateCU(OutputFile="cm2006.str",WriteHow=OverwriteFile)
# Check the results
CheckCULocations(ID="*")
WriteCheckFile(OutputFile="cm2006.str.check.html")
```

---

# Command Reference: FillCULocationsFromHydroBase()

## Fill CU Location data from HydroBase

### StateCU Command

Version 3.09.00, 2010-01-24

The `FillCULocationsFromHydroBase()` command fills missing data in existing CU Locations, using HydroBase for data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCULocationsFromHydroBase() Command**

This command fills missing data in CU Locations by using data from HydroBase, matching the CU Location identifiers. Currently the CU Location type can only be a structure. In the future, counties, etc., may be supported. The following values from HydroBase are set if missing in a CU Location:

- Region1 (currently can only be a county)
- Region2 (currently can only be a Hydrologic Unit Code, HUC)
- Name (taken from the structure data)
- Latitude (taken from structure location)

Specifying the types for the location, Region1, and Region2 indicate how the data should be taken from HydroBase.

CU location ID:	<input type="text" value="*"/>	Required - the CU locations to fill (use * for wildcard).
CU location type:	<input type="button" value="Structure"/>	Optional - the location type for the data to be filled (default=Structure).
Region1 Type:	<input type="button" value="County"/>	Optional - the type of region associated with Region1 (default=County).
Region2 type:	<input type="button" value="HUC"/>	Optional - the type of region associated with Region2 (default=HUC).
If not found:	<input type="button" value="Warn"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillCULocationsFromHydroBase ( ID="*", CULocType=Structure, Region1Type=County, Region2Type=HUC)
```

OK Cancel

FillCULocationsFromHydroBase

**FillCULocationsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillCULocationsFromHydroBase( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
CULocType	Indicates whether CU Locations are structures or climate stations, to indicate which HydroBase data to query.	Structure
Region1Type	The meaning of Region1 in the data set, to indicate which HydroBase data to query.	County
Region2Type	The meaning of Region2 in the data set, to indicate which HydroBase data to query.	HUC
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

# Command Reference: FillCULocationsFromList()

Fill missing CU Location data using information in a delimited file

## StateCU Command

Version 03.09.00, 2010-01-24

The `FillCULocationsFromList()` command fills missing data in existing CU Locations by reading information from a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillCULocationsFromList() Command**

This command fills missing data in CU Location(s), using the CU Location ID to look up the location.  
Data are supplied by values in a comma-delimited file.  
Use blanks in the any field to indicate no change to the existing value.  
Columns should be delimited by commas (user-specified delimiters will be added in the future).  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillCULocationsFromList

List file:

CU location ID column:

Latitude Column:

Elevation Column:

Region 1 Column:

Region 2 Column:

Name Column:

AWC Column:

If not found:

Optional - indicate action if no ID match is found (default=Warn).

Command:

FillCULocationsFromList

**FillCULocationsFromList() Command Editor**

The command syntax is as follows:

```
FillCULocationsFromList( Parameter=Value...)
```

### Command Parameters

Parameter	Description	Default
ListFile	Path to the delimited list file to read.	None – must be specified.
IDCol	The column number (1+) containing the CU Location identifiers.	None – must be specified.
LatitudeCol	The column number (1+) containing the CU Location latitude.	If not specified, the previous value will remain.
ElevationCol	The column number (1+) containing the CU Location elevation.	If not specified, the previous value will remain.
Region1Col	The column number (1+) containing the CU Location Region1.	If not specified, the previous value will remain.
Region2Col	The column number (1+) containing the CU Location Region2.	If not specified, the previous value will remain.
NameCol	The column number (1+) containing the CU Location name.	If not specified, the previous value will remain.
AWCCol	The column number (1+) containing the CU Location AWC.	If not specified, the previous value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

An example list file is shown below, which provides elevations for key locations:

```
#WDID/NAME/Elevation(ft),,
0100501,EMPIRE DITCH,4543
0100503_D,RIVERSIDE CANAL,4533
0100507_D,BIJOU CANAL,4495
0100511,WELDON VALLEY DITCH,4405
0100513,JACKSON LAKE INLET DITCH,4460
0100514,FT MORGAN CANAL,4347
0100515,UPPER PLATTE BEAVER CNL,4289
0100517,DEUEL SNYDER CANAL,4310
0100518,LOWER PLATTE BEAVER D,4247
0100519_D,TREMONT DITCH,4243
0100520,GILL STEVENS DITCH,4224
...
```

---

# Command Reference:

## FillDiversionDemandTSMonthlyAverage()

**Fill diversion demand time series (monthly) values using average monthly values**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversionDemandTSMonthlyAverage()` command fills missing diversion demand time series (monthly) data, using average monthly values. The averages are computed immediately after reading time series (e.g., from HydroBase or a file) or calculation of the time series (e.g., from IWR/Eff<sub>ave</sub>). The average values that are used during data filling are printed to the log file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversionDemandTSMonthlyAverage() Command**

This command fills missing data in monthly diversion demand time series.  
Missing values are replaced with monthly average - average values are computed immediately after reading/calculating the data.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:  Specify the diversion stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

FillDiversionDemandTSMonthlyAverage

**FillDiversionDemandTSMonthlyAverage() Command Editor**

The command syntax is as follows:

```
FillDiversionDemandTSMonthlyAverage( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference:

## FillDiversiionDemandTSMonthlyConstant()

**Fill diversion demand time series (monthly) values using a constant value**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversiionDemandTSMonthlyConstant()` command fills missing diversion demand time series (monthly) data, using a constant value. This command is useful, for example, to set demand values to zero if other fill commands are unable to provide data estimates for missing data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversiionDemandTSMonthlyConstant() Command**

This command fills missing data in monthly diversion demand time series.  
Missing values are replaced with a constant value.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:  Specify the diversion stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Constant:  Required - constant value to use for filling.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillDiversiionDemandTSMonthlyConstant (ID="*", Constant=0)
```

FillDiversiionDemandTSMonthlyConstant

**FillDiversiionDemandTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
FillDiversionDemandTSMonthlyConstant (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
Constant	The constant value to be used to fill missing data.	None – must be specified.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## FillDiversionDemandTSMonthlyPattern()

Fill diversion demand time series (monthly) values using WET/DRY/AVG values

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillDiversionDemandTSMonthlyPattern()` command fills missing diversion demand time series (monthly) data, using average monthly wet/dry/average values. The averages are computed using patterns read by the `ReadPatternFile()` command. The average values that are used during data filling are printed to the log file. For example, if a value is missing for May 1980, the pattern for the specified pattern identifier is checked for WET, DRY, or AVG. The values of all May's for WET, DRY, or AVG are then averaged in the time series to be filled, and the resulting average used to fill missing data. This command therefore will result in filled values that are more appropriate than simple averages; however, work must be done to characterize the wet, dry, and average months.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversionDemandTSMonthlyPattern() Command**

This command fills missing data in monthly diversion demand time series.  
Missing values are replaced with monthly average - average values are computed by this command using the pattern data.  
One or more ReadPatternFile() commands must be used before this command.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:	<input type="text" value="90*"/>	Specify the diversion stations to fill (use * for wildcard)
Fill start:	<input type="text"/>	Optional - start date or blank to fill all.
Fill end:	<input type="text"/>	Optional - end date or blank to fill all.
Pattern identifier:	<input type="text" value="08220000"/>	Required - pattern ID to use for filling.
<= zero values in average?:	<input type="checkbox"/>	Optional - are values <= zero used in averages (default=True).
Fill flag:	<input type="text"/>	Optional - 1-character flag to track filled values, or "Auto".
If not found:	<input type="checkbox"/>	Optional - indicate action if no match is found (default=Warn).

Command:  
`FillDiversionDemandTSMonthlyPattern ( ID="90*", PatternID="08220000" )`

OK Cancel

FillDiversionDemandTSMonthlyPattern

### FillDiversionDemandTSMonthlyPattern() Command Editor

The command syntax is as follows:

```
FillDiversionDemandTSMonthlyPattern(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
PatternID	The pattern identifier for data read with a ReadPatternFile() command.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing monthly averages.	True
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## FillDiversiionHistoricalTSMonthlyAverage()

**Fill diversion historical time series (monthly) values using average monthly values**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversiionHistoricalTSMonthlyAverage()` command fills missing diversion historical time series (monthly) data, using average monthly values. The historical averages are computed immediately after reading time series (e.g., from HydroBase or a file). The average values that are used during data filling are printed to the log file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversiionHistoricalTSMonthlyAverage() Command**

This command fills missing data in monthly diversion historical time series.  
Missing values are replaced with monthly average - average values are computed immediately after reading/calculating the data.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
Collections (diversion aggregates and systems) may have already been filled during initial processing and can be skipped.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:  Specify the diversion stations to fill (use \* for wildcard)

Include collections:  Optional - default=True.

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

FillDiversiionHistoricalTSMonthlyAverage

**FillDiversiionHistoricalTSMonthlyAverage() Command Editor**

The command syntax is as follows:

```
FillDiversionHistoricalTSMonthlyAverage(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IncludeCollections	Indicates whether time series for collections (diversion stations that are aggregates or systems) are included in the processing. If the time series for these stations have been filled during the read, then it may not be necessary to fill again. On the other hand, it may be necessary to use the sum of the time series to fill missing data.	True
FillStart	The first date to fill.	If not specified, fill the full period.
FillEnd	The last date to fill.	If not specified, fill the full period.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following abbreviated command file illustrates how the StateMod diversion historical time series file can be produced. Note that an initial diversion stations file is read and is then updated based on time series information.

```

StartLog(LogFile="ddh.commands.StateDMI.log")
# ddh.commands.StateDMI
#
# StateDMI command file to create the historical diversion file
# and the "step 2" direct diversion structure file, updated so structure
# capacity = maximum historical diversion
#
# Step 1 - set time-series period and year type
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read structure list from preliminary direct diversion structure file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 3 - read aggregate and diversion system structure assignments. Note that
# want to combine historical diversions for aggs and diversion systems, but
# historical diversions are separate for primary and secondary components
# of multistructures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
#
# Step 4 - read historical diversions from HydroBase. Note that want individual structures
# in aggregates and diversion systems to be filled first, then diversions
# combined.
#
ReadDiversionHistoricalTSMonthlyFromHydroBase(ID="*",IncludeCollections=False,
    UseDiversionComments=True)
#
# Step 5 - read fill pattern file, and assign patterns to water districts
#
ReadPatternFile(InputFile="fill2005.pat")
ReadDiversionHistoricalTSMonthlyFromHydroBase(ID="36*",IncludeExplicit=False,
    UseDiversionComments=True,
    PatternID="09037500",FillPatternOrder=1,FillAverageOrder=2)
#
# Step 6 - assign transbasin diversions from streamflow gages
#
SetDiversionHistoricalTSMonthly(ID="364626",TSID="09047300.DWR.Streamflow.Month~HydroBase")
...similar commands omitted...
# note that adams tunnel streamgage ID changed in 10/1996 from 09013000 to ADANETCO
SetDiversionHistoricalTSMonthly(ID="514634",TSID="514634...MONTH~StateMod~514634.stm")
# Con-Hoosier System - Blue River Diversion, driven by operating rules to con-hoosier
# summary demand
SetDiversionHistoricalTSMonthly(ID="364683",TSID="364683...MONTH~StateMod~zero.stm")
SetDiversionHistoricalTSMonthly(ID="364699",TSID="364699...MONTH~StateMod~zero.stm")
# Fryingpan-Arkansas Project
SetDiversionHistoricalTSMonthly(ID="381594",TSID="381594...MONTH~StateMod~381594.stm")
SetDiversionHistoricalTSMonthly(ID="384625",TSID="384625...MONTH~StateMod~384625.stm")
SetDiversionHistoricalTSMonthly(ID="954699",TSID="954699...MONTH~StateMod~zero.stm")
...similar commands omitted...
#
# Step 7 - set diversions from external time-series files
#

```

```

# The following commands are added to access Task 11.2 replacement files
SetDiversionHistoricalTSMonthly(ID="380757",TSID="380757...MONTH~StateMod~380757.stm")
...similar commands omitted...#
# The following structures are set for Municipal and Industrial Diversions
SetDiversionHistoricalTSMonthly(ID="360784",TSID="360784...MONTH~StateMod~360784.stm")
...similar commands omitted...
#
# Set transbasin diversions to "0" prior to construction
#
#       Wurtz Ditch
SetDiversionHistoricalTSMonthlyConstant(ID="374648",Constant=0,SetEnd="01/1929")
...similar commands omitted...
#
# Step 8 - fill historical diversion using pattern approach
#
FillDiversionHistoricalTSMonthlyPattern(ID="36*",PatternID="09034500")
...similar commands omitted...
#
# Step 9 - Fill remaining missing with month average
#
FillDiversionHistoricalTSMonthlyAverage(ID="*")
#
# Step 10 - Limit filled diversion to water rights. Exceptions include structure
#           receiving significant reservoir supply, carrier structures, etc.
#
LimitDiversionHistoricalTSMonthlyToRights(InputFile="..\statemod\cm2005.ddr",
    ID="*",IgnoreID="954683,952001,950010,950011")
#
# Step 11 - sort structures and create historical diversion file
#
SortDiversionHistoricalTSMonthly(Order=Ascending)
WriteDiversionHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
#
# Step 12 - update capacities and create final direct diversion station file
#
SetDiversionStationCapacitiesFromTS(ID="*")
WriteDiversionStationsToStateMod(OutputFile="..\statemod\cm2005.dds")
#
# Check the results.
CheckDiversionHistoricalTSMonthly(ID="*")
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")

```



---

# Command Reference:

## FillDiversiionHistoricalTSMonthlyConstant()

**Fill diversion historical time series (monthly) values using a constant value**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversiionHistoricalTSMonthlyConstant()` command fills missing diversion historical time series (monthly) data, using a constant value. This command is useful, for example, to set diversion values to zero if other fill commands are unable to provide data estimates for missing data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversiionHistoricalTSMonthlyConstant() Command**

This command fills missing data in monthly diversion historical time series.  
Missing values are replaced with a constant value.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
Collections (diversion aggregates and systems) may have already been filled during initial processing and can be skipped.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:  Specify the diversion stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Constant:  Required - constant value to use for filling.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillDiversiionHistoricalTSMonthlyConstant ( ID="*", Constant=0)
```

FillDiversiionHistoricalTSMonthlyConstant

**FillDiversiionHistoricalTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
FillDiversionHistoricalTSMonthlyConstant(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
Constant	The constant value to be used to fill missing data.	None – must be specified.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## FillDiversiionHistoricalTSMonthlyPattern()

**Fill diversion historical time series (monthly) values using WET/DRY/AVG values**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversiionHistoricalTSMonthlyPattern()` command fills missing diversion historical time series (monthly) data, using average monthly wet/dry/average values. The historical averages are computed using patterns read by the `ReadPatternFile()` command. The average values that are used during data filling are printed to the log file. For example, if a value is missing for May 1980, the pattern for the specified pattern identifier is checked for WET, DRY, or AVG. The values of all May's for WET, DRY, or AVG are then averaged in the time series to be filled, and the resulting average used to fill missing data. This command therefore will result in filled values that are more appropriate than simple averages; however, work must be done to characterize the wet, dry, and average months.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversiionHistoricalTSMonthlyPattern() Command**

This command fills missing data in monthly diversion historical time series.  
Missing values are replaced with monthly average - average values are computed by this command using the pattern data.  
One or more `ReadPatternFile()` commands must be used before this command.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
Collections (diversion aggregates and systems) may have already been filled during initial processing and can be skipped.  
The fill period can optionally be specified. Only months in the output period can be filled.

Diversion station ID:	<input type="text" value="36*"/>	Specify the diversion stations to fill (use * for wildcard)
Include collections:	<input type="button" value="v"/>	Optional - default=True.
Fill start:	<input type="text"/>	Optional - start date or blank to fill all.
Fill end:	<input type="text"/>	Optional - end date or blank to fill all.
Pattern identifier:	<input type="text" value="09034500"/>	Required - pattern ID to use for filling.
<= zero values in average?:	<input type="button" value="v"/>	Optional - are values <= zero used in averages (default=True).
Fill flag:	<input type="text"/>	Optional - 1-character flag to track filled values, or "Auto".
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillDiversiionHistoricalTSMonthlyPattern(ID="36*", PatternID="09034500")
```

OK Cancel

**FillDiversiionHistoricalTSMonthlyPattern() Command Editor**

The command syntax is as follows:

```
FillDiversionHistoricalTSMonthlyPattern(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IncludeCollections	Indicates whether time series for collections (diversion stations that are aggregates or systems) are included in the processing. If the time series for these stations have been filled during the read, then it may not be necessary to fill again. On the other hand, it may be necessary to use the sum of the time series to fill missing data.	True
FillStart	The first date to fill.	If not specified, fill the full period.
FillEnd	The last date to fill.	If not specified, fill the full period.
PatternID	The pattern identifier for data read with a <code>ReadPatternFile()</code> command.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing monthly averages.	True
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: FillDiversiionRight()

## Fill diversion right data

### StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `FillDiversiionRight()` command fills missing data in existing diversion rights. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversiionRight() Command**

This command fills missing data in diversion right(s), using the diversion right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="22MS03*"/>	Required - specify the right(s) to fill (use * for wildcard)
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text"/>	Optional - station identifier.
Administration number:	<input type="text"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text"/>	Optional - decree amount, CFS.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="button" value="Warn"/>	Optional - indicate action if no match is found (default=Warn).
Command:	<pre>FillDiversiionRight (ID="22MS03*", OnOff=1)</pre>	

OK Cancel

FillDiversiionRight

**FillDiversiionRight() Command Editor**

The command syntax is as follows:

```
FillDiversionRight (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single diversion right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching diversion right identifiers with missing name.	If not specified, the original value will remain.
StationID	The diversion station identifier to be assigned for all matching diversion right identifiers with missing diversion station identifier.	If not specified, the original value will remain.
AdministrationNumber	The administration number to be assigned for all matching diversion right identifiers with missing administration number.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching diversion right identifiers with missing administration decree.	If not specified, the original value will remain.
OnOff	The on/off switch to be assigned for all matching diversion right identifiers with missing on/off switch, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: FillDiversiionStation()

## Fill diversion station data

### StateMod Command

Version 3.09.01, 2010-02-01

The FillDiversiionStation() command fills missing data in existing diversion stations. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit FillDiversiionStation() Command

This command fills missing data in diversion station(s), using the diversion station ID to look up the location.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
Use blanks in the any field to indicate no change to the existing value.  
Monthly efficiencies should be separated by commas, with January first.  
Returns should be specified as triplets of location, percent, and return table ID:  
08123456,50.0,1;08234567,50.0,2

Diversion station ID:	*	Required - ID for stations to fill (use * for wildcard)
Name:		Optional - up to 24 characters for StateMod.
River node ID:	ID	Optional - the river node identifier, or "ID" to use the station ID.
On/Off:		Optional - is station on/off in data set?
Capacity:	999	Optional - diversion capacity, CFS.
Replacement Res. Option:	-1 - Provide depletion replacement	Optional - replacement reservoir option.
Daily ID:	4	Optional - the daily identifier, "ID", or StateMod flag).
User name:		Optional - specify the user name.
Demand type:	1 - Monthly total demand	Optional - monthly demand time series type.
Irrigated acres:		Optional - typically for the most recent year.
Use type:	1 - Irrigation	Optional - water use type.
Demand source:	2 - Irrigated acres from structure file (tia)	Optional - water demand source.
Efficiency (Annual):	50	Optional - annual efficiency, percent (ignore if setting monthly).
Efficiencies (Monthly):		Optional - percent, annual is recomputed as average.
Returns (optional):		
If not found:		Optional - indicate action if no match is found.
Command:	FillDiversiionStation (ID="*",RiverNodeID="ID",Capacity=999,ReplaceResOption=-1,DailyID="4",DemandType=1,UseType=1,DemandSource=2,EffAnnual=50)	

OK

Cancel

FillDiversiionStation

## FillDiversiionStation() Command Editor

The command syntax is as follows:

```
FillDiversionStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching diversion stations with missing name.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching diversion stations with missing river node identifier. Specify ID to assign to the diversion station identifier.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching diversion stations with missing switch, either 1 for on or 0 for off.	If not specified, the original value will remain.
Capacity	The diversion station capacity to be assigned for all matching diversion stations with missing capacity, CFS.	If not specified, the original value will remain.
ReplaceResOption	The replacement reservoir option to be assigned for all matching diversion stations with missing option, as per the StateMod documentation.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching diversion stations with missing daily identifier.	If not specified, the original value will remain.
UserName	The diversion user name (owner) to be assigned for all matching diversion stations with missing user name.	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching diversion stations with missing demand type (see StateMod documentation).	If not specified, the original value will remain.
IrrigatedAcres	The irrigated acres to be assigned for all matching diversion stations with missing irrigated acres.	If not specified, the original value will remain.
UseType	The use type to be assigned for all matching diversion stations with missing user type (see StateMod documentation).	If not specified, the original value will remain.
DemandSource	The demand source to be assigned for all matching diversion stations with missing demand source (see StateMod documentation).	If not specified, the original value will remain.
EffAnnual	The annual efficiency (percent, 0 - 100) to be assigned for all matching diversion stations with missing annual efficiency (see StateMod documentation). Monthly efficiencies will be set to the same value (but not used).	If not specified, the original value will remain.
EffMonthly	The monthly efficiencies (percent, 0 – 100) to be assigned for all matching diversion stations with missing data, specified as 12 comma-separated values, January to December. The annual efficiency will be set to the average value. The	If not specified, the original value will remain.



Parameter	Description	Default
	order of the values in the output file will be according to the output year type set by <code>setOutputYearType()</code> , or calendar by default.	
Returns	The return flows to be assigned for all matching diversion stations with missing returns. Specify as <code>StationID,Percent,DelayTableID; StationID,Percent,DelayTableID;</code> etc.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

This page is intentionally blank.

---

# Command Reference: FillDiversionsFromHydroBase()

Fill diversion station data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillDiversionsFromHydroBase()` command fills missing data in existing diversion stations, using HydroBase for data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversionsFromHydroBase() Command**

This command fills missing data in diversion stations using data from HydroBase, matching the diversion station identifiers. The following values from HydroBase are set if missing in a station:

- Name
- Capacity (can be reset when historical time series are processed)
- Demand source (checks whether GIS data are available)
- User name
- Area (first available from GIS, diversion comments, structure TIA)

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillDiversionsFromHydroBase ( ID="*" )
```

OK Cancel

FillDiversionsFromHydroBase

**FillDiversionsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillDiversionStationsFromHydroBase( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: FillDiversionsFromNetwork()

Fill diversion station data from a StateMod network

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillDiversionsFromNetwork()` command fills missing data in diversion stations, using a StateMod network for data. This command is usually used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of diversion stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillDiversionsFromNetwork() Command**

This command fills missing data in diversion stations using data from the network, matching the diversion station identifiers. This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

Name

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `FillDiversionsFromNetwork ( ID="*" )`

OK Cancel

FillDiversionsFromNetwork

**FillDiversionsFromNetwork() Command Editor**

The command syntax is as follows:

```
FillDiversionStationsFromNetwork( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

The following example illustrates how to fill diversion station names from the network. A command to fill from HydroBase or another source will often be run before the second command below.

```
ReadDiversionStationsFromNetwork( InputFile="sp2005.net" )  
FillDiversionStationsFromNetwork( ID="*" )
```

# Command Reference: FillInstreamFlowRight()

Fill instream flow right data

## StateMod Command

Version 3.09.01, 2010-02-02

The `FillInstreamFlowRight()` command fills missing data in existing instream flow rights. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillInstreamFlowRight() Command**

This command fills missing data in instream flow right(s), using the instream flow right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="220831.02"/>	Required - specify the right(s) to fill (use * for wildcard)
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text"/>	Optional - station identifier.
Administration number:	<input type="text" value="99999.99999"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text" value="7"/>	Optional - decree amount, CFS.
On/Off:	<input type="text" value=""/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="text" value=""/>	Optional - indicate action if no match is found (default=Warn).

Command: `FillInstreamFlowRight ( ID="220831.02", AdministrationNumber=99999.99999, Decree=7, OnOff=null )`

OK Cancel

FillInstreamFlowRight

**FillInstreamFlowRight() Command Editor**

The command syntax is as follows:

```
FillInstreamFlowRight(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching instream flow right identifiers with missing name.	If not specified, the original value will remain.
StationID	The instream flow station identifier to be assigned for all matching instream flow right identifiers with missing instream flow station identifier.	If not specified, the original value will remain.
AdministrationNumber	The administration number to be assigned for all matching instream flow right identifiers with missing administration number.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching instream flow right identifiers with missing administration decree.	If not specified, the original value will remain.
OnOff	The on/off switch to be assigned for all matching instream flow right identifiers with missing on/off switch, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



# Command Reference: FillInstreamFlowStation()

## Fill instream flow station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillInstreamFlowStation()` command fills missing data in existing instream flow stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillInstreamFlowStation() Command**

This command fills missing data in instream flow station(s), using the instream flow station ID to look up the location. The instream flow station ID can contain a \* wildcard pattern to match one or more locations. Use blanks in the any field to indicate no change to the existing value.

Instream flow station ID:	<input type="text" value="*"/>	Required - instream flow stations to fill (use * for wildcard).
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
Upstream river node ID:	<input type="text"/>	Optional - upstream river node identifier.
Downstream river node ID:	<input type="text"/>	Optional - downstream river node identifier.
On/Off:	<input type="button" value="v"/>	Optional - is instream flow station on/off in dataset?
Daily ID:	<input type="text" value="0"/>	Optional - daily identifier, "ID" to match ID, or StateMod flag).
Demand type:	<input type="button" value="2 - Average monthly demand v"/>	Optional - demand time series type.
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillInstreamFlowStation ( ID="*", DailyID="0", DemandType=2 )
```

OK Cancel

FillInstreamFlowStation

**FillInstreamFlowStation() Command Editor**

The command syntax is as follows:

```
FillInstreamFlowStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching instream flow station identifiers with missing name.	If not specified, the original value will remain.
UpstreamRiverNodeID	The upstream river node identifier to be assigned for all matching instream flow station identifiers with missing river node identifier.	If not specified, the original value will remain.
DownstreamRiverNodeID	The downstream river node identifier to be assigned for all matching instream flow station identifiers with missing river node identifier.	If not specified, the original value will remain.
OnOff	The on/off switch to be assigned for all matching instream flow station identifiers with missing river node identifier, either 1 for on or 0 for off.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching stream gage identifiers with missing river node identifier	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching instream flow stations with missing demand type, one of:  1 – Monthly demand 2 – Average monthly demand	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillInstreamFlowStationsFromHydroBase()

Fill instream flow station data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillInstreamFlowStationsFromHydroBase()` command fills missing data in existing instream flow stations, using HydroBase for data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillInstreamFlowStationsFromHydroBase() Command**

This command fills missing data in instream flow stations using data from HydroBase,  
The following values from HydroBase are set if missing in a station:

Name

Station ID: \* Required - specify the stations to fill (use \* for wildcard).

If not found: [dropdown] Optional - indicate action if no match is found (default=Warn).

Command:

```
FillInstreamFlowStationsFromHydroBase ( ID="*" )
```

OK Cancel

FillInstreamFlowStationsFromHydroBase

**FillInstreamFlowStationsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillInstreamFlowStationsFromHydroBase( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: FillInstreamFlowStationsFromNetwork()

Fill instream flow station data from a StateMod network

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillInstreamFlowStationsFromNetwork()` command fills missing data in instream flow stations, using a StateMod network for data. This command is usually used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of instream flow stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillInstreamFlowStationsFromNetwork() Command**

This command fills missing data in instream flow stations using data from the network, This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

Name

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `FillInstreamFlowStationsFromNetwork ( ID="*" )`

OK Cancel

FillInstreamFlowStationsFromNetwork

**FillInstreamFlowStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
FillInstreamFlowStationsFromNetwork( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.

The following example illustrates how to fill instream flow station names from the network. A command to fill from HydroBase or another source will often be run before the second command below.

```
ReadInstreamFlowStationsFromNetwork( InputFile="sp2005.net" )  
FillInstreamFlowStationsFromNetwork( ID="*" )
```

---

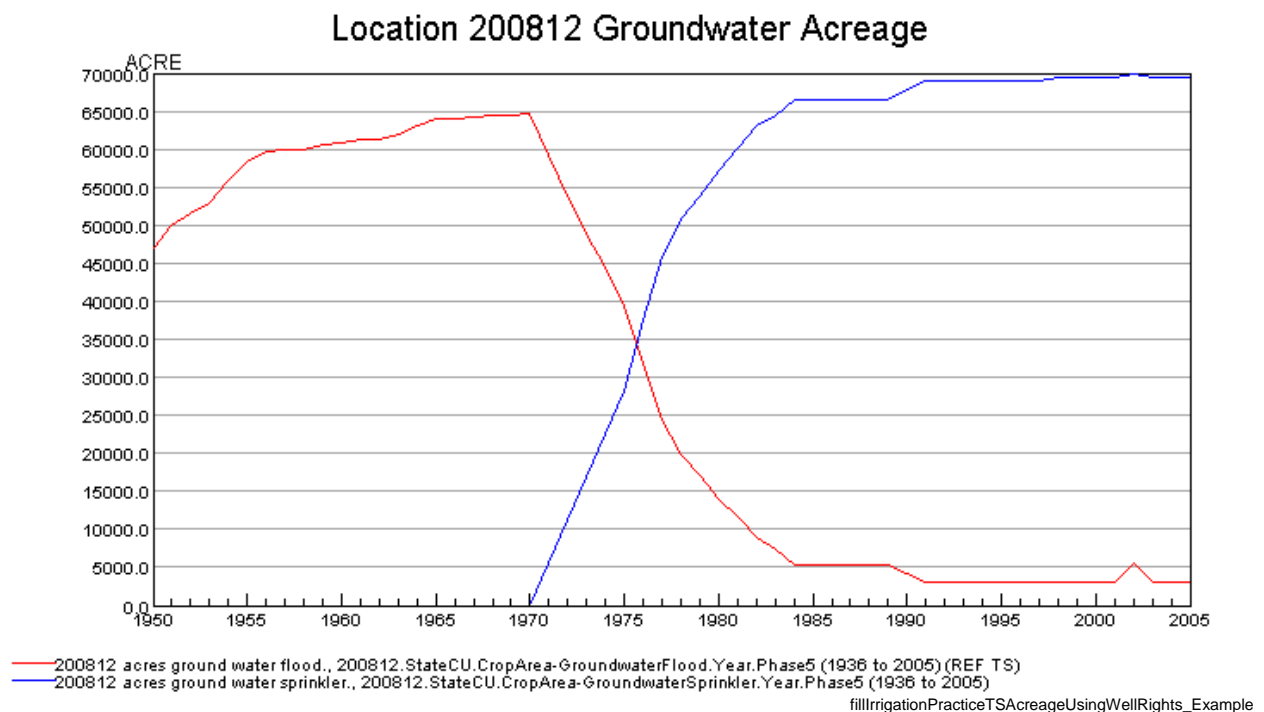
# Command Reference:

## FillIrrigationPracticeTSAcreageUsingWellRights()

**Fill missing irrigation practice time series (yearly) acreage values using well rights**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `FillIrrigationPracticeTSAcreageUsingWellRights()` fills missing irrigation practice groundwater acreage time series (yearly) information for CU locations using well rights. This command should only be used to fill data in the period before the earliest modeling year for which data are available in HydroBase and helps initialize the acreage data in the early period. For example, in the Rio Grande, 1998 parcel data and associated rights are used to fill the earlier period. The parcels associated with groundwater are turned off earlier in time, in years when no well water rights are associated with parcels. This results in the groundwater acreage decreasing back in time, as shown in the following figure (in this case there is only a slight decrease from approximately 70,000 acres to 47,000 acres, with more change being in the irrigation method):



### Prerequisites:

1. This command should be executed after the irrigation practice time series are read from HydroBase (see `ReadIrrigationPracticeTSFromHydroBase()`, which saves a list of parcels associated with each location during processing). Data for lands that are not in HydroBase should have been specified with `SetIrrigationPracticeTSFromList()` commands.

2. Total acreage has been set to the crop pattern time series total (see `SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()`). The total acres are needed for checks.
3. A non-merged, non-aggregated well water right file should have been read using the `ReadWellRightsFromStateMod()` or similar command. A StateMod well rights file with comments including parcel year and parcel are needed to ensure that rights matching the parcels for `ParcelYear` are available (see parameter description below).

The steps executed by the command are described below. Note that “CU location” refers to the StateCU model identifier.

1. For each parcel found in the water rights data, create a yearly time series of decree. The resulting time series indicates for a parcel the decreed water rights (y-axis) associated with the parcel over time (x-axis).
2. Loop through each CU location that matches the ID pattern and perform the following:
  - a. Get the list of parcels associated with the location for `ParcelYear`, taken from the irrigation practice time series. The list of parcels will have been saved when the `ReadIrrigationPracticeFromHydroBase()` command was processed.
  - b. For each year being processed, if acreage time series are missing, loop over the list of parcels for the location (note that the parcel area will be multiplied by the ditch coverage percent irrigated if the parcel is for a D&W node):
    - i. If no parcels were found for the location in the `ParcelYear`, set all groundwater acreage time series to zero. Consequently, an estimate of zero acreage will occur.
    - ii. Otherwise, set the groundwater acreage values as follows:
      - A. If the decree time series for the parcel is zero in a year, set the groundwater acreage to zero for the year.
      - B. If the parcel has groundwater supply (one or more wells in `ParcelYear`): if high efficiency irrigation method (DRIP or SPRINKLER), increment the groundwater sprinkler acreage; if low efficiency (all other irrigation methods), increment the groundwater flood acreage.
      - C. If the result is missing, set the groundwater sprinkler and flood to zero.
    - iii. Recompute the groundwater acreage by method and total. If either term is missing, set the groundwater total to missing. Otherwise, set the groundwater total to the sum of the parts.
    - iv. Recompute the surface water acreage by method and total. If either term is missing, set the surface water total to missing. Otherwise, set the surface water total to the sum of the parts.
    - v. Adjust the groundwater acres to total acres and cascade changes:
      - A. If groundwater only, set the groundwater acres to total acres if they do not already match. Else, only adjust groundwater acres down to the total (because surface water can take up the remainder).
      - B. Adjust the groundwater parts (SPRINKLER and FLOOD) to agree with the new groundwater total. If one part is zero, adjust only the non-zero part to match the total. Otherwise, prorate based on the original groundwater total and acreage split.
      - C. Adjust the surface water total and parts (SPRINKLER and FLOOD) to agree with the new surface water total. The surface water total is first set to the total acreage minus the groundwater acreage. Next adjust the parts. If one part is zero, adjust only the non-zero part to match the total. Otherwise, prorate based on the original surface water total and acreage split.



The following dialog is used to edit the command and illustrates the syntax of the command:

**Edit FillIrrigationPracticeTSAcreageUsingWellRights() Command**
✕

This command fills missing acreage data in irrigation practice time series using well water rights, which have been read with a previous command. Parcels for the specified year are used during the fill period and are turned on if corresponding well water rights were active in the year. The CU Location ID can contain a \* wildcard pattern to match one or more time series. The fill period can optionally be specified. Only years in the output period can be filled.

CU Location ID:	<input type="text" value="*"/>	Required - CU locations to process (use * for wildcard).
Parcel data year:	<input type="text" value="1998"/>	Required - 4-digit year for parcel data to turn acreage on/off.
Include surface water supply?:	<input type="text" value="True"/> ▼	Optional - include locations with surface water supply? (default=True).
Include groundwater only supply?:	<input type="text" value="True"/> ▼	Optional - include locations with only groundwater supply? (default=True).
Fill start (year):	<input type="text" value="1937"/>	Optional - start year as 4-digits (default=fill all).
Fill end (year):	<input type="text" value="1997"/>	Optional - end year as 4-digits (default=fill all).
If not found:	<input type="text" value="Warn"/> ▼	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillIrrigationPracticeTSAcreageUsingWellRights (ID="*", IncludeSurfaceWaterSupply=True, IncludeGroundwaterOnlySupply="True", FillStart=1937, FillEnd=1997, ParcelYear=1998)
```

FillIrrigationPracticeTSAcreageUsingWellRights

### FillIrrigationPracticeTSAcreageUsingWellRights() Command Editor

The command syntax is as follows:

```
FillIrrigationPracticeTSAcreageUsingWellRights(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
ParcelYear	A calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems. Only the water rights generated from parcels in this year will be used to limit groundwater acreage.	None – must be specified.
IncludeSurfaceWaterSupply	Indicate whether locations with surface water supply should be processed. Locations will only be processed if they also have groundwater supply.	True
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply should be processed.	True
FillStart	A starting year to fill data, normally the start of the output period.	The output period start.
FillEnd	An ending year to fill data, normally one year prior to the ParcelYear.	The output period end.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates the use of the command:

```
#
# Sp2008L_DDH.StateDMI
#
#
# StartLog(LogFile="SP_IPY.log")
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2006")
# Step 1 - Read CU Locations from list
#
ReadCULocationsFromList(ListFile="..\Sp2008L_StructList.csv",IDCol=1)
#
# Step 2 - Read SW aggregates, GW aggregates, and divsystems
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,
    NameCol=2,PartIDCol=3,PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,
    NameCol=2,PartIDCol=3,PartsListedHow=InRow)
```

```

#
SetWellSystemFromList(ListFile="..\SP_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDCol=2,PartsListedHow=InColumn)
#
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="")
#
# Step 4 - Set conveyance efficiencies from file for key and sw aggregate structures - NOT in HydroBase
SetIrrigationPracticeTSFromList(ListFile="Sp2008L_Eff.csv",ID="",
    SetStart=1950,SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="3")
#
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
SetIrrigationPracticeTS(ID="",SetStart=1950,SetEnd=2006,FloodAppEffMax=.6,SprinklerAppEffMax=.8,GWMode=2)
#
# Step 6 - Read well rights file and Set Max pumping (use merged *.wer file)
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
# Step 7 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="",Div="1")
#
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="Sp2008L.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="")
#
# Step 9 - Estimate 1950 ground water acreage based on active wells as defined in the non-merged *.wer file
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 10 - Fill Interpolate Acreage Type (SW and GW) 1956-2006
# Step 11a - estimate total GW and total SW
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-GroundWater",FillStart="1956",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-GroundWater",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-GroundWater",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-GroundWater",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
GroundWater",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 11b - set sprinkler to zero in early period
SetIrrigationPracticeTS(ID="",SetStart=1950,SetEnd=1969,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 11c - fill remaining irrigation method values
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="",DataType="CropArea-
GroundWaterSprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="",DataType="CropArea-

```

```

GroundWaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 12 - Set Acreage = 0 for structures that are in diversion systems, so acreage is not double accounted
SetIrrigationPracticeTS(ID="0100503_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100507_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100687",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="0200834",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400511_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 13 - Set Acreage = 0, 1950-2006
SetIrrigationPracticeTS(ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100513",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100829",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400519",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 14 - Write final ipy file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateMod\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)

```

---

# Command Reference:

## FillIrrigationPracticeTSInterpolate()

Fill irrigation practice time series values using interpolation

**StateCU Command**

Version 3.09.01, 2010-02-01

The `FillIrrigationPracticeTSInterpolate()` command fills irrigation practice time series data for a CU Location, using interpolation. Data will not be extrapolated past the end-points and therefore another fill method (e.g., `FillIrrigationPracticeTSRepeat()`) may be required after the interpolation command. Filling is currently always in a forward direction. Setting acreage values results in a cascade of adjustments to maintain sums, and will be noted in the log file. Preference is given to maintaining the total acreage, then groundwater acreage, and then surface water acreage. Irrigation method within groundwater will agree with the total and the sprinkler and flood acreage will be prorated based on previous values if necessary to adjust to the total. Similar adjustments are made to surface water acreage.

If applied to acreage, it is typical to first fill the groundwater acreage separately and then use this command to interpolate the surface water acreage by interpolating between years with observations.

Prerequisites for acreage filling:

1. This command should be executed after the irrigation practice time series are read from HydroBase (see `ReadIrrigationPracticeTSFromHydroBase()`).
2. Total acreage has been set to the crop pattern time series total (see `SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()`). The total acres are needed for checks.
3. The groundwater acreage should also have been filled using well rights before the first year of observations using `FillIrrigationPracticeTSAcreageUsingWellRights()`.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillIrrigationPracticeTSInterpolate() Command**

This command fills missing data in irrigation practice time series, using the CU Location ID, time series data type, and year to uniquely identify time series. Missing values are replaced by interpolating between known values.  
 Max Intervals indicates the maximum number of intervals to fill in a gap.  
 The CU Location ID can contain a \* wildcard pattern to match one or more time series.  
 The fill period can optionally be specified. Only years in the output period can be filled.

CU Location ID:  Required - CU Location(s) to fill (use \* for wildcard).

Data Type:  Optional - data type to fill (blank for all).

Fill start (year):  Optional - start year as 4-digits (default=fill all).

Fill end (year):  Optional - end year as 4-digits or blank to fill all (default=fill all).

Max Intervals:  Optional - max years to fill (default=all).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
FillIrrigationPracticeTSInterpolate (ID="*",DataType="CropArea-GroundWater",FillStart="1956",FillEnd="1976")
```

OK Cancel

FillIrrigationPracticeTSInterpolate

### FillIrrigationPracticeTSInterpolate() Command Editor

Currently, the following check is always done after interpolation on any acreage data to adjust acreage parts to the total acreage:

1. If any of the acreage terms (surface water flood, surface water sprinkler, groundwater flood, groundwater sprinkler) is missing, print a warning. This should not occur if valid `FillStart` and `FillEnd` are specified, with observations at endpoints.
2. Compute the “target” surface water acreage as the total acreage minus the groundwater terms (groundwater flood + sprinkler). Compute the actual surface water (from current in-memory data) as the total of surface water flood and sprinkler acreage.
  - a. If the target is less than zero, the groundwater acres are greater than the total. Adjust the groundwater acreage to the total, maintaining the ratio of flood and sprinkler acres to the total as with the previous groundwater total. Recompute the target surface water total acres (will be zero).
  - b. If the surface water actual is zero and its terms cannot be adjusted to the target, attempt to adjust the groundwater acreage to make up the difference.
    - i. If the groundwater acreage terms are zero, print a warning – no adjustment is possible. The user will need to take action.
    - ii. Else, adjust the groundwater to equal the total, maintaining the ratio of flood and sprinkler acres to the total as with the previous groundwater total.
  - c. Else if the surface water actual is not zero, adjust the surface water total to match the target, maintaining a ratio of surface water flood and sprinkler consistent with the previous values. This may result in the surface water terms being set to zero.

The command syntax is as follows:

```
FillIrrigationPracticeTSInterpolate(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DataType	A single data type or blank for all data types (see command editor for choices).	If not specified, fill all data types.
FillStart	The first year to fill. Include an endpoint with observations because they will be needed for interpolation.	If not specified, fill the full period.
FillEnd	The last year to fill. Include an endpoint with observations because they will be needed for interpolation.	If not specified, fill the full period.
MaxIntervals	The maximum number of intervals to fill in any gap.	If not specified, fill the entire gap.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how to process the irrigation practice time series file where groundwater supply is used:

```
#
# Sp2008L_DDH.StateDMI
#
# StartLog(LogFile="SP_IPY.log")
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2006")
# Step 1 - Read CU Locations from list
#
ReadCULocationsFromList(ListFile="..\Sp2008L_StructList.csv",IDCol=1)
#
# Step 2 - Read SW aggregates, GW aggregates, and divsystems
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,
    NameCol=2,PartIDSCol=3,PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,
    NameCol=2,PartIDSCol=3,PartsListedHow=InRow)
#
SetWellSystemFromList(ListFile="..\SP_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDSCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDSCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDSCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDSCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2005.csv",Year=2005,Div=1,
```

```

PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumnn)
#
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
#
# Step 4 - Set conveyance efficiencies from file for key and sw aggregate structures - NOT in HydroBase
SetIrrigationPracticeTSFromList(ListFile="Sp2008L_Eff.csv",ID="*",
    SetStart=1950,SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="3")
#
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=2006,FloodAppEffMax=.6,SprinklerAppEffMax=.8,GWmode=2)
#
# Step 6 - Read well rights file and Set Max pumping (use merged *.wer file)
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
# Step 7 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Div="1")
#
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="Sp2008L.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
#
# Step 9 - Estimate 1950 ground water acreage based on active wells as defined in the non-merged *.wer
file
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 10 - Fill Interpolate Acreage Type (SW and GW) 1956-2006
# Step 11a - estimate total GW and total SW
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1956",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 11b - set sprinkler to zero in early period
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=1969,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 11c - fill remaining irrigation method values
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 12 - Set Acreage = 0 for structures that are in diversion systems, so acreage is not double
accounted
SetIrrigationPracticeTS(ID="0100503_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,

```



```

    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100507_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100687",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="0200834",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400511_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 13 - Set Acreage = 0, 1950-2006
SetIrrigationPracticeTS(ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100513",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100829",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400519",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 14 - Write final ipy file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateMod\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)

```

This page is intentionally blank.

---

# Command Reference: FillIrrigationPracticeTSRepeat()

Fill irrigation practice time series values by repeating values

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `FillIrrigationPracticeTSRepeat()` command fills irrigation practice time series data for a CU Location, by repeating known values. Filling can occur forward or backward in time, but not both. Therefore, it may be necessary to use two similar commands, one filling forward, and one filling backward, in order to completely fill the ends of time series. Setting acreage values results in a cascade of adjustments to maintain sums, and will be noted in the log file. Preference is given to maintaining the total acreage, then groundwater acreage, and then surface water acreage. Irrigation method within groundwater will agree with the total and the sprinkler and flood acreage will be prorated based on previous values if necessary to adjust to the total. Similar adjustments are made to surface water acreage.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillIrrigationPracticeTSRepeat() Command**

This command fills missing data in irrigation practice time series, using the CU Location ID, time series data type, and year to uniquely identify time series. Missing values are replaced by repeating known values.  
Max Intervals indicates the maximum number of intervals to fill in a gap.  
The CU Location ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only years in the output period can be filled.

CU Location ID:  Required - CU Location(s) to fill (use \* for wildcard).

Data Type:  Optional - data type to fill (blank for all).

Fill start (year):  Optional - start year as 4-digits (default=fill all).

Fill end (year):  Optional - end year as 4-digits or blank to fill all (default=fill all).

Fill Direction:  Optional - direction to process data (default=Forward).

Max Intervals:  Optional - max years to fill (default=all).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillIrrigationPracticeTSRepeat (ID="*",DataType="CropArea-Ground WaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection=" Forward")`

OK Cancel

FillIrrigationPracticeTSRepeat

**FillIrrigationPracticeTSRepeat() Command Editor**

Currently, the following check is always done after filling on any acreage data to adjust acreage parts to the total acreage:

1. If any of the acreage terms (surface water flood, surface water sprinkler, groundwater flood, groundwater sprinkler) is missing, print a warning. This should not occur if valid `FillStart` and `FillEnd` are specified, with observations at endpoints.
2. Compute the “target” surface water acreage as the total acreage minus the groundwater terms (groundwater flood + sprinkler). Compute the actual surface water (from current in-memory data) as the total of surface water flood and sprinkler acreage.
  - a. If the target is less than zero, the groundwater acres are greater than the total. Adjust the groundwater acreage to the total, maintaining the ratio of flood and sprinkler acres to the total as with the previous groundwater total. Recompute the target surface water total acres (will be zero).
  - b. If the surface water actual is zero and its terms cannot be adjusted to the target, attempt to adjust the groundwater acreage to make up the difference.
    - i. If the groundwater acreage terms are zero, print a warning – no adjustment is possible. The user will need to take action.
    - ii. Else, adjust the groundwater to equal the total, maintaining the ratio of flood and sprinkler acres to the total as with the previous groundwater total.
  - c. Else if the surface water actual is not zero, adjust the surface water total to match the target, maintaining a ratio of surface water flood and sprinkler consistent with the previous values. This may result in the surface water terms being set to zero.

The command syntax is as follows:

```
FillIrrigationPracticeTSRepeat(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DataType	A single data type, CropArea-AllSurfaceAcreageParts (for surface water sprinkler and surface water flood), or blank for all data types.	If not specified, fill all data types.
FillStart	The first year to fill. Specify as a year with complete data if filling forward.	If not specified, fill the full period.
FillEnd	The last year to fill. Specify as a year with complete data if filling backward.	If not specified, fill the full period.
FillDirection	The direction to fill, either Forward or Backward.	Forward
MaxIntervals	The maximum number of intervals to fill in any gap.	If not specified, fill the entire gap.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

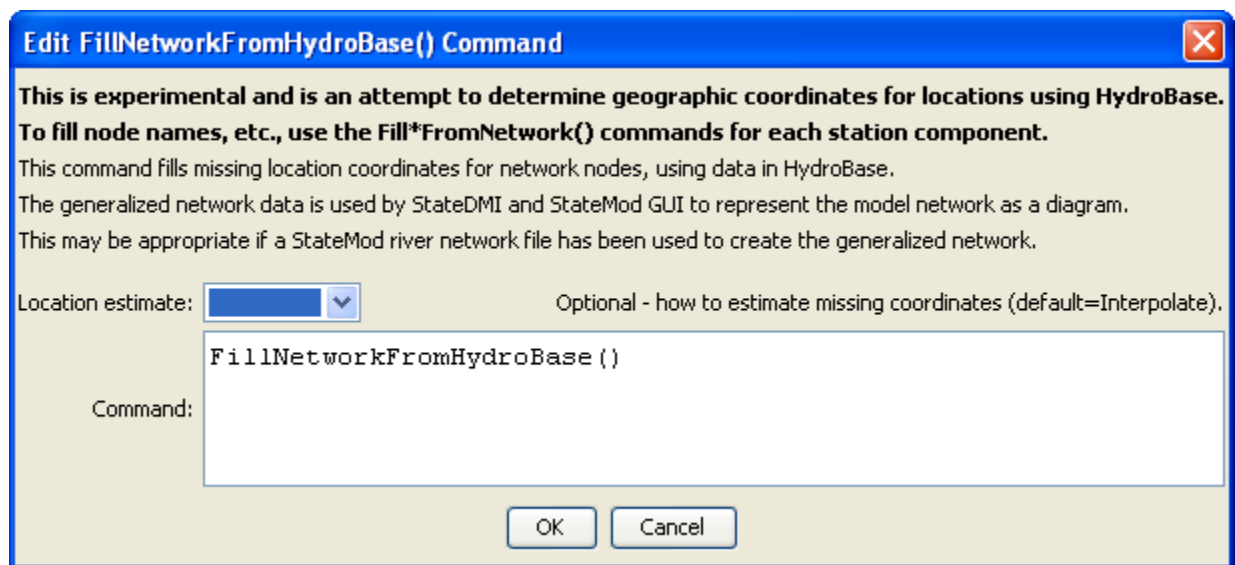
# Command Reference: FillNetworkFromHydroBase()

Fill generalized network data from HydroBase

StateMod Command

Version 3.09.01, 2010-02-01

The `FillNetworkFromHydroBase()` command fills missing location data in the generalized network, using HydroBase for data. This is used, for example, when a generalized network has been created from a StateMod river network. The following dialog is used to edit the command and illustrates the syntax of the command.



FillNetworkFromHydroBase

FillNetworkFromHydroBase() Command Editor

The command syntax is as follows:

```
FillNetworkFromHydroBase( Parameter=Value, ... )
```

### Command Parameters

Parameter	Description	Default
LocationEstimate	Indicates how to estimate missing coordinates, currently only: <ul style="list-style-type: none"> <li>Interpolate – linearly interpolate between known node locations.</li> </ul>	Interpolate

The following example command file illustrates how the command might be used:

```
# Create a generalized XML network from individual StateMod files
# Read the network, which contains upstream to downstream connectivity but does
# not indicate node types
ReadRiverNetworkFromStateMod(InputFile=cm2005.rin)
# Read the stations, which imply the node types
ReadRiverStreamGageStationsFromStateMod(InputFile=cm2005.ris)
ReadRiverDiversionStationsFromStateMod(InputFile=cm2005.dds)
ReadRiverReservoirStationsFromStateMod(InputFile=cm2005.res)
ReadRiverInstreamFlowStationsFromStateMod(InputFile=cm2005.ifs)
ReadRiverWellStationsFromStateMod(InputFile=cm2005.wes)
# To be developed...
#ReadRiverPlanStationsFromStateMod()
ReadRiverStreamEstimateStationsFromStateMod(InputFile=cm2005.ris)
# Now create the generalized network, using the connectivity and node types
CreateNetworkFromRiverNetwork()
# Fill in node names and locations from HydroBase, if any is still missing
FillNetworkFromHydroBase()
# Write the generalized network
WriteNetworkToStateMod(OutputFile="cm2005.net")
# Check for errors (the following is not yet implemented)
#CheckNetwork()
WriteCheckFile(OutputFile="cm2005.net.check.html")
```

# Command Reference: FillReservoirRight()

## Fill reservoir right data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillReservoirRight()` command fills missing data in existing reservoir rights. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit FillReservoirRight() Command

This command fills missing data in reservoir right(s), using the reservoir right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="CostRes.06"/>	Required - specify the right(s) to fill (use * for wildcard)
Name:	<input type="text" value="CO_CostR6"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text" value="CostRes"/>	Optional - station identifier.
Administration number:	<input type="text" value="1900.00000"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text" value="430"/>	Optional - decree amount, AF.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
Account served by right:	<input type="button"/>	Optional - account(s) served by right.
Right type:	<input type="button" value="1 - Standard"/>	Optional - right type.
Fill type:	<input type="button" value="1 - First fill"/>	Optional - fill type.
Operational rightID:	<input type="text"/>	Optional - specify only if right type is -1
If not found:	<input type="button"/>	Optional - indicate action if no match is found (default=Warn).
Command:	<pre>FillReservoirRight (ID="CostRes.06",Name="CO_CostR6",StationID="CostRes",AdministrationNumber=1900.00000,Decree=430,OnOff=1,AccountDist="null",RightType=1,FillType=1)</pre>	

FillReservoirRight

FillReservoirRight() Command Editor

The command syntax is as follows:

```
FillReservoirRight (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single reservoir right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching reservoir right identifiers with missing name.	If not specified, the original value will remain.
StationID	The reservoir station identifier to be assigned for all matching reservoir right identifiers with missing reservoir station identifier.	If not specified, the original value will remain.
Administration Number	The administration number to be assigned for all matching reservoir right identifiers with missing administration number.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching reservoir right identifiers with missing administration decree.	If not specified, the original value will remain.
OnOff	The on/off switch to be assigned for all matching reservoir right identifiers with missing on/off switch, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
AccountDist	The account distribution option to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
RightType	The reservoir right type to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
FillType	The reservoir right fill type to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
OpRightID	The out-of-priority associated operational right (see StateMod documentation).	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference: FillReservoirStation()

**Fill reservoir station data**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillReservoirStation()` command fills missing data in existing reservoir stations.

Currently, accounts cannot be filled, and if specified with this command, are set as if the `SetReservoirStation()` command is being used.

The following dialog is used to edit the command and illustrates the syntax of the command.

Edit FillReservoirStation() Command	
<p>This command fills missing data in reservoir station(s), using the reservoir station ID to look up the location.            The reservoir station ID can contain a * wildcard pattern to match one or more locations.            Use blanks in the any field to indicate no change to the existing value.            Only one account can be edited with each command (use additional commands for account 2+).</p>	
Reservoir station ID: <input type="text"/>	Required - reserivior stations to fill (use * for wildcard).
Name: <input type="text"/>	Optional - up to 24 characters for StateMod.
River node ID: <input type="text"/>	Optional - river node identifier or "ID".
On/Off: <input type="button" value="1 - On, do not store above reservoir targets"/>	Optional - is reservoir station on/off in data set?
One fill rule: <input type="button" value="1 - January"/>	Optional - one fill rule administration switch.
Daily ID: <input type="text"/>	Optional - daily identifier, "ID", or StateMod flag).
Content (minimum): <input type="text"/>	Optional - minimum content, AF.
Content (maximum): <input type="text"/>	Optional - maximum content, AF.
Release (maximum): <input type="text"/>	Optional - maximum release, CFS.
Dead storage: <input type="text"/>	Optional - dead storage, AF.
<b>Accounts (must use one command for each account - use multiple commands with only this section complete if necessary)</b>	
Account ID: <input type="text"/>	Required if setting account - account number (position in data), 1+.
Account name: <input type="text"/>	Optional - account name.
Maximum storage: <input type="text"/>	Optional - account maximum storage (ACFT).
Initial storage: <input type="text"/>	Optional - account initial storage (ACFT).
Evaporation distribution: <input type="button" value=""/>	Optional - evaporation distribution option.
One fill rule calculation: <input type="button" value=""/>	Optional - ownership for one fill rule calculation.
Evaporation stations: <input type="text"/>	Optional - ID1,%; ID2,%...
Precipitation stations: <input type="text"/>	Optional - ID1,%; ID2,%...
Content/Area/Seepage: <input type="text"/>	Optional - use format: Content,Area,Seepage  (one record per line)
If not found: <input type="button" value=""/>	Optional - indicate action if no match is found (default=Warn).
<b>Command:</b> <pre>FillReservoirStation(ID="*",RiverNodeID="ID",OnOff=1,OneFillRule=1,ContentMin=0,ReleaseMax=999999)</pre>	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

### FillReservoirStation() Command Editor

FillReservoirStation

The command syntax is as follows:

```
FillReservoirStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching reservoir stations with missing name.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching reservoir stations with missing river node ID. Specify ID to assign to the reservoir station ID.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching reservoir stations with missing OnOff, either 1 for on or 0 for off.	If not specified, the original value will remain.
OneFillRule	The date for one fill rule administration (see the StateMod documentation) to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
ContentMin	The reservoir minimum content, ACFT to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
ContentMax	The reservoir maximum content, ACFT to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
ReleaseMax	The reservoir maximum release, CFS to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
DeadStorage	The reservoir dead storage, ACFT to be assigned for all matching reservoir stations with missing value.	If not specified, the original value will remain.
AccountID	A reservoir account identifier, a number 1+. Reservoir accounts in the StateMod reservoir station are identified only by the account name. This AccountID lets the software know the order of the accounts. If the AccountID is specified as 1, all the accounts are deleted and a new list of accounts is started. Therefore, specify account information in sequential order.	Must be specified when providing account information.
AccountName	A reservoir account name.	If not specified, the original value will remain.
AccountMax	The account maximum content, ACFT.	If not specified, the original value will

Parameter	Description	Default
		remain.
AccountInitial	The account initial content, ACFT.	If not specified, the original value will remain.
AccountEvap	The account evaporation distribution – see the StateMod documentation.	If not specified, the original value will remain.
AccountOneFill	The account information for one fill calculations – see the StateMod documentation.	If not specified, the original value will remain.
EvapStations	A list of evaporation stations and weights (%) for the reservoir station, using the format: ID , %; ID , %.	If not specified, the original value will remain.
PrecipStations	A list of precipitation stations and weights (%) for the reservoir station, using the format: ID , %; ID , %.	If not specified, the original value will remain.
ContentAreaSeepage	Content/area/seepage values, using the format: Content , Area , Seepage; Content , Area , Seepage.	If not specified, the original value will remain.

---

# Command Reference: FillReservoirStationsFromNetwork()

Fill reservoir station data from a StateMod network

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillReservoirStationsFromNetwork()` command fills missing data in reservoir stations, using a StateMod network for data. This command usually is used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of reservoir stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillReservoirStationsFromNetwork() Command**

This command fills missing data in reservoir stations using data from the network, matching the reservoir station identifiers. This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

Name

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillReservoirStationsFromNetwork ( ID="*" )
```

FillReservoirStationsFromNetwork

**FillReservoirStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
FillReservoirStationsFromNetwork( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single reservoir station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

The following example illustrates how to fill reservoir station names from the network. A command to fill from HydroBase or another source will often be run before the second command below.

```
ReadReservoirStationsFromList( InputFile="sp2005.csv" )  
FillReservoirStationsFromNetwork( ID="*" )
```

---

# Command Reference: FillReservoirStationsFromHydroBase()

Fill reservoir station data from HydroBase

## StateMod Command

Version 3.14.00, 2004-09-14, Color, Acrobat Distiller

The `FillReservoirStationsFromHydroBase()` command fills missing data in existing reservoir stations, using HydroBase for data.

If not in HydroBase, a bounding zero value point is inserted as the first record in the curve. A maximum bound of content 9999999 is also added, using the area and seepage of the last record from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillReservoirStationsFromHydroBase() Command**

This command fills missing data in reservoir stations using data from HydroBase, matching the reservoir station identifiers. The following values from HydroBase are set if missing in a station:

Name  
Content/area/seepage

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillReservoirStationsFromHydroBase ( ID="*" )`

OK Cancel

FillReservoirStationsFromHydroBase

**FillReservoirStationsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillReservoirStationsFromHydroBase( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single reservoir station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn



---

# Command Reference: FillRiverNetworkFromHydroBase()

Fill StateMod river network data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillRiverNetworkFromHydroBase()` command fills missing data in the StateMod river network, using HydroBase for data. This is used, for example, when the river network has been created from the generalized network and “official” node names are to be determined from HydroBase. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillRiverNetworkFromHydroBase() Command**

This command fills missing data in river network data by using data from HydroBase, matching the station identifiers. The following values from HydroBase are set if missing in a station:

- Name - pick a format to use:
  - StationName - 24 characters.
  - StationName\_NodeType - 20 characters + "\_FLO".

Station ID:  Required - specify the stations to fill (use \* for wildcard).

Name format:  Optional - pick the format for the name (default=StationName).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillRiverNetworkFromHydroBase ( ID="*", NameFormat=StationName_NodeType )`

OK Cancel

FillRiverNetworkFromHydroBase

**FillRiverNetworkFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillRiverNetworkFromHydroBase ( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
ID	A single river station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of: <ul style="list-style-type: none"> <li>StationName – use the station name from HydroBase</li> <li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + “_” + the node type (e.g., “ABC DITCH_DIV”).</li> </ul>	If blank, the original values will remain unchanged.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how a StateMod river network file can be created from the generalized network file:

```

StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#           names from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",
    CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")

```

---

# Command Reference: FillRiverNetworkFromNetwork()

**Fill StateMod river network data from the generalized network**

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillRiverNetworkFromNetwork()` command fills missing data in the StateMod river network, using the generalized network for data. This is used, for example, when the river network has been created from the generalized network and “official” node names are first filled from HydroBase. Any remaining missing names can then be filled from the generalized network, using labels for the diagram. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillRiverNetworkFromNetwork() Command**

This command fills in missing data in river network data by using data from the network, matching the node identifiers. This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

Name - pick a format to use:  
StationName - 24 characters.  
StationName\_NodeType - 20 characters + "\_FLO".

Station ID:  Required - specify the stations to fill (use \* for wildcard).

Name format:  Optional - format for the name (default=StationName).

Comment format:  Optional - format for the comment (default=StationID).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillRiverNetworkFromNetwork ( ID="*", NameFormat="StationName_NodeType", CommentFormat="StationID" )`

OK Cancel

FillRiverNetworkFromNetwork

### FillRiverNetworkFromNetwork() Command Editor

The command syntax is as follows:

```
FillRiverNetworkFromNetwork ( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single river station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of:	StationName

Parameter	Description	Default
	<ul style="list-style-type: none"> <li>StationName – use the station name from HydroBase</li> <li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + “_” + the node type.</li> </ul>	
CommentFormat	<p>The format to use for the river station comment, currently only:</p> <ul style="list-style-type: none"> <li>StationID – the river station identifier.</li> </ul>	If not specified, the original data will remain unchanged.
IfNotFound	<p>Used for error handling, one of the following:</p> <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how a StateMod river network file can be created from the generalized network file:

```

StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#          names from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",
    CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")

```

# Command Reference: FillRiverNetworkNode()

## Fill river network node data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillRiverNetworkNode()` command fills missing values in existing river network nodes. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillRiverNetworkNode() Command**

This command fills missing data in a river network node, using the river network node ID to look up the location. The river network node ID can contain a \* wildcard pattern to match one or more locations. Use blanks in the any field to indicate no change to the existing value.

ID:	<input type="text" value="*"/>	Required - river network node(s) to fill (use * for wildcard).
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
Downstream river node ID:	<input type="text"/>	Optional - up to 12 characters.
Maximum recharge limit:	<input type="text" value="0"/>	Optional - maximum recharge limit (CFS) if modeling groundwater.
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn)

Command:

```
FillRiverNetworkNode (ID="*", MaxRechargeLimit=0)
```

OK Cancel

FillRiverNetworkNode

**FillRiverNetworkNode() Command Editor**

The command syntax is as follows:

```
FillRiverNetworkNode (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single river network node identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching river network nodes.	If not specified, missing values will not be filled.
DownstreamRiverNodeID	The downstream river node identifier to be assigned for all matching river network nodes.	If not specified, missing values will not be filled.
MaxRechargeLimit	The maximum recharge limit, CFS, for groundwater modeling, assigned for all matching river network nodes.	If not specified, missing values will not be filled.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillStreamEstimateStation()

## Fill stream estimate station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillStreamEstimateStation()` command fills missing data in existing stream estimate stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamEstimateStation() Command**

This command fills missing data in stream estimate stations, using the station ID to look up the location. The station ID can contain a \* wildcard pattern to match one or more locations. Use blanks in the any field to indicate no change to the existing value.

Station ID:  Required - specify the station(s) to fill (use \* for wildcard)

Name:  Optional - up to 24 characters for StateMod.

River node ID:  Optional.

Daily ID:  Optional - corresponding daily data ID.

If not found:  Optional - indicate action if no match is found.

Command: 

```
FillStreamEstimateStation ( ID="24*", DailyID="0" )
```

FillStreamEstimateStation

### FillStreamEstimateStation() Command Editor

The command syntax is as follows:

```
FillStreamEstimateStation(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single stream estimate identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
RiverNodeID	The river node identifier to be assigned for all matching stream estimate identifiers with missing river node identifier	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching stream estimate identifiers with missing river node identifier	If not specified, the original value will remain.
Name	The name to be assigned for all matching stream estimate identifiers with missing name.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference: FillStreamEstimateStationsFromHydroBase()

Fill stream estimate station data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillStreamEstimateStationsFromHydroBase()` command fills missing data in existing stream estimate stations, using HydroBase for data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamEstimateStationsFromHydroBase() Command**

This command fills missing data in stream estimate stations by using data from HydroBase, matching the station identifiers. Stream estimate stations are locations other than stream gages (diversions, reservoirs, etc.) where flow is estimated. The following values from HydroBase are set if missing in a station:

Name - pick a format to use:  
StationName - 24 characters.  
StationName\_NodeType - 20 characters + "\_FLO".

Station ID:  Required - specify the stations to fill (use \* for wildcard).

Name format:  Optional - pick the format for the name (default=StationName).

Check structures?: ☒ Optional - check structures in addition to stations (default=False).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillStreamEstimateStationsFromHydroBase ( ID="*", NameFormat=StationName, CheckStructures=True )`

OK Cancel

FillStreamEstimateStationsFromHydroBase

**FillStreamEstimateStationsFromHydroBase() Command Editor**

The command syntax is as follows:

```
FillStreamEstimateStationsFromHydroBase(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single stream estimate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of: <ul style="list-style-type: none"> <li>StationName – use the station name from HydroBase</li> <li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + “_” + the node type.</li> </ul>	StationName
Check Structures	The old convention in StateMod was to combine stream gage and stream estimate stations in the stream gage station file. A new convention that is being evaluated is to have separate stream gage and estimate station files. Because stream estimate stations are often at HydroBase structures, filling names requires checking HydroBase structures. Since this step is not needed in the new convention, it is included as an option. Specify True to check structures when filling data from HydroBase.	False
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillStreamEstimateStationsFromNetwork()

Fill stream estimate station data from a StateMod network

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillStreamEstimateStationsFromNetwork()` command fills missing data in existing stream estimate stations, using a StateMod network for data. This command is usually used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of stream gage stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamEstimateStationsFromNetwork() Command**

This command fills in missing data in stream estimate stations by using data from the network, matching the station identifiers. Stream estimate stations are locations where flow is estimated (not historically measured). This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

- Name - pick a format to use:
  - StationName - 24 characters.
  - StationName\_NodeType - 20 characters + "\_FLO".

Station ID:  Required - specify the stations to fill (use \* for wildcard).

Name format:  Optional - format for the name (default=StationName).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillStreamEstimateStationsFromNetwork (ID="*",NameFormat="StationName")`

OK Cancel

FillStreamEstimateStationsFromNetwork() Command Editor

The command syntax is as follows:

```
FillStreamEstimateStationsFromNetwork(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single stream estimate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of: <ul style="list-style-type: none"> <li>StationName – use the station name from HydroBase</li> <li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + “_” + the node type.</li> </ul>	StationName
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillStreamGageStation()

## Fill stream gage station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillStreamGageStation()` command fills missing data in existing stream gage stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamGageStation() Command**

This command fills missing data in stream gage stations, using the station ID to look up the location. The station ID can contain a \* wildcard pattern to match one or more locations. Use blanks in the any field to indicate no change to the existing value.

Station ID:  Required - specify the station(s) to fill (use \* for wildcard)

Name:  Optional - up to 24 characters for StateMod.

River node ID:  Optional.

Daily ID:  Optional - corresponding daily data ID.

If not found:  Optional - indicate action if no match is found.

Command: 

```
FillStreamGageStation (ID="24*", DailyID="0")
```

OK Cancel

FillStreamGageStation

**FillStreamGageStation() Command Editor**

The command syntax is as follows:

```
FillStreamGageStation( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single stream gage identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching stream gage identifiers with missing name.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching stream gage identifiers with missing river node identifier	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching stream gage identifiers with missing river node identifier	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: FillStreamGageStationsFromHydroBase()

Fill stream gage station data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillStreamGageStationsFromHydroBase()` command fills missing data in existing stream gage stations, using HydroBase for data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamGageStationsFromHydroBase() Command**

This command fills missing data in stream gage stations by using data from HydroBase, matching the station identifiers. The following values from HydroBase are set if missing in a station:

Name - pick a format to use:  
StationName - 24 characters.  
StationName\_NodeType - 20 characters + "\_FLO".

Station ID: \* Required - specify the stations to fill (use \* for wildcard).

Name format: StationName Optional - pick the format for the name (default=StationName).

Check structures?: True Optional - check structures in addition to stations (default=False).

If not found: [Warning Icon] Optional - indicate action if no match is found (default=Warn).

Command: FillStreamGageStationsFromHydroBase ( ID= "\*", NameFormat=StationName, CheckStructures=True)

OK Cancel

FillStreamGageStationsFromHydroBase

## FillStreamGageStationsFromHydroBase() Command Editor

The command syntax is as follows:

```
FillStreamGageStationsFromHydroBase (param=value,param=value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single stream gage station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of: <ul style="list-style-type: none"><li>StationName – use the station name from HydroBase</li><li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + "_" + the node type.</li></ul>	StationName

Parameter	Description	Default
Check Structures	The old convention in StateMod was to combine stream gage and stream estimate stations in the stream gage station file. A new convention that is being evaluated is to have separate stream gage and estimate station files. Because stream estimate stations are often at HydroBase structures, filling names requires checking HydroBase structures. Since this step is not needed in the new convention, it is included as an option. Specify True to check structures when filling data from HydroBase.	False
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example command file illustrates the commands used to read stream gage stations from the network and create a StateMod file:

```

StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase,
#           fill in missing names from the network file
#
FillStreamGageStationsFromHydroBase(ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
...many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")

```



---

# Command Reference: FillStreamGageStationsFromNetwork()

Fill stream gage station data from a StateMod network

## StateMod Command

Version 3.09.01, 2010-02-01

The `FillStreamGageStationsFromNetwork()` command fills missing data in stream gage stations, using a StateMod network for data. This command is usually used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of stream gage stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillStreamGageStationsFromNetwork() Command**

This command fills in missing data in stream gage stations by using data from the network, matching the station identifiers. This command is useful if names for stations cannot be filled from a database or other source.

The following values from the network are set if missing in a station:

- Name - pick a format to use:
  - StationName - 24 characters.
  - StationName\_NodeType - 20 characters + "\_FLO".

Station ID:  Required - specify the stations to fill (use \* for wildcard).

Name format:  Optional - format for the name (default=StationName).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillStreamGageStationsFromNetwork (ID="*", NameFormat="StationName")`

OK Cancel

FillStreamGageStationsFromNetwork

### FillStreamGageStationsFromNetwork() Command Editor

The command syntax is as follows:

```
FillStreamGageStationsFromNetwork (Parameter=Value, ...)
```

### Command Parameters

Parameter	Description	Default
ID	A single stream gage station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
NameFormat	The format to use when setting the name, one of: <ul style="list-style-type: none"> <li>StationName – use the station name from HydroBase</li> <li>StationName_NodeType – use the first 20 characters of the name from Hydrobase + “_” + the node type.</li> </ul>	StationName
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don’t add and don’t generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example command file illustrates the commands used to read stream gage stations from the network and create a StateMod file:

```

StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase,
#           fill in missing names from the network file
#
FillStreamGageStationsFromHydroBase(ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
...many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")

```

---

# Command Reference: FillWellDemandTSMonthlyAverage()

**Fill well demand time series (monthly) values using average monthly values**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `FillWellDemandTSMonthlyAverage()` command fills missing well demand time series (monthly) data, using average monthly values. The averages are computed immediately after reading time series (e.g., from HydroBase or a file) or calculation of the time series (e.g., from  $IWR/Eff_{ave}$ ). The average values that are used during data filling are printed to the log file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellDemandTSMonthlyAverage() Command**

This command fills missing data in monthly well demand time series.  
Missing values are replaced with monthly average - average values are computed immediately after reading/calculating the data.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:  Specify the well stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillWellDemandTSMonthlyAverage (ID="**")
```

FillWellDemandTSMonthlyAverage

**FillWellDemandTSMonthlyAverage() Command Editor**

The command syntax is as follows:

```
FillWellDemandTSMonthlyAverage(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## FillWellDemandTSMonthlyConstant()

**Fill well demand time series (monthly) values using a constant value**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `FillWellDemandTSMonthlyConstant()` command fills missing well demand time series (monthly) data, using a constant value. This command is useful, for example, to set demand values to zero if other fill commands are unable to provide data estimates for missing data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellDemandTSMonthlyConstant() Command**

This command fills missing data in monthly well demand time series.  
Missing values are replaced with a constant value.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:  Specify the well stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Constant:  Required - constant value to use for filling.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillWellDemandTSMonthlyConstant ( ID="*", Constant=0)
```

FillWellDemandTSMonthlyConstant

**FillWellDemandTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
FillWellDemandTSMonthlyConstant (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
Constant	The constant value to be used to fill missing data.	None – must be specified.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillWellDemandTSMonthlyPattern()

**Fill well demand time series (monthly) values using WET/DRY/AVG values**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `FillWellDemandTSMonthlyPattern()` command fills missing well demand time series (monthly) data, using average monthly wet/dry/average values. The averages are computed using patterns read by the `ReadPatternFile()` command. The average values that are used during data filling are printed to the log file. For example, if a value is missing for May 1980, the pattern for the specified pattern identifier is checked for WET, DRY, or AVG. The values of all May's for WET, DRY, or AVG are then averaged in the time series to be filled, and the resulting average used to fill missing data. This command therefore will result in filled values that are more appropriate than simple averages; however, work must be done to characterize the wet, dry, and average months.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellDemandTSMonthlyPattern() Command**

This command fills missing data in monthly well demand time series.  
Missing values are replaced with monthly average - average values are computed by this command using the pattern data.  
One or more ReadPatternFile() commands must be used before this command.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:	<input type="text" value="90*"/>	Specify the well stations to fill (use * for wildcard)
Fill start:	<input type="text"/>	Optional - start date or blank to fill all.
Fill end:	<input type="text"/>	Optional - end date or blank to fill all.
Pattern identifier:	<input type="text" value="08220000"/>	Required - pattern ID to use for filling.
<= zero values in average?:	<input type="button" value="v"/>	Optional - are values <= zero used in averages (default=True).
Fill flag:	<input type="text"/>	Optional - 1-character flag to track filled values, or "Auto".
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillWellDemandTSMonthlyPattern ( ID="90*", PatternID="08220000")
```

OK Cancel

FillWellDemandTSMonthlyPattern

**FillWellDemandTSMonthlyPattern() Command Editor**

The command syntax is as follows:

```
FillWellDemandTSMonthlyPattern (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill.	If not specified, fill the full period.
FillEnd	The last year to fill.	If not specified, fill the full period.
PatternID	The pattern identifier for data read with a ReadPatternFile() command.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing monthly averages.	True
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference: FillWellHistoricalPumpingTSMonthlyAverage()

**Fill well historical pumping time series (monthly) values using average monthly values**

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `FillWellHistoricalPumpingTSMonthlyAverage()` command fills missing well historical pumping time series (monthly) data, using average monthly values. The averages are computed immediately after reading time series (e.g., from HydroBase or a file). The average values that are used during data filling are printed to the log file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellHistoricalPumpingTSMonthlyAverage() Command**

This command fills missing data in well historical pumping (monthly) time series.  
Missing values are replaced with monthly average - average values are computed immediately after reading/calculating the data.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:  Specify the well stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`FillWellHistoricalPumpingTSMonthlyAverage (ID="*")`

FillWellHistoricalPumpingTSMonthlyAverage

### FillWellHistoricalPumpingTSMonthlyAverage() Command Editor

The command syntax is as follows:

```
FillWellHistoricalPumpingTSMonthlyAverage( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first date to fill, YYYY-MM or MM/YYYY.	If not specified, fill the full period.
FillEnd	The last date to fill, YYYY-MM or MM/YYYY.	If not specified, fill the full period.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillWellHistoricalPumpingTSMonthlyConstant()

Fill well historical pumping time series (monthly) values using a constant value

StateCU and StateMod Command

Version 3.09.01, 2010-01-27

The `FillWellHistoricalPumpingTSMonthlyConstant()` command fills missing well historical pumping time series (monthly) data, using a constant value. This command is useful, for example, to set pumping values to zero if other fill commands are unable to provide data estimates for missing data.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellHistoricalPumpingTSMonthlyConstant() Command**

This command fills missing data in well historical pumping (monthly) time series.  
Missing values are replaced with a constant value.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:  Specify the well stations to fill (use \* for wildcard)

Fill start:  Optional - start date or blank to fill all.

Fill end:  Optional - end date or blank to fill all.

Constant:  Required - constant value to use for filling.

Fill flag:  Optional - 1-character flag to track filled values.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
FillWellHistoricalPumpingTSMonthlyConstant ( ID="*",  
Constant=0)
```

OK Cancel

FillWellHistoricalPumpingTSMonthlyConstant

**FillWellHistoricalPumpingTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
FillWellHistoricalPumpingTSMonthlyConstant (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first year to fill, using format YYYY-MM or MM/YYYY.	If not specified, fill the full period.
FillEnd	The last year to fill, using format YYYY-MM or MM/YYYY.	If not specified, fill the full period.
Constant	The constant value to be used to fill missing data.	None – must be specified.
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## FillWellHistoricalPumpingTSMonthlyPattern()

**Fill well historical pumping time series (monthly) values using WET/DRY/AVG values**

**StateCU and StateMod Command**

Version 3.09.01, 2010-01-27

The `FillWellHistoricalPumpingTSMonthlyPattern()` command fills missing well historical pumping time series (monthly) data, using average monthly wet/dry/average values. The averages are computed using patterns read by the `ReadPatternFile()` command. The average values that are used during data filling are printed to the log file. For example, if a value is missing for May 1980, the pattern for the specified pattern identifier is checked for WET, DRY, or AVG. The values of all May's for WET, DRY, or AVG are then averaged in the time series to be filled, and the resulting average used to fill missing data. This command therefore will result in filled values that are more appropriate than simple averages; however, work must be done to characterize the wet, dry, and average months.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellHistoricalPumpingTSMonthlyPattern() Command**

This command fills missing data in well historical pumping (monthly) time series.  
Missing values are replaced with monthly average - average values are computed by this command using the pattern data.  
One or more `ReadPatternFile()` commands must be used before this command.  
The station ID can contain a \* wildcard pattern to match one or more time series.  
The fill period can optionally be specified. Only months in the output period can be filled.

Well station ID:	<input type="text" value="90*"/>	Specify the well stations to fill (use * for wildcard)
Fill start:	<input type="text"/>	Optional - start date or blank to fill all.
Fill end:	<input type="text"/>	Optional - end date or blank to fill all.
Pattern identifier:	<input type="text" value="08220000"/>	Required - pattern ID to use for filling.
<= zero values in average?:	<input type="checkbox"/>	Optional - are values <= zero used in averages (default=True).
Fill flag:	<input type="text"/>	Optional - 1-character flag to track filled values, or "Auto".
If not found:	<input type="checkbox"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
FillWellHistoricalPumpingTSMonthlyPattern(ID="90*", PatternID="08220000")
```

OK Cancel

FillWellHistoricalPumpingTSMonthlyPattern

**FillWellHistoricalPumpingTSMonthlyPattern() Command Editor**

The command syntax is as follows:

```
FillWellHistoricalPumpingTSMonthlyPattern(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FillStart	The first date to fill, YYYY-MM or MM/YYYY.	If not specified, fill the full period.
FillEnd	The last date to fill, YYYY-MM or MM/YYYY.	If not specified, fill the full period.
PatternID	The pattern identifier for data read with a ReadPatternFile() command.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing monthly averages.	True
FillFlag	If specified as a single character, data flags will be enabled for the time series and each filled value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: FillWellRight()

## Fill well right data

### StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `FillWellRight()` command fills missing data in existing well rights. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellRight() Command**

This command fills missing data in well right(s), using the well right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="*"/>	Required - specify the right(s) to fill (use * for wildcard)
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text"/>	Optional - station identifier.
Administration number:	<input type="text" value="99999.99999"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text"/>	Optional - decree amount, CFS.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="button"/>	Optional - indicate action if no match is found (default=Warn).
Command:	<pre>FillWellRight ( ID="*", AdministrationNumber=99999.99999, OnOff=1)</pre>	

OK Cancel

FillWellRight

**FillWellRight() Command Editor**

The command syntax is as follows:

```
FillWellRight (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching well right identifiers with missing name.	If not specified, the original value will remain.
StationID	The well station identifier to be assigned for all matching well right identifiers with missing well station identifier.	If not specified, the original value will remain.
AdministrationNumber	The administration number to be assigned for all matching well right identifiers with missing administration number.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching well right identifiers with missing administration decree.	If not specified, the original value will remain.
OnOff	The on/off switch to be assigned for all matching well right identifiers with missing on/off switch, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



# Command Reference: FillWellStation()

## Fill well station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `FillWellStation()` command fills missing data in existing well stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellStation() Command**

This command fills missing data in well station(s), using the well station ID to look up the location.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
Use blanks in the any field to indicate no change to the existing value.  
Monthly efficiencies should be separated by commas, with January first.  
Returns and depletions should be specified as triplets of location, percent, and return table ID:  
08123456,50.0,1;08234567,50.0,2

Well station ID: *	Required - ID for stations to fill (use * for wildcard)
Name:	Optional - up to 24 characters for StateMod.
River node ID: ID	Optional - the river node identifier, or "ID" to use the station ID.
On/Off: [v]	Optional - is station on/off in data set?
Capacity: 999	Optional - well capacity, CFS.
Daily ID: 4	Optional - the daily identifier, "ID", or StateMod flag).
Admin Num. Shift: 0 - Use water right priorities [v]	Optional - shift for well station right administration number.
Diversion station ID:	Optional - diversion station this well supplements (use "ID" to use the well station ID).
Demand type: 1 - Monthly total demand [v]	Optional - monthly demand time series type.
Irrigated acres:	Optional - typically for the most recent year.
Use type: 1 - Irrigation [v]	Optional - water use type.
Demand source: 1 - Irrigated acres from GIS [v]	Optional - water demand source.
Efficiency (Annual): 60	Optional - annual efficiency, percent (ignore if setting monthly).
Efficiencies (Monthly):	Optional - percent, annual is recomputed as average.
Returns (optional):	
Depletions (optional):	
If not found: [v]	Optional - indicate action if no match is found.

Command:  
`FillWellStation(ID="",RiverNodeID="ID",Capacity=999,DailyID="4",AdminNumShift=0,DemandType=1,UseType=1,DemandSource=1,EffAnnual=60)`

OK Cancel

FillWellStation

### FillWellStation() Command Editor

The command syntax is as follows:

`FillWellStation(Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20 *).	None – must be specified.
Name	The name to be assigned for all matching well stations with missing name.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching well stations with missing river node identifier. Specify ID to assign to the well station identifier.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching well stations with missing switch, either 1 for on or 0 for off.	If not specified, the original value will remain.
Capacity	The well station capacity to be assigned for all matching well stations with missing capacity, CFS.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching well stations with missing daily identifier or use ID to use the station identifier.	If not specified, the original value will remain.
AdminNumShift	The administration number shift to be assigned to water rights for all matching well stations with missing value. See the “primary” flag in the StateMod well station documentation.	If not specified, the original value will remain.
DiversionID	For all matching well stations, the diversion station identifier associated with the well station. Typically, where well water supplements surface supply, one well station is assigned to the diversion station. Specify ID to assign to the well station identifier.	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching well stations with missing demand type (see StateMod documentation).	If not specified, the original value will remain.
IrrigatedAcres	The irrigated acres to be assigned for all matching well stations with missing irrigated acres.	If not specified, the original value will remain.
UseType	The use type to be assigned for all matching well stations with missing user type (see StateMod documentation).	If not specified, the original value will remain.
DemandSource	The demand source to be assigned for all matching well stations with missing demand source (see StateMod documentation).	If not specified, the original value will remain.
EffAnnual	The annual efficiency (percent, 0 - 100) to be assigned for all matching well stations with missing annual efficiency (see StateMod documentation). Monthly efficiencies will be set to the same value (but not used).	If not specified, the original value will remain.
EffMonthly	The monthly efficiencies (percent, 0 – 100) to be assigned for all matching well stations with missing values, specified as 12 comma-separated values, January to December. The annual efficiency will be set to the average value. The order of the values in the output file will be according to the output year	If not specified, the original value will remain.

Parameter	Description	Default
	type set by <code>setOutputYearType()</code> , or calendar by default.	
Returns	The return flows to be assigned for all matching well stations with missing returns. Specify as <code>StationID,Percent,DelayTableID; StationID,Percent,DelayTableID;</code> etc.	If not specified, the original value will remain.
Depletions	The depletions to be assigned for all matching well stations with missing depletions. Specify as <code>StationID,Percent,DelayTableID; StationID,Percent,DelayTableID;</code> etc.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

This page is intentionally blank.

---

# Command Reference: FillWellStationsFromDiversionsStations ( )

Fill well station data using diversion stations

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillWellStationsFromDiversionsStations ( )` command fills missing well station data for each location, using the corresponding diversion station (only for D&W model nodes). The diversion stations must have been read or assigned with previous commands. The following data are filled: name, demand source, demand type, use type.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellStationsFromDiversionsStations() Command**

This command fills missing data in the well stations from diversion stations, including the following:

- Well name
- Demand source
- Demand type
- Use type

Diversion stations are specified by reading a StateMod diversion stations file with a previous command. The well station ID can contain a \* wildcard pattern to match one or more locations.

Well station ID: \* Required - well stations to process (use \* for wildcard).

If not found: [dropdown] Optional - indicate action if no match is found (default=Warn)

Command:

```
FillWellStationsFromDiversionsStations ( ID="*" )
```

OK Cancel

FillWellStationsFromDiversionsStations

**FillWellStationsFromDiversionsStations() Command Editor**

The command syntax is as follows:

```
FillWellStationsFromDiversionStations (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20 *).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: FillWellStationsFromNetwork()

Fill well station data from a StateMod network

**StateMod Command**

Version 3.09.01, 2010-02-01

The `FillWellStationsFromNetwork()` command fills missing data in well stations, using a StateMod network for data. This command is usually used after filling from other sources (e.g., HydroBase), because the information in the network file may have been specified mainly for the diagram and therefore does not necessarily match official data sources. It is assumed that the network has been read in a previous command (e.g., when the list of well stations was originally read).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit FillWellStationsFromNetwork() Command**

This command fills missing data in well stations using data from the network, matching the well station identifiers. This command is useful if names for stations cannot be filled from a database or other source. The following values from the network are set if missing in a station:

Name

Station ID:  Required - specify the stations to fill (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: `FillWellStationsFromNetwork ( ID="*" )`

FillWellStationsFromNetwork

**FillWellStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
FillWellStationsFromNetwork( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

The following example illustrates how to fill well station names from the network. A command to fill from HydroBase or another source will often be run before the second command below.

```
ReadWellStationsFromNetwork( InputFile="sp2005.net" )  
FillWellStationsFromNetwork( ID="*" )
```



---

# Command Reference:

## LimitDiversiionDemandTSMonthlyToRights()

**Limit diversion demand time series (monthly) to diversion rights**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `LimitDiversiionDemandTSMonthlyToRights()` command limits diversion demand time series (monthly) values to the water rights that were in effect at the time of the diversion, based on the appropriation date corresponding to water right administration numbers. The functionality of this command is nearly the same as the `LimitDiversiionHistoricalTSMonthlyToRights()`, except that demands are NOT reset to historical diversion observations, and demands can optionally be limited to only the current rights. For each diversion station being processed, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights with administration numbers greater than or equal to 90000.00000 can be assigned an appropriate date, which is then used to compute an administration number for the check. The water rights must be supplied from a StateMod diversion rights file – they are not taken from rights that may be in memory and the rights used by this command cannot be further modified and written. For boundary purposes during the check, a zero flow condition is imposed at 1800-01-01 and carried forward until a right is found. A summary of the rights is printed to the log file.

If necessary, place set commands after the `LimitDiversiionDemandTSMonthlyToRights()` command so that the set commands will not be impacted by the `LimitDiversiionDemandTSMonthlyToRights()` command.

The water rights switch in the StateMod rights file is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to limit the data).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to YYYY-01-01 during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to (YYYY+1)-01-01 and the decree is set to negative during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.

If the administration number cannot be converted to an appropriation date, then the water right `OnOff` switch can be set to a year for each water right and `UseOnOffDate=True` should be specified.

If the sum of the water rights decrees is less than zero, it is reset to zero.

A summary of the logic is as follows:

For each diversion station (ignored stations are skipped):

1. Determine the water rights for the diversion station. If no rights are available, skip the remaining steps.
2. Determine the diversion demand time series (monthly). If no time series is available, skip the remaining steps.
3. Process the water rights for the diversion station.
  - a. Convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored. See the explanation above for how the water rights switch is handled.
  - b. Sort the rights according to the Julian day value for the appropriation date.
  - c. If the diversion station has a free water right (those with administration numbers greater than or equal to 90000.00000): If the diversion station has a senior water right, convert the free water right appropriation date to that of the senior water right (therefore the free water right is in effect since the time of the senior right). If the diversion station has no senior water right (it has only free water right[s]), use the appropriation date corresponding to the `FreeWaterAppropriationDate` parameter described below.
  - d. Add a bounding zero decree for 1800-01-01 for the early period of the step function.
  - e. Generate a step function of sorted dates and decrees using the information described above. These values will be in CFS. Because appropriation dates are used, the sort order may be different from that of the numerical administration number.
  - f. If the `LimitToCurrent` parameter value is `True`, discard all but the last value in the step function.
  - g. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month.
4. Constrain the monthly time series to the step function, where the step function is defined by a list of dates and decrees, determined from the previous step. If a value in the time series is greater than the step function, set the value to the step function. Because of the conversion from CFS to ACFT, monthly values in the step function will vary.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit LimitDiversionDemandTSMonthlyToRights() Command**

This command limits diversion demand time series (monthly) to water rights for each diversion station.  
 Diversion rights are specified by reading a StateMod diversion rights file, or use rights in memory from previous commands.  
 Stations to ignore can be specified as one or more IDs or patterns, separated by commas.  
 It is recommended that the location of the rights file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversionDemandTSMonthlyToStateMod

StateMod rights file:

Diversion station ID:  Required - stations to process (use \* for wildcard).

Ignore ID(s):  Optional - stations to ignore (use \* for wildcard).

Free water appropriation date:  Optional - appropriation date for admin numbers >= 90000

Use OnOff date?:  Optional - get date from OnOff when YYYY, -YYYY? (default=False).

Limit to current rights:  Optional - use True to limit full period to current rights (default=False).

Set flag:  Optional - 1-character flag to track reset values (default=none).

Command:

LimitDiversionDemandTSMonthlyToRights

### LimitDiversionDemandTSMonthlyToRights() Command Editor

The command syntax is as follows:

```
LimitDiversionDemandTSMonthlyToRights(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod diversion rights file, surrounded by double quotes. The rights in the file are read and are used to constrain the diversion demand time series. The rights are assumed to be sorted by structure.	None – must be specified.
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IgnoreID	A list of diversion stations to ignore when processing this command. A list of comma-separated values can be specified, where each value is a single identifier, or a pattern using wildcards (similar to ID).	Do not ignore any diversion stations.
FreeWaterAppropriationDate	A date to be used for the free water rights found in the rights file. Free water rights are typically inserted to represent very junior rights. Rights having an administration number greater than or equal to 90000.00000 are assumed to be free water rights and will use the specified free water appropriation date when constraining the time series.	The date corresponding to an administration number of 0, which is Dec 31, 1849.
UseOnOffDate	If <b>False</b> , the appropriation date is always computed from the administration number. If <b>True</b> and the value of the <b>OnOff</b> switch is <b>YYYY</b> or <b>-YYYY</b> , assign the appropriation date using the switch value (see notes earlier in the command description).	<b>False</b>
LimitToCurrent	Indicate whether only the most recent sum of rights should be used when limiting the rights. This is appropriate when generating the demands for a baseline data set representing current conditions.	<b>False</b>
SetFlag	If specified as a single character, data flags will be enabled for the time series and each set value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.

---

# Command Reference:

## LimitDiversiOnHistoricalTSMonthlyToRights()

Limit diversion historical time series (monthly) to diversion rights

StateMod Command

Version 3.09.01, 2010-02-01

The `LimitDiversiOnHistoricalTSMonthlyToRights()` command limits diversion historical time series (monthly) values to the water rights that were in effect at the time of the diversion, based on the appropriation date corresponding to water right administration numbers. For each diversion station being processed, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights with administration numbers greater than or equal to 90000.00000 can be assigned an appropriate date, which is then used to compute an administration number for the check. The water rights must be supplied from a StateMod diversion rights file – they are not taken from rights that may be in memory and the rights used by this command cannot be further modified and written. For boundary purposes during the check, a zero flow condition is imposed at 1800-01-01 and carried forward until a right is found. A summary of the rights is printed to the log file.

This command does NOT reset recorded diversions. In order to detect recorded diversion values, StateDMI checks the command file for the `LimitDiversiOnHistoricalTSMonthlyToRights()` command. If the command is found, then after reading data using the `ReadDiversiOnHistoricalTSMonthlyFromHydroBase()` and `SetDiversiOnHistoricalTSMonthly()` commands, each time series is copied into a backup. Any subsequent filling of the time series does not alter this backup. When limiting to rights, the backup diversion data are checked and any observed values are enforced in the result. Consequently, the rights values are only used for estimated data. A side effect of using the original data is that any values that may have been set with other commands will be reset back to observed values. If necessary, place set commands after the `LimitDiversiOnHistoricalTSMonthlyToRights()` command so that the set commands will not be impacted by the `LimitDiversiOnHistoricalTSMonthlyToRights()` command(s).

The water rights switch in the StateMod rights file is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to limit the data).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to YYYY-01-01 during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to (YYYY+1)-01-01 and the decree is set to negative during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.

If the administration number cannot be converted to an appropriation date, then the water right `OnOff` switch can be set to a year for each water right and `UseOnOffDate=True` should be specified.

If the sum of the water rights decrees is less than zero, it is reset to zero.

A summary of the logic is as follows:

For each diversion station (ignored stations are skipped):

1. Determine the water rights for the diversion station. If no rights are available, skip the remaining steps.
2. Determine the diversion historical time series (monthly). If no time series is available, skip the remaining steps.
3. Process the water rights for the diversion station.
  - a. Convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored. See the explanation above for how the water rights switch is handled.
  - b. Sort the rights according to the Julian day value for the appropriation date.
  - c. If the diversion station has a free water right (those with administration numbers greater than or equal to 90000.00000): If the diversion station has a senior water right, convert the free water right appropriation date to that of the senior water right (therefore the free water right is in effect since the time of the senior right). If the diversion station has no senior water right (it has only free water right[s]), use the appropriation date corresponding to the FreeWaterAppropriationDate parameter described below.
  - d. Add a bounding zero decree for 1800-01-01 for the early period of the step function.
  - e. Generate a step function of sorted dates and decrees using the information described above. These values will be in CFS. Because appropriation dates are used, the sort order may be different from that of the numerical administration number.
  - f. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month.
4. Constrain the monthly time series to the step function, where the step function is defined by a list of dates and decrees, determined from the previous step. If a value in the time series is greater than the step function, set the value to the step function. Because of the conversion from CSFS to ACFT, monthly values in the step function will vary.
5. Reset observed values from the original data (as read from HydroBase or a replacement time series read from a StateMod file – time series that are NOT read from HydroBase or StateMod will NOT have a copy saved as original data).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit LimitDiversiionHistoricalTSMonthlyToRights() Command**

This command limits diversion historical time series (monthly) to water rights for each diversion station.  
 Diversion rights are specified by reading a StateMod diversion rights file, or use rights in memory from previous commands.  
 Stations to ignore can be specified as one or more IDs or patterns, separated by commas.  
 It is recommended that the location of the rights file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversiionHistoricalTSMonthlyFromHydroBase

StateMod rights file:

Diversion station ID:  Required - stations to process (use \* for wildcard).

Ignore ID(s):  Optional - stations to ignore (use \* for wildcard).

Free water appropriation date:  Optional - appropriation date for admin numbers >= 90000

Use OnOff date?:  Optional - get date from OnOff when YYYY, -YYYY? (default=False).

Set flag:  Optional - 1-character flag to track reset values (default=none).

Command: 

```
LimitDiversiionHistoricalTSMonthlyToRights (InputFile="..\statemod\cm2005.ddd"
, ID="*", IgnoreID="954683,952001,950010,950011")
```

**LimitDiversiionHistoricalTSMonthlyToRights() Command Editor**

The command syntax is as follows:

`LimitDiversiionHistoricalTSMonthlyToRights (Parameter=Value,...)`

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
InputFile	The name of the StateMod diversion rights file, surrounded by double quotes. The rights in the file are read and are used to constrain the historical diversion time series. The rights are assumed to be sorted by structure.	None – must be specified.
IgnoreID	A list of diversion stations to ignore when processing this command. A list of comma-separated values can be specified, where each value is a single identifier, or a pattern using wildcards (similar to ID).	Do not ignore any diversion stations.
FreeWaterAppropriationDate	A date to be used for the free water rights found in the rights file. Free water rights are typically inserted to represent very junior rights. Rights having an administration number	The date corresponding to an administration number of 0, which is Dec 31,

Parameter	Description	Default
	greater than or equal to 90000.00000 are assumed to be free water rights and will use the specified free water appropriation date when constraining the time series.	1849.
UseOnOffDate	If False, the appropriation date is always computed from the administration number. If True and the value of the OnOff switch is YYYY or -YYYY, assign the appropriation date using the switch value (see notes earlier in the command description).	False
SetFlag	If specified as a single character, data flags will be enabled for the time series and each set value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.

The following command file excerpt illustrates how time series can be limited to rights prior to writing the StateMod time series file:

```
#
# Step 8 - fill historical diversion using pattern approach
#
FillDiversionHistoricalTSMonthlyPattern(ID="36*",PatternID="09034500")
...similar commands omitted...
#
# Step 9 - Fill remaining missing with month average
#
FillDiversionHistoricalTSMonthlyAverage(ID="*")
#
# Step 10 - Limit filled diversion to water rights. Exceptions include structure
#           receiving significant reservoir supply, carrier structures, etc.
#
LimitDiversionHistoricalTSMonthlyToRights(InputFile="..\statemod\cm2005.ddr",
      ID="*",IgnoreID="954683,952001,950010,950011")
#
# Step 11 - sort structures and create historical diversion file
#
SortDiversionHistoricalTSMonthly(Order=Ascending)
WriteDiversionHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
#
# Step 12 - update capacities and create final direct diversion station file
#
SetDiversionStationCapacitiesFromTS(ID="*")
WriteDiversionStationsToStateMod(OutputFile="..\statemod\cm2005.dds")
#
# Check the results.
CheckDiversionHistoricalTSMonthly(ID="*")
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")
```



---

# Command Reference:

## LimitWellDemandTSMonthlyToRights()

Limit well demand time series (monthly) to diversion rights

**StateMod Command**

Version 3.09.01, 2010-02-01

The `LimitWellDemandTSMonthlyToRights()` command limits well demand time series (monthly) values to the water rights that were in effect at the time of the well pumping, based on the appropriation date corresponding to water right administration numbers. The functionality of this command is nearly the same as the `LimitDiversionDemandTSMonthlyToRights()`. For each well station being processed, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights with administration numbers greater than or equal to 90000.00000 can be assigned an appropriate date, which is then used to compute an administration number for the check. The water rights must be supplied from a StateMod well rights file – they are not taken from rights that may be in memory and the rights used by this command cannot be further modified and written. For boundary purposes during the check, a zero flow condition is imposed at 1800-01-01 and carried forward until a right is found. A summary of the rights is printed to the log file.

If necessary, place set commands after the `LimitWellDemandTSMonthlyToRights()` command so that the set commands will not be impacted by the `LimitWellDemandTSMonthlyToRights()` command.

The water rights switch in the StateMod rights file is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to limit the data).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to YYYY-01-01 during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to (YYYY+1)-01-01 and the decree is set to negative during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.

If the administration number cannot be converted to an appropriation date, then the water right `OnOff` switch can be set to a year for each water right and `UseOnOffDate=True` should be specified.

If the sum of the water rights decrees is less than zero, it is reset to zero.

A summary of the logic is as follows:

For each well station (ignored stations are skipped):

1. Determine the water rights for the well station. If no rights are available, skip the remaining steps.
2. Determine the well demand time series (monthly). If no time series is available, skip the remaining steps.
3. Process the water rights for the well station.
  - a. Convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored. See the explanation above for how the water rights switch is handled.
  - b. Sort the rights according to the Julian day value for the appropriation date.
  - c. If the well station has a free water right (those with administration numbers greater than or equal to 90000.00000): If the well station has a senior water right, convert the free water right appropriation date to that of the senior water right (therefore the free water right is in effect since the time of the senior right). If the well station has no senior water right (it has only free water right[s]), use the appropriation date corresponding to the FreeWaterAppropriationDate parameter described below.
  - d. Add a bounding zero decree for 1800-01-01 for the early period of the step function.
  - e. Generate a step function of sorted dates and decrees using the information described above. These values will be in CFS. Because appropriation dates are used, the sort order may be different from that of the numerical administration number.
  - f. If the LimitToCurrent parameter value is True, discard all but the last value in the step function.
  - g. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month.
4. Constrain the monthly time series to the step function, where the step function is defined by a list of dates and decrees, determined from the previous step. If a value in the time series is greater than the step function, set the value to the step function. Because of the conversion from CFS to ACFT, monthly values in the step function will vary.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit LimitWellDemandTSMonthlyToRights() Command**

This command limits well demand time series (monthly) to water rights for each well station.  
 Water rights are specified by reading a StateMod well rights file, or use rights in memory from previous commands.  
 Stations to ignore can be specified as one or more IDs or patterns, separated by commas.  
 It is recommended that the location of the rights file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork

StateMod rights file:

Well station ID:  Required - stations to process (use \* for wildcard).

Ignore ID(s):  Optional - stations to ignore (use \* for wildcard).

Free water appropriation date:  Optional - appropriation date for admin numbers >= 90000

Use OnOff date?:  Optional - get date from OnOff when YYYY, -YYYY? (default=False).

Limit to current rights:  Optional - use True to limit full period to current rights (default=False).

Set flag:  Optional - 1-character flag to track reset values (default=none).

Command: `LimitWellDemandTSMonthlyToRights (InputFile="rgTW.ddd", ID="*", FreeWaterAppropriationDate=1908-10-01)`

LimitWellDemandTSMonthlyToRights

### LimitWellDemandTSMonthlyToRights() Command Editor

The command syntax is as follows:

```
LimitWellDemandTSMonthlyToRights( Parameter=Value, ...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod well rights file, surrounded by double quotes. The rights in the file are read and are used to constrain the well demand time series.	None – must be specified.
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IgnoreID	A list of well stations to ignore when processing this command. A list of comma-separated values can be specified, where each value is a single identifier, or a pattern using wildcards (similar to ID).	Do not ignore any well stations.
FreeWater Appropriation Date	A date to be used for the free water rights found in the rights file. Free water rights are typically inserted to represent very junior rights. Rights having an administration number greater than or equal to 90000.00000 are assumed to be free water rights and will use the specified free water appropriation date when constraining the time series.	The date corresponding to an administration number of 0, which is Dec 31, 1849.
UseOnOffDate	If False, the appropriation date is always computed from the administration number. If True and the value of the OnOff switch is YYYY or -YYYY, assign the appropriation date using the switch value (see notes earlier in the command description).	False
LimitToCurrent	Indicate whether only the most recent sum of rights should be used when limiting the rights. This is appropriate when generating the demands for a baseline data set representing current conditions.	False
SetFlag	If specified as a single character, data flags will be enabled for the time series and each set value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.

---

# Command Reference: LimitWellHistoricalPumpingTSMonthlyToRights()

## Limit well historical pumping time series (monthly) to well rights

### StateCU and StateMod Command

Version 3.09.01, 2010-01-27

The `LimitWellHistoricalPumpingTSMonthlyToRights()` command limits well historical pumping time series (monthly) values to the water rights that were in effect at the time of the well, based on the appropriation date corresponding to water right administration numbers. For each well station being processed, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights with administration numbers greater than or equal to 90000.00000 can be assigned an appropriate date, which is then used to compute an administration number for the check. The water rights typically are supplied from a StateMod well rights file, although they can be taken from rights in memory. If the rights are read from a file, they cannot be further modified and written with other commands. For boundary purposes during the check, a zero flow condition is imposed at 1800-01-01 and carried forward until a right is found. A summary of the rights is printed to the log file.

If necessary, place set commands after the `LimitWellHistoricalPumpingTSMonthlyToRights()` command so that the set commands will not be impacted by the `LimitWellHistoricalPumpingTSMonthlyToRights()` command.

The water rights switch in the StateMod rights file is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to limit the data).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to YYYY-01-01 during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to (YYYY+1)-01-01 and the decree is set to negative during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.

If the administration number cannot be converted to an appropriation date, then the water right `OnOff` switch can be set to a year for each water right and `UseOnOffDate=True` should be specified.

If the sum of the water rights decrees is less than zero, it is reset to zero.

A summary of the logic is as follows:

For each well station (ignored stations are skipped):

1. Determine the water rights for the well station. If no rights are available, skip the remaining steps.
2. Determine the historical pumping time series (monthly). If no time series is available, skip the remaining steps.
3. Process the water rights for the well station.
  - a. Convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored. See the explanation above for how the water rights switch is handled.
  - b. Sort the rights according to the Julian day value for the appropriation date.
  - c. If the well station has a free water right (those with administration numbers greater than or equal to 90000.00000): If the well station has a senior water right, convert the free water right appropriation date to that of the senior water right (therefore the free water right is in effect since the time of the senior right). If the well station has no senior water right (it has only free water right[s]), use the appropriation date corresponding to the FreeWaterAppropriationDate parameter described below.
  - d. Add a bounding zero decree for 1800-01-01 for the early period of the step function.
  - e. Generate a step function of sorted dates and decrees using the information described above. These values will be in CFS. Because appropriation dates are used, the sort order may be different from that of the numerical administration number.
  - f. If the LimitToCurrent parameter value is True, discard all but the last value in the step function.
  - g. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month.
4. Constrain the monthly time series to the step function, where the step function is defined by a list of dates and decrees, determined from the previous step. If a value in the time series is greater than the step function, set the value to the step function. Because of the conversion from CSFS to ACFT, monthly values in the step function will vary.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit LimitWellHistoricalPumpingTSMonthlyToRights() Command**

This command limits well historical pumping time series (monthly) to water rights for each well station.  
 Water rights are specified by reading a StateMod well rights file, or use rights in memory from previous commands.  
 Stations to ignore can be specified as one or more IDs or patterns, separated by commas.  
 It is recommended that the location of the rights file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef

StateMod rights file:

Well station ID:  Required - stations to process (use \* for wildcard).

Ignore ID(s):  Optional - stations to ignore (use \* for wildcard).

Free water appropriation date:  Optional - appropriation date for admin numbers >= 90000

Use OnOff date?:  Optional - get date from OnOff when YYYY, -YYYY? (default=False).

Set flag:  Optional - 1-character flag to track reset values (default=none).

Command: 

```
LimitWellHistoricalPumpingTSMonthlyToRights (InputFile="rg2007.wer", ID="*", FreeWaterAppropriationDate=1908-10-01)
```

LimitWellHistoricalPumpingTSMonthlyToRights

### LimitWellHistoricalPumpingTSMonthlyToRights() Command Editor

The command syntax is as follows:

```
LimitWellHistoricalPumpingTSMonthlyToRights( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
InputFile	The name of the StateMod well rights file, surrounded by double quotes. The rights in the file are read and are used to constrain the well historical pumping time series.	Use well rights that have previously been defined.
IgnoreID	A list of well stations to ignore when processing this command. A list of comma-separated values can be specified, where each value is a single identifier, or a pattern using wildcards (similar to ID).	Do not ignore any well stations.
FreeWater Appropriation Date	A date to be used for the free water rights found in the rights file. Free water rights are typically inserted to represent very junior rights. Rights having an administration number greater than or equal to 90000.00000 are assumed to be free water rights and will use the specified free water appropriation date when constraining the time series.	The date corresponding to an administration number of 0, which is Dec 31, 1849.
LimitToCurrent	Indicate whether only the most recent sum of rights should be used when limiting the rights. This is appropriate when generating the demands for a baseline data set representing current conditions.	False
UseOnOffDate	If False, the appropriation date is always computed from the administration number. If True and the value of the OnOff switch is YYYY or -YYYY, assign the appropriation date using the switch value (see notes earlier in the command description).	False
SetFlag	If specified as a single character, data flags will be enabled for the time series and each set value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.



# Command Reference: MergeListFileColumns()

## Merge columns in a list file and write a new list file

### General Command

Version 3.09.00, 2010-01-16

The `MergeListFileColumns()` command reads a comma-delimited list file, uses information from existing columns to create a new column, and writes the new list file. As currently implemented, whitespace will be trimmed from the original data. This command is useful for example, if the WD and ID columns in a file need to be merged into a WDID column.

The following dialog is used to edit the command and illustrates command syntax:

**Edit MergeListFileColumns() Command**

This command merges columns in a list file, creating a new column in a new list file.  
This is useful, for example, when station identifiers need to be created from data in multiple columns.  
Columns should be delimited by commas (user-specified delimiters will be added in the future).  
It is recommended that the location of the files be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\commands\MergeListFileColumns

List file:

Output file:

Columns:  Required - specify as comma-separated numbers.

NewColumnName:  Required - new column that will be added at end.

Merge format:  Optional - (e.g., 2,5 or 02,05 to pad with zeros).

Command:  

```
MergeListFileColumns(ListFile="Data\SimpleList.csv", OutputFile="Results\Test_MergeListFileColumns_1_out.csv", Columns="1,2", NewColumnName="WDID", SimpleMergeFormat="02,05")
```

The above command can be copied for batch processing.

MergeListFileColumns

### MergeListFileColumns() Command Editor

The command syntax is as follows:

```
MergeListFileColumns(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to read.	None – must be specified.
OutputFile	The name of the output list file to create. This file will have the same contents as the original file, with the new column at the end.	None – must be specified.
Columns	A comma-separated list of columns to merge. Column numbers have a value $\geq 1$ .	None – must be specified.
NewColumnName	Name of the column to be created, which will be added at the end of the original columns.	None – must be specified.
SimpleMergeFormat	Comma-separated list of formats indicating how to format the merged column, one of the following: <ul style="list-style-type: none"> <li>Blank – concatenate the values.</li> <li>NN,NN (e.g., 2, 5) – indicate the widths for each part and merge as strings where each part is right justified and padded with spaces.</li> <li>0N,0N (e.g., 02, 05) – indicate the widths for each part and merge as integers where each part is right justified and padded with zeros.</li> </ul> A more flexible formatting capability may be added in the future.	Concatenate values.

The following example concatenates the WD and ID columns into a single WDID column by padding with zeros. The first row of the list file should contain double-quoted column headers.

```
MergeListFileColumns(ListFile="poudre1.csv",OutputFile="poudre2.csv",
Columns="2,3",NewColumnName="WDID",SimpleMergeFormat="02,05")
```

---

# Command Reference: MergeWellRights ()

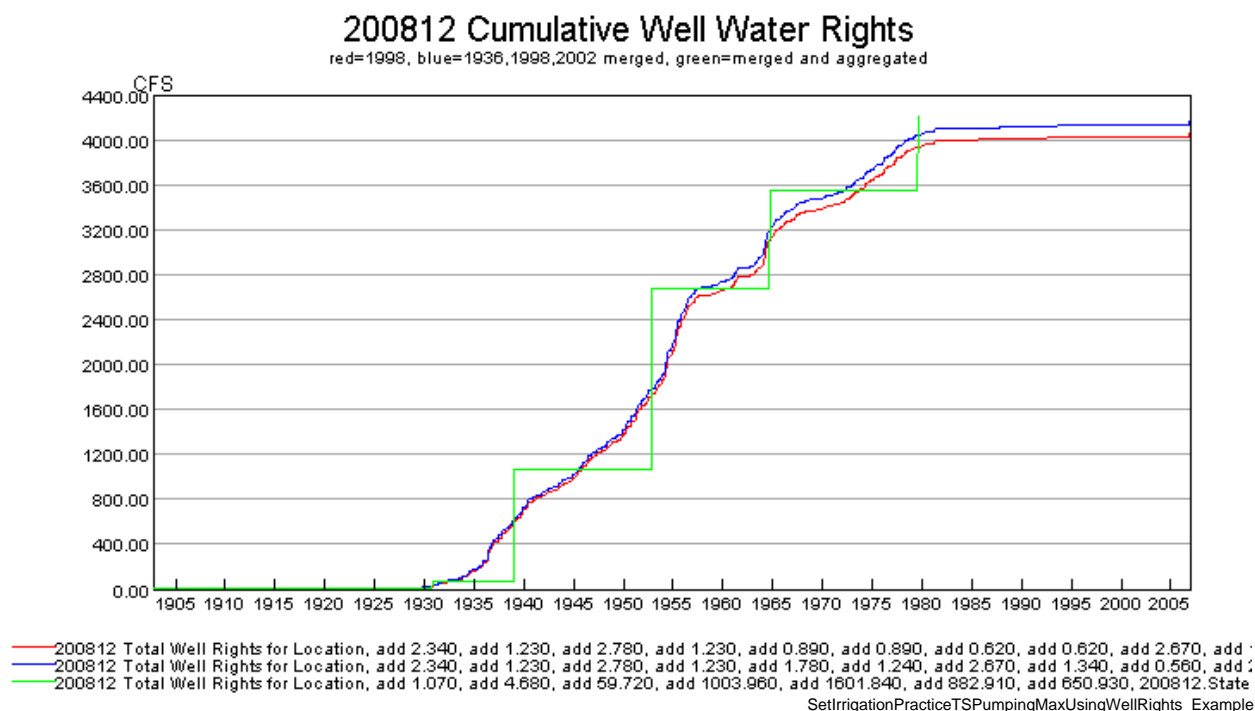
Merge well right data from multiple parcel years into a common set of well rights

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `MergeWellRights()` command merges well rights from multiple years of irrigated lands parcel data as read with the `ReadWellRightsFromHydroBase()` command. Well rights initially are generated for each parcel year by using well to parcel matching relationships. Because the parcel identifiers change each year and new wells are added over time, the mix of well rights associated with parcels changes. Well rights are generated from HydroBase for each independent year and then need to be merged in order to NOT double count the well rights for modeling. The `ReadWellRightsFromHydroBase()` and `WriteWellRightsToStateMod(..., WriteDataComments=True, ...)` commands are used to generate a well rights file that contain the parcel data, typically with a file name similar to *rg2007\_NotMerged.wer*.

The following figure illustrates the difference between raw rights data for one year, merged rights for multiple years, and aggregated rights created from the merged rights. Merged rights will typically be higher than the raw rights because a few rights appear in some years but not in others (they are additive). Aggregating rights reduces the number of rights in the data set by grouping rights into administration number classes (see the `AggregateWellRights()` command) – this reduced the time needed to run the model but limits the ability to refer to individual rights (e.g., in augmentation plans).



An excerpt from a well rights file with data comments is shown below. The parcel year, well/parcel matching class, and parcel ID are shown on the far right and are not part of the standard StateMod well

right file. See CDSS technical memoranda for a description of well classes (SPDSS Task Memorandum “SPDSS, Spatial System Integration Component, Well Class Adjustments”, March 15<sup>th</sup>, 2007).

#>	ID	Name	Struct	Admin #	Decree	On/Off	PYr--Cls--PID
#>-----eb-----eb-----eb-----eb-----eb-----eb-----exb--exb--exb-----e							
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1936 1 3107
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956	1936 1 3107
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1998 2 11016
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956	1998 2 11016
2005001	W0006	WELL NO 01	200812	31592.00000	1.19	1936	2002 2 20901
2005001	W0006	WELL NO 01	200812	38836.00000	0.62	1956	2002 2 20901
2005001	W0006	WELL NO 01	200812	31592.00000	1.15	1936	2002 5 20902
2005001	W0006	WELL NO 01	200812	38836.00000	0.61	1956	2002 5 20902

For the example shown, the well rights are the same for the 1936 and 1998 irrigated lands parcel data. However, in 2002 there is a change. This may be due to several reasons, including:

- The results for each irrigated lands assessment were created with the HydroBase and other data available at that time. HydroBase subsequently changed and original parcel/well matching work was not redone.
- The parcel configuration changes in different years, resulting in different well match classes (note class 5 wells above). For example a parcel may be whole in one year and split in another year, due to changes in physical configuration or data processing procedures. If a parcel is split between ditch service areas, a factor is applied to split the well among the ditches.

In order to generate a merged file that represents rights for the full period (and all active parcels and wells in that period), it is necessary to compare the rights from each year and remove duplicates. The logic that is implemented in this command for merging rights is as follows:

1. Extract any records that have a parcel year of -999 (indicative of explicit well rights not associated with parcels that need to apply to the entire period) and do not process with the following steps – these rights are added to the list at the end.
2. Determine the unique list of locations from the well rights, and the unique list of parcel years.
3. Merge the first and second years of data, then merge the results with the third year of data, etc.:
  - a. For each location ID, get the list of rights for each parcel year (or from the previous results and the next parcel year being merged).
  - b. For the above list, get the list of water rights identifiers for the first year (or previous merge). This divides the long list of water rights for a location into a more manageable list. The administration number and well/permit match class are not considered.
  - c. Compare the two lists.
    - i. If the lists are exactly the same (same number and rights exactly match), then include the rights from the first year (or previous merge). The parcel year is retained for further comparisons.
    - ii. If the rights are not exactly the same, determine the sum of the decrees from each year and include the rights for the year with the highest decree total. In some cases, a new set of rights will be added, which were not present in the previous results. **This assumes that well use increases over time – currently the case where wells are turned off is not handled.**

Repeat step c for each well right identifier for a location, for the years of data that are being compared.
  - d. If any rights were not considered above, add them to the list. For example, a year may include right identifiers that did not exist in another year. In this case it is not possible to compare the sum of rights – the sum in one year will be zero.

Repeat step 3 for each year of data, comparing the next year with the results from the previous merge. In many cases the rights will simply be carried forward during the comparison, but in some years a block of rights will be replaced.

4. Add to the final list the well rights that had a parcel ID of -999.
5. If an output file was specified with the `OutputFile` parameter, write the intermediate results to a file (see the `OutputFile` parameter description in the table below).
6. If `SumDecreases=True`, further process the rights to sum decrees where the right identifier, administration number, and on/off switch are equal. This reduces the number of rights, but the overall decree will be the same. Water rights that are associated with estimated wells (class 4 or 9) are passed through without change in order to retain the original information for these relationships. For example the same original well may be copied multiple times for an estimated well and this information is evident when the original rights are retained. A more complete evaluation of estimated wells could be performed if the merge process considered well locations.

After merging the rights, the file is typically written using `WriteWellRightsToStateMod()` with a name similar to *rg2007.wer*. This file can be used to set irrigation practice time series pumping maximum and can be used for `StateMod` modeling. It cannot be used to fill crop pattern and irrigation practice acreage time series because a specific parcel year is needed (use the original non-merged rights file with all years for filling acreage). If the well rights will be aggregated, as has been done in the Río Grande modeling, then use the `AggregateWellRights()` command and create a third rights file for use with `StateMod`. For the initial example above, the merged results are as follows:

#>	ID	Name	Struct	Admin #	Decree	On/Off	PYr--Cls--PID
#>	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	exb--exb--exb---e
2005001		W0006 WELL NO 01	200812	31592.00000	2.34	1936	
2005001		W0006 WELL NO 01	200812	38836.00000	1.23	1956	

The following dialog is used to edit the command and illustrates the syntax of the command.

Edit MergeWellRights() Command

This command merges well water rights created using multiple years of irrigated parcel data (output from the `ReadWellRightsFromHydroBase()` command). The original water rights for multiple years are replaced by merged rights that apply for the full period. Rights present in adjacent years with matching location ID, right ID, well type (class), and administration number are retained in the result, using the earliest year for the parcel year. If rights do not match, the list of rights for a location ID and right ID with the highest decree total are retained in the result (assuming increasing water use). Therefore, rights found in only one year are retained in the result. The first two years of data are compared first and subsequent years of data are then compared to the result. This command does not aggregate rights spatially or by combining a range of administration numbers. See the `AggregateWellRights()` command, which can be run after this command. Specify the output file the same as the final \*.wer to create a sequence of files showing intermediate results including the rights for the first year, after merging the second year, etc.

A final step sums the decrees for rights with the same identifier, administration number, and on/off switch. Water rights that are parcel supply class 4 and 9 are not summed because they represent unique estimated wells.

Each step is controlled by optional parameters to allow evaluation of the processing steps.

Output file:

Merge parcel years?:  Optional - merge well rights for multiple parcel years (default=True).

Sum right decrees?:  Optional - at end, sum rights with same ID and admin number (default=True).

Command:

MergeWellRights

### MergeWellRights() Command Editor

The command syntax is as follows:

```
MergeWellRights (Parameter=value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	If specified, the output file typically is the same as the output file for the WriteWellRightsToStateMod() command. The filename will be pre-pended with wer-merged-YYYY- to indicate the results of the merge after considering the parcel year indicated by YYYY.	No file is created to indicate results of intermediate processing.
MergeParcelYears	If True, merge the water rights for multiple parcel years as indicated by this documentation. If False, do not merge (but can still sum decrees). This parameter allows evaluation of combinations of the processing steps.	True
SumDecrees	If True, process the rights after the main merge logic to combine rights where the identifier, administration number, and on/off switch are the same, increasing the decree to the combined value. Water rights that are class 4 or 9 parcel matches (indicating estimated wells) are not changed, to better understand the impact of estimated wells.	True

The following example command file illustrates how well rights can be defined, sorted, merged, checked, and written to a StateMod file:

```
# Well Rights File (*.wer)
#
StartLog(LogFile="Sp2008L_WER.log")
#
# Step 1 - Read all structures
#
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
SortWellStations()
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=1,IfNotFound=Ignore)
#
# Step 5 - Read rights from HydroBase
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1956,1976,1987,2001,2005",
    Div="1",DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
    ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds & ipy acreage filling
# Write Data Comments="False" provides merged file used for seting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Sp2008L.wer.check.html",Title="Well Rights Check File")
```

The following TSTool command file illustrates how various StateMod well right files can be processed to generate time series of decrees, similar to the graph shown above. The command will automatically generate a data set total time series. This visual check provides an understanding of the decrees in a basin over time.

```
# Read the unmerged and merged StateDMI *wer files to compare
SetOutputPeriod(OutputStart="1900-01",OutputEnd="2008-12")
ReadStateMod( InputFile="Sp2008L_Unmerged.wer",Alias="%L-Unmerged",Interval="Month" )
ReadStateMod( InputFile="Sp2008L.wer",Alias="%L",Interval="Month" )
```



# Command Reference: OpenHydroBase()

## Open a connection to a HydroBase database

**General Command**  
Version 3.08.02, 2010-01-06

The `OpenHydroBase()` command opens a connection to a HydroBase database, allowing data to be read from the database. This command is not typically used for interactive sessions but may be inserted to run in batch only mode to allow a specific database and commands files to be distributed.

The following dialog is used to edit this command and illustrates the command syntax. The **Database type** is used to control settings for parameters and is not itself a parameter.

**Edit OpenHydroBase() command**

This command opens a connection to a HydroBase database, closing the previous connection with the same input name.  
This command is used, for example, when making connections to more than one HydroBase database, or running in batch mode.  
The RunMode can also be set to control whether the command is run in batch mode, in interactive sessions, or both.  
The connection can be made either by specifying a database server/name for SQL Server,  
or an ODBC Data Source Name (DSN) for a Microsoft Access HydroBase database (only for very old HydroBase databases).

Database type:  Required - indicates whether a database server/name or ODBC DSN is specified below.

Database server:  Required - when using SQL Server.

Database name:  Required - when using SQL Server.

ODBC DSN:  Optional - only used with Microsoft Access.

Input name:  Optional - input name for connection, to be used with time series identifiers.

Use stored procedures?: ☒ Optional - default is true for SQL Server, ignored for Access.

Run Mode:  Optional - default is GUI and batch mode.

Command:

OpenHydroBase

**OpenHydroBase() Command Editor**

The command syntax is as follows:

```
OpenHydroBase (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
DatabaseServer	Used with a SQL Server HydroBase. Specify the SQL Server database machine name. A list of choices will be shown, corresponding to properties in the <i>CDSS.cfg</i> configuration file.	Required if a SQL Server database is used, and accepts the generic value <code>DatabaseServer=local</code> , which will automatically be translated to the name of the local computer.
DatabaseName	Used with a SQL Server HydroBase.	HydroBase

Parameter	Description	Default
	The name of the database typically follows a pattern similar to: HydroBase_CO_YYYYMMDD. A list of choices will be shown, corresponding to properties in the <i>CDSS.cfg</i> configuration file.	
OdbcDsn	The ODBC DSN to use for the connection, used only when working with a Microsoft Access database.	Required if a Microsoft Access database is used.
InputName	The input name corresponding to the ~InputType~InputName information in time series identifiers. This is used when more than one HydroBase connection is used in the same command file.	Blank (no input name).
UseStoredProcedures	Used with SQL Server, indicating whether stored procedures are used. Stored procedures are the default and should be used except when testing software.	True (used stored procedures).
RunMode	Indicates when the command should be run, one of:  BatchOnly – run the command only in batch mode. GUIOnly – run the command only in GUI mode. GUIAndBatch – run the command in batch and GUI mode.	GUIAndBatch

The following example command file illustrates how to connect to a SQL Server database running on a machine named “sopris”:

```
StartLog(LogFile="Results/Example_OpenHydroBase_DatabaseName.StateDMI.log")
OpenHydroBase(DatabaseServer="sopris",DatabaseName="HydroBase_CO_20060816")
```

Example\_OpenHydroBase\_DatabaseName

# Command Reference: PrintNetwork()

Print network to printer and/or file

**StateMod Command**

Version 3.09.01, 2010-02-01

**This command is under development.**

The `PrintNetwork()` command prints the network to a printer and/or to a file. This command is useful for automating the output of network products. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit PrintNetwork() Command**

**This is an experimental command and is under development.**

This command prints a StateMod XML network file.

The network can be printed directly to a printer or can be printed to a file, which is useful for testing and reviewing drafts.

If the network file is not specified, it is assumed that the network has already been read in the network editor or by another command.

If the network file is specified, it is read and will be available to later commands.

The layout that is selected must use a paper size that is available on the selected printer.

It is recommended that the path to files be specified relative to the working directory.

The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI

StateMod network file:

Page layout:   Required - layout as shown in the network editor.

Printer:  Required - printer (e.g., //ServerName/PrinterName).

Print file:

Command: 

```
PrintNetwork(InputFile="cm2005.net", PageLayout="11x17", Printer="//p  
rintserver/printer1")
```

PrintNetwork

**PrintNetwork() Command Editor**

The command syntax is as follows:

```
PrintNetwork(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to process.	If not specified, the network that has been previously read into memory will be used.
PageLayout	The page layout that is configured for the network, as shown in the network editor.	None – must be specified.
Printer	The name of the printer to use as //ServerName/PrinterName	Nothing will be sent to a printer.
OutputFile	The name of the output file to create.	No file will be created.

---

# Command Reference: ReadBlaneyCriddleFromHydroBase()

Read Blaney-Criddle crop coefficients data from HydroBase

**StateCU Command**  
Version 3.08.02, 2010-01-07

The ReadBlaneyCriddleFromHydroBase() command reads a list of Blaney-Criddle crop coefficients from the HydroBase database. The crop coefficients can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadBlaneyCriddleFromHydroBase() Command**

This command reads Blaney-Criddle crop coefficients from HydroBase.  
Blaney-Criddle crop coefficients estimate crop water requirements for each crop during the year, for standard conditions.  
Characteristics can be queried for a specific Blaney-Criddle method.

Blaney-Criddle Method:  Required - Blaney-Criddle method.

Command:

OK Cancel

ReadBlaneyCriddleFromHydroBase

**ReadBlaneyCriddleFromHydroBase() Command Editor**

The command syntax is as follows:

`ReadBlaneyCriddleFromHydroBase(Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
BlaneyCriddleMethod	The Blaney-Criddle method that is defined in HydroBase for the crop type and its coefficients.	None – must be specified.

The crop type (e.g., ALFALFA) is used as the unique identifier. Any previous crop coefficients objects will be added to (or replaced if identifiers match).

The BlaneyCriddleMethod parameter corresponds to a value in HydroBase and allows regional crop characteristics to be defined.

The following example command file illustrates how to read Blaney-Criddle coefficients from HydroBase, sort the data, create a StateCU file, and check the results:

```
StartLog(LogFile="Crops_KBC.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Blaney-Criddle coefficients File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-23 SAM, RTi               Update for Rio Grande Phase 5.
#
# Step 1 - read data from HydroBase
#
# Read the general Blaney-Criddle coefficients first and then override with Rio Grande
# data.
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 3 - write the file
#
SortBlaneyCriddle(Order=Ascending)
WriteBlaneyCriddleToStateCU(OutputFile="rg2007.kbc")
#
# Check the results
#
CheckBlaneyCriddle(ID="*")
WriteCheckFile(OutputFile="rg2007.kbc.check.html")
```

---

# Command Reference: ReadBlaneyCriddleFromStateCU()

**Read Blaney-Criddle crop coefficients data from a StateCU Blaney-Criddle crop coefficients file**

**StateCU Command**

Version 3.08.02, 2010-01-07

The `ReadBlaneyCriddleFromStateCU()` command reads Blaney-Criddle crop coefficients from a StateCU Blaney-Criddle crop coefficients file and defines crop coefficients in memory. The crop coefficients can then be manipulated and output with other commands. This command can be used to adjust an existing crop coefficients file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadBlaneyCriddleFromStateCU() Command**

This command reads Blaney-Criddle crop coefficients from a StateCU Blaney-Criddle crop coefficients file. Blaney-Criddle crop coefficients estimate crop water requirements for each crop during the year, for standard conditions. It is recommended that the file be specified using a path relative to the working directory. The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadBlaneyCriddleFromStateCU

Input file:

Command:  

```
ReadBlaneyCriddleFromStateCU ( InputFile="rg2007.kbc" )
```

ReadBlaneyCriddleFromStateCU

**ReadBlaneyCriddleFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadBlaneyCriddleFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.



# Command Reference: ReadClimateStationsFromList()

Read climate station data from a list file to define climate stations

**StateCU Command**

Version 03.08.02, 2010-01-05

The `ReadClimateStationsFromList()` command reads a list of climate stations from a delimited list file and defines climate stations in memory. The climate stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadClimateStationsFromList() Command**

This command reads climate station from a list file containing columns of information. Climate station data are used by the consumptive use model to estimate water requirement. The climate stations, once defined, can be associated with locations where consumptive use is estimated, using Region1 (e.g., county) and Region2 (e.g., HUC). Columns should be delimited by commas (user-specified delimiters will be added in the future). It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadClimateStationsFromList

List file:

Climate station ID column:  Required - column (1+) for station identifier.

Name column:  Optional - column (1+) for station name.

Latitude column:  Optional - column (1+) for latitude.

Elevation column:  Optional - column (1+) for elevation.

Region1 column:  Optional - column (1+) for region 1 (e.g., county).

Region2 column:  Optional - column (1+) for region 2 (e.g., HUC).

Command: 

```
ReadClimateStationsFromList(ListFile="climsta.lst", IDCol=1)
```

ReadClimateStationsFromList

**ReadClimateStationsFromList() Command Editor**

The command syntax is as follows:

```
ReadClimateStationsFromList (Parameter=Value, ...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the climate station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the climate station name.	None – optional (name will be initialized to blank).
LatitudeCol	The column number (1+) containing the climate station latitude.	None – optional (latitude will be initialized to missing data).
ElevationCol	The column number (1+) containing the climate station elevation.	None – optional (elevation will be initialized to missing data).
Region1Col	The column number (1+) containing the climate station Region1.	None – optional (Region1 will be initialized to blank).
Region2Col	The column number (1+) containing the climate station Region2.	None – optional (Region2 will be initialized to blank).

At a minimum, the list file must contain a column with station identifiers. Lines starting with the # character are treated as commas. Column names can be specified in the first non-comment line by enclosing each column name in quotes.

An example list file is shown below, for example created from CDSS TSTool software:

```
# Climate stations
"ID", "Name"
0130, "ALAMOSA SAN LUIS VALLEY RGNL"
0776, "BLANCA"
1458, "CENTER 4 SSW"
2184, "DEL NORTE 2 E"
3541, "GREAT SAND DUNES N M"
3951, "HERMIT 7 ESE"
5322, "MANASSA"
5706, "MONTE VISTA 2 W"
7337, "SAGUACHE"
```

The following example command file illustrates how climate stations can be defined from a list and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst", IDCol=1)
FillClimateStationsFromHydroBase(ID="*")
SetClimateStation(ID="3016", Region2="14080106", IfNotFound=Warn)
SetClimateStation(ID="1018", Region2="14040106", IfNotFound=Warn)
SetClimateStation(ID="1928", Elevation=6440, IfNotFound=Warn)
SetClimateStation(ID="0484", Region1="MOFFAT", IfNotFound=Add)
WriteClimateStationsToStateCU( outputFile="COclim2006.cli")
```

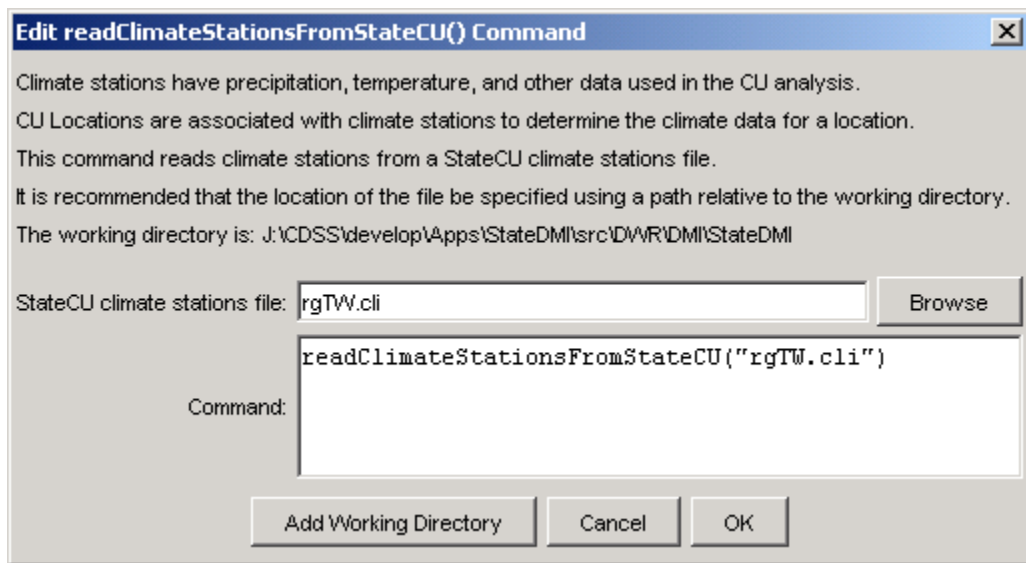
---

# Command Reference: ReadClimateStationsFromStateCU()

**Read climate station data from a StateCU climate stations file to define climate stations**

**StateCU Command**  
Version 3.08.02, 2010-01-05

The `ReadClimateStationsFromStateCU()` command reads a list of climate stations from a StateCU climate stations file and defines climate stations in memory. The climate stations can then be manipulated and output with other commands. This command can be used to adjust an existing climate stations file. The following dialog is used to edit the command and illustrates the syntax of the command.



ReadClimateStationsFromStateCU

**ReadClimateStationsFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadClimateStationsFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.

The following example command file illustrates how climate stations can be read from a StateCU file:

```
ReadClimateStationsFromStateCU(InputFile="COclim2006.cli")
```

---

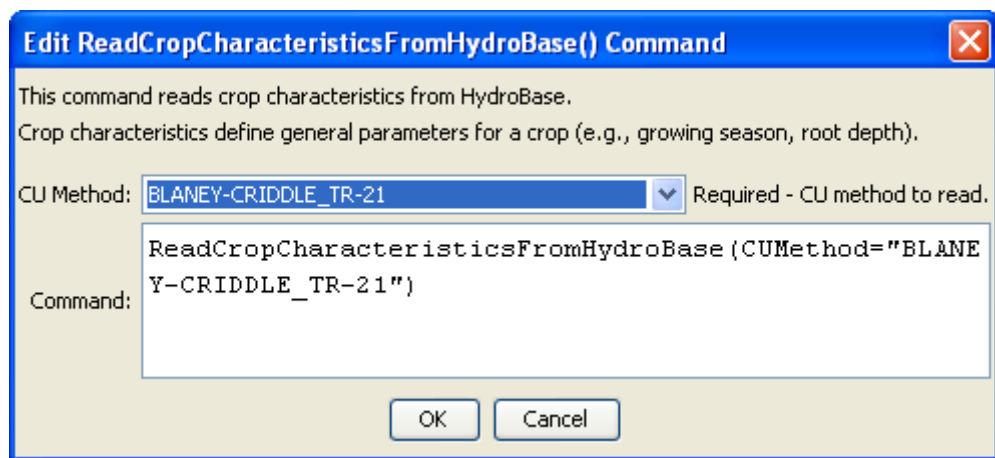
# Command Reference: ReadCropCharacteristicsFromHydroBase()

Read crop characteristics data from HydroBase

**StateCU Command**

Version 3.08.02, 2010-01-07

The `ReadCropCharacteristicsFromHydroBase()` command reads a list of crops and their characteristics from a HydroBase database. The crop characteristics can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.



ReadCropCharacteristicsFromHydroBase

**ReadCropCharacteristicsFromHydroBase() Command Editor**

The command syntax is as follows:

```
ReadCropCharacteristicsFromHydroBase( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
CUMethod	The CU method that is defined in HydroBase for the crop type and its characteristics.	None – must be specified.

The crop type (e.g., ALFALFA) is used as the unique identifier. Any previous crop characteristics objects will be added to (or replaced if identifiers match). The crop types in HydroBase may actually include some land use types that are not appropriate for StateCU (e.g., Water, NO\_DATA). Currently these crop types are still queried from HydroBase.

To allow for some flexibility in defining crop characteristics, a *CU Method* is used in HydroBase and can be used to adjust crop characteristics for regional differences. For example, read the **Soil Conservation Service Irrigation Water Requirements Technical Release No. 21 (TR-21)** characteristics first and then reset the characteristics for a crop due to local conditions.

The following example illustrates how to create a StateCU crop characteristics file with data from HydroBase:

```
StartLog(LogFile="Crops_CCH.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Crop Characteristics File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-22 SAM, RTi               Use new directory structure, current
#                                   software and HydroBase.
#
# Step 1 - read data from HydroBase
#
# Read the general TR-21 characteristics first and then override with Rio Grande
# data.
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 2 - adjust crop characteristics if needed
#       No resets are needed.
#
# Step 3 - write the file
#
WriteCropCharacteristicsToStateCU(OutputFile="rg2007.cch")
#
# Check the results
#
CheckCropCharacteristics(ID="*")
WriteCheckFile(OutputFile="rg2007.cch.check.html")
```

---

# Command Reference: ReadCropCharacteristicsFromStateCU()

Read crop characteristics data from a StateCU crop characteristics file

**StateCU Command**

Version 3.08.02, 2010-01-07

The `ReadCropCharacteristicsFromStateCU()` command reads a list of crops and their characteristics from a StateCU crop characteristics file and defines crop characteristics in memory. The crop characteristics can then be manipulated and output with other commands. This command can be used to adjust an existing crop characteristics file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadCropCharacteristicsFromStateCU() Command**

This command reads crop characteristics from a StateCU crop characteristics file.  
Crop characteristics define general parameters for a crop (e.g., growing season, root depth).  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCropCharacteristicsFromStateCU

Input file:

Command:  

```
ReadCropCharacteristicsFromStateCU (InputFile="rg2007.cch")
```

ReadCropCharacteristicsFromStateCU

**ReadCropCharacteristicsFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadCropCharacteristicsFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.



---

# Command Reference: ReadCropPatternTSFromHydroBase()

Read crop pattern time series data from HydroBase

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `ReadCropPatternTSFromHydroBase()` command reads crop pattern time series from HydroBase and defines crop pattern time series in memory. The crop pattern time series can then be manipulated and output with other commands. If a CU Location is a diversion, the crop pattern data are read from HydroBase tables that contain irrigated acres for the ditch service area. If the CU Location is an aggregate of parcels, the area is determined from the parcel data.

When processing crop pattern time series, data from HydroBase may need to be combined with user-specified data. A single location or location that is part of an aggregate/system can have its data specified with a `SetCropPatternTS(...,ProcessWhen=WithParcels,...)` or `SetCropPatternTSFromList(...,ProcessWhen=WithParcels,...)` command. In this case, it is expected that the acreage will not be found in HydroBase. Use set commands before the `ReadCropPatternTSFromHydroBase()` command. It is recommended that a `SetCropPatternTSFromList(...,ProcessWhen=WithParcels,...)` command be used for each year of HydroBase data that is processed.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadCropPatternTSFromHydroBase() Command**

This command reads crop pattern time series from HydroBase for CU Locations.  
Crop patterns for a CU Location are defined by crop name (type), area, and year.  
All available years will be read, unless an input period is specified.  
Crop patterns defined with `SetCropPatternTS(...,ProcessWhen=WithParcels,...)` and `SetCropPatternTSFromList(...,ProcessWhen=WithParcels,...)` also will be processed as data are read from HydroBase.

CU location ID:  Required - locations to process (use \* for wildcard).

Input start (year):  Optional - starting year to read data (blank for full period).

Input end (year):  Optional - ending year to read data (blank for full period).

Command:  

```
ReadCropPatternTSFromHydroBase ( ID="*" )
```

Cancel OK

**ReadCropPatternTSFromHydroBase() Command Editor**

ReadCropPatternTSFromHydroBase\_True

The command syntax is as follows:

```
ReadCropPatternTSFromHydroBase( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
InputStart	Starting year to read data.	All available data will be read.
InputEnd	Ending year to read data.	All available data will be read.

The following command file illustrates how to create a crop pattern time series file:

```
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
# Step 4 - Need to translate crops out of HB to include TR21 suffix
# Translate all crops from HB to include .TR21 suffix
TranslateCropPatternTS(ID="*",OldCropType="GRASS_PASTURE",NewCropType="GRASS_PASTURE.TR21")
TranslateCropPatternTS(ID="*",OldCropType="CORN_GRAIN",NewCropType="CORN_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SMALL_GRAINS",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="VEGETABLES",NewCropType="VEGETABLES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WO_COVER",NewCropType="ORCHARD_WO_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ORCHARD_WITH_COVER",NewCropType="ORCHARD_WITH_COVER.TR21")
TranslateCropPatternTS(ID="*",OldCropType="DRY_BEANS",NewCropType="DRY_BEANS.TR21")
TranslateCropPatternTS(ID="*",OldCropType="GRAPES",NewCropType="GRAPES.TR21")
TranslateCropPatternTS(ID="*",OldCropType="WHEAT",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SUNFLOWER",NewCropType="SPRING_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="SOD_FARM",NewCropType="GRASS_PASTURE.TR21")
# Step 5 - Translate crop names
# use high-altitude coefficients for structures with more than 50% of
# irrigated acreage above 6500 feet
TranslateCropPatternTS(ListFile="cm2005_HA.lst",IDCol=1,
    OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE.DWHA")
# Step 6 - Fill Acreage
# Fill SW structure acreage backward from 1999 to 1950
# Fill acreage forward for all structures from 2000 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1993,FillDirection=Backward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1993,FillEnd=1999,FillDirection=Forward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2000,FillEnd=2006,FillDirection=Forward)
# Step 7 - Write final *.cds file
WriteCropPatternTSToStateCU(OutputFile="..\StateCU\cm2006.cds",
    WriteCropArea=True,WriteHow=OverwriteFile)
# Check the results
CheckCropPatternTS(ID="*")
WriteCheckFile(OutputFile="cm2006.cds.StateDMI.check.html")
```

---

# Command Reference: ReadCropPatternTSFromStateCU()

Read crop pattern time series data from a StateCU file

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `ReadCropPatternTSFromStateCU()` command reads crop pattern time series data from a StateCU crop pattern time series file and defines crop patterns in memory. The crop pattern time series can then be manipulated and output with other commands. This command can be used to adjust an existing crop pattern file or to set the total acreage in the irrigation practice time series file (see the `SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()` command). **Warning: when writing the crop pattern time series, the total acreage and the fractions corresponding to each crop (three digits) are written. The acreage for each crop is also now written but was not included in older versions of files. When reading the file with this command, the default is to read the individual crop acreages and the total and fractions are computed based on the individual crop acreages. Because the fraction is only three digits, crop areas computed from the total and fraction may differ from the raw crop acreages. Consequently, comparing old and new files may result in differences.**

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadCULocationsFromStateMod() Command**

This command reads CU locations from a StateMod direct diversion station file or well station file.  
A CU Location is a location where water requirement is estimated.  
Only the list of identifiers and names are read (latitude, region1, etc., must be assigned with other commands).  
Only diversions and wells with the following agricultural demand source (demsrc) are read:  
1 (GIS), 2 (TIA), 3 (GIS primary), 4 (TIA primary), and 8 (user defined).  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCULocationsFromStateCU

StateMod file:

Command:

ReadCropPatternTSFromStateCU

**ReadCropPatternTSFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadCropPatternTSFromStateCU(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
Version	A StateCU version, to allow backward compatibility with files from an older software version. Refer to StateCU documentation for a description of older file formats.	Use the file format for the most current StateCU version.
ReadDataFrom	Indicate how to read crop values, one of: <ul style="list-style-type: none"> <li>CropArea – read the detailed crop acreage values from the file (may not be available in very old files)</li> <li>TotalAreaAndCropFraction – read the total area and crop fractions and compute the crop area from this information. Because fractions are only 3 digits, the crop areas will only be accurate to three digits (and may therefore not agree with HydroBase or other input data).</li> </ul>	CropArea

---

# Command Reference: ReadCULocationsFromList()

Read CU Locations data from a list file

**StateCU Command**

Version 3.08.02, 2010-01-07

The `ReadCULocationsFromList()` command reads a list of CU Locations from a delimited list file and defines CU Locations in memory. The CU Locations can then be manipulated and output with other commands. The identifier column is required.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadCULocationsFromList() Command**

This command reads CU locations from a list file containing columns of information.  
A CU Location is a location where water requirement is estimated.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCULocationsFromList

List file:

ID column:	<input type="text" value="1"/>	Required - column (1+) for station identifier.
Name column:	<input type="text" value="6"/>	Optional - column (1+) for station name.
Latitude column:	<input type="text"/>	Optional - column (1+) for latitude.
Elevation column:	<input type="text"/>	Optional - column (1+) for elevation.
Region1 column:	<input type="text"/>	Optional - column (1+) for region 1 (e.g., county).
Region2 column:	<input type="text"/>	Optional - column (1+) for region 2 (e.g., HUC).
AWC column:	<input type="text"/>	Optional - column (1+) for AWC.

Command: 

```
ReadCULocationsFromList (ListFile="cmstrlist.csv", IDCol=1, NameCol=6)
```

ReadCULocationsFromList

**ReadCULocationsFromList() Command Editor**

The command syntax is as follows:

```
ReadCULocationsFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the CU Location identifiers.	None – must be specified.
NameCol	The column number (1+) containing the CU Location name.	The name will be initialized to blank.
LatitudeCol	The column number (1+) containing the CU Location latitude.	The latitude will be missing.
ElevationCol	The column number (1+) containing the CU Location elevation.	The elevation will be missing.
Region1Col	The column number (1+) containing the CU Location Region1.	The region1 will be initialized to blank.
Region2Col	The column number (1+) containing the CU Location Region2.	The region2 will be initialized to blank.
AWCCol	The column number (1+) containing the CU Location AWC.	The AWC will be missing.

At a minimum, the list file must contain a column with CU Location identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
#wdid,lat,long,county,huc,name,ceff,fleff,speff,gmode,awc
360645,39.84,-106.35,SUMMIT,14010002,GUTHRIE THOMAS DITCH,1,0.6,0.8,2,0.08
360649,39.78,-106.18,SUMMIT,14010002,HAMILTON DAVIDSON DITCH,1,0.6,0.8,2,0.08
360660,39.8,-106.16,SUMMIT,14010002,HIGH MILLER DITCH,1,0.6,0.8,2,0.12
360662,39.97,-106.38,SUMMIT,14010002,HOAGLAND CANAL,1,0.6,0.8,2,0.12
360671,39.74,-106.13,SUMMIT,14010002,INDEPENDENT BLUE DITCH,1,0.6,0.8,2,0.13
360687,39.77,-106.18,SUMMIT,14010002,KIRKWOOD DITCH,1,0.6,0.8,2,0.08
360709,40.01,-106.38,GRAND,14010002,LOBACK DITCH,1,0.6,0.8,2,0.09
360725,39.82,-106.25,SUMMIT,14010002,MARY DITCH,1,0.6,0.8,2,0.08
360728,39.83,-106.25,SUMMIT,14010002,MAT NO 1 DITCH,1,0.6,0.8,2,0.08
```

---

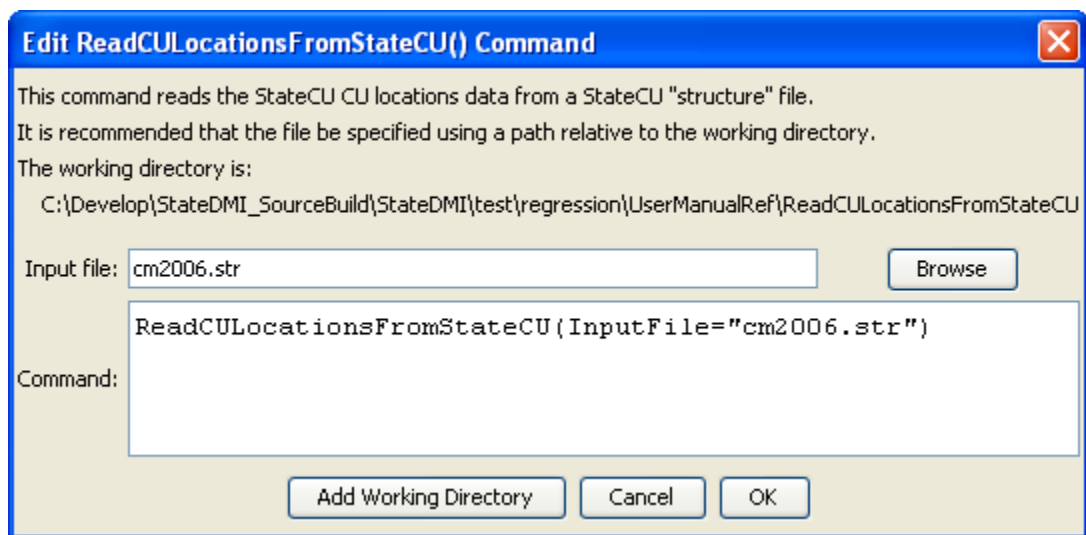
# Command Reference: ReadCULocationsFromStateCU()

Read CU Locations data from a StateCU structure file

## StateCU Command

Version 3.08.02, 2010-01-07

The ReadCULocationsFromStateCU( ) command reads a list of CU Locations from a StateCU structure file and defines CU Locations in memory. The CU Locations can then be manipulated and output with other commands. This command can be used to adjust an existing CU Locations file. The following dialog is used to edit the command and illustrates the syntax of the command.



ReadCULocationsFromStateCU

**ReadCULocationsFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadCULocationsFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.



---

# Command Reference: ReadCULocationsFromStateMod()

**Read CU Locations data from a StateMod diversion or well stations file**

**StateCU Command**  
Version 3.08.02, 2010-01-02

The `ReadCULocationsFromStateMod()` command reads a list of CU Locations from a StateMod diversion or well stations file and defines CU Locations in memory. The CU Locations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadCULocationsFromStateMod() Command**

This command reads CU locations from a StateMod direct diversion station file or well station file.  
A CU Location is a location where water requirement is estimated.  
Only the list of identifiers and names are read (latitude, region1, etc., must be assigned with other commands).  
Only diversions and wells with the following agricultural demand source (demsrc) are read:  
1 (GIS), 2 (TIA), 3 (GIS primary), 4 (TIA primary), and 8 (user defined).  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCULocationsFromStateCU

StateMod file:

Command:

ReadCULocationssFromStateMod

**ReadCULocationsFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadCULocationsFromStateMod( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.

---

# Command Reference: ReadDelayTablesMonthlyFromStateMod()

Read delay tables (monthly) data from a StateMod file

StateCU, StateMod Command

Version 3.09.01, 2010-02-01

The `ReadDelayTablesMonthlyFromStateMod()` command reads delay tables (monthly) from a StateMod delay tables file. For example, this command may be used to convert a delay table file between fraction and percent.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadDelayTablesMonthlyFromStateMod() Command**

This command reads monthly delay tables from a StateMod delay tables file.  
Delay tables indicate how returns (or depletions) are distributed over time.  
These data are used in spatial delay/depletion assignments for diversions and wells.  
By default, is assumed that tables are read with values as percent (0 to 100).  
If necessary, use the scale to multiply the values as they are read, to convert fraction (0-1) to percent (0-100).  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateRiverNetworkFromNetwork

StateMod file:

Scale:  Optional - scale to/from fraction/percent (default=no scale)

Command: 

```
ReadDelayTablesMonthlyFromStateMod ( InputFile="cm2005.dly", Scale=100)
```

ReadDelayTablesMonthlyFromStateMod  
**ReadDelayTablesMonthlyFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadDelayTablesMonthlyFromStateMod( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
Scale	If the StateMod delay table values are specified as fractions rather than percent, a scale of 100 can be used to convert the StateMod delay tables to percent.	If not specified, no scale is applied to the delay values.

---

# Command Reference: ReadDiversiionDemandTSMonthlyFromStateMod()

Read diversion demand time series (monthly) data from a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadDiversiionDemandTSMonthlyFromStateMod()` command reads diversion demand time series (monthly). All time series are read, whether or not they match the list of diversion stations.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadDiversiionDemandTSMonthlyFromStateMod() Command**

This command reads diversion demand time series (monthly) from a StateMod time series file.  
Diversion demand time series (monthly) are associated with diversion stations.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversiionDemandTSMonthlyToStateMod

StateMod file:

Command:

ReadDiversiionDemandTSMonthlyFromStateMod

## ReadDiversiionDemandTSMonthlyFromStateMod() Command Editor

The command syntax is as follows:

```
ReadDiversionDemandTSMonthlyFromStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod diversion demand time series (monthly) file to read.	None – must be specified.

---

# Command Reference: ReadDiversioHistoricalTSMonthlyFromHydro Base()

Read diversion historical time series (monthly) data from HydroBase

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadDiversioHistoricalTSMonthlyFromHydroBase()` command reads diversion historical time series (monthly) from HydroBase, for each diversion station identifier that is specified. Diversion comments and structure “currently in use” (CIU) can be checked for each time series in order to set zero values that are not indicated in monthly data. The period from the `SetOutputPeriod()` command is used to request data from HydroBase. After reading, the time series can be manipulated and output with other commands.

If the diversion station is an aggregate/system, the time series from HydroBase will be added together. If filling is not done at read time, then later commands can be used to fill data, operating on the sum. In most cases, it is more technically correct to fill the aggregate/system part time series before adding them together. The summed time series can then also be filled with other commands after the initial read, to account for still missing data (e.g., fill with a constant value of zero). Attempts are made to fill the complete output period.

**Features are not currently enabled to fill the time series for explicit structures during the read – use other fill commands after the read.** Explicit structures and collections are typically processed with separate commands. The `IncludeExplicit` and `IncludeCollections` parameters can be used to indicate which diversion stations should be processed. For example, if pattern filling is done while reading data for aggregate/systems, separate read commands will be needed to specify the pattern identifier for each fill. A single read command can then be used to read the time series for explicit structures.

## Diversion Comment “Not Used” Flag

HydroBase contains diversion comment data with a `not_used` field. If the `not_used` value matches one of the values shown in the following table for an irrigation year (November of the previous year to October of the irrigation year), the diversion (or reservoir release) data for the specified irrigation year can be interpreted as zero (see the **State of Colorado’s Water Commissioner Manual** for more information):

Diversion Comment `not_used` Flag Resulting in Additional Zero Values

<code>not_used</code>	Meaning (reason why diversion is zero)
A	Structure is not usable
B	No water is available
C	Water available, but not taken
D	Water taken in another structure

## Structure “Currently in Use” Flag

The HydroBase structure data contains a “currently in use” (CIU) field. Unlike diversion comments, this is a single value that is consistent with the current status of a structure (it is not a time series). The following CIU values are used.

### Structure CIU Flag Values and Meaning

CIU	Meaning
A	Active structure with contemporary diversion records
B	Structure abandoned by the court
C	Conditional structure
D	Duplicate; ID no longer used
F	Structure used as FROM number; located in another water district
H	Historical structure only-no longer exists or has records, but has historical data
I	Inactive structure which physically exists but no diversion records are kept
N	Non-existent structure with no contemporary or historical records
U	Active structure but diversion records are not maintained

If UseCIU=True is specified for this command, the following logic will be used to fill missing time series values:

1. If the HydroBase CIU value is H or I for the structure associated with the time series:
  - a. Fill using the diversion comments (see above for interpretation of comments).
  - b. The limits of the time series are recomputed based on diversion data and comments, and missing data at the end of the period are filled with zeros.
  - c. Missing data values at the end of the period will be filled with zeros, reflecting the fact that the structure is off-line. These values are not included in historical averages because they do not occur in the active life of the structure.
  - d. Missing data within the data period remain missing, and can be filled with other commands such as FillHistMonthAverage().
  - e. Missing data prior to the first diversion values or comments remain missing, and can be filled with other commands as appropriate, perhaps specific to each location.
2. If in HydroBase CIU=N:
  - a. Fill using the diversion comments (see above for interpretation of comments).
  - b. The limits of the time series are recomputed based on diversion data and comments, and missing data at the beginning of the period are filled with zeros.
  - c. The remaining missing data in the active data period or at the end of the period remain missing and can be filled with using other parameters or commands.

The specific logic for the command is as follows:

Loop through the diversion stations that have been read with previous commands.

1. If the diversion station identifier does not match the given ID pattern, do not complete the following steps.
2. If explicit stations are being processed and the station is not an explicit station, do not complete the following steps.
3. If collection stations (aggregates and systems) are being processed and the station is not a collection, do not complete the following steps.



4. Process the time series for the station:

If a collection (aggregate or system), perform the following by looping through each part (this guarantees that a time series will result, possibly only with missing data):

- a. If a part identifier is not a WDID, generate an error (only WDIDs should be specified as parts). This type of error should not be ignored and should be corrected.
- b. Read the monthly diversion records for the part. If an error occurs (no data), create an empty time series for the part with all missing data. Important – parts must be read from HydroBase. There is no way to substitute another time series for a part.
- c. If requested (`UseDiversionsComments=True`), read the diversion comments and fill missing values with additional zeros for irrigation years where diversion comments are available.
- d. If requested (`FillUsingCIU=True`) also fill with CIU as per the logic described above.
- e. If the first part of an aggregate is being processed, initialize the total time series to the first part. Also initialize the backup copy that contains only observations.
- f. If the second or greater part, add the observations to the backup copy total time series.
- g. If filling has been requested using pattern and/or average, fill the part's time series.
- h. Add the part's time series to the total. This represents the total of filled data, whereas the backup copy contains only the observed values.
- i. If all parts have been processed, calculate the monthly average limits of the backup copy (observations only), which can be used in later fill commands. **This may be a problem with CIU since some zeros should not be in the average.**

If an explicit station, perform the following:

- a. If the station ID is not a WDID, generate a warning. Otherwise, read the time series diversion records from HydroBase.
- b. If the station ID is a WDID and diversion comments were requested (`UseDiversionsComments=True`), read the diversion comments from HydroBase and fill additional values with zeros for irrigation years where diversion comments are available.
- c. If requested (`FillUsingCIU=True`) also fill with CIU as per the logic described above.
- d. If no time series was read, create an empty time series.
- e. Calculate the monthly average limits of the time series, which can be used in later fill commands. **Need to evaluate CIU impact.**

5. Add the time series to the list of time series being maintained for output.

6. Add a copy of the time series to the backup, to be used to set observed values when processing the `LimitDiversionsHistoricalTSMonthlyToRights()` command. Since explicit stations' time series are not filled, copy the time series as is. For aggregate time series, use the backup copy of the time series.

The following command combinations will provide the same results:

- Process all diversion stations with one command (`ID="*"`) with no filling.
- Process explicit diversion stations with one command (`ID="*", IncludeCollections=False`) and collections with one or more commands (`ID="X*", IncludeExplicit=False`), with no filling.

If historical patterns are not used for filling, then a smaller number of commands can be used.

Note that a time series is automatically added only if the station ID is initially matched by the ID pattern. A station that does not match any of the ID patterns in read commands will not automatically have a time series added.

If multiple read commands are used, it may be necessary to use a `SortDiversionHistoricalTSMonthly()` command to sort time series to match the diversion stations.

The following dialog is used to edit the command and illustrates the syntax of the command, when processing explicit structures (no aggregates or collections):

**Edit ReadDiversionHistoricalTSMonthlyFromHydroBase() Command**

This command reads diversion historical monthly time series from HydroBase, using the diversion station identifiers to find data. The available period is read if no period is provided.

Explicit diversion stations and collections (aggregates and systems) can be included or can be skipped.

If data are not found in HydroBase, a time series with missing data is added for the station.

For diversion aggregates/systems, the data are added. Missing data are ignored.

Each aggregate/system part time series can each be filled using averages. The filled time series are then added.

Diversion station ID:  Required - diversion stations to read (use \* for wildcard).

Include explicit:  Optional - process explicit diversion stations? (default=True).

Include collections:  Optional - process aggregates/systems? (default=True).

<= zero values in average?:  Optional - are values <= 0 used in averages (used in filling)? (default=True).

Use diversion comments:  Optional - use diversion comments for more zero values? (default=True).

Fill using CIU:  Optional - use currently in use CIU for more zeros? (default=False).

Fill CIU flag:  Optional - 1-character (or "Auto") flag to indicate fill (default=none).

Period to read (optional):  to

If filling aggregates/systems:

Pattern identifier:  Required if filling with pattern - pattern ID to use for filling aggregates/systems.

Fill pattern order:  Optional - order to use pattern filling (default=no fill).

Pattern fill flag:  Optional - 1-character flag to use for filled values, or "Auto".

Fill average order:  Optional - order to use monthly average filling (default=no fill).

Average fill flag:  Optional - 1-character flag to use for filled values.

Command: 

```
ReadDiversionHistoricalTSMonthlyFromHydroBase ( ID="*", UseDiversionComments=True )
```

OK Cancel

ReadDiversionHistoricalTSMonthlyFromHydroBase\_Explicit

### ReadDiversionHistoricalTSMonthlyFromHydroBase() Command Editor for Explicit Diversions

The following dialog is used to edit the command and illustrates the syntax of the command, when processing collections (aggregates or collections):

**Edit ReadDiversiionHistoricalTSMonthlyFromHydroBase() Command**
✖

This command reads diversion historical monthly time series from HydroBase, using the diversion station identifiers to find data. The available period is read if no period is provided.  
 Explicit diversion stations and collections (aggregates and systems) can be included or can be skipped.  
 If data are not found in HydroBase, a time series with missing data is added for the station.  
 For diversion aggregates/systems, the data are added. Missing data are ignored.  
 Each aggregate/system part time series can each be filled using averages. The filled time series are then added.

Diversion station ID: <input type="text" value="36*"/>	Required - diversion stations to read (use * for wildcard).
Include explicit: <input type="button" value="False"/>	Optional - process explicit diversion stations? (default=True).
Include collections: <input type="button"/>	Optional - process aggregates/systems? (default=True).
<= zero values in average?: <input type="button"/>	Optional - are values <= 0 used in averages (used in filling)? (default=True).
Use diversion comments: <input type="button" value="True"/>	Optional - use diversion comments for more zero values? (default=True).
Fill using CIU: <input type="button"/>	Optional - use currently in use CIU for more zeros? (default=False).
Fill CIU flag: <input type="text"/>	Optional - 1-character (or "Auto") flag to indicate fill (default=none).
Period to read (optional): <input type="text"/> to <input type="text"/>	

If filling aggregates/systems:

Pattern identifier: <input type="text" value="09037500"/>	Required if filling with pattern - pattern ID to use for filling aggregates/systems.
Fill pattern order: <input type="button" value="1"/>	Optional - order to use pattern filling (default=no fill).
Pattern fill flag: <input type="text"/>	Optional - 1-character flag to use for filled values, or "Auto".
Fill average order: <input type="button" value="2"/>	Optional - order to use monthly average filling (default=no fill).
Average fill flag: <input type="text"/>	Optional - 1-character flag to use for filled values.

Command:

```
ReadDiversiionHistoricalTSMonthlyFromHydroBase ( ID="36*", IncludeExplicit=False, UseDiversionComments=True, PatternID="09037500", FillPatternOrder=1, FillAverageOrder=2)
```

ReadDiversiionHistoricalTSMonthlyFromHydroBase\_Explicit

### ReadDiversiionHistoricalTSMonthlyFromHydroBase() Command Editor for Aggregates/Systems

The command syntax is as follows:

`ReadDiversionHistoricalTSMonthlyFromHydroBase( Parameter=Value, ... )`

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IncludeExplicit	Indicates whether explicit diversion stations (those that are not aggregates or systems) should be included in processing.	True
IncludeCollections	Indicates whether diversion stations that are collections (aggregates and systems) should be included in processing.	True
LEZeroInAverage	Indicate whether values $\leq 0$ should be considered when computing historical averages.	True
UseDiversionComments	Indicate whether diversion comments should be checked when reading time series data. Diversion comments may indicate additional zero values.	True
FillUsingCIU	Indicates whether the “currently in use” (CIU) information is used to fill missing data. This will result in additional zeros at the beginning or end of the time series, depending on CIU value. See the description of the logic above.	False (CIU information is not used to fill missing data).
FillUsingCIUFlag	For each missing data value that is filled using the CIU information, tag the filled value as follows: <ul style="list-style-type: none"> <li>If FillUsingCIUFlag is specified as a single character, tag filled values with the specified character.</li> <li>If FillUsingCIUFlag=Auto is specified, the CIU value (H, I, or N) from HydroBase is used for the flag.</li> </ul> The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.
ReadStart	A date, to monthly precision, indicating the start of the read.	Read all available data.
ReadEnd	A date, to monthly precision, indicating the end of the read.	Read all available data.
PatternID	The pattern identifier for data read with the ReadPatternFile() command.	None – filling with monthly average pattern is not done.
FillPatternOrder	If filling aggregates and systems during the read, specify the order that monthly average pattern filling should occur.	None – filling with monthly average pattern is not done.
PatternFillFlag	If specified as a single character, data flags will be enabled for the time series and each value	No flag is assigned.

Parameter	Description	Default
	filled using a pattern will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	
FillAverageOrder	If filling aggregates and systems during the read, specify the order that monthly average filling should occur.	None – filling with monthly average is not done.
AverageFillFlag	If specified as a single character, data flags will be enabled for the time series and each value filled using the historical average will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary.	No flag is assigned.

The following abbreviated command file illustrates how the StateMod diversion historical time series file can be produced. Note that an initial diversion stations file is read and is then updated based on time series information.

```

StartLog(LogFile="ddh.commands.StateDMI.log")
# ddh.commands.StateDMI
#
# StateDMI command file to create the historical diversion file
# and the "step 2" direct diversion structure file, updated so structure
# capacity = maximum historical diversion
#
# Step 1 - set time-series period and year type
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read structure list from preliminary direct diversion structure file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 3 - read aggregate and diversion system structure assignments. Note that
# want to combine historical diversions for aggs and diversion systems, but
# historical diversions are separate for primary and secondary components
# of multistructures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
PartsListedHow=InRow)
#
# Step 4 - read historical diversions from HydroBase. Note that want individual structures
# in aggregates and diversion systems to be filled first, then diversions
# combined.
#
ReadDiversionHistoricalTSMonthlyFromHydroBase(ID="*",IncludeCollections=False,
UseDiversionComments=True)
#
# Step 5 - read fill pattern file, and assign patterns to water districts
#
ReadPatternFile(InputFile="fill2005.pat")
ReadDiversionHistoricalTSMonthlyFromHydroBase(ID="36*",IncludeExplicit=False,
UseDiversionComments=True,
PatternID="09037500",FillPatternOrder=1,FillAverageOrder=2)

```

```

#
# Step 6 - assign transbasin diversions from streamflow gages
#
SetDiversionHistoricalTSMonthly(ID="364626",TSID="09047300.DWR.Streamflow.Month~HydroBase")
...similar commands omitted...
# note that adams tunnel streamgage ID changed in 10/1996 from 09013000 to ADANETCO
SetDiversionHistoricalTSMonthly(ID="514634",TSID="514634...MONTH~StateMod~514634.stm")
# Con-Hoosier System - Blue River Diversion, driven by operating rules to con-hoosier
summary demand
SetDiversionHistoricalTSMonthly(ID="364683",TSID="364683...MONTH~StateMod~zero.stm")
SetDiversionHistoricalTSMonthly(ID="364699",TSID="364699...MONTH~StateMod~zero.stm")
# Fryingpan-Arkansas Project
SetDiversionHistoricalTSMonthly(ID="381594",TSID="381594...MONTH~StateMod~381594.stm")
SetDiversionHistoricalTSMonthly(ID="384625",TSID="384625...MONTH~StateMod~384625.stm")
SetDiversionHistoricalTSMonthly(ID="954699",TSID="954699...MONTH~StateMod~zero.stm")
...similar commands omitted...
#
# Step 7 - set diversions from external time-series files
#
# The following commands are added to access Task 11.2 replacement files
SetDiversionHistoricalTSMonthly(ID="380757",TSID="380757...MONTH~StateMod~380757.stm")
...similar commands omitted...#
# The following structures are set for Municipal and Industrial Diversions
SetDiversionHistoricalTSMonthly(ID="360784",TSID="360784...MONTH~StateMod~360784.stm")
...similar commands omitted...
#
# Set transbasin diversions to "0" prior to construction
#
# Wurtz Ditch
SetDiversionHistoricalTSMonthlyConstant(ID="374648",Constant=0,SetEnd="01/1929")
...similar commands omitted...
#
# Step 8 - fill historical diversion using pattern approach
#
FillDiversionHistoricalTSMonthlyPattern(ID="36*",PatternID="09034500")
...similar commands omitted...
#
# Step 9 - Fill remaining missing with month average
#
FillDiversionHistoricalTSMonthlyAverage(ID="*")
#
# Step 10 - Limit filled diversion to water rights. Exceptions include structure
# receiving significant reservoir supply, carrier structures, etc.
#
LimitDiversionHistoricalTSMonthlyToRights(InputFile="..\statemod\cm2005.ddy",
ID="*",IgnoreID="954683,952001,950010,950011")
#
# Step 11 - sort structures and create historical diversion file
#
SortDiversionHistoricalTSMonthly(Order=Ascending)
WriteDiversionHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
#
# Step 12 - update capacities and create final direct diversion station file
#
SetDiversionStationCapacitiesFromTS(ID="*")
WriteDiversionStationsToStateMod(OutputFile="..\statemod\cm2005.dds")
#
# Check the results.
CheckDiversionHistoricalTSMonthly(ID="*")
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")

```

---

# Command Reference: ReadDiversiOnHistoricalTSMonthlyFromStateMod()

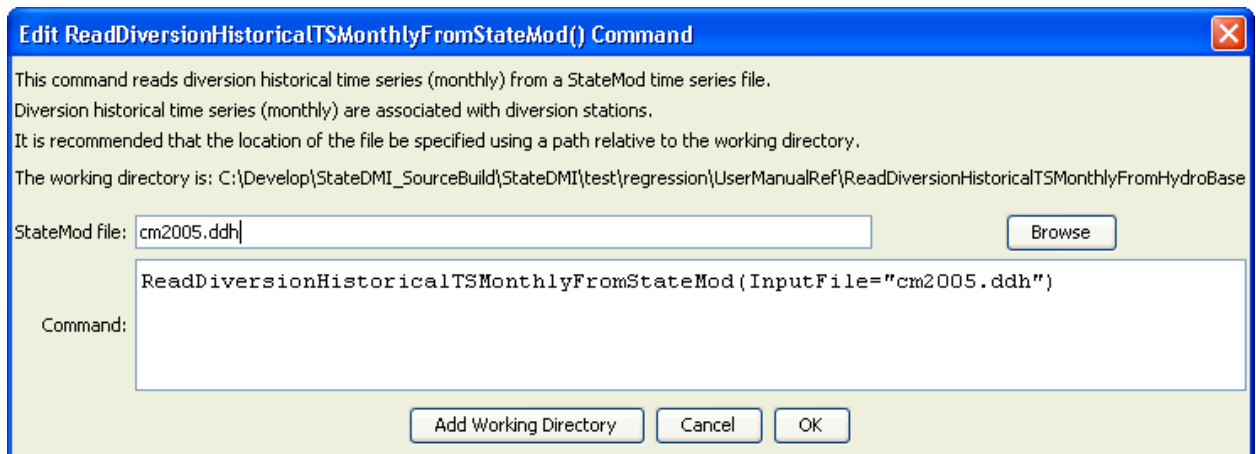
**Read diversion historical time series (monthly) data from a StateMod file**

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadDiversiOnHistoricalTSMonthlyFromStateMod()` command reads diversion historical time series (monthly). This command is used when estimating average efficiencies and calculating demand time series. All time series are read, whether or not they match the list of diversion stations. Copies of the time series are NOT made for use as original data with the `Limit*ToRights()` commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadDiversiOnHistoricalTSMonthlyFromStateMod

## ReadDiversiOnHistoricalTSMonthlyFromStateMod() Command Editor

The command syntax is as follows:

```
ReadDiversionHistoricalTSMonthlyFromStateMod( Parameter=Value,... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod diversion historical time series (monthly) file to read.	None – must be specified.



---

# Command Reference: ReadDiversiionRightsFromHydroBase()

Read diversion right data from HydroBase

StateCU and StateMod Command

Version 3.09.07, 2010-01-26

The `ReadDiversiionRightsFromHydroBase()` command reads diversion net amount water rights from HydroBase, for each diversion station that is defined. The diversion rights can then be manipulated and output with other commands. Within a diversion station, rights are sorted by administration number and order number. In some cases, multiple rights for the diversion station may be listed, each with the same administration number. This is because the order number is different; however, the order number is not listed in the StateMod output. In such cases, the individual rights are retained to allow comparison with HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadDiversiionRightsFromHydroBase() Command**

This command reads diversion rights from HydroBase, using the diversion station identifiers to find rights. Net absolute water rights are read by default. If the diversion stations list contains aggregates, specify the administration number classes to group rights (indicate administration numbers separated by commas). The output water right identifier is assigned as the diversion station identifier + "." + a two digit number.

Diversion station ID:  Required - diversion stations to read (use \* for wildcard).

Decree minimum:  Optional - minimum decree to include (blank = .0005 CFS).

Ignore use type(s):  Optional - Comma-separated HydroBase use types (e.g., "STO,IND") (default=read all).

Admin. number classes:

OnOff default:  Optional - default OnOff switch (default=AppropriationDate).

Command: 

```
ReadDiversiionRightsFromHydroBase (ID="*", OnOffDefault=1, AdminNumClasses="14854.00000,20427.18999,22729.21241,30895.21241,31258.00000,32023.28989,39095.38998,43621.42906,46674.00000,48966.00000,99999.")
```

OK Cancel

ReadDiversiionRightsFromHydroBase

## ReadDiversiionRightsFromHydroBase() Command Editor

If aggregating rights, the following steps occur (diversion systems use steps 1-2 and are then explicitly added):

1. Water rights for each part of the aggregate are read from HydroBase, reporting errors as necessary.
2. The rights are added to a list and are sorted by administration number. This ensures that the cumulative list of rights is listed in order of administration number (in particular, this step is important for diversion systems).

3. Water rights are defined for each class (see the AdminNumClasses parameter description below), initializing the decree to zero.
4. For each class, the following sums are calculated:  $\text{sum}(\text{decree} * \text{AdminNum})$  and  $\text{sum}(\text{decree})$ , where the administration number is determined from the appropriation date derived from the original HydroBase administration number (it will not have a remainder).
5. The final administration number for the class is determined (it will not have a remainder):  
 $\text{int}(\text{sum}(\text{decree} * \text{AdminNum}) / \text{sum}(\text{decree}))$

Water rights that are less than the decree minimum are ignored.

The command syntax is as follows:

`ReadDiversionRightsFromHydroBase( Parameter=Value , ... )`

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DecreeMin	The minimum decree to accept as a valid right.	0.0 – read all rights.
IgnoreUseType	A comma-separated list of HydroBase water right use types to ignore (e.g., STO, IND), needed to prevent double counting of some rights.	Include all right types.
AdminNumClasses	A list of administration numbers, separated by spaces or commas, to define the breaks for aggregate water rights, for diversion aggregates. For example, if the class breaks are 10000.000, 20000.00000, and 99999.99999, the first group will contain water rights with administration numbers $\leq 10000.00000$ , the second will contain water rights with administration number $> 10000.00000$ and $\leq 20000.00000$ , and the third will contain water rights with administration number $> 20000.00000$ and $\leq 99999.99999$ .	If not specified, diversion aggregates will be treated as diversion systems, with all water rights explicitly included in output.
OnOffDefault	Indicates how to set the on/off switch for all water rights that are processed. A value of 1 indicates that the right is on for the whole period. If the value is AppropriationDate, the switch is set to the year corresponding to the appropriation date, indicating that the right will be turned on starting in the year. Use set commands to reset the switch to other values.	Appropriation Date

---

# Command Reference: ReadDiversiionRightsFromStateMod()

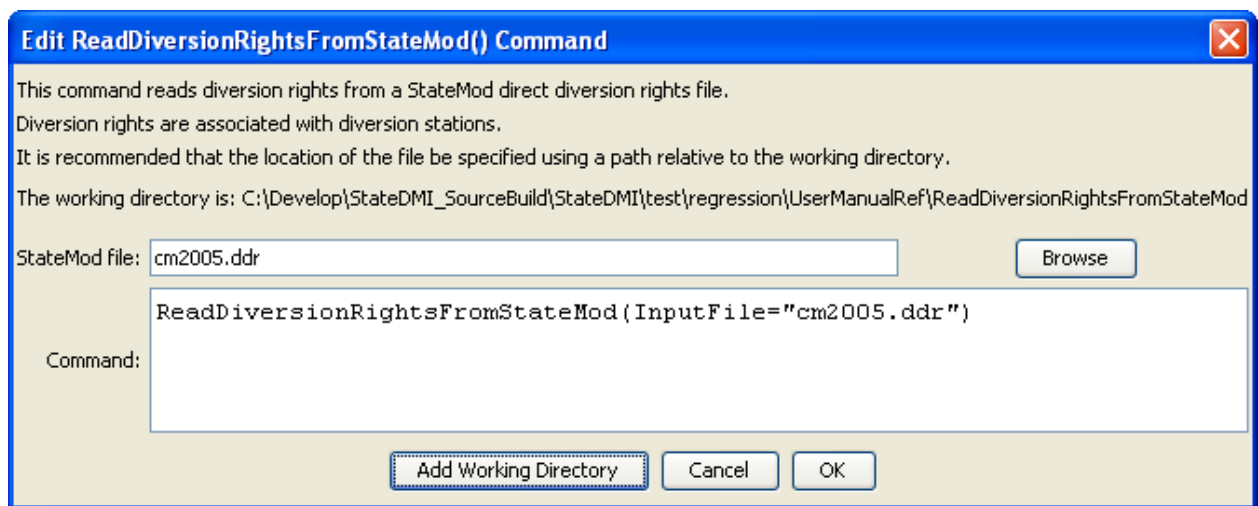
Read diversion right data from a StateMod diversion rights file

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `ReadDiversiionRightsFromStateMod()` command reads diversion rights from a StateMod diversion rights file. The diversion rights can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadDiversiionRightsFromStateMod

### ReadDiversiionRightsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadDiversionRightsFromStateMod( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod diversion rights file to be read.	None – must be specified.

---

# Command Reference: ReadDiversionsFromList()

Read diversion stations data from a list file

**StateCU and StateMod Command**

Version 3.09.01, 2010-01-26

The `ReadDiversionsFromList()` command reads a list of diversion stations from a delimited list file and defines diversion stations in memory. The diversion stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadDiversionsFromList() Command**

This command reads diversion stations from a list file containing columns of information. Diversion stations indicate locations where water is diverted from a river, lake, or reservoir. Columns should be delimited by commas (user-specified delimiters will be added in the future). Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversionsFromStateMod

List file:

ID column:  Required - column (1+) for ID.

Name column:  Optional - column (1+) for name.

Command: 

```
ReadDiversionsFromList (ListFile="rgdssall.csv", IDCol=1, NameCol=5)
```

ReadDiversionsFromList

**ReadDiversionsFromList() Command Editor**

The command syntax is as follows:

```
ReadDiversionStationsFromList(Parameter=value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the diversion station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the diversion station names.	None – optional (name will be initialized to blank).

At a minimum, the list file must contain a column with diversion station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
# Diversions as list file
#
#
"ID","Latitude","County","HUC","Name"
200505,37.5,ALAMOSA,13010002,ALAMOSA D
200511,37.68,RIO GRANDE,13010001,ANACONDA D
200512,37.61,RIO GRANDE,13010002,ANDERSON D
200513,37.68,RIO GRANDE,13010002,ANNA RABER D
...
```

---

# Command Reference: ReadDiversionsFromNetwork()

Read diversion station data from a network file

StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `ReadDiversionsFromNetwork()` command reads a list of diversion stations from a StateMod network file (XML or Makenet) and defines diversion stations in memory. The diversion stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadDiversionsFromNetwork() Command**

This command reads diversion station data from a StateMod XML network file.  
Diversion stations indicate locations where water is diverted from a river, lake, or reservoir.  
Diversion and D&W (diversion with well[s]) nodes are read from the network file.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
River node ID - set to station ID  
Daily ID - set to the nearest downstream streamflow station ID.  
See also the `Fill*FromNetwork()` commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversionsFromNetwork

StateMod network file:

Command:

ReadDiversionsFromNetwork

**ReadDiversionsFromNetwork() Command Editor**

The command syntax is as follows:

```
ReadDiversionStationsFromNetwork( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to be read.	None – must be specified.



---

# Command Reference: ReadDiversionsFromStateMod()

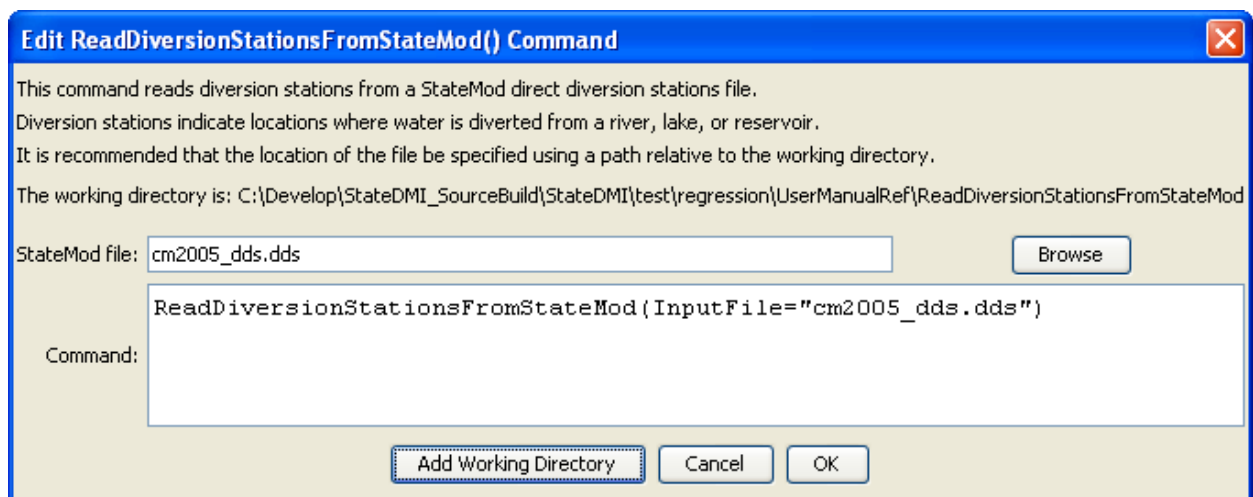
Read diversion station data from a StateMod diversion stations file

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `ReadDiversionsFromStateMod()` command reads a list of diversion stations from a StateMod diversion stations file and defines diversion stations in memory. The diversion stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadDiversionsFromStateMod

## ReadDiversionsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadDiversionStationsFromStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod diversion stations file to be read.	None – must be specified.

---

# Command Reference:

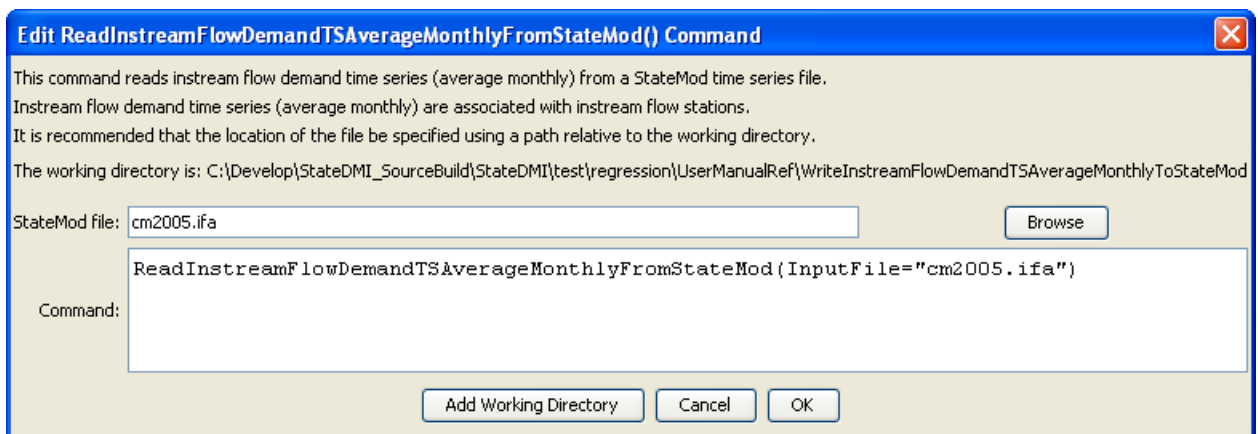
## ReadInstreamFlowDemandTSAverageMonthlyFromStateMod()

**Read instream flow demand time series (average monthly) data from a StateMod instream flow demand time series (average monthly) file**

**StateMod Command**  
Version 3.09.01, 2010-02-02

The `ReadInstreamFlowDemandTSAverageMonthlyFromStateMod()` command reads instream flow demand time series (average monthly) from a StateMod instream flow demand time series (average monthly) file. The instream flow demand time series can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadInstreamFlowDemandTSAverageMonthlyFromStateMod  
**ReadInstreamFlowDemandTSAverageMonthlyFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadInstreamFlowDemandTSAverageMonthlyFromStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod instream flow demand time series (average monthly) file to be read.	None – must be specified.

---

# Command Reference: ReadInstreamFlowRightsFromHydroBase()

Read instream flow right data from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadInstreamFlowRightsFromHydroBase()` command reads instream flow rights from HydroBase, for each instream flow station that is defined. The instream flow rights can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadInstreamFlowRightsFromHydroBase() Command**

This command reads instream flow rights from HydroBase, using the instream flow station identifiers to find rights. Net absolute water rights are read by default. The output water right identifier is assigned as the instream flow station identifier + "." + a two digit number.

Instream flow station ID:  Required - instream flow stations to read (use \* for wildcard).

OnOff default:  Optional - default OnOff switch (default=AppropriationDate).

Command:

OK Cancel

ReadInstreamFlowRightsFromHydroBase

**ReadInstreamFlowRightsFromHydroBase() Command Editor**

The command syntax is as follows:

```
ReadInstreamFlowRightsFromHydroBase( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
OnOffDefault	Indicates how to set the on/off switch for all water rights that are processed. A value of 1 indicates that the right is on for the whole period. If the value is AppropriationDate, the switch is set to the year corresponding to the appropriation date, indicating that the right will be turned on starting in the year. Use set commands to reset the switch to other values.	Appropriation Date

---

# Command Reference: ReadInstreamFlowRightsFromStateMod()

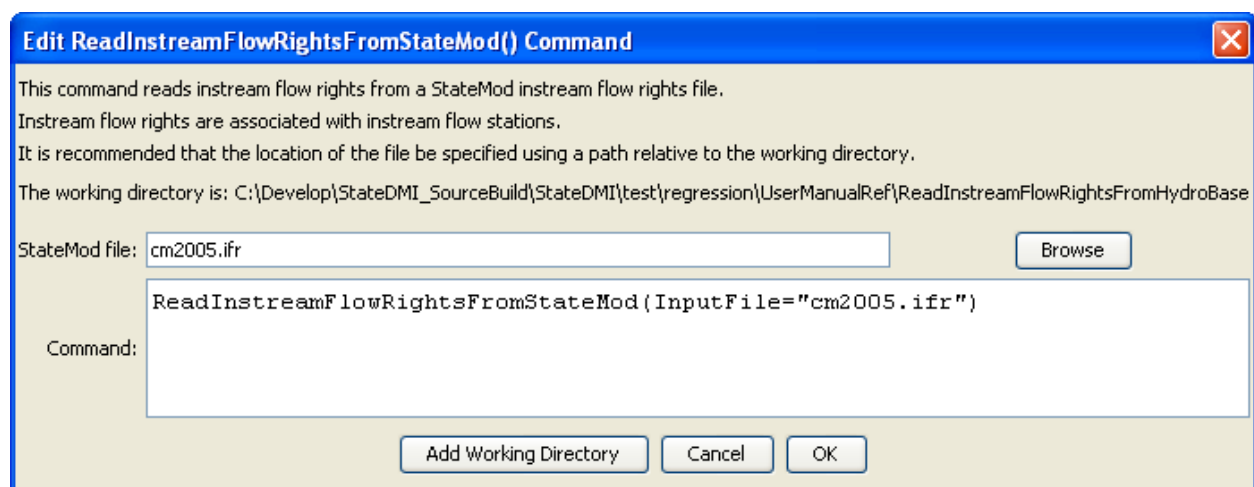
Read instream flow right data from a StateMod instream flow rights file

## StateMod Command

Version 3.09.01, 2010-02-02

The `ReadInstreamFlowRightsFromStateMod()` command reads instream flow rights from a StateMod instream flow rights file. The instream flow rights can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadInstreamFlowRightsFromStateMod

### ReadInstreamFlowRightsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadInstreamFlowRightsFromStateMod( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod instream flow rights file to be read.	None – must be specified.



---

# Command Reference: ReadInstreamFlowStationsFromList()

Read instream flow station data from a list file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadInstreamFlowStationsFromList()` command reads a list of instream flow stations from a delimited list file and defines instream flow stations in memory. The instream flow stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadInstreamFlowStationsFromList() Command**

This command reads instream flow stations from a list file containing columns of information. Instream flow stations indicate locations where surface water flow can be associated with a minimum flow constraint. Columns should be delimited by commas (user-specified delimiters will be added in the future). Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillInstreamFlowStationsFromHydroBase

List file:

ID column:  Required - column (1+) for ID.

Name column:  Optional - column (1+) for name.

Command:

ReadInstreamFlowStationsFromList

**ReadInstreamFlowStationsFromList() Command Editor**

The command syntax is as follows:

```
ReadInstreamFlowStationsFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the stream gage station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the stream gage station names.	None – optional (name will be initialized to blank).

At a minimum, the list file must contain a column with instream flow station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
# Stream gage stations as a list file
#
"ID", "Name"
IFS1, "INSTREAM FLOW REACH 1"
IFS2, "INSTREAM FLOW REACH 2"
...
```

---

# Command Reference: ReadInstreamFlowStationsFromNetwork()

Read instream flow station data from a network file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadInstreamFlowStationsFromNetwork()` command reads a list of instream flow stations from a StateMod network file (XML or old Makenet) and defines instream flow stations in memory. The instream flow stations can then be manipulated and output with other commands. Instream flow stations are actually modeled as a reach defined by upstream and downstream nodes. Both nodes must be included in the network but the instream flow station file has a single record for each reach.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadInstreamFlowStationsFromNetwork() Command**

This command reads instream flow station data from a StateMod XML network file.  
Instream flow stations indicate locations where surface water flow can be associated with a minimum flow constraint.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
Upstream river node ID is set to the station ID.  
Downstream river node ID is set to the station ID + "\_Dwn" (these nodes must be defined in the network)  
See also the Fill\*FromNetwork() commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillInstreamFlowStationsFromHydroBase

StateMod network file:

Command: 

```
ReadInstreamFlowStationsFromNetwork( InputFile="..\Network\cm2005.net")
```

ReadInstreamFlowStationsFromNetwork

**ReadInstreamFlowStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
ReadInstreamFlowStationsFromNetwork( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to be read.	None – must be specified.

---

# Command Reference:

## ReadInstreamFlowStationsFromStateMod()

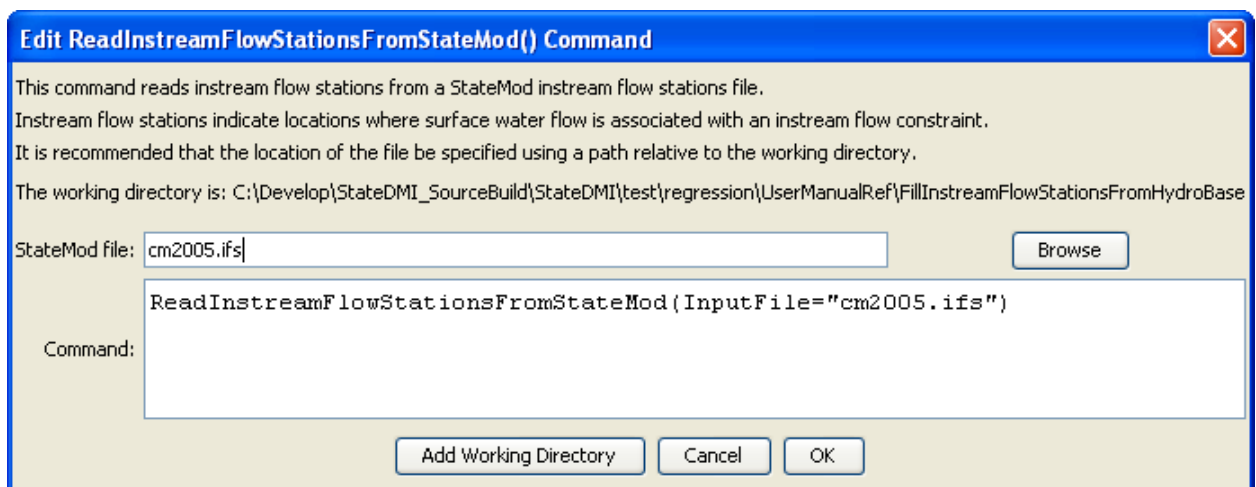
Read instream flow station data from a StateMod instream flow stations file

### StateMod Command

Version 3.09.01, 2010-02-01

The `ReadInstreamFlowStationsFromStateMod()` command reads a list of instream flow stations from a StateMod instream flow stations file and defines instream flow stations in memory. The instream flow stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadInstreamFlowStationsFromStateMod

### ReadInstreamFlowStationsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadInstreamFlowStationsFromStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod instream flow stations file to be read.	None – must be specified.

---

# Command Reference: ReadIrrigationPracticeTSFromHydroBase()

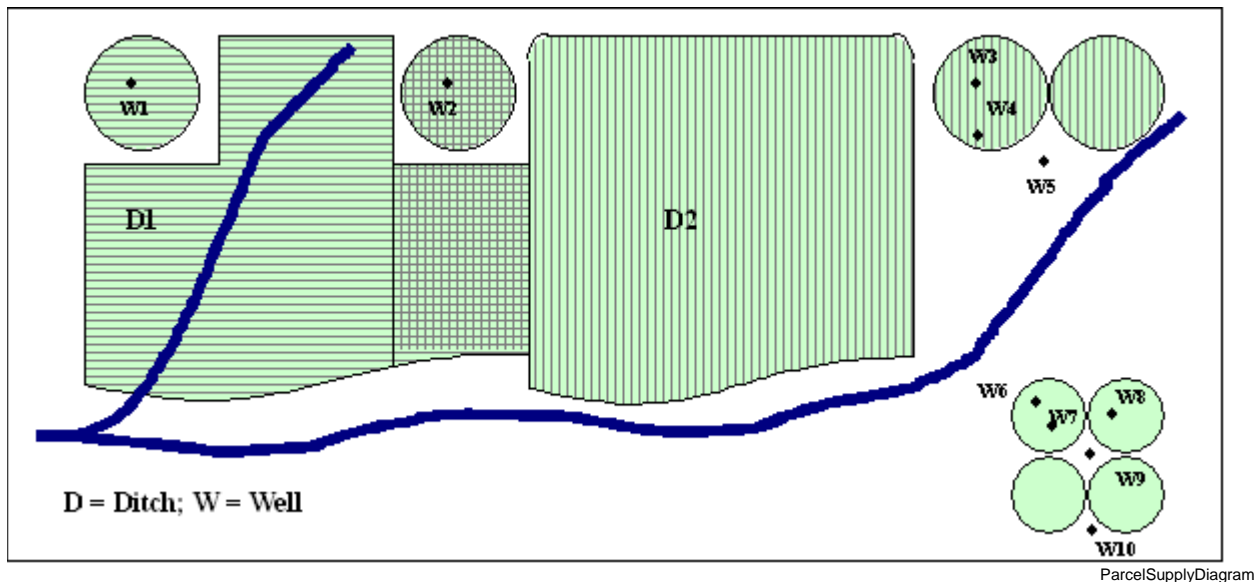
Read irrigation practice time series (yearly) from HydroBase

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadIrrigationPracticeTSFromHydroBase()` command reads parcel acreage data from HydroBase and sets the irrigation practice acreage time series (yearly) information for CU locations. Only years with data in HydroBase are processed. Values for other years must be estimated using other commands (see `FillIrrigationPracticeTS*()`). This command should be executed before other commands that estimate or set irrigation practice acreage time series.

The following figure illustrates possible water supply for parcels.



**Example Supply for Parcels**

In this example, two ditches (D1 and D2, each represented with different hatching) provide surface water supply to the indicated parcels. In some cases, only one ditch provides supply. Between the ditches, both supply water to shared parcels. Wells can supplement surface water supply (parcels above the river) or can be the sole supplier of water (lower right) and wells do not need to be physically located on a parcel to provide supply to the parcel. For StateCU, well-only lands are identified by CU locations that are defined by a collection (aggregate/system) of parcels. For StateMod, well-only lands are well stations that do not have a related diversion station. In both cases, lands irrigated by surface water are identified with ditch identifiers and parcels are associated with the ditches in HydroBase. Typically, well-only lands are grouped and multiple wells provide supply to the collection of parcels. Processing logic is different for ditch and well-only lands only in how the list of parcels is obtained.

The steps used to process irrigation practice time series are described below. Note that “CU location” refers to the StateCU model identifier (which can be a collection of wells) and “well” refers to a hole in the ground that has physical characteristics, water rights, and/or well permits.

Loop through each CU location that matches the ID pattern and perform the following:

For each year being processed (specified by the `Year` parameter or by default all available data in HydroBase), perform the following:

1. Initialize all irrigation practice acreage time series to zero. Consequently, if no data are found in a year, an “observation” of zero acreage will occur. Any previous data are reset.
2. Get the list of parcels associated with the location (note that in a given year there may be zero or more parcels associated with a location):
  - a. If the location is a groundwater only location, get the list of parcels from the aggregate/system definitions.
  - b. If the location is a ditch that is supplemented by groundwater (assumed as possible because the StateCU CU Location data does not indicate whether a location has groundwater supply and all may – one purpose of the IPY file is to indicate groundwater supply over time):
    - i. If the ditch is explicit (no aggregate/system information has been provided for the location), get the list of parcels associated with the single ditch.
    - ii. If the ditch is an aggregate/system, get the list of parcels associated with each part of the aggregate/system and form one list of parcels.
3. Read the parcel data using the parcel identifiers.
  - a. Query HydroBase to get the parcel data, using the year, division, and parcel identifier.
  - b. Acreage not in HydroBase is appended to the parcel list. This acreage is supplied by `SetIrrigationPracticeTS()` and `SetIrrigationPracticeTSFromList()` commands with the `ProcessWhen=WithParcels` parameter. These commands must be specified before the `ReadIrrigationPracticeTSFromHydroBase()` command. For the sake of processing, user-supplied acreage is treated as one parcel for the specified year.
4. Process the parcels for the location:
  - a. If the parcel was associated with a ditch, the parcel area is multiplied by the ditch service area percent irrigated value (actually a fraction in HydroBase), reflecting the fact that only a portion of the parcel area is associated with the location.
  - b. The appropriate irrigation practice acreage time series are incremented. Total acreage is always incremented. For IPY acreage purposes, SPRINKLER and DRIP irrigation methods are treated as SPRINKLER (high efficiency) and all other irrigation methods as FLOOD (low efficiency). A parcel is considered to have groundwater supply if there is at least one well associated with the parcel for the specific year (use supplied data must specify whether the supply is ground or surface water). The combination of irrigation method and whether ground/surface supply indicates the acreage time series that are incremented. If a location does not have groundwater supply, it has surface supply only and the surface water acreage values are incremented.
  - c. The total groundwater acres are set to the sum of the acres for SPRINKLER and FLOOD. The total surface water acres are set to the sum of the acres for SPRINKLER and FLOOD.



The following dialog is used to edit the command and illustrates the syntax of the command.

ReadIrrigationPracticeFromHydroBase

### ReadIrrigationPracticeTSFromHydroBase() Command Editor

The command syntax is as follows:

`ReadIrrigationPracticeTSFromHydroBase (Parameter=Value,...)`

### Command Parameters

Parameter	Description	Default
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Div	A water division to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems.	None – must be specified.
Year	A calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems. Separate multiple years by commas.	All years in HydroBase will be processed.

The following command file illustrates how to process the irrigation practice time series file where groundwater supply is not used:

```
# Step 1 - Set output period and read CU locations from structure file
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
# Set Max SW Eff = 1.0
```

```

SetIrrigationPracticeTS(ID="*",SurfaceDelEffMax=1.0,FloodAppEffMax=.60,
    SprinklerAppEffMax=.80,PumpingMax=0,GWMode=2)
SetIrrigationPracticeTSFromList(ListFile="cmstrlist.csv",ID="*",SetStart=1950,
    SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="7",FloodAppEffMaxCol="8",SprinklerAppEffMaxCol="9")
# Step 6 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Year="1993,2000",Div="5")
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="..\StateCU\cm2006.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
# Step 9 - Fill all land use acreage
# Fill groundwater acreage first
# Fill surface water sprinkler and flood 1950-2006
# Fill ground water sprinkler and flood 1950-2006
# Step 9a - estimate total GW and total SW
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1993",FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2000",FillEnd="2006",FillDirection="Forward")
# Step 9b - fill remaining irrigation method values
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-SurfaceWaterOnlySprinkler",
    FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1993",
    FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2000",
    FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="1950",
    FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="1993",
    FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="2000",
    FillEnd="2006",FillDirection="Forward")
# Step 10 - Write final ipy file
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\cm2006.ipy")
# Check the results
CheckIrrigationPracticeTS(ID="*")
WriteCheckFile(OutputFile="cm2006.ipy.StateDMI.check.html")

```

The following command file illustrates how to process the irrigation practice time series file where groundwater supply is used:

```

#
# Sp2008L_DDH.StateDMI
#
#
# StartLog(LogFile="SP_IPY.log")
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2006")
# Step 1 - Read CU Locations from list
#
ReadCULocationsFromList(ListFile="..\Sp2008L_StructList.csv",IDCol=1)
#
# Step 2 - Read SW aggregates, GW aggregates, and divsystems
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
#
SetWellSystemFromList(ListFile="..\SP_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2001.csv",Year=2001,Div=1,

```

```

PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2005.csv",Year=2005,Div=1,
PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
#
# Step 4 - Set conveyance efficiencies from file for key and sw aggregate structures - NOT in HydroBase
SetIrrigationPracticeTSFromList(ListFile="Sp2008L_Eff.csv",ID="*",
SetStart=1950,SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="3")
#
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=2006,FloodAppEffMax=.6,SprinklerAppEffMax=.8,GWMode=2)
#
# Step 6 - Read well rights file and Set Max pumping (use merged *.wer file)
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
# Step 7 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Div="1")
#
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="Sp2008L.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
#
# Step 9 - Estimate 1950 ground water acreage based on active wells as defined in the non-merged *.wer
file
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
IncludeGroundwaterOnlySupply="True",FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 10 - Fill Interpolate Acreage Type (SW and GW) 1956-2006
# Step 11a - estimate total GW and total SW
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1956",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 11b - set sprinkler to zero in early period
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=1969,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 11c - fill remaining irrigation method values
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 12 - Set Acreage = 0 for structures that are in diversion systems, so acreage is not double

```

```

accounted
SetIrrigationPracticeTS(ID="0100503_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100507_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100687",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="0200834",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400511_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 13 - Set Acreage = 0, 1950-2006
SetIrrigationPracticeTS(ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100513",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100829",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400519",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
  AcresGWFFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 14 - Write final ipy file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateMod\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)

```

# Command Reference: ReadIrrigationPracticeTSFromList()

Read irrigation practice time series data from information in a delimited file

**StateCU Command**

Version 3.09.01, 2010-02-01

The `ReadIrrigationPracticeTSFromList()` command reads irrigation practice time series data for existing CU Locations by reading information from a delimited file. New locations are not added and the information is added to existing locations. HydroBase may not contain all irrigated lands data. For example, additional lands may have been identified after HydroBase was populated or acreage must be set for a model identifier that is not a structure WDID in HydroBase (e.g., out of state lands). In this case, the command can be used to provide additional data to supplement HydroBase.

**Edit ReadIrrigationPracticeTSFromList() Command**

This command reads irrigation practice time series data from a delimited list file, using the CU Location ID to look up the location. The data will be added to previous results. Data specified for part of an aggregate/system will be processed, and additional data can be read with `ReadIrrigationPracticeTSFromHydroBase()`. A comma-delimited list file is used to supply data, with values being set one of the following ways:

- 1) If the input start and end years are specified and a year column is not specified, the file data values are applied to each year in the specified input period.
- 2) If a year column is specified, year and corresponding values are read from the list file (the input period then controls how many years are processed).

Acres, irrigation method, and supply type must be specified. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: `C:\Develop\StateDMI_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase`

List file:

CU Location ID:  Required - CU Location(s) to read (use \* for wildcard).

Input start (year):  Optional - starting year to read data (default=process all).

Input end (year):  Optional - ending year to read data (default=process all).

Year column:  Optional - column in file for year.

CU location ID column:  Required - column in file for CU location ID.

Irrigated acres column:  Required - column in file for irrigated acres.

Irrigation method column:  Required - column in file for irrigation method (column containing SPRINKLER, FLOOD).

Supply type column:  Required - column in file for supply type (column containing Ground or Surface).

Command:  

```
ReadIrrigationPracticeTSFromList(ListFile="ipy-additions.csv", ID="*", InputStart=x, YearCol=2, IDCol="1", AcresCol="4", IrrigationMethodCol="5", SupplyTypeCol="6")
```

ReadIrrigationPracticeTSFromList

**ReadIrrigationPracticeTSFromList() Command Editor – Provide Parcel Data not in HydroBase**

The command syntax is as follows:

```
ReadIrrigationPracticeTSFromList(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	Path to the delimited list file to read.	None – must be specified.
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
InputStart	The first year to read from the file.	If not specified, all years are read from the file.
InputEnd	The last year to read from the file.	If not specified, all years are read from the file.
YearCol	The column number (1+) containing the year for data.	The file values are applied to each year in the data set.
IDCol	The column number (1+) containing the CU Location identifiers. These values are matched against CU Location identifiers in the existing irrigation practice data.	None – must be specified.
AcresCol	The column number (1+) containing the crop area.	If not specified, the previous data values will remain.
IrrigationMethodCol	The column number (1+) containing the irrigation method, consistent with HydroBase (e.g., SPRINKLER, FLOOD).	If not specified, the previous data values will remain.
SupplyTypeCol	The column number (1+) containing the supply type (Surface or Ground).	If not specified, the previous data values will remain.

Data file lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings. An example list file for specifying acreage data (not in HydroBase) is shown below. Currently, supplemental acreage data can have only a single irrigation method and supply type, to support irrigation practice time series processing. Therefore, break supplemental acreage into multiple “parcels” if necessary.

```
# The following data provide acreage for structures that did not have GIS data
# and consequently no data in HydroBase. The data are specific to 1998 and are
# used to set the CDS and IPY acres. The crop is used to provide CDS data. The
# irrigation method and source are used to provide IPY data.
"ID","Crop","Acres","IrrigationMethod","SupplySource"
200500,1998,GRASS_PASTURE,0,Flood,Surface
200506,1998,GRASS_PASTURE,100,Flood,Surface
200507,1998,GRASS_PASTURE,50,Flood,Surface
200508,1998,GRASS_PASTURE,40,Flood,Surface
200522,1998,GRASS_PASTURE,40,Flood,Surface
200523,1998,GRASS_PASTURE,50,Flood,Surface
200526,1998,GRASS_PASTURE,40,Flood,Surface
200529,1998,GRASS_PASTURE,5,Flood,Surface
... etc...
```

---

# Command Reference: ReadIrrigationPracticeTSFromStateCU()

Read irrigation practice time series data from a StateCU file

**StateCU Command**

Version 3.09.01, 2010-02-01

The `ReadIrrigationPracticeTSFromStateCU()` command reads irrigation practice time series data from a StateCU irrigation practice time series file and defines the data in memory. The irrigation practice time series can then be manipulated and output with other commands. This command can be used to adjust an existing irrigation practice file. See also the TSTool software, which can be used to read, manipulate, and view irrigation practice time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadIrrigationPracticeTSFromStateCU() Command**

This command reads CU irrigation practice time series from a StateCU irrigation practice time series file.  
StateCU irrigation practice time series are defined for each CU Location.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase

Input file:

Version:  Optional - StateCU program version (default=most current).

Command:  

```
ReadIrrigationPracticeTSFromStateCU (InputFile="cm2006.ipy")
```

ReadIrrigationPracticeTSFromStateCU

**ReadIrrigationPracticeTSFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadIrrigationPracticeTSFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the file to read, surrounded by double quotes.	None – must be specified.
Version	The StateCU version number, to allow backward compatibility with file formats from an earlier StateCU version.	Use the most current known format.



# Command Reference: ReadIrrigationWaterRequirementTSMonthlyFrom StateCU()

Read irrigation water requirement time series data from a StateCU file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadIrrigationWaterRequirementTSMonthlyFromStateCU()` command reads irrigation water requirement time series data from a StateCU irrigation water requirement time series file and defines the data in memory. Currently this command is meant to read the IWR time series for use in estimating average efficiencies and demands for StateMod – it is not supported in StateCU commands (e.g., to read and modify the time series file). All time series are read, whether or not they match the list of diversion stations. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadIrrigationWaterRequirementTSMonthlyFromStateCU() Command**

This command reads IWR or CU requirement time series (monthly) from a StateCU/StateMod time series file. Consumptive water requirement time series (monthly) are associated with diversion and well stations. For agricultural stations, the irrigation water requirement (IWR) time series from the consumptive use model are equivalent. It is recommended that the file be specified using a path relative to the working directory. The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversionDemandTSMonthlyToStateMod

Input file:

Command:

ReadIrrigationWaterRequirementTSMonthlyFromStateCU

### ReadIrrigationWaterRequirementTSMonthlyFromStateCU() Command Editor

The command syntax is as follows:

```
ReadIrrigationWaterRequirementTSMonthlyFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateCU irrigation water requirement file (StateMod time series format) to read.	None – must be specified.

The following abbreviated command file illustrates how irrigation water requirement time series can be processed into average demand time series:

```

StartLog(LogFile="Cddm.commands.StateDMI.log")
# Cddm.commands.StateDMI
#
# StateDMI command file to create the Calculated demand file
#
# Step 1 - set the output period, used to compute averages...
#
SetOutputPeriod(OutputStart="10/1908",OutputEnd="09/2005")
SetOutputYearType(OutputYearType=Water)
#
# Step 2 - read historical diversion file -defines structures for *.ddm file
#           plus read *.ddh file
#
ReadDiversionStationsFromStateMod(InputFile="..\StateMod\cm2005.dds")
ReadDiversionHistoricalTSMonthlyFromStateMod(InputFile="..\StateMod\cm2005.ddh")
#
# Step 3 - read StateCU *.iwr and *.def files (irrigation requirements and average efficiencies)
#
ReadIrrigationWaterRequirementTSMonthlyFromStateCU(InputFile="..\StateMod\cm2005.iwr")
# calculateDiversionStationEfficiencies(ID="*",EffMin=0,EffMax=60,
#   EffCalcStart=10/1974,EffCalcEnd=9/2004,LEZeroInAverage=False)
SetDiversionStationsFromList(ListFile="cm2005.def",IDCol="1",EffMonthlyCol="2",
  Delim="Space",MergeDelim=True)
#
# Step 4 - determine calculated demand = iwr/efficiency
#           - take max of calculated demand and historical diversion
#
CalculateDiversionDemandTSMonthly(ID="*")
CalculateDiversionDemandTSMonthlyAsMax(ID="*")
#
# Step 5 - set carriers nodes demand to 0, set full demand and summary demand nodes
#
# set carrier "transbasin" diversion to Divide Creek to "0", use operating rules to satisfy demand
SetDiversionDemandTSMonthlyConstant(ID="724721",Constant=0)
# place summary demand at the Moffat Tunnel, zero out collection points
SetDiversionDemandTSMonthly(ID="514655",TSID="514655..DivTotal.Month~StateMod~514655.stm")
... similar commands omitted...
#
# Step 6 - set calculated demand to historic for structures whose historical acreage is
#           different from current
#
SetDiversionDemandTSMonthly(ID="360687",TSID="360687..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
SetDiversionDemandTSMonthly(ID="360725",TSID="360725..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
...similar commands omitted...
#
# Set Ute WCD demand node structure and set other structures to zero
SetDiversionDemandTSMonthly(ID="950020",TSID="950020..DivTotal.Month~StateMod~950020.stm")
SetDiversionDemandTSMonthlyConstant(ID="950030",Constant=0)
... similar commands omitted...
#
# Set Orchard Mesa Check
SetDiversionDemandTSMonthly(ID="950003",TSID="950003..DivTotal.MONTH~StateMod~..\StateMod\cm2005H.ddm")
#
# Set Excess HUP node demands for Homestake, Dillon, Williams Fork, and Wolford Reservoirs
SetDiversionDemandTSMonthlyConstant(ID="954516D",Constant=999999)
...similar commands omitted...
# Step 7 - write out calculated demand file
#
WriteDiversionDemandTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005C.ddm")
#
# Check the results
CheckDiversionDemandTSMonthly(ID="*")
WriteCheckFile(OutputFile="Cddm.commands.StateDMI.check.html")

```

---

# Command Reference: ReadNetworkFromStateMod()

**Read generalized network from a StateMod XML network file**

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadNetworkFromStateMod()` command reads the generalized network from a StateMod XML network file. The network can then be manipulated and utilized by other commands. Normally the generalized network is edited interactively in StateDMI (or StateMod GUI) and is used to generate lists of stations, for further processing. However, this command can be used to read the network and allow manipulation based on river upstream/downstream connectivity. See also commands like `ReadDiversionStationsFromNetwork()`, which read a subset of the network, to facilitate creation of specific model data files.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadNetworkFromStateMod() Command**

This command reads the generalized network from a StateMod network file (old Makenet format or new XML format). The network can then be used, for example, to create the river network specifically needed by StateMod.

The following data are read:

- Identifier (ID)
- River node identifier - set to the identifier.

The name is not read because it is often filled from a database with other commands.

It is recommended that the location of the file be specified using a path relative to the working directory.

The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI

StateMod file:

Command:

```
ReadNetworkFromStateMod ( InputFile="..\Network\cm2005.net" )
```

ReadNetworkFromStateMod

**ReadNetworkFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadNetworkFromStateMod( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod XML network file to read.	None – must be specified.

# Command Reference: ReadPatternFile()

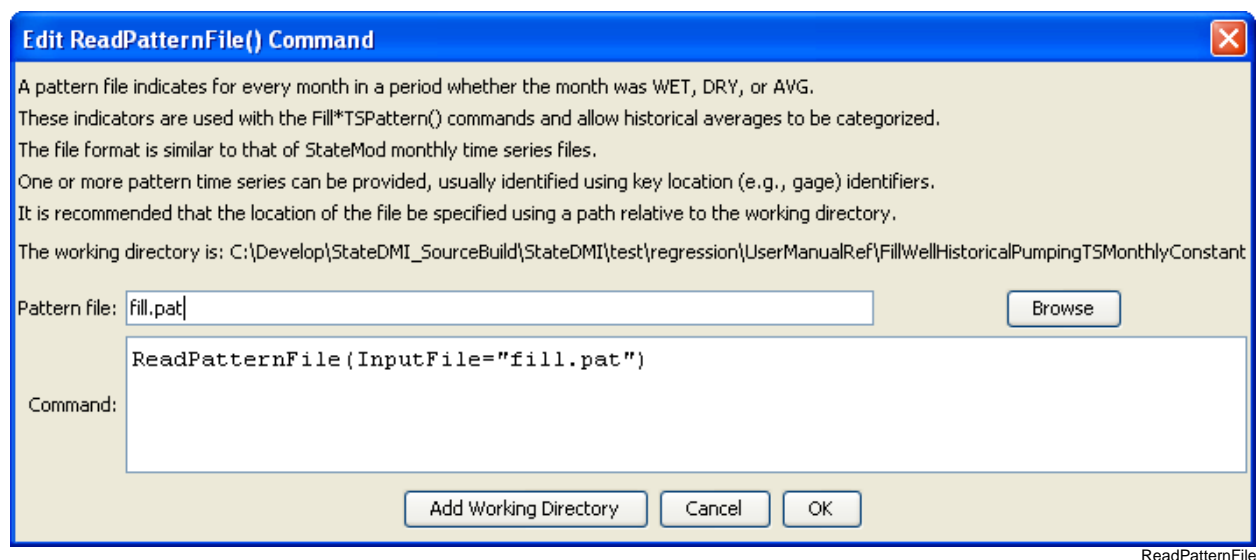
## Read WET/DRY/AVG monthly pattern data file

### StateCU and StateMod Command

Version 3.09.01, 2010-01-27

The `ReadPatternFile()` command reads monthly WET/DRY/AVG patterns from a file. This information is used with `Fill*Pattern()` commands, which are more refined than commands that fill with historical averages.

The following dialog is used to edit the command and illustrates the syntax of the command.



**ReadPatternFile() Command Editor**

The command syntax is as follows:

```
ReadPatternFile(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the pattern file to be read.	None – must be specified.

The pattern file format is very similar to the StateMod monthly time series file format except that instead of monthly numerical values, the strings WET, DRY, or AVG are used to indicate the conditions during the particular month. The pattern file contains pattern time series for one or more key locations, often stream gages with long periods of record. The AnalyzePattern( ) command provided by the TSTool software can be used to create the pattern file.

```
# Monthly Streamflow Characterizations for Estimation of Missing Diversion Data
# *****
#
# Years Shown = Calendar Years
#
# Time series identifier      = 08220000.CRDSS_USGS.QME.MONTH.1
# Description                = RIO GRANDE AT DEL NORTE
# Located in water div, district = 3, 20
# Located in county, state   =
# Located in HUC             =
# Latitude, longitude        =
#
# Characterizations are based on 25% and 75% cutoffs assuming a normal distribution
#
# 25% (ACFT)      8063      8410      11336      26024      107908      118306      37082      24135      15410      15075      10404      8946
# 75% (ACFT)      11564      11594      17564      49094      179836      241828      123255      63532      31008      29759      18286      12808
#
# Yr-----JAN      FEB      MAR      APR      MAY      JUN      JUL      AUG      SEPT      OCT      NOV      DEC
# -e-b-----eb-----eb-----eb-----eb-----eb-----eb-----eb-----eb-----e
# File generated by:
#
# Program:      Excel Spreatsheet
# User:         E. Armbruster
# File:         g:\projects\297\task6\rgmodel\ts_files\fill.pat
# Date:         October 26, 1999
#
#
# Yr-----JAN      FEB      MAR      APR      MAY      JUN      JUL      AUG      SEPT      OCT      NOV      DEC
# -e-b-----eb-----eb-----eb-----eb-----eb-----eb-----eb-----eb-----e
# 01/1950 -      12/1997 ACFT      CYR
1950 08220000      WET      WET      AVG      WET      AVG      AVG      AVG      AVG      DRY      DRY      DRY      DRY
1951 08220000      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      AVG
1952 08220000      AVG      AVG      DRY      WET      WET      WET      WET      WET      AVG      AVG      AVG      AVG
1953 08220000      AVG      AVG      AVG      AVG      DRY      AVG      AVG      DRY      DRY      DRY      AVG      AVG
1954 08220000      AVG      WET      DRY      AVG      AVG      DRY      AVG      AVG      AVG      AVG      AVG      DRY
1955 08220000      AVG      AVG      DRY      DRY      DRY      AVG      DRY      AVG      DRY      DRY      DRY      DRY
1956 08220000      AVG      AVG      AVG      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY
1957 08220000      DRY      DRY      DRY      AVG      DRY      WET      WET      WET      WET      AVG      WET      WET
1958 08220000      AVG      WET      AVG      AVG      WET      AVG      AVG      AVG      AVG      AVG      AVG      AVG
1959 08220000      DRY      DRY      DRY      DRY      DRY      DRY      DRY      AVG      DRY      AVG      WET      AVG
1960 08220000      AVG      AVG      WET      WET      AVG      AVG      AVG      DRY      DRY      DRY      DRY      AVG
1961 08220000      DRY      AVG      AVG      AVG      AVG      AVG      DRY      AVG      AVG      AVG      AVG      WET
1962 08220000      WET      WET      AVG      WET      WET      AVG      AVG      AVG      AVG      AVG      AVG      AVG
1963 08220000      AVG      AVG      AVG      AVG      AVG      DRY      DRY      DRY      AVG      DRY      DRY      DRY
1964 08220000      DRY      DRY      DRY      DRY      AVG      DRY      DRY      DRY      AVG      DRY      DRY      DRY
1965 08220000      DRY      DRY      DRY      WET      WET      WET      WET      WET      WET      WET      WET      WET
1966 08220000      WET      AVG      WET      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG
1967 08220000      DRY      DRY      AVG      DRY      DRY      DRY      AVG      AVG      AVG      DRY      DRY      AVG
1968 08220000      AVG      AVG      AVG      DRY      AVG      AVG      WET      WET      WET      AVG      AVG      AVG
1969 08220000      AVG      AVG      AVG      AVG      AVG      AVG      WET      WET      WET      WET      WET      WET
1970 08220000      AVG      AVG      DRY      WET      WET      AVG      WET      WET      WET      WET      WET      WET
1971 08220000      WET      WET      WET      AVG      DRY      AVG      DRY      AVG      AVG      AVG      AVG      AVG
1972 08220000      AVG      AVG      WET      WET      AVG      DRY      DRY      DRY      DRY      AVG      AVG      AVG
1973 08220000      AVG      AVG      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET
1974 08220000      AVG      AVG      AVG      DRY      AVG      DRY      DRY      DRY      DRY      DRY      DRY      DRY
1975 08220000      DRY      DRY      DRY      DRY      WET      WET      WET      WET      WET      WET      WET      WET
1976 08220000      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG      AVG      DRY
1977 08220000      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY      DRY
1978 08220000      DRY      DRY      DRY      DRY      DRY      AVG      AVG      DRY      DRY      AVG      DRY      DRY
1979 08220000      DRY      DRY      AVG      WET      WET      WET      WET      WET      WET      DRY      AVG      AVG
1980 08220000      WET      WET      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET
1981 08220000      AVG      AVG      DRY      AVG      DRY      DRY      DRY      WET      WET      WET      WET      WET
1982 08220000      AVG      AVG      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET
1983 08220000      WET      AVG      DRY      WET      WET      WET      WET      WET      WET      WET      WET      WET
1984 08220000      AVG      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET
1985 08220000      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1986 08220000      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1987 08220000      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1988 08220000      WET      WET      WET      WET      DRY      WET      WET      WET      WET      WET      WET      WET
1989 08220000      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1990 08220000      DRY      DRY      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET
1991 08220000      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1992 08220000      AVG      DRY      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET
1993 08220000      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1994 08220000      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1995 08220000      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1996 08220000      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
1997 08220000      AVG      AVG      WET      WET      WET      WET      WET      WET      WET      WET      WET      WET
```

---

# Command Reference: ReadPenmanMonteithFromHydroBase()

Read Penman-Monteith crop coefficients data from HydroBase

**StateCU Command**  
Version 3.10.00, 2010-04-02

The ReadPenmanMonteithFromHydroBase() command reads a list of Penman-Monteith crop coefficients from the HydroBase database. The crop coefficients can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadPenmanMonteithFromHydroBase() Command**

This command reads Penman-Monteith crop coefficients from HydroBase.  
Penman-Monteith crop coefficients estimate crop water requirements for each crop during the year, for standard conditions.  
Characteristics can be queried for a specific Penman-Monteith method.

PenmanMonteith Method:  Required - Penman-Monteith method.

Command: `ReadPenmanMonteithFromHydroBase (PenmanMonteithMethod="PENMAN-MONTEITH_ALFALFA")`

OK Cancel

ReadPenmanMonteithFromHydroBase() Command Editor

The command syntax is as follows:

`ReadPenmanMonteithFromHydroBase (Parameter=Value,...)`

## Command Parameters

Parameter	Description	Default
PenmanMonteithMethod	The Penman-Monteith method that is defined in HydroBase for the crop type and its coefficients.	None – must be specified.

The crop type (e.g., ALFALFA) is used as the unique identifier. Any previous crop coefficients objects will be added to (or replaced if identifiers match).

The PenmanMonteithMethod parameter corresponds to a value in HydroBase and allows variations on crop characteristics to be defined. In general the ASCE standardized coefficients are used.

The following example command file illustrates how to read Penman-Monteith coefficients from HydroBase, sort the data, create a StateCU file, and check the results:

```
StartLog(LogFile="Crops_KPM.StatedMI.log")
#
# StatedMI commands to create the Penman-Monteith crop coefficients file
#
# Step 1 - read data from HydroBase
#
# Read the general ASCE standardized coefficients
ReadPenmanMonteithFromHydroBase(PenmanMonteithMethod="PENMAN-
MONTEITH_ALFALFA")
#
# Step 3 - write the file
#
SortPenmanMonteith(Order=Ascending)
WritePenmanMonteithToStateCU(OutputFile="rg2007.kpm")
#
# Check the results
#
CheckPenmanMonteith(ID="*")
WriteCheckFile(OutputFile="Crops_KPM.StatedMI.check.html")
```



---

# Command Reference: ReadPenmanMonteithFromStateCU()

**Read Penman-Monteith crop coefficients data from a StateCU Penman-Monteith crop coefficients file**

**StateCU Command**

Version 3.10.00, 2010-04-02

The `ReadPenmanMonteithFromStateCU()` command reads Penman-Monteith crop coefficients from a StateCU Penman-Monteith crop coefficients file and defines crop coefficients in memory. The crop coefficients can then be manipulated and output with other commands. This command can be used to adjust an existing crop coefficients file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadPenmanMonteithFromStateCU() Command**

This command reads Penman-Monteith crop coefficients from a StateCU Penman-Monteith crop coefficients file. Penman-Monteith crop coefficients estimate crop water requirements for each crop during the year, for standard conditions. The ASCE Standardized Penman-Monteith equation is also supported. It is recommended that the file be specified using a path relative to the working directory. The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadPenmanMonteithFromHydroBase

Input file:

Command:

ReadPenmanMonteithFromStateCU

**ReadPenmanMonteithFromStateCU() Command Editor**

The command syntax is as follows:

```
ReadPenmanMonteithFromStateCU( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.

---

# Command Reference: ReadReservoirRightsFromHydroBase()

Read reservoir right data from HydroBase

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `ReadReservoirRightsFromHydroBase()` command reads reservoir rights from HydroBase, for each reservoir station that is defined. The reservoir rights can then be manipulated and output with other commands. Within a reservoir station, rights are sorted by administration number and order number. In some cases, multiple rights for the reservoir may be listed, each with the same administration number. This is because the order number is different; however, the order number is not listed in the StateMod output.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadReservoirRightsFromHydroBase() Command**

This command reads reservoir rights from HydroBase, using the reservoir station identifiers to find rights. Net absolute water rights are read by default.  
If the reservoir stations contain aggregates, use the administration number classes to group rights (indicate administration numbers separated by commas).  
The output water right identifier is assigned as the reservoir station identifier + "." + a three digit number.

Reservoir station ID:  Required - reservoir stations to read (use \* for wildcard).

Decree minimum:  Optional - minimum decree to include (default = .0005).

Admin. number classes:

OnOff default:  Optional - default OnOff switch (default=AppropriationDate).

Command: 

```
ReadReservoirRightsFromHydroBase ( ID="*", OnOffDefault=1)
```

OK Cancel

ReadReservoirRightsFromHydroBase

**ReadReservoirRightsFromHydroBase() Command Editor**

If aggregating rights, the following steps:

1. Water rights for each part of the aggregate are read from HydroBase, reporting errors as necessary.
2. The rights are added to a list and are sorted by administration number. This ensures that the cumulative list of rights is listed in order of administration number (this step will be necessary if reservoir systems, similar to diversion systems, are supported – currently they are not).
3. Water rights are defined for each class (see the `AdminNumClasses` parameter description below), initializing the decree to zero.

4. For each class, the following sums are calculated:  $\text{sum}(\text{decree} * \text{AdminNum})$  and  $\text{sum}(\text{decree})$ , where the administration number is determined from the appropriation date derived from the original HydroBase administration number (it will not have a remainder).
5. The final administration number for the class is determined (it will not have a remainder):  
 $\text{int}(\text{sum}(\text{decree} * \text{AdminNum}) / \text{sum}(\text{decree}))$

Water rights that are less than the decree minimum are ignored.

The command syntax is as follows:

`ReadReservoirRightsFromHydroBase(Parameter=Value,...)`

#### Command Parameters

Parameter	Description	Default
ID	A single reservoir station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DecreeMin	The minimum decree to accept as a valid right.	0.0 – read all rights.
AdminNumClasses	A list of administration numbers, separated by spaces or commas, to define the breaks for aggregate water rights, for reservoir aggregates. For example, if the class breaks are 10000.0000, 20000.00000, and 99999.99999, the first group will contain water rights with administration numbers $\leq 10000.00000$ , the second will contain water rights with administration number $> 10000.00000$ and $\leq 20000.00000$ , and the third will contain water rights with administration number $> 20000.00000$ and $\leq 99999.99999$ .	If not specified, diversion aggregates will be treated as diversion systems, with all water rights explicitly included in output.
OnOffDefault	Indicates how to set the on/off switch for all water rights that are processed. A value of 1 indicates that the right is on for the whole period. If the value is AppropriationDate, the switch is set to the year corresponding to the appropriation date, indicating that the right will be turned on starting in the year. Use set commands to reset the switch to other values.	Appropriation Date

---

# Command Reference: ReadReservoirRightsFromStateMod()

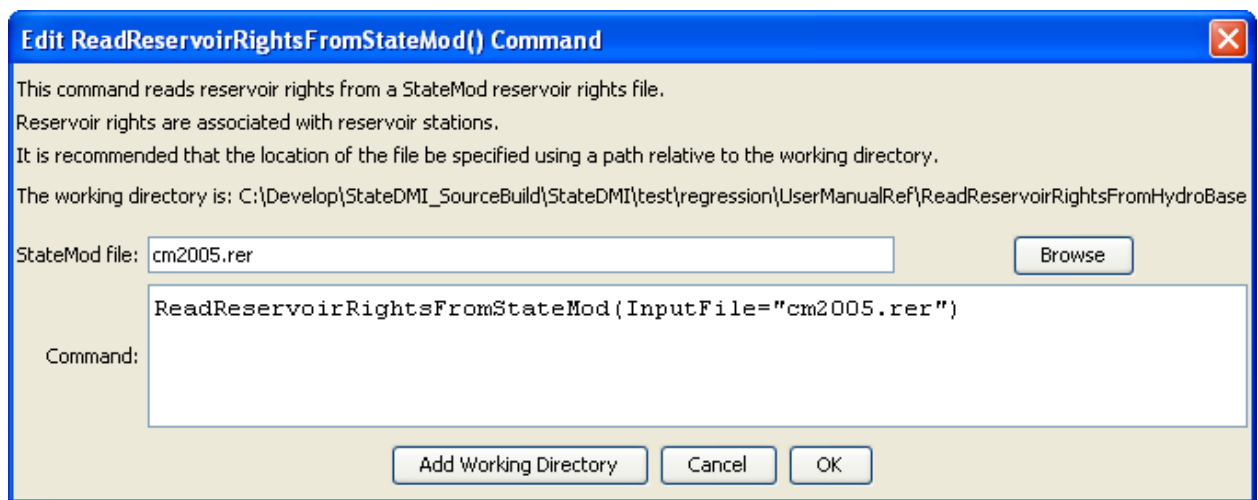
Read reservoir right data from a StateMod reservoir rights file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadReservoirRightsFromStateMod()` command reads reservoir rights from a StateMod reservoir rights file. The reservoir rights can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadReservoirRightsFromStateMod

### ReadReservoirRightsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadReservoirRightsFromStateMod( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod reservoir rights file to read.	None – must be specified.

---

# Command Reference: ReadReservoirStationsFromList()

Read reservoir stations data from a list file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadReservoirStationsFromList()` command reads a list of reservoir stations from a delimited list file and defines reservoir stations in memory. The reservoir stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadReservoirStationsFromList() Command**

This command reads reservoir stations from a list file containing columns of information. Reservoir stations indicate locations where water can be stored for use at a later date. Columns should be delimited by commas (user-specified delimiters will be added in the future). Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillReservoirStationsFromHydroBase

List file:

ID column:  Required - column (1+) for ID.

Name column:  Optional - column (1+) for name.

Command:

ReadReservoirStationsFromList

**ReadReservoirStationsFromList() Command Editor**

The command syntax is as follows:

```
ReadReservoirStationsFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the reservoir station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the reservoir station names.	None – optional (name will be initialized to blank).

At a minimum, the list file must contain a column with diversion station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
# Reservoir stations as a list file
#
"ID", "Name"
203536, "Reservoir 1"
203558, "Reservoir 2"
...
```



---

# Command Reference: ReadReservoirStationsFromNetwork()

Read reservoir station data from a network file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadReservoirStationsFromNetwork()` command reads a list of reservoir stations from a StateMod network file (XML or older Makenet network file) and defines reservoir stations in memory. The reservoir stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadReservoirStationsFromNetwork() Command**

This command reads reservoir station data from StateMod XML network file.  
Reservoir stations indicate locations where water can be stored for use at a later date.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
River node ID - set to station ID  
Daily ID - set to the nearest downstream streamflow station ID.  
See also the Fill\*FromNetwork() commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillReservoirStationsFromHydroBase

StateMod network file:

Command:

ReadReservoirStationsFromNetwork

### ReadReservoirStationsFromNetwork() Command Editor

The command syntax is as follows:

```
ReadReservoirStationsFromNetwork( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to be read.	None – must be specified.

---

# Command Reference: ReadReservoirStationsFromStateMod()

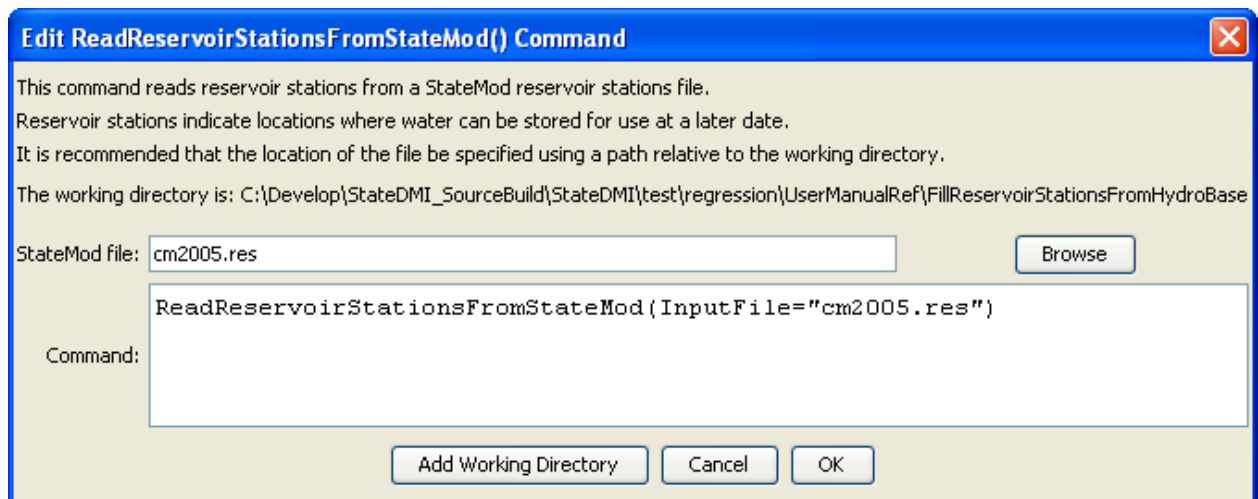
Read reservoir station data from a StateMod reservoir stations file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadReservoirStationsFromStateMod()` command reads a list of reservoir stations from a StateMod reservoir stations file and defines reservoir stations in memory. The reservoir stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



**ReadReservoirStationsFromStateMod() Command Editor**

ReadReservoirStationsFromStateMod

The command syntax is as follows:

```
ReadReservoirStationsFromStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod reservoir stations file to be read.	None – must be specified.

---

# Command Reference: ReadRiverNetworkFromStateMod()

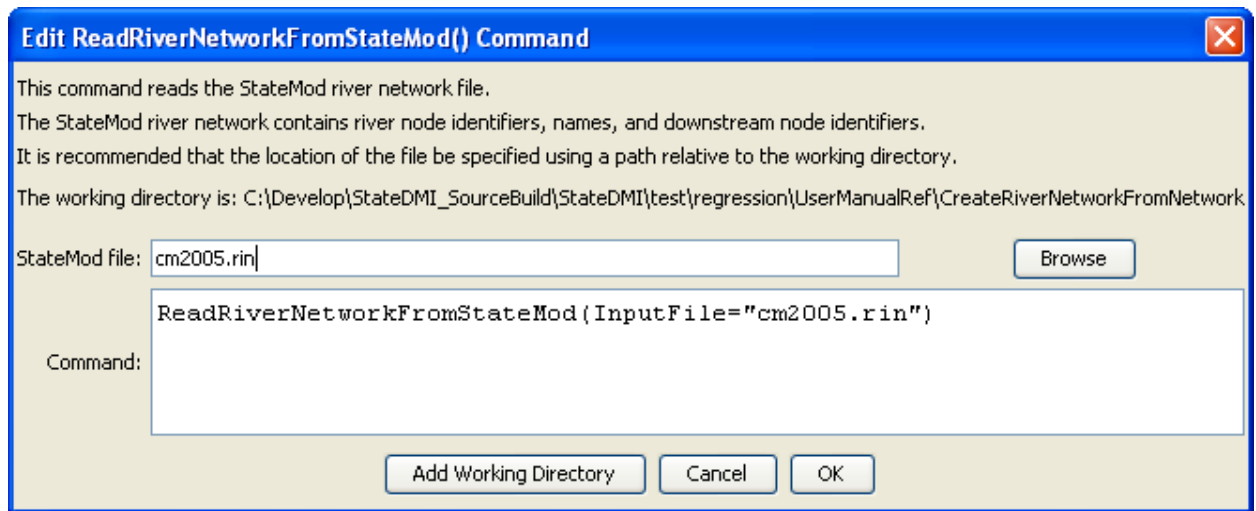
Read river network from a StateMod river network file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadRiverNetworkFromStateMod()` command reads the river network from a StateMod river network file. The river network can then be manipulated and utilized by other commands. Normally the StateMod river network is only created as output, but it may be read if it is being converted to a generalized network file.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadRiverNetworkFromStateMod

**ReadRiverNetworkFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadRiverNetworkFromStateMod( Parameter=Value, ...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod river network file to be read.	None – must be specified.

The following example command file illustrates how the command might be used:

```
# Create a generalized XML network from individual StateMod files
# Read the network, which contains upstream to downstream connectivity but does
# not indicate node types
ReadRiverNetworkFromStateMod(InputFile=cm2005.rin)
# Read the stations, which imply the node types
ReadRiverStreamGageStationsFromStateMod(InputFile=cm2005.ris)
ReadRiverDiversionStationsFromStateMod(InputFile=cm2005.dds)
ReadRiverReservoirStationsFromStateMod(InputFile=cm2005.res)
ReadRiverInstreamFlowStationsFromStateMod(InputFile=cm2005.ifs)
ReadRiverWellStationsFromStateMod(InputFile=cm2005.wes)
# To be developed...
#ReadRiverPlanStationsFromStateMod()
ReadRiverStreamEstimateStationsFromStateMod(InputFile=cm2005.ris)
# Now create the generalized network, using the connectivity and node types
CreateNetworkFromRiverNetwork()
# Fill in node names and locations from HydroBase, if any is still missing
FillNetworkFromHydroBase()
# Write the generalized network
WriteNetworkToStateMod(OutputFile="cm2005.net")
# Check for errors (the following is not yet implemented)
#CheckNetwork()
WriteCheckFile(OutputFile="cm2005.net.check.html")
```

---

# Command Reference: ReadStreamEstimateCoefficientsFromStateMod()

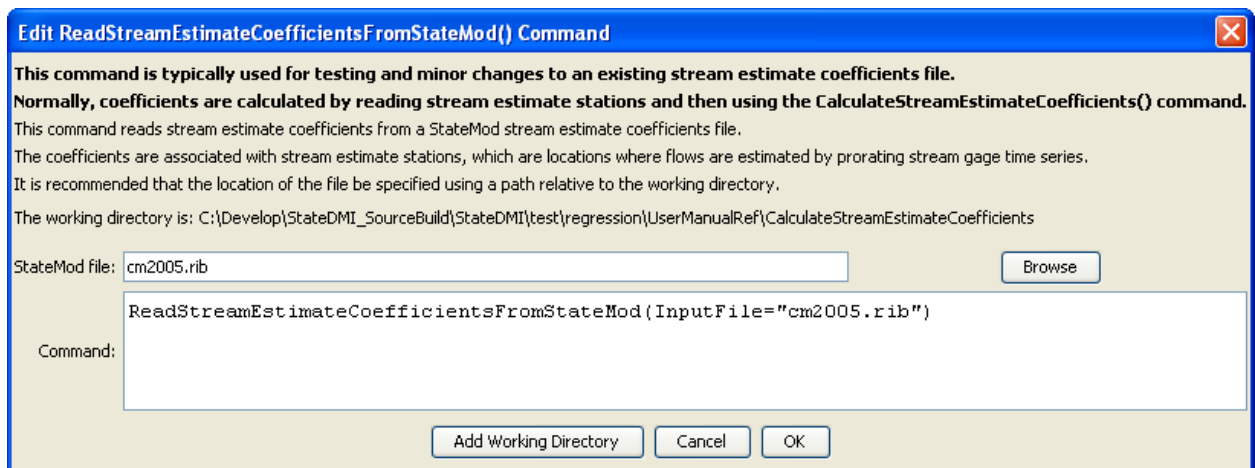
**Read stream estimate coefficient data from a StateMod stream estimate coefficients file**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadStreamEstimateCoefficientsFromStateMod()` command reads stream estimate coefficients from a StateMod stream estimate coefficients file. This information is associated with stream estimate stations using the station identifier as the lookup. Stream estimate coefficients define how streamflow is estimated at ungaged locations (stream estimate stations). The stream estimate coefficients that are read can be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadStreamEstimateCoefficientsFromStateMod

**ReadStreamEstimateCoefficientsFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadStreamEstimateCoefficientsFromStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod stream estimate coefficients file to be read.	None – must be specified.



---

# Command Reference: ReadStreamEstimateStationsFromList()

Read stream estimate stations data from a list file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadStreamEstimateStationsFromList()` command reads a list of stream estimate stations from a delimited list file and defines stream estimate stations in memory. The stream estimate stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadStreamEstimateStationsFromList() Command**

This command reads stream estimate stations from a list file containing columns of information. Stream estimate stations indicate locations where stream flow is estimated (not historically measured). Columns should be delimited by commas (user-specified delimiters will be added in the future). Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamEstimateStationsFromHydroBase

List file:

ID column:  Required - column (1+) for ID.

Name column:  Optional - column (1+) for name.

River node ID column:  Optional - column (1+) for river network ID.

Daily ID column:  Optional - column (1+) for daily ID.

Command:  
`ReadStreamEstimateStationsFromList (ListFile="rgtw_estimategations.csv",  
IDCol=1,NameCol=2)`

ReadStreamEstimateStationsFromList

### ReadStreamEstimateStationsFromList() Command Editor

The command syntax is as follows:

```
ReadStreamEstimateStationsFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the stream estimate station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the stream estimate station names.	If not specified, set to blank.
RiverNodeIDCol	The column number (1+) containing the river node identifier.	If not specified, set to blank.
DailyIDCol	The column number (1+) containing the daily identifier (for estimating time series).	If not specified, set to blank.

At a minimum, the list file must contain a column with stream estimate station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
# Stream estimate stations as a list file
#
"ID", "Name"
NF1, "Natural flow 1"
NF2, "Natural flow 2"
...
```

---

# Command Reference: ReadStreamEstimateStationsFromNetwork()

Read stream estimate station data from a network file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `ReadStreamEstimateStationsFromNetwork()` command reads a list of stream estimate stations from a StateMod network file (XML or Makenet) and defines stream estimate stations in memory. Stream estimate stations are stations not of type FLOW but which are indicated as natural flow nodes in the network. The default output order is that of the stream network, upstream to downstream. The StateMod model requires that the stream gage station file be in the same order as the river network file. The stream estimate stations that are read can be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadStreamEstimateStationsFromNetwork() Command**

This command reads stream estimate station data from a StateMod XML network file.  
Stream estimate stations indicate locations where flow is estimated (not historically measured).  
Stream estimate stations are nodes other than streamflow nodes that are natural flow nodes.  
Nodes indicated as natural flow nodes that are not FLOW nodes are read.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
River node ID - set to station ID  
Daily ID - set to the nearest downstream streamflow station ID.  
See also the Fill\*FromNetwork() commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamEstimateStationsFromHydroBase

StateMod network file:

Command: 

```
ReadStreamEstimateStationsFromNetwork( InputFile="..\Network\cm2005.net" )
```

ReadStreamEstimateStationsFromNetwork

**ReadStreamEstimateStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
ReadStreamEstimateStationsFromNetwork( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to be read.	Use the network that has previously been read with other commands.

---

# Command Reference: ReadStreamEstimateStationsFromStateMod()

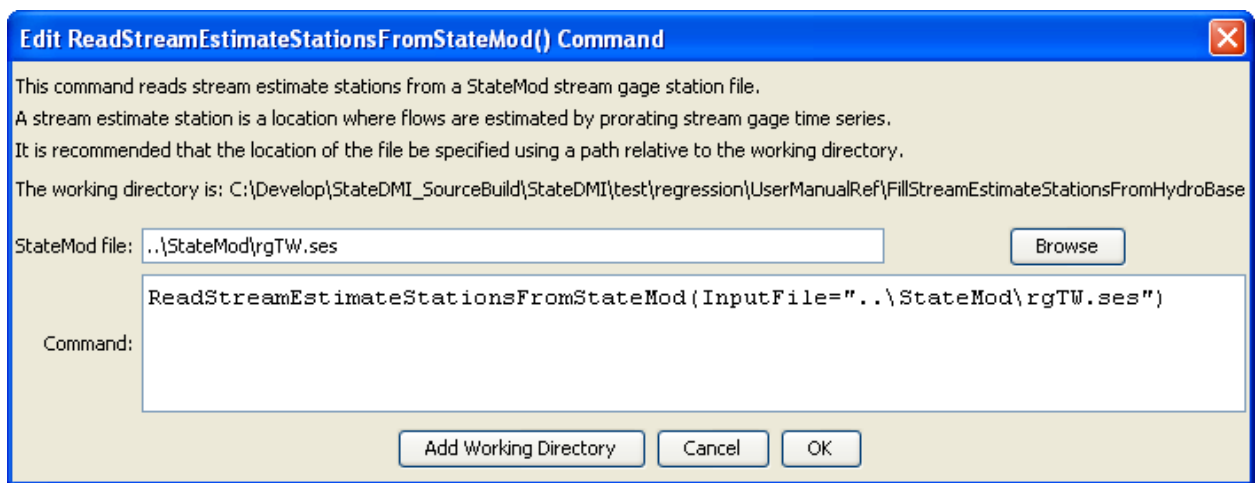
Read stream estimate station data from a StateMod stream estimate stations file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadStreamEstimateStationsFromStateMod()` command reads a list of stream estimate stations from a StateMod stream estimate stations file and defines stream estimate stations in memory. The stream estimate stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadStreamEstimateStationsFromStateMod

## ReadStreamEstimateStationsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadStreamEstimateStationsFromStateMod( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod stream estimate stations file to be read.	None – must be specified.

---

# Command Reference: ReadStreamGageStationsFromList()

Read stream gage station data from a list file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `ReadStreamGageStationsFromList()` command reads a list of stream gage stations from a delimited list file and defines stream gage stations in memory. The stream gage stations can then be manipulated and output with other commands. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadStreamGageStationsFromList() Command**

This command reads stream gage stations from a list file containing columns of information. Stream gage stations indicate locations where historical flow observations are available. Columns should be delimited by commas (user-specified delimiters will be added in the future). Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers. It is recommended that the location of the file be specified using a path relative to the working directory.

The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamGageStationsFromHydroBase

List file:

ID column:   Required - column (1+) for ID.

Name column:   Optional - column (1+) for name.

River node ID column:   Optional - column (1+) for river network ID.

Daily ID column:   Optional - column (1+) for daily ID.

Command: 

```
ReadStreamGageStationsFromList(ListFile="rgTW_stations.csv", IDCol=1, NameCol=2)
```

ReadStreamGageStationsFromList

**ReadStreamGageStationsFromList() Command Editor**

The command syntax is as follows:

```
ReadStreamGageStationsFromList(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the stream gage station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the stream gage station names.	If not specified, set to blank.
RiverNodeIDCol	The column number (1+) containing the river node identifier.	If not specified, set to blank.
DailyIDCol	The column number (1+) containing the daily identifier (for estimating time series).	If not specified, set to blank.

At a minimum, the list file must contain a column with stream gage station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
# Stream gage stations as a list file
#
"ID", "Name"
08213500, "RIO GRANDE RIVER AT THIRTY MILE BRIDGE NEAR CREEDE"
08214500, "NORTH CLEAR CREEK BELOW CONTINENTAL RESERVOIR"
...
```



---

# Command Reference: ReadStreamGageStationsFromNetwork()

Read stream gage station data from a network file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `ReadStreamGageStationsFromNetwork()` command reads a list of stream gage stations from a StateMod network file (XML or Makenet) and defines stream gage stations in memory. The stream gage stations can then be manipulated and output with other commands. The default output order is that of the stream network, upstream to downstream. The StateMod model requires that the stream gage station file be in the same order as the river network file.

Stream gages in the network are those defined as node type FLOW that are natural flow nodes. Stream gages that are included in the network but which are not identified as natural flow nodes are omitted from the stream gage station file – these nodes are typically treated as OTHER nodes in the network and will be included in the river network file but not other station files.

If stream estimate stations are also included in processing, all nodes identified as natural flow nodes are processed. See also the `ReadStreamEstimateStationsFromNetwork()` command.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadStreamGageStationsFromNetwork() Command**

This command reads stream gage station data from a StateMod XML network file.  
Stream gage stations indicate locations where historical observations are available.  
Nodes of type FLOW that are natural flow nodes are read.  
Stream estimate stations can also be read - these are locations where streamflow is estimated using proration factors.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
River node ID - set to station ID  
Daily ID - if a stream gage set to the station ID, otherwise (if stream estimate stations are included) set to the nearest downstream flow station ID.  
See also the `Fill*FromNetwork()` commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamGageStationsFromHydroBase

StateMod network file:

Include stream estimate stations?:  Optional - use True if creating RIS file with no SES file (default=False).

Command:  

```
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net", IncludeStreamEstimateStations="True")
```

**ReadStreamGageStationsFromNetwork() Command Editor**

The command syntax is as follows:

```
ReadStreamGageStationsFromNetwork( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to be read.	Use the network that has been previously read.
IncludeStreamEstimateStations	Indicate whether stream estimate stations should also be included. If False, only stream gage stations will be read. If True, stream gage and estimate stations will be read and will be treated as stream gage stations (separate stream gage and stream estimate station files are being evaluated but traditionally have been saved to the <i>*ris</i> file).	False

The following example command file illustrates the commands used to read stream gage stations from the network and create a StateMod file:

```
StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase,
#           fill in missing names from the network file
#
FillStreamGageStationsFromHydroBase( ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
...many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")
```

---

# Command Reference: ReadStreamGageStationsFromStateMod()

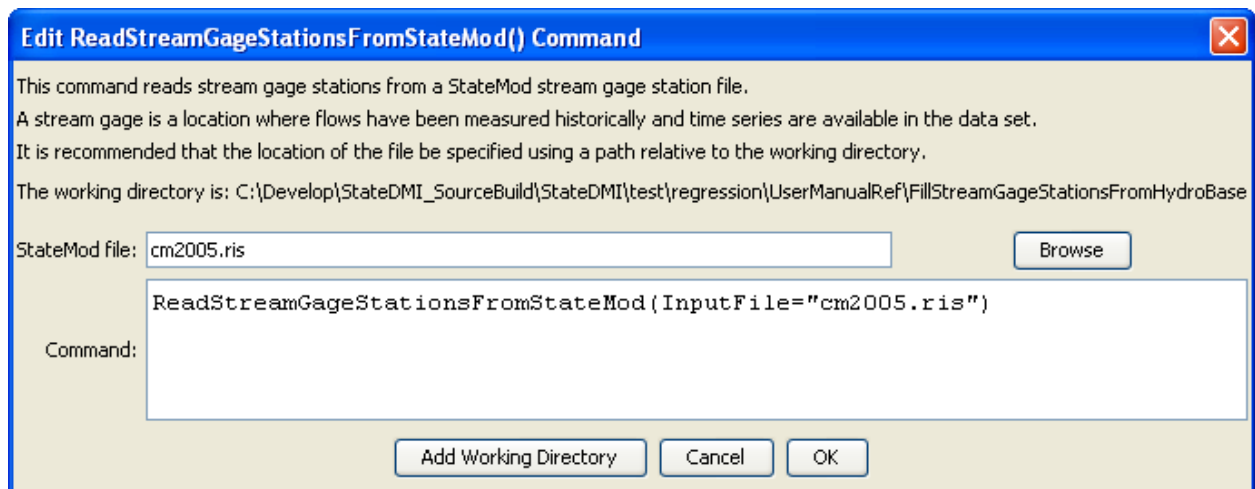
Read stream gage station data from a StateMod stream gage stations file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadStreamGageStationsFromStateMod()` command reads a list of stream gage stations from a StateMod stream gage stations file and defines stream gage stations in memory. The stream gage stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadStreamGageStationsFromStateMod

## ReadStreamGageStationsFromStateMod() Command Editor

The command syntax is as follows:

```
ReadStreamGageStationsFromStateMod( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod stream gage stations file to be read.	None – must be specified.

---

# Command Reference: ReadWellDemandTSMonthlyFromStateMod()

Read well demand time series (monthly) from a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `ReadWellDemandTSMonthlyFromStateMod()` command reads a list of well demand time series (monthly) from a StateMod monthly time series file. The file does not need to be a demand file (e.g., it could be a historical pumping file); however, once read with this command, the data will need to be processed with demand commands.

The StateMod well stations file contains stations for which only groundwater supply is available and stations for which groundwater supply supplements surface water supply of a diversion station (in this case the well station data includes the diversion station identifier). Parameters are available in this command to read all demand time series or only demands for some well stations, to allow flexibility in demand data processing. By default, all time series are read and are processed, whether they correspond to well stations or not.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellDemandTSMonthlyFromStateMod() Command**

This command reads well demand time series (monthly) from a StateMod well demand time series file.  
Well demand time series (monthly) are associated with well stations.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\commands\SortCropPatternTS

StateMod file:

Ignore diversion/well stations?:

Ignore well stations?:

Command:

ReadWellDemandTSMonthlyFromStateMod

## ReadWellDemandTSMonthlyFromStateMod() Command Editor

The command syntax is as follows:

```
ReadWellDemandTSMonthlyFromStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod monthly time series file to read.	None – must be specified.
IgnoreWells	Indicate whether the well nodes should be ignored. These are locations where only well supply is used. This requires that well stations have been read and the “associated diversion” values are set.	False
IgnoreDWs	Indicate whether the D&W nodes should be ignored. These are locations where well supply supplements surface water (diversion) supply. This requires that well stations have been read and the “associated diversion” values are set.	False

---

# Command Reference: ReadWellHistoricalPumpingTSMonthlyFromStateCU()

Read well historical pumping time series (monthly) data from a StateCU file

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `ReadWellHistoricalPumpingTSMonthlyFromStateCU()` command reads well historical pumping time series (monthly) and defines the data in memory. This command is used when estimating average efficiencies and calculating demand time series. All time series are read, whether or not they match the list of well stations. This command is equivalent to the `ReadWellHistoricalPumpingTSMonthlyFromStateMod()` command – use the commands as appropriate depending on which data set file is being read (the file format is the StateMod time series file format).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellHistoricalPumpingTSMonthlyFromStateCU() Command**

This command reads well demand time series (monthly) from a StateCU (or StateMod) well pumping time series file. Well demand time series (monthly) are associated with well stations. It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellHistoricalPumpingTSMonthlyFromStateMod

StateCU file:

Ignore diversion stations?:  Optional - ignore diversion stations in read (default=False).

Command:

ReadWellHistoricalPumpingTSMonthlyFromStateCU

## ReadWellHistoricalPumpingTSMonthlyFromStateCU() Command Editor

The command syntax is as follows:

```
ReadWellHisoricalPumpingTSMonthlyFromStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the well historical pumping time series (monthly) file to read.	None – must be specified.



---

# Command Reference:

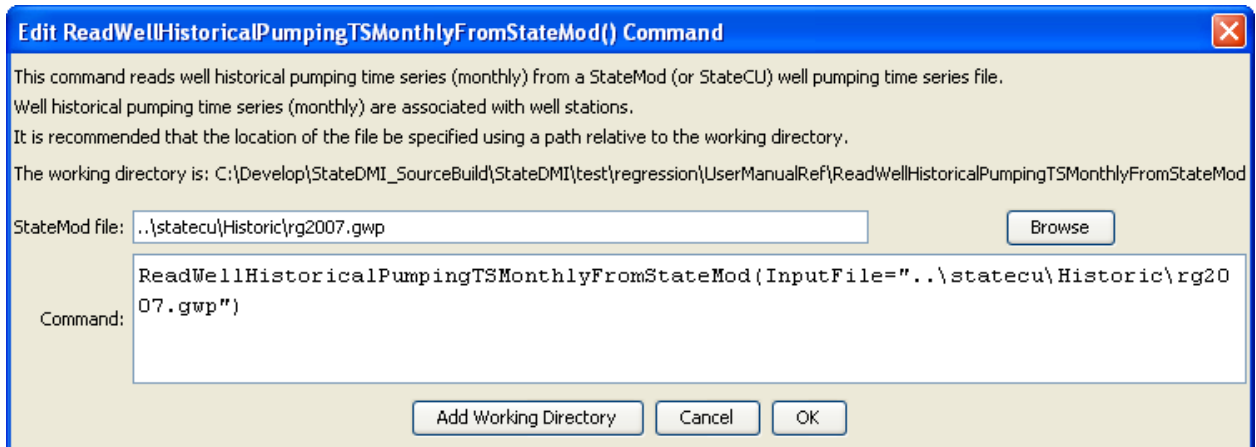
## ReadWellHistoricalPumpingTSMonthlyFromStateMod()

Read well historical pumping time series (monthly) data from a StateMod file

**StateMod Command**  
Version 3.09.00, 2010-01-26

The `ReadWellHistoricalPumpingTSMonthlyFromStateMod()` command reads well historical pumping time series (monthly) and defines the data in memory. This command is used when estimating average efficiencies and calculating demand time series. All time series are read, whether or not they match the list of well stations. This command is equivalent to the `ReadWellHistoricalPumpingTSMonthlyFromStateCU()` command – use the commands as appropriate depending on which data set file is being read (the file format is the StateMod time series file format).

The following dialog is used to edit the command and illustrates the syntax of the command.



ReadWellHistoricalPumpingTSMonthlyFromStateMod() Command Editor

The command syntax is as follows:

```
ReadWellHisoricalPumpingTSMonthlyFromStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod well historical pumping time series (monthly) file to read.	None – must be specified.

---

# Command Reference: ReadWellRightsFromHydroBase()

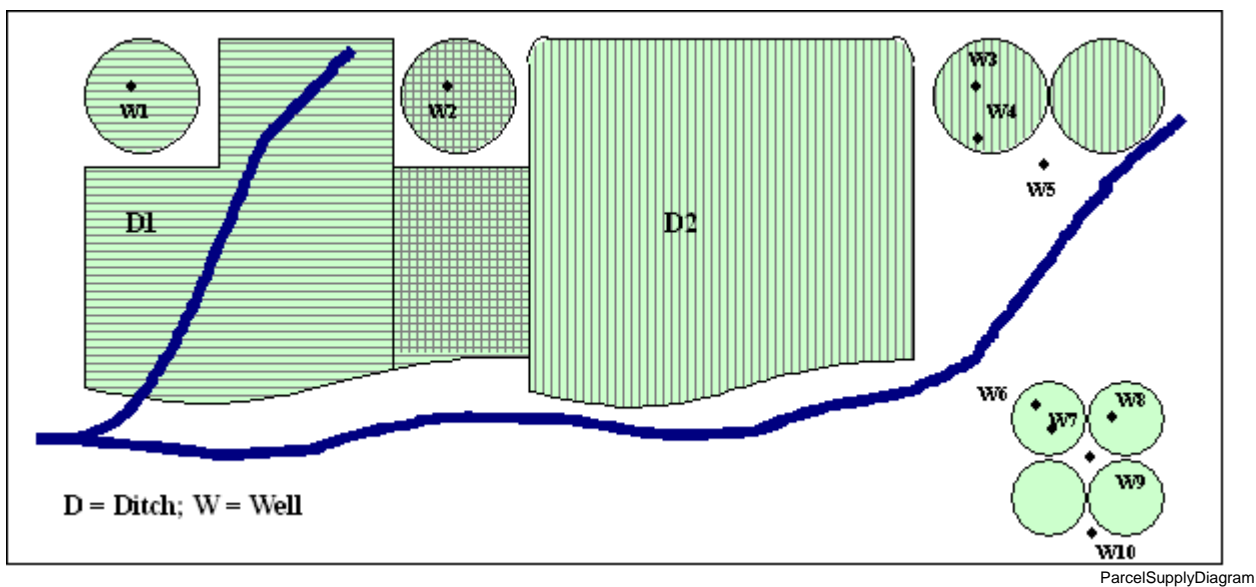
Read well right data from HydroBase

StateCU and StateMod Command

Version 3.09.00, 2010-01-25

The `ReadWellRightsFromHydroBase()` command reads well rights from HydroBase for each well station that is defined. The well rights can then be manipulated and output with other commands.

The following figure illustrates possible water supply for parcels.



Example Supply for Parcels

In this example, two ditches (D1 and D2, represented with hatching in vertical and horizontal directions) provide surface water supply to the indicated parcels. In some cases, only one ditch provides supply. Both ditches supply water to shared parcels that are indicated by cross-hatching in the figure. Wells can supplement surface water supply (parcels shown above the river in the figure) or can be the sole supplier of water (lower right) and wells do not need to be physically located on a parcel to provide supply to the parcel. For StateCU, well-only lands are identified by CU locations that are defined by a collection (aggregate/system) of parcels. For StateMod, well-only lands are well stations that do not have a related diversion station (and consequently also are defined by a list of parcels). Lands irrigated by surface water are identified with ditch identifiers and parcels are associated with the ditches in HydroBase. Processing logic is different for ditch and well-only lands only in how the list of parcels is obtained. Once a list of parcels is obtained, the wells and corresponding rights/permits associated with the parcels can be processed. Explicit wells and groups of such wells can also be modeled, in which case a list of WDIDs is provided for the wells. StateMod and StateCU files do not contain enough detail to indicate all of these conditions and therefore well station aggregate and system information is used by StateDMI (see the `SetWellAggregate()`, `SetWellAggregateFromList()`, `SetWellSystem()`, and `SetWellSystemFromList()` commands).

A well (hole in the ground) in HydroBase can be a structure with water rights, a well permit, or both (a matched location). In HydroBase, the relationship between well structure and well permit has been determined in CDSS projects by using a common well attributes (e.g., name) or by spatial proximity analysis using GIS tools. For general well data in HydroBase, there has been no explicit link to help identify when a well structure matched a well permit: well structures do not reference permits and well permits don't reference well structures. **This relationship is only available as a result of DSS projects for modeling.** Well permit records can be difficult to interpret because of replacement wells. Typically, major wells do have water rights, although a corresponding permit may also exist, perhaps with different date and other information. The CDSS projects have attempted to uniquely identify holes in the ground such that subsequent data processing can treat the hole as a structure or permit, but not both (to avoid double-counting). Wells were first modeled in the Rio Grande RGDSS project and subsequently the South Platte.

The steps used to determine well rights are described below. Note that “well station” refers to the StateMod model node (which is often a collection of wells associated with groundwater-only lands, a ditch, or explicit well structures with WDIDs) and “well” refers to a hole in the ground that has physical characteristics, water rights, and/or well permits, and a relationship with one or more parcels.

Loop through each location that matches the ID pattern and perform the following:

For each year being processed (specified by the `Year` parameter or by default all available parcel years in HydroBase for the specified water division), perform the following:

1. Evaluate the type of location to set up further processing
  - a. If the location is a diversion station or collection specified with part type `Ditch`, go to step 2.
  - b. If the location is a well station or collection specified with part type `Parcel`, go to step 2.
  - c. If the location is an explicit well (with `WDID`) or collection specified with part type `Well`, go to step 4 (no need to involve parcels in processing).
2. Get the list of parcels associated with the location (note that in a given year there may be zero or more parcels associated with a location):
  - a. If the location is a groundwater-only location, get the list of parcels from the aggregate/system definitions, where `PartType=Parcel`.
  - b. If the location diversion+well node (and/or an aggregate/system where `PartType=Ditch`):
    - i. If the ditch is explicit (no aggregate/system information has been provided for the location), get the list of parcels associated with the single ditch.
    - ii. If the ditch is an aggregate/system, get the list of parcels associated with each part of the aggregate/system and form one list of parcels.
3. Get the list of wells (holes in the ground) from the joined parcel/well data using the parcel identifiers.
  - a. Query HydroBase to get the joined parcel/well data, using the parcel year, division, and parcel identifier.
4. Get the HydroBase well right/permit detailed data. Based on command parameters, read the HydroBase well rights and permits as follows:
  - If the `ReadWellRights=False`, use the well/parcel matching data without further reads; consequently the resulting well right information may not exactly match all the rights that are available in HydroBase because the well matching results are a sum of net amount rights.
  - If `ReadWellRights=True` and a well has a `WDID`, the well rights are re-read from the HydroBase net amounts table. This ensures that all information is considered, including APEX. This parameter setting is recommended and will always be used for explicit wells (those with no associated diversion).

- In either case, well permits are taken from the well/parcel matching data for quality control reasons and because HydroBase traditionally has not been distributed with well permit data. Use the `DefineRightHow` parameter value to determine how to define the right. If the value of `DefineRightHow=RightIfAvailable` (recommended in current procedures):
  - Set the date.
    - If `ReadWellRights=True`, read the individual well rights from HydroBase. If a water right is available, use the appropriation date (and corresponding administration number) for the water right. If no date is available for the water right (this should not happen), assign the administration number to the value corresponding to the `DefaultAppropriationDate` parameter value or 99999.99999 as a final default.
    - If `ReadWellRights=False`, use the processed appropriation date determined during the irrigated lands load process.
  - Set the decree amount.
    - If `ReadWellRights=True`, use the decree from the water rights (CFS). If `UseApex=True`, the alternate point/exchange values will also be added to the well right decree. Because well rights typically have either the decree or the APEX (not both), this will result in water rights that are either the decree or the APEX value. Multiply the right amount by the percent of the well that irrigates the parcel (AND the percent of the parcel that is irrigated by the ditch if the lands are associated with a ditch). If warnings are generated, it may be due to older well matching data indicating that well rights should be in HydroBase; however, subsequent changes now result in no net amounts in the database. Additional evaluation of loaded data may need to occur.
    - If `ReadWellRights=False`, assign the decree as the well yield determined from well matching (converted from GPM to CFS), multiplied by the percent of the well that irrigates the parcel (AND the percent of the parcel that is irrigated by the ditch if the lands are associated with a ditch).

Else if `DefineRightHow=EarliestDate` (used with Phase 4 Rio Grande data set):

- From the DSS well matching data, use the earliest of the right's appropriation date and permit's permit date. Convert the date to an administration number. If no date is available, assign the administration number to the value corresponding to the `DefaultAppropriationDate` parameter value or 99999.99999 as a final default.
- Assign the decree as the well yield, converted from GPM to CFS, multiplied by the percent of the well that irrigates the parcel (AND the percent of the parcel that is irrigated by the ditch if the lands are associated with a ditch).
- This option currently does not allow reading well right net amounts.

Else if `DefineRightHow=LatestDate` (used experimentally): similar to above, except the latest date is used.

5. Add the StateMod well rights for the location by converting the HydroBase rights to StateMod rights.
  - Water rights from HydroBase that are less than the decree minimum (.0005 CFS, as per previously determined conventions) are ignored and during final output, water rights with a decree of 0.00 (the StateMod file format) are ignored.
  - The identifier will be assigned as specified by the `IDFormat` parameter.
  - The name of the final right will include either water right (WDID and name) or permit information (number, suffix, and replacement), depending on the input that was used.

In the above process, status messages and warnings are printed to the log file as appropriate and command status messages are added. For example, the following information is listed in the log file: the number of

parcels for a well station, the number of wells for the parcel, and the number of rights/permits for the well.

After reading the well rights from HydroBase, it is typical to write the results to a file similar to *rg2007\_NotMerged.wer*. This file can then be used to fill crop pattern and irrigation practice acreage time series. The water rights determined from multiple years can then be processed with the `MergeWellRights()` command, resulting in a file that can be used for modeling (if all rights are to be modeled) and to set the irrigation practice pumping maximum time series – this file typically has a name similar to *rg2007.wer*. Finally, if aggregation of well rights by administration number class is desired, the `AggregateWellRights()` command can be used, and the results written to a file with a name similar to *rg2007\_Agg.wer*.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellRightsFromHydroBase() Command**

This command reads well rights from HydroBase, using the well station identifiers to find rights. Water rights are determined from summarized well right and permit data, which have been matched with wells and parcels. Summary data can be used as is, or well rights can be requested to obtain individual net amount rights. Alternate point or exchange (APEX) decrees provide additional rights. If the well rights are to be aggregates, use the `AggregateWellRights()` command to reduce the number (but not decree sum) of rights in the model. See also `MergeWellRights()` command, which minimizes duplicate rights due to multiple parcel years.

Well station ID:	<input type="text" value="*"/>	Required - well stations to read (use * for wildcard).
Water Division (Div):	<input type="text" value="1"/>	Required - water division for the parcels.
Year:	<input type="text" value="1987,2001,2005"/>	Optional - year(s) for the parcels, separated by commas (default=all available).
Decree minimum:	<input type="text"/>	Optional - minimum decree to include (default = .0005 CFS).
Right ID format:	<input type="text" value="HydroBaseID"/>	Optional - format for right identifiers (default=StationIDW.NN).
Default appropriation date:	<input type="text" value="1950-01-01"/>	Optional - use if date is not available from right or permit (default=99999.99999 admin. num.).
Define right how?:	<input type="text" value="RightIfAvailable"/>	Optional - how to define right from HydroBase right/permit (default=EarliestDate).
Read well rights?:	<input type="text" value="True"/>	Optional - read well rights rather than relying on well matching results (default=True).
Use Apex?:	<input type="text" value="True"/>	Optional - add APEX amount to right amount (default=False).
OnOff default:	<input type="text" value="AppropriationDate"/>	Optional - default StateMod OnOff switch (default=AppropriationDate).
Optimization level:	<input type="text"/>	Optional - optimize performance (default=UseMoreMemory).

Command:

```
ReadWellRightsFromHydroBase (ID="*", IDFormat="HydroBaseID", Year="1956, 1976, 1987, 2001, 2005", Div="1", DefaultAppropriationDate="1950-01-01", DefineRightHow=RightIfAvailable, ReadWellRights=True, UseApex=True, OnOffDefault=AppropriationDate)
```

OK Cancel

ReadWellRightsFromHydroBase

### ReadWellRightsFromHydroBase() Command Editor

An excerpt from a StateMod well rights file with data comments is shown below. The parcel year, well/parcel matching class, and parcel ID are shown on the far right and are not part of the standard StateMod well right file. Well class 4 and 9 are “estimated wells”, which are essentially a copy of other wells. These values are used by the `MergeWellRights()` command. See CDSS technical memoranda for a description of well classes (SPDSS Task Memorandum “SPDSS, Spatial System Integration Component, Well Class Adjustments”, March 15<sup>th</sup>, 2007)

#>	ID	Name	Struct	Admin #	Decree	On/Off	PYr--Cls--PID
#>	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	exb--exb--exb--e
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1936 1 3107
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956	1936 1 3107
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1998 2 11016
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956	1998 2 11016
2005001	W0006	WELL NO 01	200812	31592.00000	1.19	1936	2002 2 20901
2005001	W0006	WELL NO 01	200812	38836.00000	0.62	1956	2002 2 20901
2005001	W0006	WELL NO 01	200812	31592.00000	1.15	1936	2002 5 20902
2005001	W0006	WELL NO 01	200812	38836.00000	0.61	1956	2002 5 20902

The command syntax is as follows:

```
ReadWellRightsFromHydroBase( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Div	A water division to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems.	None – must be specified.
Year	A calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems. Separate multiple years with commas. If years are specified and data for a year in HydroBase is omitted, the results will be generated by ignoring the HydroBase data year – this is only advised if a year of data in HydroBase is purposefully being ignored for some reason.	Read all parcel years in HydroBase.
DecreeMin	Minimum decree to include, CFS. Well permits are converted from GPM to CFS prior to checking the value. Note that StateMod well right files typically have a precision of two digits after the decimal and therefore including small rights may result in a decree of zero (unless the rights sum/aggregate to a larger number).	.0005
IDFormat	Indicate the format to be used for water right identifiers, one of: <ul style="list-style-type: none"> <li>HydroBaseID – use the 7-digit WDID if the well structure identifier is used. If a well permit, use the well receipt number followed by :P (see note below about estimated wells). The identifier that is used is controlled by the DefineRightHow parameter. This value should be used when wells are being explicitly modeled (no water right aggregation), such as on the South Platte.</li> <li>StationIDW.NN – use the well station identifier concatenated with W. and a two digit number. This convention matches the approach that has traditionally been used in earlier CDSS modeling, in particular in Phase 4 Río Grande modeling where</li> </ul>	StationIDW.NN (because this was used in the Rio Grande; however, HydroBaseID is recommended when not aggregating rights, such as in the South Platte).

Parameter	Description	Default
	<p>well rights are aggregated. Modeling in the South Platte requires that wells are not aggregated and using the HydroBaseID is necessary.</p> <p>Estimated wells, as defined by well supply to parcel matching classes 4 and 9, have identifiers that are concatenated with :PE if a permit or :WE if a well right. This allows the wells to be uniquely identified when processed with the MergeWellRights() command.</p>	
Default Appropriation Date	Some right/permit data does not have a date in data records. For example, very old well permits may not have a date. In these cases a default date can be assigned to be used as the appropriation date in the well water right. The appropriation date will be converted to a State of Colorado administration number in StateMod water rights.	The administration number is set to 99999.99999.
DefineRightHow	<p>Wells (holes in the ground) are matched with water rights, well permits, and occasionally “estimated” wells necessary because a water right or permit could not be found. In some cases a right and permit will both exist for a well, each with their own dates. This parameter indicates how to define the right in these cases and has a value of:</p> <ul style="list-style-type: none"> <li>• EarliestDate – will use the earliest date determined from the right’s appropriation date and the permit’s permit date from well matching data. ReadWellRights=True is not enabled or used.</li> <li>• LatestDate – will use the latest date determined from the right’s appropriation date and the permit’s permit date from well matching data. ReadWellRights=True is not enabled or used.</li> <li>• RightIfAvailable – will always use the water right appropriation date, if available. If ReadWellRights=True (see below), the net amount rights are read. If ReadWellRights=False, the processed well data determined when irrigated lands are loaded into HydroBase are used.</li> </ul>	EarliestDate
ReadWellRights	<p>This parameter is only used when DefineRightHow=RightIfAvailable, and indicates whether individual water rights should be read from HydroBase. The following values are recognized:</p> <ul style="list-style-type: none"> <li>• True – the net amounts data are read, which may result in multiple well water rights for a well WDID. See also the UseApex parameter.</li> <li>• False – a single processed water right will be returned, which is the sum of net amount rights, using the oldest appropriation date found for the rights (APEX is not considered). This information is</li> </ul>	True



Parameter	Description	Default
	taken from the well/parcel matching results.	
UseApex	<p>This parameter indicates whether to use alternate point/exchange values when processing rights. The following values are recognized:</p> <ul style="list-style-type: none"> <li>• True – the APEX values corresponding to well rights are added to the net amount right values, resulting in a larger decree being considered for some rights.</li> <li>• False – the APEX values are not added to net amount rights.</li> </ul> <p>Because net amount rights usually either have a decreed rate or an APEX amount, using True will generally result in more water rights, where the resulting right amount is either the decree or APEX.</p>	False
OnOffDefault	Indicates how to set the on/off switch for all water rights that are processed. A value of 1 indicates that the right is on for the whole period. If the value is AppropriationDate, the switch is set to the year corresponding to the appropriation date, indicating that the right will be turned on starting in the year. Use set commands to reset the switch to other values.	Appropriation Date
Optimization	<p>Indicate how queries are performed, one of:</p> <ul style="list-style-type: none"> <li>• UseLessMemory – run time will be slower, but this may be required on computers that do not have enough memory for optimization</li> <li>• UseMoreMemory – run time will be faster, but more computer memory is required</li> </ul>	UseMoreMemory

The following example command file illustrates how well rights can be defined, sorted, checked, and written to a StateMod file:

```
# Well Rights File (*.wer)
#
StartLog(LogFile="Sp2008L_WER.log")
#
# Step 1 - Read all structures
#
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
SortWellStations()
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,Div=1,
```

```

PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=1,IfNotFound=Ignore)
#
# Step 5 - Read rights from HydroBase
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1956,1976,1987,2001,2005",
    Div="1",DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
    ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds & ipy acreage filling
# Write Data Comments="False" provides merged file used for seting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Sp2008L.wer.check.html",Title="Well Rights Check File")

```

---

# Command Reference: ReadWellRightsFromStateMod()

Read well right data from a StateMod well rights file

StateCU and StateMod Command

Version 3.09.00, 2010-01-00

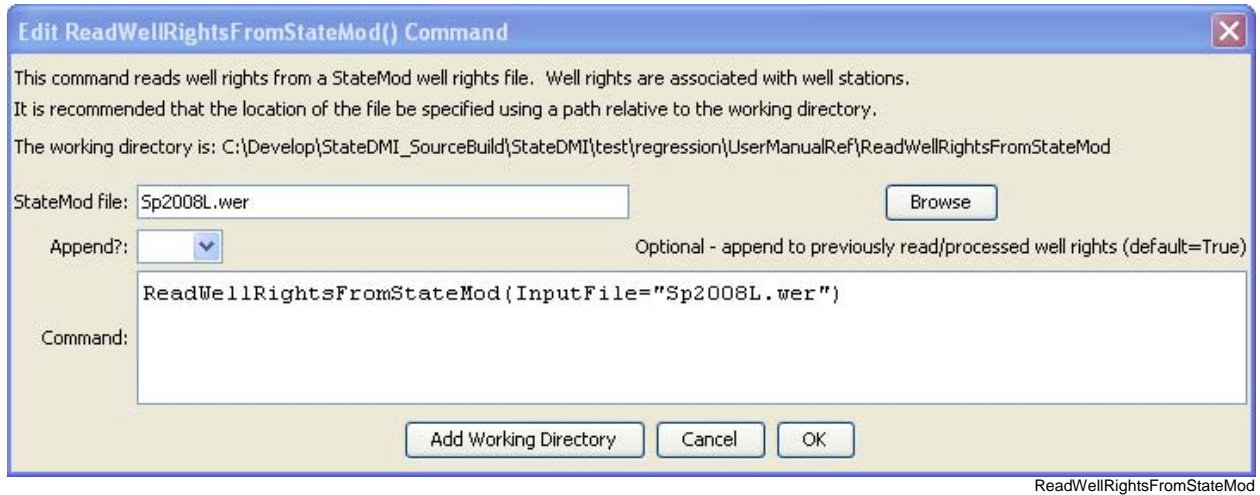
The `ReadWellRightsFromStateMod()` command reads well rights from a StateMod well rights file. The well rights can then be manipulated and output with other commands. Current procedures for processing some data files involve utilizing StateMod well rights file(s) as input rather than rereading well rights from HydroBase. This ensures that the content and quality of the well rights file can be verified and is used consistently when processing various model files. For example, StateMod well rights files may be used for the following purposes:

1. Turning off groundwater only parcels irrigated lands (crop pattern and irrigation practice acreage time series) prior to a specific year of parcel data. Parcels that do not have a well right in a year are estimated to not be irrigated for the year (and prior years). This step typically uses a well rights file that has NOT had water rights for various parcel years merged – data for a single parcel year will be used. See the `WriteWellRightsToStateMod(..., WriteDataComments=True, ...)` command.
2. Setting the pumping maximum in the irrigation practice time series. This step typically uses a well rights file that includes well rights for different parcel years that have been merged. For example, well rights from 1998 parcels and those from 2002 parcels are merged to not double count rights. See the `MergeWellRights()` command.
3. Setting the well station capacity to the total of the well rights. This step may use a merged rights file or one that has been aggregated (well rights summed into administration number classes). Aggregation is used in some data sets to decrease the complexity of the model. However, aggregation is generally NOT performed in data sets such as the South Platte where augmentation plans reference individual well rights. See the `AggregateWellRights()` command.

The well rights files mentioned above are typically written to separate files with unique names. To support cases 1 and 2 above, the StateMod file should include “data comments” on the far right, which include the well to parcel matching data (parcel year, class, parcel ID). An excerpt from a well rights file with this information is shown below:

#>	ID	Name	Struct	Admin #	Decree	On/Off	PYr--Cls--PID
#>	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	-----eb-----	exb--exb--exb-----e
#>							
#>							
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936 1936	1 3107
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956 1936	1 3107
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936 1998	2 11016
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956 1998	2 11016
2005001	W0006	WELL NO 01	200812	31592.00000	1.19	1936 2002	2 20901
2005001	W0006	WELL NO 01	200812	38836.00000	0.62	1956 2002	2 20901
2005001	W0006	WELL NO 01	200812	31592.00000	1.15	1936 2002	5 20902
2005001	W0006	WELL NO 01	200812	38836.00000	0.61	1956 2002	5 20902

The following dialog is used to edit the command and illustrates the syntax of the command.



**ReadWellRightsFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadWellRightsFromStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod well rights file to be read.	None – must be specified.
Append	Indicate whether the data should be appended to in-memory well rights. Specify <code>False</code> if processing well rights for independent tasks and the current task should not be influenced by previously read well rights.	True

# Command Reference: ReadWellStationsFromList()

Read well stations data from a list file

**StateMod Command**

Version 3.09.00, 2010-01-24

The `ReadWellStationsFromList()` command reads a list of well stations from a delimited list file and defines well stations in memory. The well stations can then be manipulated and output with other commands. Reading from a list is more general than reading from a StateMod file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellStationsFromList() Command**

This command reads well stations from a list file containing columns of information.  
A well station can be one of the following:  
1) location where only groundwater supply is used to meet demand  
2) location where groundwater supply supplements surface water (diversion) supply to meet demand.  
In this case, see also diversion stations that are supplemented by well pumping -  
both diversion and well stations must be defined and the Diversion ID must be specified for the well (see below).  
Columns should be delimited by commas (user-specified delimiters will be added in the future).  
Identifiers and names (and in some cases other information) can be read - most subsequent commands only need a list of identifiers.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellStationsFromList

List file:

ID column:  Required - column (1+) for ID.

Name column:  Optional - column (1+) for name.

Diversion ID column:  Optional - column (1+) to link the well to a diversion location.

Command:  

```
ReadWellStationsFromList(ListFile="Sp2008L_AugRchWells.csv", IDCol=1)
```

ReadWellStationsFromList

**ReadWellStationsFromList() Command Editor**

The command syntax is as follows:

ReadWellStationsFromList(Parameter=Value,...)

## Command Parameters

Parameter	Description	Default
ListFile	The name of the list file to be read.	None – must be specified.
IDCol	The column number (1+) containing the well station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the well station names.	None – optional (name will be initialized to blank).
DiversionIDCol	The column number (1+) containing the diversion identifiers associated with the well stations. This is needed in some cases to determine when a location is a diversion with supplemental well supply, for example, when processing well rights.	None – optional (diversion ID will be initialized to blank).

At a minimum, the list file must contain a column with well station identifiers. Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

A sample list file is shown below:

```
#
# sp2007L_AugRchWells.csv
# _____
#
# rrb Augmentation and recharge wells from SmOpr
#      SmOpr
#      State of Colorado
#      Version: 1.00
#      Last revision date: 2006/10/27
#
# rrb 2008/10/08 count = 17 + 4 21
# _____
#
#
0102522_AuW , "RIVERSIDE AUG          ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
0102528_AuW , "FT MORGAN CNL AUG PLAN   ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
0102529_AuW , "UPPER PLATTE BEAVER AUG    ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
0102535_ReW , "LOWER PLATTE BEAVER AUG    ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
0102535_AuW , "LOWER PLATTE BEAVER AUG    ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
6402015_ReW , "TAMARAK                     ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
6402027_ReW , "OVERLAND                    ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
6402518_AuW , "HARMONY Aug Well           ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
6402519_ReW , "DINSDALE AUG               ", ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'NA'
...

```

# Command Reference: ReadWellStationsFromNetwork()

Read well station data from a network file

StateCU and StateMod Command

Version 3.09.00, 2010-01-25

The `ReadWellStationsFromNetwork()` command reads a list of well stations from a StateMod network file (XML or Makenet) and defines well stations in memory. The well stations can then be manipulated and output with other commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellStationsFromNetwork() Command**

This command reads well station data from a StateMod XML network file.  
Well stations indicate locations where ONLY groundwater supply is used to meet demand,  
and locations where wells supplement surface water supply for diversion stations.  
Well and D&W (diversion with well[s]) nodes are read from the network file.  
If the network file is not specified, it is assumed that the network has already been read by the network editor or another command.  
If the network file specified, it is read and will be available to later commands.  
The following station data are set from the network:  
Identifier (station ID)  
River node ID - set to station ID  
Daily ID - set to the nearest downstream streamflow station ID.  
See also the Fill\*FromNetwork() commands to fill missing data.  
It is recommended that the path to the file be specified relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellStationsFromNetwork

StateMod network file:

Command:

ReadWellStationsFromNetwork

## ReadWellStationsFromNetwork() Command Editor

The command syntax is as follows:

```
ReadWellStationsFromNetwork(param=value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the network file to read.	None – must be specified.

The following example command file illustrates how well rights can be defined, sorted, checked, and written to a StateMod file:

```
# Well Rights File (*.wer)
#
StartLog(LogFile="Sp2008L_WER.log")
#
# Step 1 - Read all structures
#
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
SortWellStations()
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=1,IfNotFound=Ignore)
#
# Step 5 - Read rights from HydroBase
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1956,1976,1987,2001,2005",
    Div="1",DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
    ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds & ipy acreage filling
# Write Data Comments="False" provides merged file used for seting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",
    WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Sp2008L.wer.check.html",Title="Well Rights Check File")
```



---

# Command Reference: ReadWellStationsFromStateMod()

Read well station data from a StateMod well stations file

## StateCU and StateMod Command

Version 3.09.00, 2010-01-24

The `ReadWellStationsFromStateMod()` command reads a list of well stations from a StateMod well stations file and defines well stations in memory. The well stations can then be manipulated and output with other commands. The StateMod well stations file contains stations for which only groundwater supply is available and stations for which groundwater supply supplements surface water supply of a diversion station (in this case the well station data includes the diversion station identifier). For some data (e.g., demands), StateMod accepts data from multiple files. For example, diversion and diversion+well stations may read total demands from the diversion demands file and well (groundwater only) stations may read demands from the well demands file. Parameters are available in this command to read all well stations or only a subset, to allow flexibility in data processing. Other commands may also process a subset, regardless of what is read.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit ReadWellStationsFromStateMod() Command**

This command reads well stations from a StateMod well stations file.  
Well stations indicate locations where ONLY groundwater supply is used to meet demand  
and locations where groundwater supply supplements surface water (diversion) supply.  
(See also diversion stations that are supplemented by well pumping.)  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellStationsFromStateMod

StateMod file:

Ignore diversion/well stations?:  Optional - ignore diversion/well stations in read (default=False).

Ignore well stations?:  Optional - ignore well stations in read (default=False).

Command:

ReadWellStationsFromStateMod

**ReadWellStationsFromStateMod() Command Editor**

The command syntax is as follows:

```
ReadWellStationsFromStateMod( Parameter=Value , ...)
```

#### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod well stations file to be read.	None – must be specified.
IgnoreDws	Indicate whether the D&W well nodes should be ignored. These are locations where well supply supplements surface water (diversion) supply.	False
IgnoreWells	Indicate whether the well nodes should be ignored. These are locations where only well supply is used.	False

---

# Command Reference: RemoveCropPatternTS()

## Remove crop pattern time series

### StateCU Command

Version 3.09.01, 2010-02-01

The RemoveCropPatternTS() command removes crop pattern time series data. This is useful when inappropriate crop types have been processed into the crop pattern time series (e.g., unirrigated parcels), which may be the case during when using preliminary data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit RemoveCropPatternTS() Command**

This command removes a crop type from the crop pattern time series.  
This is sometimes necessary because the irrigated lands data may include non-irrigated lands for some crop types.  
The time series to be removed can be identified with CU locations and/or crop types.

CU Location ID:  Required - specify the CU Location(s) to fill (use \* for wildcard)

Crop type:  Required - for example "USER-MEADOW", use \* to match all.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
RemoveCropPatternTS ( ID="*", CropType="USER-MEADOW" )
```

RemoveCropPatternTS

### RemoveCropPatternTS() Command Editor

The command syntax is as follows:

```
RemoveCropPatternTS(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier or pattern to match (e.g., 20*).	None – must be specified.
CropType	A single crop type identifier to match.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: RunCommands()

## Run a command file

Version 3.08.02, 2010-01-07

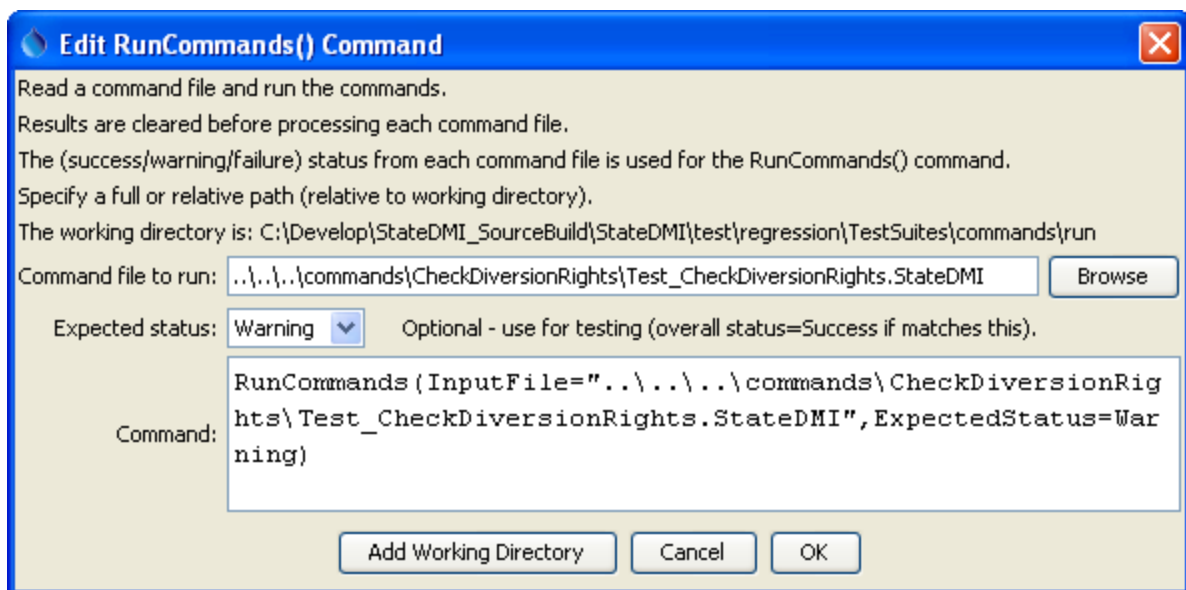
The `RunCommands()` command runs a command file. This command can be used to manage workflow where multiple commands files are run and is also useful for testing, where a test suite consists of running separate test case command files.

Command files that are run can themselves include `RunCommands()` commands. Each command file that is run has knowledge of its initial working directory and relative paths referenced in the command file are relative to this directory. This allows a master command file to reside in a different location than the individual command files that are being run. The current working directory is reset to that of the command file being run.

Currently the properties from the parent command file are NOT applied to the initial conditions when running the command file. Therefore, global properties like input and output period are reset to defaults before running the command file. A future enhancement may implement a property to indicate whether to inherit the properties. The output from the command is also not added to the parent processor. Again, a future enhancement may be to append output so that one final set of output is generated.

There is currently no special handling of log files; consequently, if the main command file opens a log file and then a command file is run that opens a new log file, the main log file will be closed. This behavior is being evaluated.

The following dialog is used to edit the command and illustrates the syntax for the command.



RunCommands

**RunCommands() Command Editor**

The command syntax is as follows:

```
RunCommands ( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the command file to run, enclosed in double quotes if the file contains spaces or other special characters. A path relative to the master command file can be specified.	None – must be specified.
ExpectedStatus	Used for testing – indicates the expected status from the command, one of: <ul style="list-style-type: none"> <li>Unknown</li> <li>Success</li> <li>Warning</li> <li>Failure</li> </ul>	Success
AppendResults	Envisioned for implementation in the future. Indicate whether time series results from each command file should be appended to the overall time series results. This parameter currently always defaults to False, but support for True may be implemented in the future. Consequently, only the time series results from the last command file that is run will be displayed.	Currently always False

The following example illustrates how the RunCommands ( ) command can be used to test software (or any implementation of commands that represent a standard process). First, individual command files are implemented to test specific functionality, which will result in warnings if a test fails:

```
# Test check diversion rights data where each checked value is in error
# The set command won't let invalid data be set from parameters so read bad data
# to trigger the check warnings.
# Compare the data csv to make sure the data are being produced as expected
# and the check file csv to make sure the checks are working.
# The expected status is Warning because the check will detect the missing values.
#@expectedStatus Warning
StartLog(LogFile="Results/Test_CheckDiversionRights.StateDMI.log")
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out.csv",IfNotFound=Ignore)
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out_check.csv",IfNotFound=Ignore)
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out_check.html",IfNotFound=Ignore)
# Define a diversion station to trigger the check of stations
SetDiversionStation(ID="Diversion1",IfNotFound=Add)
SetDiversionRight(ID="Location1",IfNotFound=Add)
# Also read some bad data...
ReadDiversionRightsFromStateMod(InputFile="Data\simple.ddd")
# Uncomment the following command to regenerate the expected results.
#
WriteDiversionRightsToList(OutputFile="ExpectedResults/Test_CheckDiversionRights_out.csv")
WriteDiversionRightsToList(OutputFile="Results/Test_CheckDiversionRights_out.csv")
CompareFiles(InputFile1="ExpectedResults/Test_CheckDiversionRights_out.csv",
InputFile2="Results/Test_CheckDiversionRights_out.csv",WarnIfDifferent=True)
#
# Check the data and create the check file.
CheckDiversionRights(ID="*")
# Uncomment the following command to regenerate the expected results.
# WriteCheckFile(OutputFile="ExpectedResults/Test_CheckDiversionRights_out_check.csv")
```

```
WriteCheckFile(OutputFile="Results/Test_CheckDiversionRights_out_check.csv")
WriteCheckFile(OutputFile="Results/Test_CheckDiversionRights_out_check.html")
CompareFiles( InputFile1="ExpectedResults/Test_CheckDiversionRights_out_check.csv",
  InputFile2="Results/Test_CheckDiversionRights_out_check.csv",WarnIfDifferent=True)
```

Next, use the RunCommands ( ) command to run one or more tests:

```
StartRegressionTestResultsReport(
  OutputFile="RunRegressionTest_commands_general.StateDMI.out.txt")
...
RunCommands(
  InputFile="..\..\..\commands\CheckDiversionRights\Test_CheckDiversionRights.StateDMI",
  ExpectedStatus=Warning)...
```

Each of the above command files should produce expected results, without warnings. If any command file unexpectedly produces a warning, a warning will also be visible in StateDMI. The issue can then be evaluated to determine whether a software or configuration change is necessary.

This page is intentionally blank.



# Command Reference: RunProgram()

## Run an external program

### General Command

Version 3.08.02, 2010-01-07

**This command is under development. Preliminary development has occurred in the TSTool software.** The `RunProgram()` command runs an external program, given the full command line or individual command line parts, and waits until the program is finished before processing additional commands. The command will indicate a failure if the exit status from the program being run is non-zero. It is therefore possible to call an external program that reads and/or writes recognized time series formats to perform processing that StateDMI cannot. It is also useful to use StateDMI's testing features to implement quality control checks for other software tools.

StateDMI internally maintains a working directory that is used to convert relative paths to absolute paths to locate files. The working directory is by default the location of the last command file that was opened. The external program may assume that the working directory is the location from which StateDMI software was started (or the installation location if started from a menu). Therefore, it may be necessary to run StateDMI in batch mode from the directory where the external software's data files exist, use absolute paths to files, or use the `${WorkingDir}` property in the command line. Use `\` in the command line or arguments to surround whitespace. Some operating systems may have limitations on command line length. The following dialog is used to edit the command and illustrates command syntax.

**Edit RunProgram() command**

This command runs another program, and TSTool waits for it to complete before continuing. Commands must use a full path to files, TSTool must be started from the directory where files exist to use relative paths, or use `${WorkingDir}` in the command line to specify files relative to the working directory. Use `"` to indicate double quotes if needed to surround program name or program command-line parameters - this may be needed if there are spaces in paths. Specify the exit status indicator if program output messages must be used to determine the program exit status (e.g., "Status:"). Specify the program to run using the command line OR separate arguments - the latter makes it simpler to know how to treat whitespace in command line arguments. The program by default will be run with a command shell (e.g., cmd.exe on Windows) - specify as False if it is known that the program is an executable (not a shell command or script).

Command to run (with arguments):

```
echo Hello > ${WorkingDir}/Results/Test_RunProgram_CommandLine_echo_out.txt
```

Program to run:  Required - if full command line is not specified above.

Program argument 1:  Optional - as needed if Program is specified.

Program argument 2:  Optional - as needed if Program is specified.

Program argument 3:  Optional - as needed if Program is specified.

Program argument 4:  Optional - as needed if Program is specified.

Program argument 5:  Optional - as needed if Program is specified.

Program argument 6:  Optional - as needed if Program is specified.

Program argument 7:  Optional - as needed if Program is specified.

Program argument 8:  Optional - as needed if Program is specified.

Use command shell: ☐ Optional - use command shell (default=True).

Timeout (seconds):  Optional - default is no timeout.

Exit status indicator:  Optional - output string to indicate status (default=use process exit status).

Command:

```
RunProgram(CommandLine="echo Hello > ${WorkingDir}/Results/Test_RunProgram_CommandLine_echo_out.txt")
```

Cancel OK

RunProgram

### RunProgram() Command Editor when Specifying Command Line

The command syntax is as follows:

```
RunProgram(Parameter=Value...)
```

### Command Parameters

Parameter	Description	Default
CommandLine	<p>The full program command line, with arguments. If the program executable is found in the PATH environment variable, then only the program name needs to be specified. Otherwise, specify an absolute path to the program or run StateDMI from a command shell the same directory.</p> <p>The <code>\${WorkingDir}</code> property can be used in the command line to indicate the working directory (command file location) when specifying file names.</p> <p>For Windows, it may be necessary to place a <code>\</code> at the start and end of the command line, if a full command line is specified.</p>	<p>Must be specified if the Program parameter is not specified.</p> <p>The Program parameter will be used if both are specified.</p>
Program	The name of the program to run. Program arguments are specified using the ProgramArg# parameter(s). See the CommandLine parameter for more information about parameter formatting and locating the executable.	Must be specified if the CommandLine parameter is not specified.
ProgramArg1, ProgramArg2, etc.	Command like arguments used with Program. If necessary, use <code>\${WorkingDir}</code> to specify the working directory to locate files.	No arguments will be used with Program.
UseCommandShell	If specified as False, the program will be run without using a command shell. A command shell is needed if the program is a script (batch file), a shell command, or uses <code>&gt;</code> , <code> </code> , etc.	True, using <code>cmd.exe /C</code> on Windows and <code>/bin/sh -c</code> on UNIX/Linux.
Timeout	The timeout in seconds – if the program has not yet returned, the process will be ended. Zero indicates no timeout. <b>This behavior varies and is being enhanced.</b>	No timeout.
ExitStatus Indicator	By default, the program exit status is determined from the process that is run. Normally 0 means success and non-zero indicates an error. However, the program may not exit with a non-zero exit status when an error occurs. If the program instead uses an output string like STOP 3 to indicate the status, use this parameter to indicate the leading string, which is followed by the exit status (e.g., STOP).	Determine the exit status from the process exit value.

The following figure illustrates how a command would be entered using the program name and parts, and use the command shell to run. Note that the output redirection character “>” is entered as a program argument. The *echo* program on Windows is actually internal to the *cmd.exe* command shell and therefore must be run using the command shell (the default behavior).

**Edit RunProgram() command**

This command runs another program, and TSTool waits for it to complete before continuing.  
 Commands must use a full path to files, TSTool must be started from the directory where files exist to use relative paths, or use \${WorkingDir} in the command line to specify files relative to the working directory.  
 Use \" to indicate double quotes if needed to surround program name or program command-line parameters - this may be needed if there are spaces in paths.  
 Specify the exit status indicator if program output messages must be used to determine the program exit status (e.g., "Status:").  
 Specify the program to run using the command line OR separate arguments - the latter makes it simpler to know how to treat whitespace in command line arguments.  
 The program by default will be run with a command shell (e.g., cmd.exe on Windows) - specify as False if it is known that the program is an executable (not a shell command or script).

Command to run (with arguments):

Program to run:  Required - if full command line is not specified above.

Program argument 1:  Optional - as needed if Program is specified.

Program argument 2:  Optional - as needed if Program is specified.

Program argument 3:  Optional - as needed if Program is specified.

Program argument 4:  Optional - as needed if Program is specified.

Program argument 5:  Optional - as needed if Program is specified.

Program argument 6:  Optional - as needed if Program is specified.

Program argument 7:  Optional - as needed if Program is specified.

Program argument 8:  Optional - as needed if Program is specified.

Use command shell: ☒ Optional - use command shell (default=True).

Timeout (seconds):  Optional - default is no timeout.

Exit status indicator:  Optional - output string to indicate status (default=use process exit status).

Command:

Cancel OK

RunProgram\_Program

**RunProgram() Command Editor when Specifying Program and Arguments**

The following figure illustrates how a command can be run without a command shell and using the program output to determine the exit status. The *testecho.exe* program is a compiled executable and can therefore be run without a command shell. Because the standard output is being evaluated for the exit value, the output cannot be redirected to a file with `>` (this would result in no output being available to StateDMI to evaluate), and `>` is only recognized if running with a command shell in any case.

The following approach is suitable, for example, when running a compiled model or data analysis tool. However, if the tool is run using a script or batch file, then a command shell must be used.

**Edit RunProgram() command**

This command runs another program, and TSTool waits for it to complete before continuing.  
 Commands must use a full path to files, TSTool must be started from the directory where files exist to use relative paths, or use `${WorkingDir}` in the command line to specify files relative to the working directory.  
 Use `'\'` to indicate double quotes if needed to surround program name or program command-line parameters - this may be needed if there are spaces in paths.  
 Specify the exit status indicator if program output messages must be used to determine the program exit status (e.g., "Status:").  
 Specify the program to run using the command line OR separate arguments - the latter makes it simpler to know how to treat whitespace in command line arguments.  
 The program by default will be run with a command shell (e.g., `cmd.exe` on Windows) - specify as `False` if it is known that the program is an executable (not a shell command or script).

Command to run (with arguments):

Program to run: `${WorkingDir}/testecho.exe` Required - if full command line is not specified above.

Program argument 1: `STOP 2` Optional - as needed if Program is specified.

Program argument 2: Optional - as needed if Program is specified.

Program argument 3: Optional - as needed if Program is specified.

Program argument 4: Optional - as needed if Program is specified.

Program argument 5: Optional - as needed if Program is specified.

Program argument 6: Optional - as needed if Program is specified.

Program argument 7: Optional - as needed if Program is specified.

Program argument 8: Optional - as needed if Program is specified.

Use command shell: ☐ Optional - use command shell (default=True).

Timeout (seconds): Optional - default is no timeout.

Exit status indicator: `STOP` Optional - output string to indicate status (default=use process exit status).

Command:

```
RunProgram(Program="${WorkingDir}/testecho.exe", ProgramArg1="STOP
2", ExitStatusIndicator="STOP")
```

Cancel OK

RunProgram\_Program\_ExitStatusIndicator

**RunProgram() Command Editor when Specifying Program, Arguments, and Exit Status Indicator**

---

# Command Reference: RunPython()

## Run a Python script

### General Command

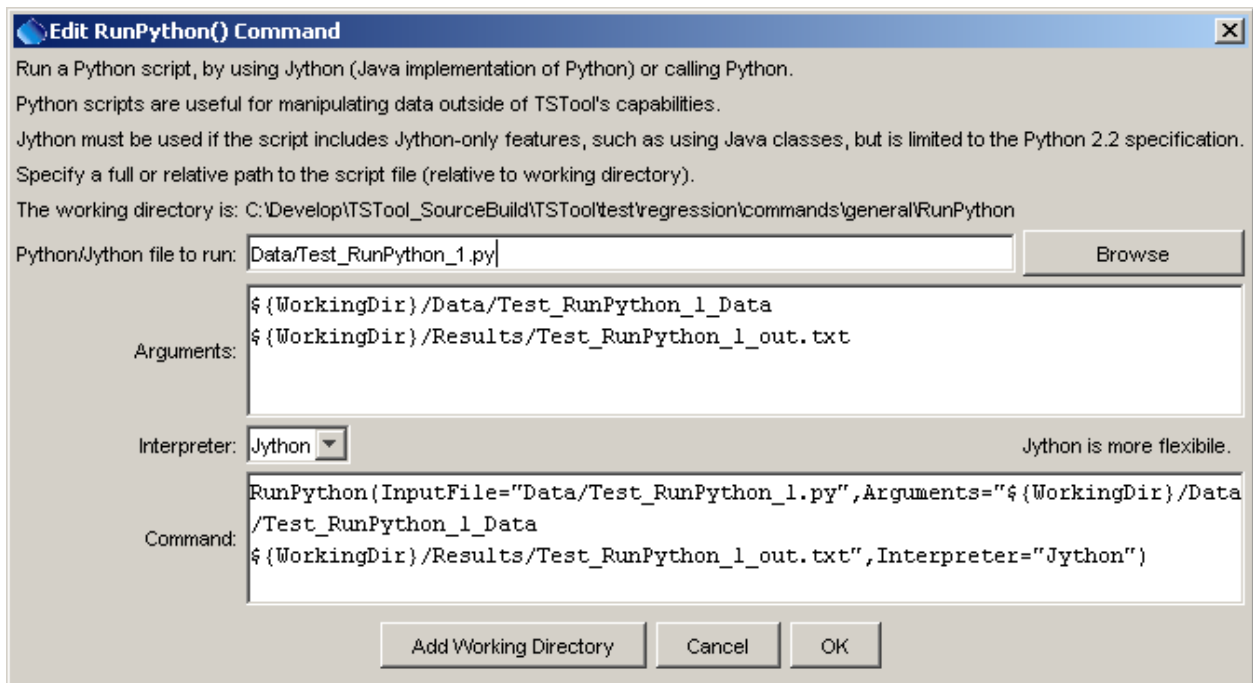
Version 3.08.02, 2010-01-07

**This command is under development. Initial development is occurring in the TSTool software.** The RunPython( ) command runs a Python or Jython script, waiting until the script is finished before processing additional commands. Python is a powerful scripting language that is widely used (see <http://www.python.org>). This command allows either Python or Jython to be used as the interpreter. If Python is used, then Python must be installed on the computer and be in the PATH environment variable. The script is then run by running:

#### python Arguments

Jython (see <http://www.jython.org>) is a Java implementation of Python, allowing most Python scripts to be run and also allowing integration with Java software, including using Java classes in scripts. This allows existing Java components to be used and supports more robust integration and error handling. If the Jython interpreter is used, no additional software needs to be installed. Additional features to utilize Jython will be provided in future software releases (e.g., directly passing time series objects to/from python scripts). Currently Jython only supports the Python 2.2 specification; however, future releases are expected to support later specifications.

The following dialog is used to edit the command and illustrates the command syntax.



RunPython

**RunPython() Command Editor**

The command syntax is as follows:

```
RunPython(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The Python/Jython script to run, specified as an absolute path or relative to the command file.	None – must be specified.
Arguments	Arguments to pass to the script, such as the names of files to process. Use the \${WorkingDir} property to specify the location of the command file. Separate arguments by a space.	None – arguments are optional.
Interpreter	The Python interpreter to run, either Python or Jython.	Jython

The following command example illustrates how to run a Python script.

```
RunPython(InputFile="Data/Test_RunPython_1.py",
Interpreter="Jython",Arguments="${WorkingDir}/Data/Test_RunPython_1_Data
${WorkingDir}/Results/Test_RunPython_1_out.txt")
```

The corresponding Python script is as follows:

```
#
# Test command for running Python script from TSTool
#
import sys
import os
print "start of script"
print 'os.getcwd()="' + os.getcwd() + '"'
infile = None
outfile = None
if ( len(sys.argv) < 3 ):
    print "Error. Expecting input file name as first command line argument,
output file name as second."
    sys.exit(1)
else:
    infile = sys.argv[1]
    outfile = sys.argv[2]
    print 'Input file to process is "' + infile + '"'
    print 'Output file to create is "' + outfile + '"'

inf=open(infile,'r')
outf=open(outfile,'w')
for line in inf:
    outf.write("out: " + line)
inf.close()
outf.close()
print "end of script"
```

The data file is as follows:

```
Line 1 (first line)
Line 2
Line 3
Line 4
Line 5 (last line)
```

The output file is as follows:

```
out: Line 1 (first line)
out: Line 2
out: Line 3
out: Line 4
out: Line 5 (last line)
```

This page is intentionally blank.



# Command Reference: SetBlaneyCriddle()

## Set Blaney-Criddle crop coefficients data

### StateCU Command

Version 3.08.02, 2010-01-07

The `SetBlaneyCriddle()` command sets data in existing Blaney-Criddle crop coefficients or adds a new crop type with crop coefficients. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetBlaneyCriddle() Command**

This command edits Blaney-Criddle crop coefficient data for a specified crop type (name).  
If the crop coefficients are based on the day of year, specify 25 values, for the month boundaries and midpoints.  
If the crop coefficients are based on the percent of growing season, specify 21 values, for 0, 5, ..., 100%.  
If the crop type does not contain a \* wildcard pattern and does not match existing data,  
a new curve will be added if the "If not found" option is set to Add.  
Single values in a curve cannot be set - the entire curve must be set.

Crop Type:	<input type="text" value="ALFALFA"/>	Required - crop type (use * for wildcard).
Curve Type:	<input type="text" value="Day - Perennial Crop"/>	Required - see comments above.
Blaney-Criddle Method:	<input type="text" value="0 - SCS Modified Blaney-Criddle"/>	Required - Blaney-Criddle method to use.
Coefficients:	<input type="text" value=".11, .12, .13, .14, .15, .16, .17, .18, .19, .20, .21, .22, .23, .22, .21, .20, .19, .18, .17, .16, .15, .14, .13, .12, .11"/>	Required - separate by commas.
If not found:	<input type="text" value="Add"/>	Optional - action if no match is found (default=Warn).
Command:	<pre>SetBlaneyCriddle(CropType="ALFALFA", CurveType=Day, BlaneyCriddleMethod=0, Coefficients=".11, .12, .13, .14, .15, .16, .17, .18, .19, .20, .21, .22, .23, .22, .21, .20, .19, .18, .17, .16, .15, .14, .13, .12, .11", IfNotFound=Add)</pre>	

SetBlaneyCriddle

### SetBlaneyCriddle() Command Editor

The command syntax is as follows:

```
SetBlaneyCriddle(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
CropType	A crop type to match or a pattern using wildcards (e.g., ALFALFA*).	None – must be specified.
CurveType	Specify Percent for an annual crop or Day for a perennial crop.	If not specified, the original value will remain.
Coefficients	A list of coefficients, surrounded by double quotes.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the crop if not found using the provided information</li> <li>• Fail – generate a failure message if the crop is not found</li> <li>• Ignore – ignore (don't add and don't generate a message) if the crop is not found</li> <li>• Warn – generate a warning message if the crop is not found</li> </ul>	Warn

---

# Command Reference: SetClimateStation()

## Set climate station data

### StateCU Command

Version 03.08.02, 2010-01-05

The `SetClimateStation()` command sets data in existing climate stations or adds a new climate station. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetClimateStation() Command**

This command sets (edits) data in climate station(s), using the climate station ID to look up the station. The climate station ID can contain a \* wildcard pattern to match one or more stations. If the climate station ID does not contain a \* wildcard pattern and does not match an ID, the location will be added if the "IfNotFound" parameter is set to Add. Use blanks in the any field to indicate no change to the existing value.

Climate station ID:  Required - climate station(s) to fill (use \* for wildcard).

Latitude:  Optional - decimal degrees.

Elevation:  Optional - feet.

Region 1:  Optional - primary region for the climate station (typically county).

Region 2:  Optional - secondary region for the climate station (traditionally HUC or blank).

Name:  Optional - up to 28 characters for StateCU.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetClimateStation(ID="1928", Elevation=6440, IfNotFound=Warn)
```

OK Cancel

SetClimateStation

### SetClimateStation() Command Editor

The command syntax is as follows:

```
SetClimateStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single climate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Latitude	The climate station latitude to be assigned for all matching climate stations.	If not specified, the original value will remain.
Elevation	The climate station elevation to be assigned for all matching climate stations.	If not specified, the original value will remain.
Region1	The climate station Region1 (typically county) to be assigned for all matching climate stations.	If not specified, the original value will remain.
Region2	The climate station Region2 (typically the HUC basin) to be assigned for all matching climate stations.	If not specified, the original value will remain.
Name	The climate station name to be assigned for all matching climate stations.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the climate station if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example command file illustrates how climate stations can be defined and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
FillClimateStationsFromHydroBase(ID="*")
SetClimateStation(ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation(ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation(ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation(ID="0484",Region1="MOFFAT",IfNotFound=Add)
WriteClimateStationsToStateCU(OutputFile="COclim2006.cli")
```

# Command Reference: SetCropCharacteristics()

## Set crop characteristics data

**StateCU Command**  
Version 3.08.02, 2010-01-07

The `SetCropCharacteristics()` command sets data in existing crop characteristics or adds a new crop type with crop characteristics. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetCropCharacteristics() Command

This command sets (edits) crop characteristics data, using the crop type (name) to look up the crop. The crop name can contain a \* wildcard pattern to match one or more crops. If the crop name does not contain a \* wildcard pattern and does not match a crop name, the crop will be added if the "If not found" option is set to Add.

Crop type:	ALFALFA	Required - crop type to set (use * for wildcard)	
Planting month:	3	Day: 1	Month: 1-12, Day: 1-31.
Harvest month:	6	Day: 1	Month: 1-12, Day: 1-31.
Days to full cover:	60	Days to full cover	
Length of season:	120	Days from planting.	
Earliest moisture use temperature:	41	Degrees F	
Latest moisture use temperature:	42	Degrees F	
Maximum root zone depth:	2.5	Feet.	
Maximum application depth:	6	Inches.	
Spring frost flag:	0 - Mean Temperature	Indicate how frost is handled.	
Fall frost flag:	0 - Mean Temperature	Indicate how frost is handled.	
Days to 2nd Cut:	32	Alfalfa only - days between 1st and 2nd cuts.	
Days to 3rd Cut:	33	Alfalfa only - days between 2nd and 3rd cuts.	
If not found:	Add	Optional - action if no match is found (default=Warn).	
Command:	<pre>SetCropCharacteristics (CropType="ALFALFA", PlantingMonth=3, PlantingDay=1, HarvestMonth=6, HarvestDay=1, DaysToFullCover=60, LengthOfSeason=120, LatestMoistureUseTemp=42, EarliestMoistureUseTemp=41, MaxRootZoneDepth=2.5, MaxAppDepth=6, SpringFrostFlag=0, FallFrostFlag=0, DaysTo2ndCut=32, DaysTo3rdCut=33, IfNotFound=Add)</pre>		

OK

Cancel

SetCropCharacteristics

**SetCropCharacteristics() Command Editor**

The command syntax is as follows:

```
SetCropCharacteristics(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
CropType	A crop type to match or a pattern using wildcards (e.g., ALFALFA*).	None – must be specified.
PlantingMonth	The planting month for the crop, as an integer (1=January).	If not specified, the original value will remain.
PlantingDay	The planting day of month for the crop, for the planning month.	If not specified, the original value will remain.
HarvestMonth	The harvest month for the crop, as an integer (1=January).	If not specified, the original value will remain.
HarvestDay	The harvest day of month for the crop, for the planning month.	If not specified, the original value will remain.
DaysToFullCover	Days to full cover.	If not specified, the original value will remain.
LengthOfSeason	Length of growing season, days.	If not specified, the original value will remain.
EarliestMoistureUseTemp	Earliest moisture use temperature, F.	If not specified, the original value will remain.
LatestMoistureUseTemp	Latest moisture use temperature, F.	If not specified, the original value will remain.
MaxRootZoneDepth	Maximum root zone depth.	If not specified, the original value will remain.
MaxAppDepth	Maximum application depth.	If not specified, the original value will remain.
SpringFrostFlag	Spring frost flag.	0 (mean)
FallFrostFlag	Fall frost flag.	0 (mean)
DaysTo2ndCut	Days between first and second cuts (alfalfa).	If not specified, the original value will remain.
DaysTo3rdCut	Days between second and third cuts (alfalfa).	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the climate station if not found using the provided information</li> <li>• Fail – generate a failure message if the climate station is not found</li> <li>• Ignore – ignore (don't add and don't generate a message) if the climate station is not found</li> <li>• Warn – generate a warning message if the climate station is not found</li> </ul>	Warn

---

# Command Reference: SetCropPatternTS()

**Set crop pattern time series values**

**StateCU Command**

Version 3.09.01, 2010-02-01

The `SetCropPatternTS()` command sets crop pattern time series data for a CU Location. The combination of location ID, crop type, and year identify the data. It is recommended that the `SetCropPatternTSFromList()` command be used instead to shorten commands files and allow sharing of the data with `SetIrrigationPracticeTSFromList()` commands. There are two uses for this command:

1. Specify crop data for a location, to be processed with parcel data. For example, an irrigated lands assessment using GIS might show zero acreage for a ditch but other information indicates that the ditch irrigates lands. The ditch may be an individual (key) structure or may be part of an aggregate/system. In this case, the specified data values contribute to the final data values in output. The following dialog is used to edit the command and illustrates the syntax of the command. The years typically agree with an irrigated lands assessment and the **Process when** value must be specified as **WithParcels**. In this case, the `SetCropPatternTS()` commands should be specified before `ReadCropPatternTSFromHydroBase()` or other similar commands. The data will be processed as if they were read from `HydroBase`.

Edit SetCropPatternTS() Command

This command edits crop pattern time series data, using the CU Location ID to look up the location.

Crop patterns should be specified using the format:

Crop1,Area1,Crop2,Area2

For example:

ALFALFA,300,POTATOES,150

If ProcessWhen=Now, previous crop patterns for matching CU locations (created with `CreateCropPatternTSForCULocations()`) are reset when the command is processed. Acreage for other crops at the location and date (e.g., from other commands) will be set to zero. Therefore this command completely defines the crop pattern for a location at a point in time. Used in this way, the command usually comes AFTER commands that process crop patterns from parcels, and changes are made to final CU Locations (not parts in an aggregate/system).

If ProcessWhen=WithParcels, the crop patterns are processed with `HydroBase` irrigated parcels data. Each crop/area/year triplet is treated as if it were determined from parcels, to supplement later processing. Used in this way, the commands should come BEFORE commands that process crop patterns from parcels, and data can be defined for parts of an aggregate/system.

CU location ID:

Set start (year):

Set end (year):

Crop pattern:

Irrigation method:

Supply type:

Set to missing:

Process when:

If not found:

Required - the CU location(s) to fill (use \* for wildcard)

Optional - starting year to set data (blank for full period).

Optional - ending year to set data (blank for full period).

Required - irrigation method for crops.

Required - supply type for crops.

Optional - set data to missing (no crops) - can then be filled.

Optional - indicate when to process the data (default=Now).

Optional - indicate action if no match is found (default=Warn).

Command:

```
SetCropPatternTS (ID="200506", SetStart=1998, SetEnd=1998, CropPattern="GRASS_PASTURE, 100", IrrigationMethod=Flood, SupplyType=Surface, ProcessWhen=WithParcels)
```

OK
Cancel

SetCropPatternTS\_WithParcels

### SetCropPatternTS() Command Editor (to specify parcel information)



- Specify crop data to override (or supply) crop pattern data for a structure. In this case, the specified data will be visible as the final data values in output and will not be considered when irrigated lands parcels are processed. The **Process when** flag should be blank or Now. In this case, the SetCropPatternTS( ) commands should be specified after ReadCropPatternTSFromHydroBase( ) or other similar commands. It is recommended that the previous alternative be used, in particular when multiple years of data are being processed and need to be quality controlled.

Edit SetCropPatternTSFromList() Command

This command sets crop pattern time series data, using the CU Location ID to look up the location.

If ProcessWhen=Now, supplied data will be applied when the command is processed.

If ProcessWhen=WithParcels, data will supplement HydroBase data when readCropPatternTSFromHydroBase() is processed.

A comma-delimited list file is used to supply data, with values being set one of the following ways.

If the set start and end years are specified and a year column is not specified, the file values are applied to each year in the set period.

If a year column is specified, year and corresponding values are read from the list file. The set period limits the years that are processed.

The crop pattern data will be reset to new values (or data will supplement HydroBase data, as per ProcessWhen). All crops not set will be set to zero.

Blanks in column fields will result in no change to the data.

It is recommended that the location of the file be specified using a path relative to the working directory.

The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCropPatternTSFromHydroBase

List file: NoGIS\_1936.csv
Browse

CU Location ID: \*
Required - CU Location(s) to fill (use \* for wildcard).

Set start (year): 1936
Optional - starting year to set data (default=output period).

Set end (year): 1936
Optional - ending year to set data (default=output period).

CU location ID column: 1
Required - column in file for CU location ID.

Year column:
Optional - column in file for year.

Crop type column: 2
Optional - column in file for crop type.

Area (ACRE): 3
Optional - column in file for crop area.

Irrigation method column: 4
Optional - column in file for irrigation method (column containing SPRINKLER, FLOOD).

Supply type column: 5
Optional - column in file for supply type (column containing Ground or Surface).

Process when?: WithParcels
Optional - when to process the data (default=Now).

Command:

```
SetCropPatternTSFromList (ListFile="NoGIS_1936.csv", ID="*", SetStart=1936, SetEnd=1936, IDCol="1", CropTypeCol="2", AreaCol="3", ProcessWhen=WithParcels, IrrigationMethodCol="4", SupplyTypeCol="5")
```

Add Working Directory
Cancel
OK

SetCropPatternTS

### SetCropPatternTS() Command Editor (to edit crop pattern time series)

The command syntax is as follows:

```
SetCropPatternTS( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
SetStart	The first year to set data values.	If not specified, data are set for the full output period.
SetEnd	The last year to set data values.	If not specified, data are set for the full output period.
CropPattern	A sequence of crop type and area values, to set as data for the specified period.	None – must be specified.
SetStart	Starting year to set data.	Set for the full period.
SetEnd	Ending year to set data.	Set for the full period.
SetToMissing	Indicate whether the crop pattern for the specified years should be set to missing, instead of supplying data values. This was used in the Río Grande as follows: Read 1936, 1998, and 2002 data, resulting in crop pattern time series. It is necessary to include all years in order to get a complete list of crops over the period, even if zero or missing in some years. After reading all years, 2002 is set to missing using this command and a standard filling approach is used for the full period. Then, 2002 is read at the end. The overall result is that 2002's crops are listed in the full period but only have non-zero observations in 2002.	False
ProcessWhen	Indicates when the specified data values should be processed. If the parameter value is WithParcels, then the values will be considered when irrigated lands data are processed with later ReadCropPatternTSFromHydroBase( ).	Now, indicating that the acreage should be set when the command is processed (not when later read commands are processed).
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: SetCropPatternTSFromList()

## Set crop pattern time series data from information in a delimited file

### StateCU Command

Version 3.09.02, 2010-03-12

The `SetCropPatternTSFromList()` command sets crop pattern data for existing CU Locations by reading information from a delimited file. New locations are not added. The data are set to zero for a year being reset, and then values are applied for the year (acreage does not add to previous values). The command can be used to set values over a period of 1+ years as follows:

1. If the `SetStart` and `SetEnd` parameters are specified and the year column is not specified, then repeat the values from the file for each year in the set period. For example, this can be used to provide acreage data not in HydroBase, for a specific year (set `SetStart` and `SetEnd` to the same value).
2. If the year column is provided, use the year in the file to specify the year for the set. In this case, `SetStart` and `SetEnd` control the period of data that will be processed from the file.

The command provides irrigated parcel acreage for two data processing situations:

1. **Supplement HydroBase Acreage.** If the `ProcessWhen=WithParcels`, the supplied data will be processed when parcel data are read from HydroBase with `ReadCropPatternTSFromHydroBase()`. HydroBase may not contain all irrigated lands data. For example, additional lands may have been identified after HydroBase was populated or acreage must be set for a model identifier that is not a structure WDID in HydroBase (e.g., out of state lands). In this case, the command can be used to provide additional data to supplement HydroBase. For example, this may be appropriate for diversion aggregates where part of the data are in HydroBase for a year, but some parts of the aggregate require data to be provided by this command. Therefore, it is important that the data are processed when reading from HydroBase (otherwise a reset of HydroBase data might occur). Specifying the supply type allows the `FillCropPatternTSUsingWellRights()` command to determine that a parcel has groundwater only supply.
2. **Provide Acreage Independent of HydroBase.** If `ProcessWhen=Now`, the provided information will be applied as the command is processed. This may be appropriate to explicitly set data values, in the following cases:
  - a. Set acreage before processing HydroBase data are processed (where no data exist in HydroBase).
  - b. Override values after HydroBase data have been processed (where HydroBase data are inappropriate).
  - c. Supply acreage values independent of HydroBase.

For clarity in data management, it may be appropriate to use separate `SetCropPatternTSFromList()` commands for each year of data. However, the command does allow multiple years of data to be included in a single list file.

The following dialog is used to edit the command and illustrates the syntax of the command for providing acreage data that are not in HydroBase. The data will be processed when HydroBase data are read. The file that is used is the same one used with the `SetIrrigationPracticeTSFromList()` and the crop type and area should be specified (irrigation method and supply type are shown for consistency with irrigation practice time series processing).

**Edit SetCropPatternTSFromList() Command**

This command sets crop pattern time series data, using the CU Location ID to look up the location.  
 If ProcessWhen=Now, supplied data will be applied when the command is processed.  
 The crop pattern data will be reset to new values. All crops not set will be set to zero.  
 If ProcessWhen=WithParcels, data will supplement HydroBase data when ReadCropPatternTSFromHydroBase() is processed.  
 A comma-delimited list file is used to supply data, with values being set one of the following ways:  
 1) If the set start and end years are specified and a year column is not specified, the file values are applied to each year in the set period.  
 2) If a year column is specified, year and corresponding values are read from the list file. The set period limits the years that are processed.  
 It is recommended that the location of the file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetCropPatternTSFromList

List file:

CU Location ID:  Required - CU Location(s) to set (use \* for wildcard).

Set start (year):  Optional - starting year to set data (default=output period).

Set end (year):  Optional - ending year to set data (default=output period).

CU location ID column:  Required - column in file for CU location ID.

Year column:  Optional - column in file for year.

Crop type column:  Optional - column in file for crop type.

Area (ACRE):  Optional - column in file for crop area.

Irrigation method column:  Optional - column in file for irrigation method (column containing SPRINKLER, FLOOD).

Supply type column:  Optional - column in file for supply type (column containing Ground or Surface).

Process when?:  Optional - when to process the data (default=Now).

Command: 

```
SetCropPatternTSFromList (ListFile="NoGIS_1936.csv", ID="*", SetStart=1936, SetEnd=1936, IDCol="1", CropTypeCol="2", AreaCol="3", ProcessWhen=WithParcels, IrrigationMethodCol="4", SupplyTypeCol="5")
```

SetCropPatternTSFromList

### SetCropPatternTSFromList() Command Editor – Provide Parcel Data not in HydroBase

The command syntax is as follows:

```
SetCropPatternTSFromList (Parameter=Value,...)
```

**Command Parameters**

Parameter	Description	Default
ListFile	Path to the delimited list file to read.	None – must be specified.
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
SetStart	The first year to set data values.	If not specified, data are set for the full output period.
SetEnd	The last year to set data values.	If not specified, data are set for the full output period.
IDCol	The column number (1+) containing the CU Location identifiers. These values are matched against CU Location identifiers in the existing irrigation practice data.	None – must be specified.
YearCol	The column number (1+) containing the year for data.	The file values are applied to each year in the data set.
CropTypeCol	The column number (1+) containing the crop type.	If not specified, the previous data values will remain.
AreaCol	The column number (1+) containing the crop area.	If not specified, the previous data values will remain.
Irrigation MethodCol	The column number (1+) containing the irrigation method, consistent with HydroBase (e.g., SPRINKLER, FLOOD).	If not specified, the previous data values will remain.
SupplyTypeCol	The column number (1+) containing the supply type (Surface or Ground).	If not specified, the previous data values will remain.
ProcessWhen	When to process the data, one of: <ul style="list-style-type: none"> <li>Now – set the values in the time series when the command is encountered</li> <li>WithParcels – treat the data as raw parcel data that should be processed when parcels are read from HydroBase (see the ReadCropPatternTSFromHydroBase() command).</li> </ul>	Now

Data file lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings. An example list file for specifying acreage data (not in HydroBase) is shown below. Currently, supplemental acreage data can have only a single irrigation method and supply type, to support irrigation practice time series processing. Therefore, break supplemental acreage into multiple "parcels" if necessary.

```
# The following data provide acreage for structures that did not have GIS data
# and consequently no data in HydroBase. The data are specific to 1998 and are
# used to set the CDS and IPY acres. The crop is used to provide CDS data. The
# irrigation method and source are used to provide IPY data.
"ID","Crop","Acres","IrrigationMethod","SupplySource"
200500,GRASS_PASTURE,0,Flood,Surface
200506,GRASS_PASTURE,100,Flood,Surface
200507,GRASS_PASTURE,50,Flood,Surface
200508,GRASS_PASTURE,40,Flood,Surface
200522,GRASS_PASTURE,40,Flood,Surface
200523,GRASS_PASTURE,50,Flood,Surface
200526,GRASS_PASTURE,40,Flood,Surface
200529,GRASS_PASTURE,5,Flood,Surface
... etc...
```

This page is intentionally blank.

# Command Reference: SetCULocation()

## Set CU Location data

### StateCU Command

Version 3.08.02, 2010-01-02

The `SetCULocation()` command sets data in existing CU Locations or adds a new CU Location. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetCULocation() Command**

This command sets (edits) data in CU Location(s), using the CU Location ID to look up the location.  
The CU Location ID can contain a \* wildcard pattern to match one or more locations.  
If the CU Location ID does not contain a \* wildcard pattern and does not match an ID,  
the location will be added if the "IfNotFound" parameter is set to Add.  
Use blanks in the any field to indicate no change to the existing value.  
See also the `SetCULocationClimateStationWeights()` command.

CU Location ID:	<input type="text" value="Station1"/>	Required - specify the CU Location(s) to fill (use * for wildcard)
Name:	<input type="text" value="Station 1"/>	Optional - up to 28 characters for StateCU.
Latitude:	<input type="text" value="44"/>	Optional - decimal degrees.
Elevation:	<input type="text" value="4001"/>	Optional - feet.
Region 1:	<input type="text" value="ADAMS"/>	Optional - primary region for the CU location (typically county).
Region 2:	<input type="text"/>	Optional - secondary region for the CU location (traditionally HUC or blank).
AWC:	<input type="text" value=".5"/>	Optional - Available Water Content, fraction (0-1).
If not found:	<input type="text" value="Add"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
SetCULocation( ID="Station1",Name="Station 1",Latitude=44,Elevation=4001,Region1="ADAMS",AWC=.5,IfNotFo  
und=Add)
```

SetCULocation

### SetCULocation() Command Editor

The command syntax is as follows:

```
SetCULocation(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Latitude	The latitude to be assigned for all matching CU Locations.	If not specified, the original value will remain.
Elevation	The elevation to be assigned for all matching CU Locations.	If not specified, the original value will remain.
Region1	The Region1 to be assigned for all matching CU Locations.	If not specified, the original value will remain.
Region2	The Region2 to be assigned for all matching CU Locations.	If not specified, the original value will remain.
Name	The name to be assigned for all matching CU Locations.	If not specified, the original value will remain.
AWC	The available water content (AWC) to be assigned for all matching CU Locations.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn



# Command Reference: SetCULocationClimateStationWeights ( )

Set CU Location climate station weights data

**StateCU Command**

Version 3.09.00, 2010-01-24

The SetCULocationClimateStationWeights( ) command sets climate station weights data in existing CU Locations. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetCULocationClimateStationWeights() Command

This command assigns the climate station identifiers and weights associated with a CU location.  
The previous data will be reset to new values.  
Each CU location must be associated with one or more precipitation and temperature stations.  
The data from each station is weighted, and the weights should add to 1.0.  
The climate station weights should be specified using the format (, and ; are equivalent):  
StationID,TempWt,PrecWt;StationID,TempWt,PrecWt,...  
For example:  
2184,.7,.7,3951,.3,.3  
The weights are specified as a fraction 0.0 to 1.0.  
If orographic adjustment factors are included, insert the factors after the weights in the order of temperature adjustment, precipitation adjustment, where the temperature adjustment is degrees F/1000 ft. of elevation and the precipitation adjustment is a fraction (0.0 to 1.0).

CU location ID:  Required - specify the CU Location(s) to process (use \* for wildcard)

Include orographic temperature adjustment?:  Optional - include orographic temperature adjustment factor in data (default=False).

Include orographic precipitation adjustment?:  Optional - include orographic precipitation adjustment factor in data (default=False).

Climate station weights:

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

SetCULocationClimateStationWeights (ID="20\*",Weights="2184,.7,.7,3951,.3,.3")

SetCULocationClimateStationWeights

**SetCULocationClimateStationWeights() Command Editor**

The command syntax is as follows:

```
SetCULocationClimateStationWeights( Parameter=Value, ... )
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Include Orographic TempAdj	If True, include the orographic temperature adjustment factor, after the Weights described below, specified as degrees/1000 feet.	False
Include Orographic PrecAdj	If True, include the orographic precipitation adjustment factor, after the Weights described below, specified as a fraction 0.0 to 1.0. Place after the orographic temperature adjustment factor if it is specified.	False
Weights	A repeating pattern of StationID, TempWt, PrecWt, where the station identifiers match climate station identifiers and the weights are specified as fractions in the range 0.0 to 1.0. Also include the orographic temperature and/or orographic precipitation adjustment factors if the above parameters are True.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

---

# Command Reference: setCULocationClimateStationWeightsFromHydroBase()

**Set CU Location climate station weights data from HydroBase**

## StateCU Command

Version 01.07.00, 2004-03-31, Color, Acrobat Distiller

The setCULocationClimateStationWeightsFromHydroBase() command sets climate station weights data in existing CU Locations, using HydroBase for data. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit setCULocationClimateStationWeightsFromHydroBase() Command**

Each CU Location must be associated with one or more precipitation and temperature stations. The data from each station is weighted, and the weights should add to 1.0. This command reads from HydroBase station weights that are associated with Region1 (county) and Region2 (HUC). The Region1 and Region2 information is then matched with the same information for the CU Locations. The climate station weights are then assigned for each CU location. The climate station weights are usually assigned after other CU Location data have been filled.

Station ID: \* Specify the CU Locations to fill (use \* for wildcard).

Command: setCULocationClimateStationWeightsFromHydroBase(ID="\*")

OK Cancel

setCULocationClimateStationWeightsFromHydroBase

**setCULocationClimateStationWeightsFromHydroBase() Command Editor**

The command syntax is as follows:

```
setCULocationClimateStationWeightsFromHydroBase(param=value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.

# Command Reference: SetCULocationClimateStationWeightsFromList()

## Set CU Location climate station weights from data in a list file

### StateCU and StateMod Command

Version 3.09.00, 2010-01-24

The SetCULocationClimateStationWeightsFromList() command reads climate station weights from a list file and sets the information for CU Locations.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetCULocationClimateStationWeightsFromList() Command**

This command sets CU location climate station weights from data in a delimited list file.  
Each CU Location must be associated with one or more precipitation and temperature stations.  
The data from each station is weighted, and the weights should add to 1.0.  
There are two ways to match CU locations for processing:

1. Specify the CU Location ID column - in this case weights will be set only for CU locations that match the ID in the list file.
2. Specify the Region1 and Region2 columns - in this case weights will be set for CU locations with matching Region1 and Region2 (e.g., County, HUC).

Columns in the file should be delimited by commas.  
The climate station weights are usually assigned after other CU Location data have been filled, in particular if county is filled.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetCULocationClimateStationWeightsFromList

List file:

CU location ID column:  If specified, Region1 and Region2 columns should not be specified.

Climate station ID column:  Required - indicate the climate station column.

Region 1 column:  If specified, the CU Location ID column should not be specified.

Region 2 column:  If specified, the CU Location ID column should not be specified.

Temperature station weight column:  Required - temperature station weight column.

Precipitation station weight column:  Required - precipitation station weight column.

Orographic temperature adjustment column:  Optional - orographic temperature adjustment column.

Orographic precipitation adjustment column:  Optional - orographic precipitation adjustment column.

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command:  
`SetCULocationClimateStationWeightsFromList (ListFile="cowts.csv",StationIDCol=1,Region1Col=2,Region2Col=3,TempWtCol=4,PrecWtCol=5)`

SetCULocationClimateStationWeightsFromList

### SetCULocationClimateStationWeightsFromList() Command Editor

An example list file for setting data using Region1 (county) is shown below:

```
# RGDSS.WTS - Hand built climate weights file for the RGDSS analysis
#           Base on Climate Assignment memo prepared 9/21/99
# ID, Lat, County, HUC, TempWt, PrecWt
3951,37.7667,SAN JUAN,13010001,1.000,1.000
3951,37.7667,HINSDALE,13010001,1.000,1.000
2184,37.6833,RIO GRANDE,13010001,0.700,0.700
...
```

An example list file for setting data by CU location ID is shown below:

```
#Date and Time |Thu May 03 11:31:37 2007
#Input Polygon Theme |2001_Acreage_CW
#Polygon ID Field |PARCEL_ID
#Climate Weights Workspace |S:\CDSS\GIS\Climate_Wts
#Orographic Grids Workspace |#
0200552,2220,0.19,1
0200552,3553,0.39,1
0200552,5116,0.42,1
0200805,1179,0.39,1
0200805,2220,0.1,1
0200805,3553,0.51,1
```

The command syntax is as follows:

```
SetCULocationClimateStationWeightsFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
IDCol	The column number (1+) containing the climate station identifier.	None – must be specified. If specified, Region1Col and Region2Col should not be specified.
Region1Col	The column number (1+) containing the Region1 identifier.	If specified, the ID column should not be specified.
Region2Col	The column number (1+) containing the Region2 identifier.	If specified, the ID column should not be specified.
TempWtCol	The column number (1+) containing the temperature station weights.	If not specified, the original values remain.
PrecWtCol	The column number (1+) containing the precipitation station weights.	If not specified, the original values remain.
Orographic TempAdjCol	The column number (1+) containing the orographic temperature adjustment factor (DEGF/1000 FT).	If not specified, the original values remain.
Orographic PrecAdjCol	The column number (1+) containing the orographic precipitation adjustment factor (fraction).	If not specified, the original values remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID pattern is not matched</li> <li>Ignore – ignore (don't generate a message) if the ID pattern is not matched</li> <li>Warn – generate a warning message if the ID pattern is not matched</li> </ul>	Warn

# Command Reference: SetCULocationsFromList()

Set CU Location data from information in a delimited file

**StateCU Command**

Version 3.08.02, 2010-01-07

The `SetCULocationsFromList()` command sets data in existing CU Locations by reading information from a delimited file. New locations are not added. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetCULocationsFromList() Command**

This command sets (edits) data in CU Location(s), using the CU Location ID to look up the location.  
Data are supplied by values in a comma-delimited file.  
Use blanks in the any field to indicate no change to the existing value.  
Columns should be delimited by commas (user-specified delimiters will be added in the future).  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCULocationsFromList

List file:

CU location ID column:

Latitude Column:

Elevation Column:

Region 1 Column:

Region 2 Column:

Name Column:

AWC Column:

If not found:

Optional - indicate action if no ID match is found (default=Warn).

Command:

SetCULocationsFromList

**SetCULocationsFromList() Command Editor**

The command syntax is as follows:

```
SetCULocationsFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	Path to the delimited list file to read.	None – must be specified.
IDCol	The column number (1+) containing the CU Location identifiers.	None – must be specified.
LatitudeCol	The column number (1+) containing the CU Location latitude.	If not specified, the previous value will remain.
ElevationCol	The column number (1+) containing the CU Location elevation.	If not specified, the previous value will remain.
Region1Col	The column number (1+) containing the CU Location Region1.	If not specified, the previous value will remain.
Region2Col	The column number (1+) containing the CU Location Region2.	If not specified, the previous value will remain.
NameCol	The column number (1+) containing the CU Location name.	If not specified, the previous value will remain.
AWCCol	The column number (1+) containing the CU Location AWC.	If not specified, the previous value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the climate station is not found</li> <li>Ignore – ignore (don't add and don't generate a message) if the climate station is not found</li> <li>Warn – generate a warning message if the climate station is not found</li> </ul>	Warn

Lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings.

An example list file is shown below:

```
72_ADC064,MESA2
72_ADC063,MESA2
72_ARC010,MESA2
950030,MESA2
720766,MESA2
720731,MESA2
720514,MESA2
720933,MESA2
72_ADC062,MESA2
721339,MESA2
720580,MESA2
```



---

# Command Reference: SetDebugLevel()

## Set Level for Debug Messages

### General Command

Version 3.08.02, 2010-01-06

The `SetDebugLevel ( )` command is used to set debug levels for the screen and log file. The following dialog is used to edit this command and illustrates the command syntax.

**Edit SetDebugLevel() command**

Set the level for screen and/or log file debug messages.  
Debug information is useful for troubleshooting. The default debug level is 0.  
Setting the debug level to a higher number prints more information.  
Debug levels can be increased before and decreased after specific commands to troubleshoot the commands.

Screen debug level:  0=none, 100=all, blank=no change.

Log file debug level:  0=none, 100=all, blank=no change.

Command: 

```
SetDebugLevel (ScreenLevel=0,LogFileLevel=30)
```

SetDebugLevel

### SetDebugLevel() Command Editor

Debug messages are useful during troubleshooting. A general guideline is that a debug level of 1 prints basic messages, 30 prints detailed information about processing, and 100 prints very low-level messages about input/output. Intermediate values will result in more or less output.

This command is useful for troubleshooting and can be specified multiple times to increase debug output for a specific command, if necessary.

This page is intentionally blank.

# Command Reference: SetDiversiOnAggregate ( )

## Set diversion aggregate parts

### StateCU and StateMod Command

Version 3.08.02, 2010-01-07

The `SetDiversiOnAggregate ( )` command sets diversion aggregate part identifier data for a diversion (a CU Location that corresponds to a diversion or D&W node or StateMod diversion station). Diversion aggregates are specified using a list of ditch identifiers, and the aggregation information applies for the full model period (does not vary by year). To facilitate processing, it is often best to use list files to specific aggregates (see `SetDiversiOnAggregateFromList ( )`). Aggregates by convention have their water rights grouped into classes – to represent all water rights at a location, use a system (see the similar `System` commands). See also the StateDMI **Introduction** chapter, which provides additional information about aggregates and other modeling conventions. Aggregate information should be specified after diversion locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversiOnAggregate() Command**

This command sets a Diversion Aggregate location's information.  
Each Aggregate is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with `SetDiversiOnAggregate()` when water rights will be aggregated into classes.  
A "System" is used with `SetDiversiOnSystem()` when individual water rights will be maintained.  
For example, multiple nearby or related ditches may be grouped as a single identifier.  
When grouping ditches, specify the diversion station IDs for parts.  
Separate the part IDs by spaces or commas.

Diversion Aggregate ID:  Required - specify the Diversion Aggregate ID.

Part IDs:  Required - up to 12 characters for each ID.

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command:

SetDiversiOnAggregate

### SetDiversiOnAggregate() Command Editor

The command syntax is as follows:

```
SetDiversionAggregate (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The diversion identifier to associate with the collection of individual diversions.	None – must be specified.
PartIDs	The list of part identifiers to comprise the aggregate, for example ditch WDIDs that will be found in HydroBase. The part identifiers are by default of type Ditch.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the identifier is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the identifier is not matched</li><li>• Warn – generate a warning message if the identifier is not matched</li></ul>	Warn

# Command Reference: SetDiversionAggregateFromList()

Set diversion aggregate parts from data in a list file

StateCU and StateMod Command

Version 3.08.02, 2010-01-07

The `SetDiversionAggregateFromList()` command sets diversion aggregate part identifier data for a diversion (a CU Location that corresponds to a diversion or D&W node or StateMod diversion station). Diversion aggregates are specified using a list of ditch identifiers, and the aggregation information applies for the full model period (does not vary by year). To facilitate processing, the list of parts is specified in a delimited list file. Aggregates by convention have their water rights grouped into classes – to represent all water rights at a location, use a system (see the similar `System` commands). See also the **StateDMI Introduction** chapter, which provides additional information about aggregates and other modeling conventions. Aggregate information should be specified after diversion locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionAggregateFromList() Command**

This command sets a Diversion Aggregate location's information from a list file.  
Each Aggregate is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with `SetDiversionAggregateFromList()` when water rights will be aggregated into classes.  
A "System" is used with `SetDiversionSystemFromList()` when individual water rights will be maintained.  
For example, multiple nearby or related ditches may be grouped as a single identifier.  
When grouping ditches, specify the diversion station IDs for the parts in the list file.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetDiversionAggregateFromList

List file:

ID column:  Required - column for the Diversion Aggregate IDs.

Name column:  Optional - column for the Diversion Aggregate name.

Part IDs column:  Required - first/only column for the part IDs.

Parts listed how:  Required - are part IDs listed in row or column?

Part IDs column (max):  Optional - maximum column for part IDs if in row (default is use all).

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command:  
`SetDiversionAggregateFromList (ListFile="cm_agg.csv", IDCol=1, NameCol=2, PartIDCol=3, PartsListedHow=InRow)`

SetDiversionAggregateFromList

SetDiversionAggregateFromList() Command Editor

The command syntax is as follows:

```
SetDiversionAggregateFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
IDCol	The column number (1+) containing the aggregate diversion identifiers.	None – must be specified.
NameCol	The column number (1+) containing the aggregate diversion name.	None – optional (name will remain as previously defined).
PartIDsCol	The column number (1+) for the first column having part identifiers. The identifiers are ditch WDIDs that will be found in HydroBase. The part identifiers are by default of type Ditch.	None – must be specified.
PartsListedHow	If InRow, it is expected that all parts defining an aggregate are listed in the same row (as shown in the example below). If InColumn, it is expected that the parts defining an aggregate are listed one per row, with multiple rows defining the full aggregate (PartIDsColMax is ignored in this case).	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers. Use if extra columns on the right need to be excluded from the list.	Use all available non-blank columns starting with PartIDsCol.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the aggregate identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the aggregate identifier is not matched</li> <li>Warn – generate a warning message if the aggregate identifier is not matched</li> </ul>	Warn

An example list file is shown below:

```
51_ADC001,Colorado River nr Granby,510580,510663,510703,510704,510707,510833,510841,510974,511032,511033,511048
51_ADC002,Willow Creek,510742,510818,510819,510847,510920,510930,510962
51_ADC003,Ranch Creek,510513,510568,510606,510681,510708,510727,510767
...
```

The following command file illustrates how diversion aggregates are defined with this command and how the aggregate classes are specified when reading diversion rights from HydroBase:

```
# ddr.commands.StateDMI
#
# StateDMI command file to create the direct diversion rights file for the Colorado
model
#
# Step 1 - read structures from preliminary direct diversion station file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 2 - read aggregate and diversion system structure assignments. Note that
#          want to combine water rights for aggs and diversion systems, but
#          water rights are assigned to primary and secondary components of
multistuctures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
#
# Step 3 - read diversion rights from HydroBase and define water rights classes
#          used for aggregate structures - but NOT for diversion systems
#
ReadDiversionRightsFromHydroBase(ID="*",OnOffDefault=1,
    AdminNumClasses="14854.00000,20427.18999,22729.21241,30895.21241,31258.00000,
    32023.28989,39095.38998,43621.42906,46674.00000,48966.00000,99999.")
#
# Step 4 - set water rights for structure IDs different from or not included in
HydroBase
#
# Grand Valley Area - many rights obtain water through operations
SetDiversionRight(ID="720646.02",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.03",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.05",Name="USA Power
Plant",StationID="ID",Decree=800.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.07",Name="Grand Valley
Proj",StationID="ID",AdministrationNumber=22729.19544,Decree=40.0,OnOff=1,
    IfNotFound=Add,IfFound=Set)
... commands omitted
#
# Step 7 - create direct diverison rights file
#
WriteDiversionRightsToStateMod(OutputFile="cm2005.ddr")
```

This page is intentionally blank.



# Command Reference: SetDiversionDemandTSMonthly()

Set diversion demand time series (monthly) data

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetDiversionDemandTSMonthly()` command sets the diversion demand time series (monthly) for a specific diversion station, by reading another time series from a file or HydroBase. If data already exist, the previous time series is discarded. The period of the time series is set to the output period. This command is useful if data cannot be calculated in an automated fashion (e.g., municipal demands may need to be specified manually). The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionDemandTSMonthly() Command**

This command sets a diversion demand time series (monthly) by reading the data from a file or HydroBase. The diversion station identifier is used to match the time series that is read. Time series identifiers follow the conventions used by TSTool and other CDSS software. For example, for a StateMod file:  
ID..DemTotal.Month~StateMod~..\path\to\file  
For example, for a DateValue file:  
ID..DemTotal.Month~DateValue~..\path\to\file  
For example, for HydroBase (use TSTool to determine the identifier):  
ID.DWR.DivTotal.Month~HydroBase  
It is recommended files be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversionDemandTSMonthlyToStateMod  
If the period that is read is shorter than the output period, it is extended to the output period with missing data.

Diversion station ID:  Required - stations to process.

Time series ID:

<= zero values in average?: ☐ Optional - are values <= zero used in averages? (default=True; used later in filling)

If not found: ☐ Optional - indicate action if no match is found (default=Warn).

Command: 

```
SetDiversionDemandTSMonthly ( ID="950020", TSID="950020..DivTotal.Month~StateMod~950020.stm" )
```

SetDiversionDemandTSMonthly

**SetDiversionDemandTSMonthly() Command Editor**

The command syntax is as follows:

```
SetDiversionDemandTSMonthly( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
TSID	The full time series identifier, which is used to locate and read the time series. Currently time series from StateMod and DateValue files are recognized. See the TSTool input type appendices for the formats of these files. Other input types can be enabled if necessary.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing historical averages.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: SetDiversionDemandTSMonthlyConstant()

**Set diversion demand time series (monthly) data to a constant value**

## StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversionDemandTSMonthlyConstant()` command sets diversion demand time series (monthly) data to a constant value. The output period can be set or will default to that defined by the most recent `SetOutputPeriod()` command. If a matching time series is not found, it can be added to the list of time series (at the end). The values that are set are treated the same as observations from HydroBase. To ensure that set values remain, use the `SetDiversionDemandTSMonthlyConstant()` command after other commands that may modify the time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionDemandTSMonthlyConstant() Command**

This command sets monthly diversion demand time series data to a constant.  
The original data limits are recomputed as if the data are historical data. The time series will be created if it does not exist and IfNotFound=Add.

Diversion station ID:  Required - identifier of station to process.

Constant:  Required - constant value to use.

Set start:  Optional - start date (default=output period).

Set end:  Optional - end date (default=output period).

Recalculate limits:  Recalculate original data limits after set (default=True).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

SetDiversionDemandTSMonthlyConstant

## SetDiversionDemandTSMonthlyConstant() Command Editor

The command syntax is as follows:

```
SetDiversionDemandTSMonthlyConstant (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Constant	A constant diversion demand value.	None – must be specified.
SetStart	The start of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period start.
SetEnd	The end of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period end.
RecalcLimits	If True, then the constant values will be treated as observations and the historical averages will be recalculated with the values. False will result in the time series being set but the previous averages remaining. The averages are used with fill commands.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## SetDiversionHistoricalTSMonthly()

**Set diversion historical time series (monthly) data by reading another time series**

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversionHistoricalTSMonthly()` command sets the diversion historical time series (monthly) for a specific diversion, by reading another time series from HydroBase, or a StateMod or DateValue file. This command is useful if data do not exist in the HydroBase database or are saved using a different identifier (e.g., diversion records for transbasin structures may be saved as a gaged streamflow time series). If data already exist, the previous time series is discarded. If a time series is not found, a new time series can be added at the end of the time series list (use the `SortDiversionHistoricalTSMonthly()` command if necessary before writing). The period of the time series that is read is the output period from the `SetOutputPeriod()` command. The time series are treated the same as those read from HydroBase with the `ReadDiversionHistoricalTSMonthlyFromHydroBase()` command. For example, the `LimitDiversionHistoricalTSMonthlyToRights()` command will not modify the observations in the time series. If necessary, to ensure that set values remain for output, use the `SetDiversionHistoricalTSMonthly()` command after other commands that may modify the time series.

If time series are read from HydroBase, it is useful to use TSTool to first verify the time series identifier. For example, for the Streamflow data type, the data source may be USGS, DWR, or other. Diversion comments will be applied by default if available, resulting in additional zero values for diversions. Non-fatal warnings will be generated in the log file for HydroBase time series that do not have diversion comments (e.g., streamflow time series). Warnings are generated because it can be difficult to differentiate a stream gate identifier from a diversion WDID.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionHistoricalTSMonthly() Command**

This command sets a diversion historical time series (monthly) by reading the data from a file or HydroBase.  
 The diversion station identifier is used to match the time series that is read.  
 Time series identifiers follow the conventions used by TSTool and other CDSS software.  
 For example, for a StateMod file:  
 ID..DivTotal.Month~StateMod~..\path\to\file  
 For example, for a DateValue file:  
 ID..DivTotal.Month~DateValue~..\path\to\file  
 For example, for HydroBase (use TSTool to determine the identifier):  
 ID.DWR.DivTotal.Month~HydroBase

It is recommended files be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversionHistoricalTSMonthlyFromHydroBase  
 If the period that is read is shorter than the output period, it is extended to the output period with missing data.

Diversion station ID:  Required - stations to process.

Time series ID:

<= zero values in average?:  Optional - are values <= zero used in averages? (default=True; used later in filling)

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

### SetDiversionsHistoricalTSMonthly() Command Editor

The command syntax is as follows:

```
SetDiversionHistoricalTSMonthly(Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
TSID	The full time series identifier, which is used to locate and read the time series. Currently time series from the following: HydroBase, StateMod file, DateValue file. See the TSTool input type appendices for the formats of these files. Other input types can be enabled if necessary.	None – must be specified.
LEZeroIn Average	Indicates whether values $\leq 0$ should be considered when computing historical averages. These averages are used later with the <code>FillDiversionHistoricalTSMonthlyAverage()</code> and <code>FillDiversionHistoricalTSMonthlyPattern()</code> commands.	True
IfNot Found	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the time series if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## SetDiversiionHistoricalTSMonthlyConstant()

**Set diversion historical time series (monthly) data to a constant value**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetDiversiionHistoricalTSMonthlyConstant()` command sets diversion historical time series (monthly) data to a constant value. The output period can be set or will default to that defined by the most recent `SetOutputPeriod()` command. If a matching time series is not found, it can be added to the list of time series (at the end). The values that are set are treated the same as observations from HydroBase. To ensure that set values remain, use the `SetDiversiionHistoricalTSMonthlyConstant()` command after other commands that may modify the time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversiionHistoricalTSMonthlyConstant() Command**

This command sets monthly diversion historical time series data to a constant.  
The original data limits are recomputed as if the data are historical data. The time series will be created if it does not exist and IfNotFound=Add.

Diversion station ID:  Required - identifier of station to process.

Constant:  Required - constant value to use.

Set start:  Optional - start date (default=output period).

Set end:  Optional - end date (default=output period).

Recalculate limits:  Recalculate original data limits after set (default=True).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetDiversiionHistoricalTSMonthlyConstant (ID="374648")
```

SetDiversiionHistoricalTSMonthlyConstant

**SetDiversiionHistoricalTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
SetDiversionHistoricalTSMonthlyConstant(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Constant	A constant historical diversion value.	None – must be specified.
SetStart	The start of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period start.
SetEnd	The end of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period end.
RecalcLimits	If True, then the constant values will be treated as observations and the historical averages will be recalculated with the values. False will result in the time series being set but the previous averages remaining. The averages are used with fill commands.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



# Command Reference: SetDiversiionMultiStruct()

## Set diversion MultiStruct parts

### StateCU and StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversiionMultiStruct()` command sets diversion MultiStruct part identifier data for a diversion (a CU Location or StateMod diversion station). A diversion MultiStruct indicates that multiple diversion stations take water from more than one tributary but irrigate the same lands. Each diversion station is included in the model network and retains its normal capacity and historical diversions; however, average efficiencies are calculated using the combined demand and historical diversion time series. The demands for the primary structure (the first listed) are set to the total demands, with demands for secondary stations being set to zero. Diversion MultiStruct definition commands are required only when processing the demand time series. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetDiversiionMultiStruct() Command

This command sets a Diversion MultiStruct location's information.  
A "MultiStruct" is used when demands are met using water from different tributaries.  
Each diversion station is represented in the model network  
and the historical water rights and diversion time series are distinct for each diversion station.  
However, the efficiencies are estimated using combined demand and historical diversion time series,  
and total demands are used for the primary structure, with zero demands on the other structure(s).  
Operating rules are used to handle sharing diversion water.  
The primary ID will receive all demands.  
Separate the part IDs by spaces or commas.

Diversion MultiStruct ID:

20MS02

Required - specify the Diversion MultiStruct ID.

Part IDs:

200516,200613,201004

Required - up to 12 characters for each ID.

If not found:

Optional - indicate action if no ID match is found (default=Warn).

Command:

SetDiversiionMultiStruct ( ID="20MS02", Part IDs="200516,200613,201004")

OK

Cancel

SetDiversiionMultiStruct

SetDiversiionMultiStruct() Command Editor

The command syntax is as follows:

```
SetDiversionMultiStruct (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The diversion identifier to associate with the MultiStruct part identifiers.	None – must be specified.
PartIDs	The list of part identifiers to comprise the MultiStruct, separated by commas and/or spaces. The first identifier is the primary diversion station, and the others are secondary stations.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the MultiStruct identifier is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the aggregate identifier is not matched</li><li>• Warn – generate a warning message if the aggregate identifier is not matched</li></ul>	Warn

# Command Reference: SetDiversionMultiStructFromList()

Set diversion MultiStruct parts from data in a list file

StateCU and StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversionMultiStructFromList()` command reads diversion MultiStruct part identifier data from a list file and saves the information for the diversion (a CU Location or StateMod diversion station). A diversion MultiStruct indicates that multiple diversion stations take water from more than one tributary but irrigate the same lands. Each diversion station is included in the model network and retains its normal capacity and historical diversions; however, average efficiencies are calculated using the combined demand and historical diversion time series. The demands for the primary structure (the first listed) are set to the total demands, with demands for secondary stations being set to zero. Diversion MultiStruct definition commands are required only when processing the demand time series. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionMultiStructFromList() Command**

This command sets a Diversion MultiStruct location's information from a list file.  
A "MultiStruct" is used when demands are met using water from different tributaries.  
Each diversion station is represented in the model network  
and the historical water rights and diversion time series are distinct for each diversion station.  
However, the efficiencies are estimated using combined demand and historical diversion time series,  
and total demands are used for the primary structure, with zero demands on the other structure(s).  
Operating rules are used to handle sharing diversion water.  
The primary ID will receive all demands.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversionDemandTSMonthlyToStateMod

List file:

ID column:  Required - column for the Diversion MultiStruct IDs.

Name column:  Optional - column for the Diversion MultiStruct name.

Part IDs column:  Required - first/only column for the part IDs.

Parts listed how:  Required - are part IDs listed in row or column?

Part IDs column (max):  Optional - maximum column for part IDs if in row (default is use all).

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
SetDiversionMultiStructFromList (ListFile="rgTW_multistruct.csv", IDCol=1, PartIDsCol=2, PartsListedHow=InRow)
```

SetDiversionMultiStructFromList

**SetDiversionMultiStructFromList() Command Editor**

The command syntax is as follows:

```
SetDiversionMultiStructFromList(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
IDCol	The column number (1+) containing the diversion MultiStruct primary station identifiers.	None – must be specified.
NameCol	The column number (1+) containing the diversion MultiStruct name.	None – optional (if not specified the name will not be changed).
PartIDsCol	The column number (1+) for the first column having part identifiers.	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers.	Use all available columns.
PartsListedHow	If InRow, it is expected that all parts defining a system are listed in the same row. If InColumn, it is expected that the parts defining a system are listed one per row, with multiple rows defining the full system (PartIDsColMax is ignored in this case).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the MultiStruct identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the aggregate identifier is not matched</li> <li>Warn – generate a warning message if the aggregate identifier is not matched</li> </ul>	Warn

An example list file is shown below:

```
#AggID,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,Ditch,
20MS01,200516,200613,201004
20MS02,200623,201060,210521,210522
20MS03,200706,200784
20MS04,200814,200815
20MS05,200683,200775
...
```

# Command Reference: SetDiversiionRight()

## Set diversion right data

### StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `SetDiversiionRight()` command sets data in existing diversion rights or adds a new diversion right. If a new right is added, it is added in alphabetical order according to the right identifier. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetDiversiionRight() Command

This command sets (edits) data in diversion right(s), using the diversion right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
If the right ID does not contain a \* wildcard pattern and does not match an ID,  
the right will be added if the "If not found" option is set to Add.  
If the right ID does not contain a \* wildcard pattern and does match an ID,  
the right will be reset if the "If found" option is set to Set.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="364683.01"/>	Required - specify the right(s) to set (use * for wildcard).
Name:	<input type="text" value="Con-Hoosier 1948 WR"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text" value="ID"/>	Optional - station ID or "ID" to match first part of right ID.
Administration number:	<input type="text" value="35927.00000"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text" value="540.0"/>	Optional - decree amount, CFS.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="button" value="Warn"/>	Optional - indicate action if no match is found (default=Warn).
If found:	<input type="button" value="Set"/>	Optional - indicate action if match is found (default=Warn).
Command:	<pre>SetDiversiionRight ( ID="364683.01",Name="Con-Hoosier 1948 WR",StationID="ID",AdministrationNumber=35927.00000,Decree= 540.0,OnOff=1,IfNotFound=Warn,IfFound=Set)</pre>	

OK

Cancel

SetDiversiionRight

### SetDiversiionRight() Command Editor

The command syntax is as follows:

```
SetDiversionRight(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single diversion right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching diversion rights.	If not specified, the original value will remain.
StationID	The diversion station identifier to be assigned for all matching diversion rights.	If not specified, the original value will remain.
AdministrationNumber	The administration number to be assigned for all matching diversion rights.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching diversion rights.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching diversion rights, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the water right if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn
IfFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Set – set the water right data</li> <li>• Fail – generate a failure message if the ID is matched</li> <li>• Ignore – ignore (don't set and don't generate a message) if the ID is matched</li> <li>• Warn – generate a warning message if the ID is matched</li> </ul>	Warn

# Command Reference: SetDiversionStation()

## Set diversion station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversionStation()` command sets data in existing diversion stations or adds a new diversion station. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionStation() Command**

This command sets (edits) data in diversion station(s), using the diversion station ID to look up the location.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
If the station ID does not contain a \* wildcard pattern and does not match an ID, the location will be added if the "If not found" parameter is set to Add.  
Use blanks in the any field to indicate no change to the existing value.  
Monthly efficiencies should be separated by commas, with January first.  
Returns should be specified as triplets of location, percent, and return table ID:  
08123456,50.0,1;08234567,50.0,2

Diversion station ID:	<input type="text" value="360784"/>	Required - ID for stations to fill (use * for wildcard)
Name:	<input type="text"/>	Optional - up to 24 characters for StateMod.
River node ID:	<input type="text"/>	Optional - the river node identifier, or "ID" to use the station ID.
On/Off:	<input type="button" value="v"/>	Optional - is station on/off in data set?
Capacity:	<input type="text"/>	Optional - diversion capacity, CFS.
Replacement Res. Option:	<input type="button" value="v"/>	Optional - replacement reservoir option.
Daily ID:	<input type="text"/>	Optional - the daily identifier, "ID", or StateMod flag).
User name:	<input type="text"/>	Optional - specify the user name.
Demand type:	<input type="button" value="v"/>	Optional - monthly demand time series type.
Irrigated acres:	<input type="text"/>	Optional - typically for the most recent year.
Use type:	<input type="button" value="v"/>	Optional - water use type.
Demand source:	<input type="button" value="v"/> 6 - Municipal, industrial, or transmountain structure	Optional - water demand source.
Efficiency (Annual):	<input type="text"/>	Optional - annual efficiency, percent (ignore if setting monthly).
Efficiencies (Monthly):	<input type="text" value="10,12,14,44,55,62,61,56,44,26,0,10"/>	Optional - percent, annual is recomputed as average.
Returns (optional):	<input type="text"/>	
If not found:	<input type="button" value="v"/> Warn	Optional - indicate action if no match is found.
Command:	<pre>SetDiversionStation( ID="360784", DemandSource=6, EffMonthly="10, 12, 14, 44, 55, 62, 61, 56, 44, 26, 0, 10", IfNotFound=Warn)</pre>	

OK Cancel

SetDiversionStation

### SetDiversionStation() Command Editor

The command syntax is as follows:

```
SetDiversionStation( Parameter=Value,... )
```

**Command Parameters**

<b>Parameter</b>	<b>Description</b>	<b>Default</b>
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching diversion stations.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching diversion stations. Specify ID to assign to the diversion station identifier.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching diversion stations, either 1 for on or 0 for off.	If not specified, the original value will remain.
Capacity	The diversion station capacity, CFS.	If not specified, the original value will remain.
ReplaceResOption	The replacement reservoir option, as per the StateMod documentation.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching diversion stations.	If not specified, the original value will remain.
UserName	The diversion user name (owner).	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching diversion stations (see StateMod documentation).	If not specified, the original value will remain.
IrrigatedAcres	The irrigated acres to be assigned for all matching diversion stations.	If not specified, the original value will remain.
UseType	The use type to be assigned for all matching diversion stations (see StateMod documentation).	If not specified, the original value will remain.
DemandSource	The demand source to be assigned for all matching diversion stations (see StateMod documentation).	If not specified, the original value will remain.
EffAnnual	The annual efficiency (percent, 0 - 100) to be assigned for all matching diversion stations (see StateMod documentation). Monthly efficiencies will be set to the same value (but not used).	If not specified, the original value will remain.
EffMonthly	The monthly efficiencies (percent, 0 – 100) to be assigned for all matching diversion stations, specified as 12 comma-separated values, January to December. The annual efficiency will be set to the average value. The order of the values in the output file will be according to the output year type set by SetOutputYearType( ), or calendar by default.	If not specified, the original value will remain.



Parameter	Description	Default
Returns	The return flows to be assigned for all matching diversion stations. Specify as StationID,Percent,DelayTableID; StationID,Percent,DelayTableID; etc.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Add – add the diversion station if the ID is not matched and is not a wildcard</li><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

This page is intentionally blank.

---

# Command Reference:

## SetDiversionStationCapacitiesFromTS()

**Set diversion station capacity data as maximum historical diversion**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetDiversionStationCapacitiesFromTS()` command sets diversion station capacities to the maximum historical time series (monthly) value. The historical time series must have been previously read or calculated with other commands. Monthly ACFT values are converted to CFS units by applying the conversion:

$$\text{CFS} = \text{X ACFT} / (1.9835 * \text{DaysInMonth})$$

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionStationCapacitiesFromTS() Command**

This command sets the diversion station capacity to the largest value in the diversion historical time series (monthly). This is necessary because the initial capacity value may be too small and will be a constraint in the simulation. The monthly ACFT value is converted to CFS using the number of days in the month. The capacity is reset only if the time series value is larger than the existing capacity.

Diversion station ID:  Required - stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

SetDiversionStationCapacitiesFromTS

**SetDiversionStationCapacitiesFromTS() Command Editor**

The command syntax is as follows:

```
SetDiversionStationCapacitiesFromTS(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file excerpt illustrates how time series can be limited to rights prior to writing the StateMod time series file. Note that the original diversion stations file is read and a new one is written.

```
#
# Step 2 - read structure list from preliminary direct diversion structure file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
...steps omitted...
#
# Step 8 - fill historical diversion using pattern approach
#
FillDiversionHistoricalTSMonthlyPattern(ID="36*",PatternID="09034500")
...similar commands omitted...
#
# Step 9 - Fill remaining missing with month average
#
FillDiversionHistoricalTSMonthlyAverage(ID="*")
#
# Step 10 - Limit filled diversion to water rights. Exceptions include structure
#           receiving significant reservoir supply, carrier structures, etc.
#
LimitDiversionHistoricalTSMonthlyToRights(InputFile="..\statemod\cm2005.ddr",
ID="*",IgnoreID="954683,952001,950010,950011")
#
# Step 11 - sort structures and create historical diversion file
#
SortDiversionHistoricalTSMonthly(Order=Ascending)
WriteDiversionHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
#
# Step 12 - update capacities and create final direct diversion station file
#
SetDiversionStationCapacitiesFromTS(ID="*")
WriteDiversionStationsToStateMod(OutputFile="..\statemod\cm2005.dds")
#
# Check the results.
CheckDiversionHistoricalTSMonthly(ID="*")
WriteCheckFile(OutputFile="ddh.commands.StateDMI.check.html")
```

---

# Command Reference: SetDiversionStationDelayTablesFromNetwork()

Set diversion station delay table data from the network

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetDiversionStationDelayTablesFromNetwork()` command sets delay table data in existing diversion stations using network information. A default delay table is used to assign 100% of the returns to the downstream node in the network. This command is often used to set a default before more specific delay table information is set with the `SetDiversionStationDelayTablesFromRTN()` command. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionStationDelayTablesFromNetwork() Command**

This command sets diversion station delay table data using network data.  
The network must be read with a previous command.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
By default, 100% of the returns will be returned to the downstream node.  
Specify the delay table to use for the returns.  
Separate delay table files are used for monthly and daily data sets, although table identifiers can be made to agree.

Diversion station ID:  Required - diversion stations to fill (use \* for wildcard).

Default table:  Optional - default table to use with 100% of returns (default=1).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
SetDiversionStationDelayTablesFromNetwork ( ID="*", Default Table=1)
```

SetDiversionStationDelayTablesFromNetwork

**SetDiversionStationDelayTablesFromNetwork() Command Editor**

The command syntax is as follows:

```
SetDiversionStationDelayTablesFromNetwork( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single diversion station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DefaultTable	The default delay table to use when assigning the delay tables.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: SetDiversionStationDelayTablesFromRTN()

Set diversion station delay table data from an RTN format file

## StateMod Command

Version 3.09.01, 2010-02-01

The `SetDiversionStationDelayTableFromRTN()` command sets delay table data in existing diversion stations using information in an RTN format file, which is a format that has been used in CDSS StateMod modeling, and is created by the “makertn” program. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionStationDelayTablesFromRTN() Command**

This command reads and processes delay table information from an "RTN" format file.  
Delay (return flow) table data indicate the pattern by which unused water is returned to the system.  
The file may contain default efficiency information for diversion stations.  
This information can be used and can then be reset later when average efficiencies are estimated from time series.  
This file format has been used with CDSS modeling software and is provided for backward compatibility.  
A delimited list file format may be supported in the future.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillDiversionStationsFromHydroBase

Input file:

Set efficiency?:  Optional - if True, use default efficiency information in file (default=False).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

SetDiversionStationDelayTablesFromRTN

### SetDiversionStationDelayTablesFromRTN() Command Editor

The command syntax is as follows:

```
SetDiversionStationDelayTablesFromRTN( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the RTN file to process. Specify an absolute path or a path relative to the working directory.	None – must be specified.
SetEfficiency	Indicates whether the default efficiency value in the file should be used.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

A sample RTN file is shown below:

200511	2	75	1
	200742	1	1
	200742	99	2
200742	2	75	1
	200787	1	1
	200787	99	2
200752	2	75	1
	20ADW07	1	1
	20ADW07	99	2

The first line contains the station identifier, number of return flow locations, default efficiency for the station, and the default delay table to use for the return. For the number of return flow locations, the following lines indicate the identifier for the station to receive the return, the percentage of the return to receive, and the delay table for the return.



# Command Reference: SetDiversionStationsFromList()

Set diversion station data from a list file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `SetDiversionStationsFromList()` command sets data in existing diversion stations (it currently will not add a station – use `ReadDiversionStationsFromList()`). This command is useful when data has been created from another program or process. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionStationsFromList() Command**

This command edits/sets data in diversion stations, using the station ID to look up the location.  
Data are supplied by values in a delimited file.  
Use blanks in the any field to indicate no change to the existing value.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillDiversionStationsFromHydroBase

List file:

Diversion stations ID column:	<input type="text" value="1"/>	Required - column (1+) for identifier.
Name column:	<input type="text"/>	Optional - column (1+) for name.
River node ID column:	<input type="text"/>	Optional - column (1+) for river node ID.
On/Off column:	<input type="text"/>	Optional - column (1+) for on/off switch.
Capacity column:	<input type="text"/>	Optional - column (1+) for capacity.
Replacement reservoir option column:	<input type="text"/>	Optional - column (1+) for repl. res. option.
Daily ID column:	<input type="text"/>	Optional - column (1+) for daily ID.
User name column:	<input type="text"/>	Optional - column (1+) for user name.
Demand type column:	<input type="text"/>	Optional - column (1+) for demand type.
Irrigated acres column:	<input type="text"/>	Optional - column (1+) for irrigated acres.
Use type column:	<input type="text"/>	Optional - column (1+) for use type.
Demand source column:	<input type="text"/>	Optional - column (1+) for demand source.
Efficiency (annual) column:	<input type="text"/>	Optional - column (1+) for annual efficiency.
Efficiency (monthly) column:	<input type="text" value="1"/>	Optional - first column of 12, listed Jan...Dec.
Delimiter:	<input type="text" value=","/>	Optional - delimiter character(s) (default=",").
Merge delimiters:	<input type="text"/>	Optional - treat consecutive delimiters as one (default=False).
If not found:	<input type="text"/>	Optional - indicate action if no ID match is found (default=Warn).

Command:

SetDiversionStationsFromList

**SetDiversionStationsFromList() Command Editor**

The command syntax is as follows:

```
SetDiversionStationsFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the delimited input file to read. Strings that include delimiter characters can be surrounded by double quotes in the list file. Lines starting with # are treated as comments.	None – must be specified.
IDCol	The column number (1+) containing the diversion station identifiers.	If not specified, the original value will remain.
NameCol	The column number (1+) containing the diversion station names.	If not specified, the original value will remain.
RiverNodeIDCol	The column number (1+) containing the river node identifiers.	If not specified, the original value will remain.
OnOffCol	The column number (1+) containing the on/off switch.	If not specified, the original value will remain.
CapacityCol	The column number (1+) containing the capacity.	If not specified, the original value will remain.
ReplaceResOptionCol	The column number (1+) containing the replacement reservoir option.	If not specified, the original value will remain.
DailyIDCol	The column number (1+) containing the daily identifier.	If not specified, the original value will remain.
UserNameCol	The column number (1+) containing the user name.	If not specified, the original value will remain.
DemandTypeCol	The column number (1+) containing the demand type.	If not specified, the original value will remain.
IrrigatedAcresCol	The column number (1+) containing the irrigated acres.	If not specified, the original value will remain.
UseTypeCol	The column number (1+) containing the use type.	If not specified, the original value will remain.
DemandSourceCol	The column number (1+) containing the demand source.	If not specified, the original value will remain.
EffAnnualCol	The column number (1+) containing the annual efficiency. If the annual efficiency is specified, each monthly efficiency will be set to the annual value.	If not specified, the original value will remain.

Parameter	Description	Default
EffMonthlyCol	The column number (1+) containing the monthly efficiency for January. The efficiencies for other months should be specified in columns that follow. The annual efficiency is set to the average of the monthly efficiencies. The efficiencies in the list file must be listed January to December as percent (0 to 100). The order of the values in the StateMod diversion stations will be according to the output year type set by <code>setOutputYearType()</code> , or calendar by default.	If not specified, the original values will remain.
Delim	The character(s) that delimits columns, or one of the literal words: <ul style="list-style-type: none"> <li>• Space</li> <li>• Tab</li> <li>• Whitespace – spaces and tabs.</li> </ul>	, (comma)
MergeDelim	If True, then treat consecutive delimiter characters as one delimiter. If False, separate columns will result.	False
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example illustrates how to create diversion stations from a list file and then set the efficiencies (in this case from a StateCU output file) from another list. The full data line is trimmed of whitespace before processing and data in columns are automatically trimmed of whitespace after parsing.

```
StartLog(LogFile="commands.StateDMI.log")
# For testing...
# setOutputYearType(Water)
ReadDiversionStationsFromList(ListFile="rgdssall.csv",IDCol="1")
SetDiversionStationsFromList(
    ListFile="rg2007-diveff.csv",IDCol="1",
    EffMonthlyCol="2",Delim="Space",MergeDelim=True,IfNotFound=Warn)
WriteDiversionStationsToStateMod(OutputFile="rgdssall.dds")
```

The following is an example of the list file used with the above:

```
# Card 1 Control
# format: (Free)
# NOTE EFF1 IS JANUARY, EFF2 IS FEBRUARY, ETC.
#
# ID      cwelid:  Well ID
# Eff1    eff(1)   Efficiency in month 1
# Eff1    eff(2)   Efficiency in month 2
# ...     ....    ...
# Eff1    eff(12)  Efficiency in month 12
#
#
#1 ID      Eff1    Eff2    Eff3    Eff4    Eff5    Eff6'Eff7    Eff8    Eff9    Eff10    Eff11    Eff12
#-----eb-----eb-----eb-----eb-----eb-----eb-----'-----eb-----eb-----eb-----eb-----exb-----eb-----
#
# 200505      42.    42.    42.    42.    42.    42.    42.    42.    42.    42.    42.    42. ALAMOS A D
# 200511      49.    49.    14.    8.    21.    30.    38.    35.    27.    11.    3.    4. ANACONDA D
```

# Command Reference: SetDiversionSystem()

## Set diversion system parts

### StateCU and StateMod Command

Version 3.08.02, 2010-01-07

The `SetDiversionSystem()` command sets diversion system part identifier data for a diversion (a CU Location that corresponds to a diversion or D&W node or StateMod diversion station). Diversion systems are specified using a list of ditch identifiers, and the system information applies for the full model period (does not vary by year). To facilitate processing, it is often best to use list files to specific systems (see `SetDiversionSystemFromList()`). Systems by convention have their water rights fully represented in output – to aggregate water rights at a location, use an aggregate (see the similar `Aggregate` commands). See also the StateDMI **Introduction** chapter, which provides additional information about systems and other modeling conventions. System information should be specified after diversion locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionSystem() Command**

This command sets a Diversion System location's information.  
Each System is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with `SetDiversionAggregate()` when water rights will be aggregated into classes.  
A "System" is used with `SetDiversionSystem()` when individual water rights will be maintained.  
For example, multiple nearby or related ditches may be grouped as a single identifier.  
When grouping ditches, specify the diversion station IDs for parts.  
Separate the part IDs by spaces or commas.

Diversion System ID:  Required - specify the Diversion System ID.

Part IDs: 

530555, 530519, 530521

 Required - up to 12 characters for each ID.

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

SetDiversionSystem( ID="530555", PartIDs="530555, 530519, 530521" )

SetDiversionSystem

**SetDiversionSystem() Command Editor**

The command syntax is as follows:

```
SetDiversionSystem(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The diversion identifier to associate with the collection of individual diversions.	None – must be specified.
PartIDs	The list of part identifiers to comprise the system, for example ditch WDIDs that will be found in HydroBase. The part identifiers are by default of type Ditch.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the identifier is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the identifier is not matched</li><li>• Warn – generate a warning message if the identifier is not matched</li></ul>	Warn

# Command Reference: SetDiversionSystemFromList()

Set diversion system parts from data in a list file

StateCU and StateMod Command

Version 3.08.02, 2010-01-07

The `SetDiversionSystemFromList()` command sets diversion system part identifier data for a diversion (a CU Location that corresponds to a diversion or D&W node or StateMod diversion station). Diversion systems are specified using a list of ditch identifiers, and the system information applies for the full model period (does not vary by year). To facilitate processing, the list of parts is specified in a delimited list file. Systems by convention have their water rights fully represented in output – to aggregate water rights at a location, use an aggregate (see the similar `Aggregate` commands). See also the **StateDMI Introduction** chapter, which provides additional information about systems and other modeling conventions. System information should be specified after diversion locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetDiversionSystemFromList() Command**

This command sets a Diversion System location's information from a list file.  
Each System is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with `SetDiversionAggregateFromList()` when water rights will be aggregated into classes.  
A "System" is used with `SetDiversionSystemFromList()` when individual water rights will be maintained.  
For example, multiple nearby or related ditches may be grouped as a single identifier.  
When grouping ditches, specify the diversion station IDs for the parts in the list file.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetDiversionSystemFromList

List file:

ID column:  Required - column for the Diversion System IDs.

Name column:  Optional - column for the Diversion System name.

Part IDs column:  Required - first/only column for the part IDs.

Parts listed how:  Required - are part IDs listed in row or column?

Part IDs column (max):  Optional - maximum column for part IDs if in row (default is use all).

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command:  
`SetDiversionSystemFromList (ListFile="cm_divsys.csv", IDCol=1, NameCol=2, PartIDCol=3, PartsListedHow=InRow)`

SetDiversionSystemFromList

SetDiversionSystemFromList() Command Editor

The command syntax is as follows:

```
SetDiversionSystemFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
IDCol	The column number (1+) containing the diversion system identifiers.	None – must be specified.
NameCol	The column number (1+) containing the diversion system name.	None – optional (name will remain as previously defined).
PartIDsCol	The column number (1+) for the first column having part identifiers. The identifiers are ditch WDIDs that will be found in HydroBase. The part identifiers are by default of type Ditch.	None – must be specified.
PartsListedHow	If InRow, it is expected that all parts defining a system are listed in the same row (as shown in the example below). If InColumn, it is expected that the parts defining a system are listed one per row, with multiple rows defining the full system (PartIDsColMax is ignored in this case).	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers. Use if extra columns on the right need to be excluded from the list.	Use all available non-blank columns starting with PartIDsCol.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the system identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the system identifier is not matched</li> <li>Warn – generate a warning message if the system identifier is not matched</li> </ul>	Warn



An example list file is shown below:

```
#the following are all divsystems
360649,Hamilton Davidson Div Sys,360649,360541
360662,Hoagland Div Sys,360662,360946,361018,361047,361020,361019,360945,361048,361049
380880,Mt. Sopris Div Sys,380880,381633
394725,Vulcan Ditch Div Sys,394725,390685
500734,Deberard Div Sys,500734,500548
510529,Big Lake Div Sys,510529,510584
510941,Vail Irr Div Sys,510941,511231
511309,FRASER RIVER DIVR PROJ,511309,510593
530555,Derby Div Sys,530555,530519,530521
720512,Arbogast Pump Div Sys,721072,720512
720852,RMG Div Sys,720852,720555
950050,Redlands Power Canal Irr,724713
720766,Ute WCD Carver Ranch,720766,721334
721329,Rapid Creek PP DivSys,721329,721235
720820,Park Creek DivSys,720820,720819
```

The following command file illustrates how diversion systems are defined with this command:

```
# ddr.commands.StateDMI
#
# StateDMI command file to create the direct diversion rights file for the Colorado model
#
# Step 1 - read structures from preliminary direct diversion station file
#
ReadDiversionStationsFromStateMod(InputFile="cm2005_dds.dds")
#
# Step 2 - read aggregate and diversion system structure assignments. Note that
#          want to combine water rights for aggs and diversion systems, but
#          water rights are assigned to primary and secondary components of multistructures
#
SetDiversionAggregateFromList(ListFile="cm_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
SetDiversionSystemFromList(ListFile="cm_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
#
# Step 3 - read diversion rights from HydroBase and define water rights classes
#          used for aggregate structures - but NOT for diversion systems
#
ReadDiversionRightsFromHydroBase(ID="*",OnOffDefault=1,
    AdminNumClasses="14854.00000,20427.18999,22729.21241,30895.21241,31258.00000,
    32023.28989,39095.38998,43621.42906,46674.00000,48966.00000,99999.")
#
# Step 4 - set water rights for structure IDs different from or not included in HydroBase
#
# Grand Valley Area - many rights obtain water through operations
SetDiversionRight(ID="720646.02",Name="Orchard Mesa Irr Dist
Sys",StationID="ID",OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.05",Name="USA Power
Plant",StationID="ID",Decree=800.0,OnOff=1,IfNotFound=Add,IfFound=Set)
SetDiversionRight(ID="720646.07",Name="Grand Valley
Proj",StationID="ID",AdministrationNumber=22729.19544,Decree=40.0,OnOff=1,
    IfNotFound=Add,IfFound=Set)
... commands omitted
#
# Step 7 - create direct diversion rights file
#
WriteDiversionRightsToStateMod(OutputFile="cm2005.ddr")
```

This page is intentionally blank.

---

# Command Reference:

## SetInstreamFlowDemandTSAverageMonthlyConstant()

**Set instream flow demand time series (average monthly) data to constant value(s)**

**StateMod Command**  
Version 3.09.01, 2010-02-02

The `SetInstreamFlowDemandTSAverageMonthlyConstant()` command sets instream flow demand time series (average monthly) data to constant monthly values. Typically this command is used to (re)set values after the `SetInstreamFlowDemandTSAverageMonthlyFromRights()` command is used.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetInstreamFlowDemandTSAverageMonthlyConstant() Command**

This command sets an instream flow demand time series (average monthly) to constant values.  
The instream flow station identifier is used to match the time series that is assigned.  
Specify 12 monthly values (January through December) separated by spaces or commas.

Instream flow station ID:  Required - identifier of station to process.

Monthly values  Required - monthly constant values for Jan-Dec.

Recalculate limits:  Recalculate original data limits after set (default=True).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  
`SetInstreamFlowDemandTSAverageMonthlyConstant (ID="362000", MonthValues="3.00,3.00,3.00,3.00,6.00,6.00,6.00,6.00,6.00,3.00,3.00,3.00", IfNotFound=Add)`

**SetInstreamFlowDemandTSAverageMonthlyConstant() Command Editor**

The command syntax is as follows:

```
SetInstreamFlowDemandTSAverageMonthlyConstant (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
MonthValues	Twelve monthly instream flow demand time series values for January through December.	None – must be specified.
RecalcLimits	If <b>True</b> , then the time series average limits will be recalculated. If <b>False</b> , the limits from previously set data will be used (not typically used).	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• <b>Add</b> – add the instream flow demand time series if the ID is not matched and is not a wildcard</li> <li>• <b>Fail</b> – generate a failure message if the ID is not matched</li> <li>• <b>Ignore</b> – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• <b>Warn</b> – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## SetInstreamFlowDemandTSAverageMonthlyFromRights()

**Set instream flow demand time series (average monthly) data from instream flow rights**

### StateMod Command

Version 3.09.01, 2010-02-02

The `SetInstreamFlowDemandTSAverageMonthlyFromRights()` command sets instream flow demand time series (average monthly) data using instream flow water rights data that have been previously read (e.g., from a `ReadInstreamFlowRightsFromStateMod()` command). The resulting time series at each instream flow station represents the total water rights for the specified station. The output year type is set to that defined by the most recent `SetOutputYearType()` command. For average time series, it is only important that a sequence of months be specified in the time series. If water year is used, then the data span two calendar years in memory. Incorrectly specifying the year type may result in missing data in the output.

If necessary, the constant values determined from water rights can be reset using the `SetInstreamFlowDemandTSAverageMonthlyConstant()` command.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetInstreamFlowDemandTSAverageMonthlyFromRights() Command**

This command sets instream flow demand average monthly time series using instream flow water rights. The instream flow water rights must have been read or set with a previous command. The total water rights for the instream flow station are used to define the right. The demand time series is set to a constant value, equal to the water right decree.

Instream flow station ID:  Required - instream flow station ID to set (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
SetInstreamFlowDemandTSAverageMonthlyFromRights ( ID = "3 *", IfNotFound=Add)
```

SetInstreamFlowDemandTSAverageMonthlyFromRights

### SetInstreamFlowDemandTSAverageMonthlyFromRights() Command Editor

The command syntax is as follows:

```
SetInstreamFlowDemandTSAverageMonthlyFromRights(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Add – add the instream flow demand time series if the ID is not matched and is not a wildcard</li><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

# Command Reference: SetInstreamFlowRight()

## Set instream flow right data

### StateMod Command

Version 3.09.01, 2010-03-14, Color, Acrobat Distiller

The `SetInstreamFlowRight()` command sets data in existing instream flow rights or adds a new instream flow right. If a new right is added, it is added in alphabetical order according to the right identifier. Instream flow rights may be defined for a variety of reasons for modeling purposes where a flow needs to be ensured.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetInstreamFlowRight() Command**

This command sets (edits) data in instream flow right(s), using the instream flow right ID to look up the right. The right ID can contain a \* wildcard pattern to match one or more rights. If the right ID does not contain a \* wildcard pattern and does not match an ID, the right will be added if the "If not found" option is set to Add. If the right ID does not contain a \* wildcard pattern and does match an ID, the right will be reset if the "If found" option is set to Set. Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="953508.01"/>	Required - specify the right(s) to set (use * for wildcard).
Name:	<input type="text" value="Rifle_Gap_Res_Bypass"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text" value="ID"/>	Optional - station ID or "ID" to match first part of right ID.
Administration number:	<input type="text" value="37503.36898"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text" value="5.00"/>	Optional - decree amount, CFS.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="button" value="Add"/>	Optional - indicate action if no match is found (default=Warn).
If found:	<input type="button" value="Warn"/>	Optional - indicate action if match is found (default=Warn).
Command:	<pre>SetInstreamFlowRight (ID="953508.01",Name="Rifle_Gap_Res_Bypass",StationID="ID",AdministrationNumber=37503.36898,Decree=5.00,OnOff=1,IfNotFound=Add,IfFound=Warn)</pre>	

SetInstreamFlowRight

### SetInstreamFlowRight() Command Editor

The command syntax is as follows:

```
SetInstreamFlowRight (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching instream flow rights.	If not specified, the original value will remain.
StationID	The instream flow station identifier to be assigned for all matching instream flow rights.	If not specified, the original value will remain.
Administration Number	The administration number to be assigned for all matching instream flow rights.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching instream flow rights.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching instream flow rights, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the instream flow right if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn
IfFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Set – set the instream flow right, overwriting previous values if the ID is matched</li> <li>• Fail – generate a failure message if the ID is matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is matched</li> <li>• Warn – generate a warning message if the ID is matched</li> </ul>	Warn



# Command Reference: SetInstreamFlowStation()

## Set instream flow station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetInstreamFlowStation()` command sets data in existing instream flow stations or adds a new instream flow station. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetInstreamFlowStation() Command

This command sets (edits) data in instream flow station(s), using the instream flow station ID to look up the location. The instream flow station ID can contain a \* wildcard pattern to match one or more locations. If the instream flow station ID does not contain a \* wildcard pattern and does not match an ID, the location will be added if the "If not found" option is set to Add. Use blanks in the any field to indicate no change to the existing value.

Instream flow station ID:	<input type="text" value="954512"/>	Required - instream flow stations to fill (use * for wildcard).
Name:	<input type="text" value="Dillon_Res_Min_Rel"/>	Optional - up to 24 characters for StateMod.
Upstream river node ID:	<input type="text" value="954512"/>	Optional - upstream river node identifier.
Downstream river node ID:	<input type="text" value="954512_Dwn"/>	Optional - downstream river node identifier.
On/Off:	<input type="button" value="1 - On"/>	Optional - is instream flow station on/off in dataset?
Daily ID:	<input type="text" value="0"/>	Optional - daily identifier, "ID" to match ID, or StateMod flag).
Demand type:	<input type="button" value="2 - Average monthly demand"/>	Optional - demand time series type.
If not found:	<input type="button" value="Warn"/>	Optional - indicate action if no match is found (default=Warn).
Command:	<pre>SetInstreamFlowStation(ID="954512",Name="Dillon_Res_Min_Rel",UpstreamRiverNodeID="954512",DownstreamRiverNodeID="954512_Dwn",OnOff=1,DailyID="0",DemandType=2,IfNotFound=Wa rn)</pre>	

SetInstreamFlowStation

### SetInstreamFlowStation() Command Editor

The command syntax is as follows:

```
SetInstreamFlowStation(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single instream flow station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching instream flow stations.	If not specified, the original value will remain.
UpstreamRiverNodeID	The upstream river node identifier to be assigned for all matching instream flow stations.	If not specified, the original value will remain.
DownstreamRiverNodeID	The downstream river node identifier to be assigned for all matching instream flow stations.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching instream flow stations, either 1 for on or 0 for off.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching instream flow stations.	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching instream flow stations, one of:  1 – Average monthly demand, 2 – Monthly demand.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the instream flow station if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: SetIrrigationPracticeTS()

## Set irrigation practice time series values

### StateCU Command

Version 3.09.01, 2010-02-01

The `SetIrrigationPracticeTS()` command sets irrigation practice time series data for a CU Location. Setting acreage values results in a cascade of adjustments to maintain sums, and will be noted in the log file. Preference is given to maintaining the total acreage, then groundwater acreage, and then surface water acreage. Irrigation method within groundwater will agree with the total and the sprinkler and flood acreage will be prorated based on previous values if necessary to adjust to the total. Similar adjustments are made to surface water acreage. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetIrrigationPracticeTS() Command

This command edits irrigation practice time series data, using the CU Location ID to look up the location.

The CU Location ID can contain a \* wildcard pattern to match one or more locations.

The previous irrigation practice data will be reset to new values.

Blanks will result in no change to the data.

The following can typically set with no restriction because they are usually a simple set, with no subsequent filling over time:

Efficiencies, groundwater mode, pumping maximum.

**When processing acreage, the following approaches should be used.**

**1. Supply supplemental acreage data that are not in HydroBase for a year with parcel data - see SetIrrigationPracticeTSFromList().**

**2. Override all acreage data that are in HydroBase - acreage pairs will be set and will be adjusted to previous surface water only and groundwater total acres.**

**3. Override one irrigation method acreage - the other term will be computed from the total.**

**In all cases, acreage parts will be reduced to previous totals if necessary.**

CU Location ID:	<input type="text" value="0100501"/>	Required - CU Location(s) to fill (use * for wildcard).
Set start (year):	<input type="text" value="1950"/>	Optional - starting year to set data (default=set all).
Set end (year):	<input type="text" value="2006"/>	Optional - ending year to set data (default=set all).
Surface delivery efficiency maximum:	<input type="text"/>	Optional - specify a number 0.0 to 1.0.
Flood application efficiency maximum:	<input type="text"/>	Optional - specify a number 0.0 to 1.0.
Sprinkler application efficiency maximum:	<input type="text"/>	Optional - specify a number 0.0 to 1.0.
Acres irrigated by surface water only (flood):	<input type="text" value="0"/>	Optional.
Acres irrigated by surface water only (sprinkler):	<input type="text" value="0"/>	Optional.
Acres irrigated by groundwater (flood):	<input type="text" value="0"/>	Optional.
Acres irrigated by groundwater (sprinkler):	<input type="text" value="0"/>	Optional.
Pumping maximum:	<input type="text" value="0"/>	Optional - ACFT per month.
Groundwater mode:	<input type="text"/>	Optional.
Acres irrigated, total:	<input type="text" value="0"/>	Optional.
If not found:	<input type="text"/>	Optional - indicate action if no match is found (default=Warn).

Command:

```
SetIrrigationPracticeTS (ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,GWMode=null,AcresTotal=0)
```

OK

Cancel

SetIrrigationPracticeTS

### SetIrrigationPracticeTS() Command Editor

The command syntax is as follows:

SetIrrigationPracticeTS (Parameter=Value,...)

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
SetStart	The first year to set data values.	If not specified, data are set for the full output period.
SetEnd	The last year to set data values.	If not specified, data are set for the full output period.
SurfaceDelEffMax	Surface water delivery efficiency maximum (0.0 to 1.0).	If not specified, the original value will remain.
FloodAppEffMax	Flood application efficiency maximum (0.0 to 1.0).	If not specified, the original value will remain.
SprinklerAppEffMax	Sprinkler application efficiency maximum (0.0 to 1.0).	If not specified, the original value will remain.
AcresSWFlood	Acres irrigated by surface water, flood irrigation.	If not specified, the original value will remain, or will recompute based on other set values.
AcresSWSprinkler	Acres irrigated by surface water, sprinkler irrigation.	If not specified, the original value will remain, or will recompute based on other set values.
AcresGWFlood	Acres irrigated by groundwater, flood irrigation.	If not specified, the original value will remain, or will recompute based on other set values.
AcresGWSprinkler	Acres irrigated by groundwater, sprinkler irrigation.	If not specified, the original value will remain, or will recompute based on other set values.
PumpingMax	Maximum pumping, AF/M.	If not specified, the original value will remain.
GWMode	Groundwater mode (see StateCU documentation).	If not specified, the original value will remain.
AcresTotal	Total acres for location. This is normally set from the crop pattern time series data.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how to process the irrigation practice time series file where groundwater supply is used:

```
#
# Sp2008L_DDH.StateDMI
#
#
# StartLog(LogFile="SP_IPY.log")
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2006")
# Step 1 - Read CU Locations from list
#
ReadCULocationsFromList(ListFile="..\Sp2008L_StructList.csv",IDCol=1)
#
# Step 2 - Read SW aggregates, GW aggregates, and divsystems
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
#
SetWellSystemFromList(ListFile="..\SP_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
#
# Step 4 - Set conveyance efficiencies from file for key and sw aggregate structures - NOT in HydroBase
SetIrrigationPracticeTSFromList(ListFile="Sp2008L_Eff.csv",ID="*",
    SetStart=1950,SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="3")
#
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=2006,FloodAppEffMax=.6,SprinklerAppEffMax=.8,GWMode=2)
#
# Step 6 - Read well rights file and Set Max pumping (use merged *.wer file)
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
# Step 7 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Div="1")
#
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="Sp2008L.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
#
# Step 9 - Estimate 1950 ground water acreage based on active wells as defined in the non-merged *.wer
file
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 10 - Fill Interpolate Acreage Type (SW and GW) 1956-2006
# Step 11a - estimate total GW and total SW
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1956",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWater",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
```

```

GroundWater",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 11b - set sprinkler to zero in early period
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=1969,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 11c - fill remaining irrigation method values
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 12 - Set Acreage = 0 for structures that are in diversion systems, so acreage is not double
accounted
SetIrrigationPracticeTS(ID="0100503_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100507_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100687",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="0200834",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400511_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 13 - Set Acreage = 0, 1950-2006
SetIrrigationPracticeTS(ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100513",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100829",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400519",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 14 - Write final ipy file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateMod\Historic\Sp2008L.ipy",WriteHow=OverwriteFile)

```

---

# Command Reference: setIrrigationPracticeTSFromHydroBase()

Set irrigation practice time series (yearly) from HydroBase

## StateMod Command

Version 02.14.00, 2007-07-03, Color, Acrobat Distiller

THIS COMMAND IS OBSOLETE – INSTEAD, USE THE `readIrrigationPracticeTSFromHydroBase()` COMMAND. This older command was used for Phase 4 Río Grande work, and only works with one year of parcel data (e.g., 1998). However, an entirely new procedure has now been implemented, which can be applied to all basins. The new procedure relies on processing water rights into a StateMod water rights file and then using this file as input when processing parcels for the irrigation practice time series. Other commands have also been implemented to allow more control over acreage processing.

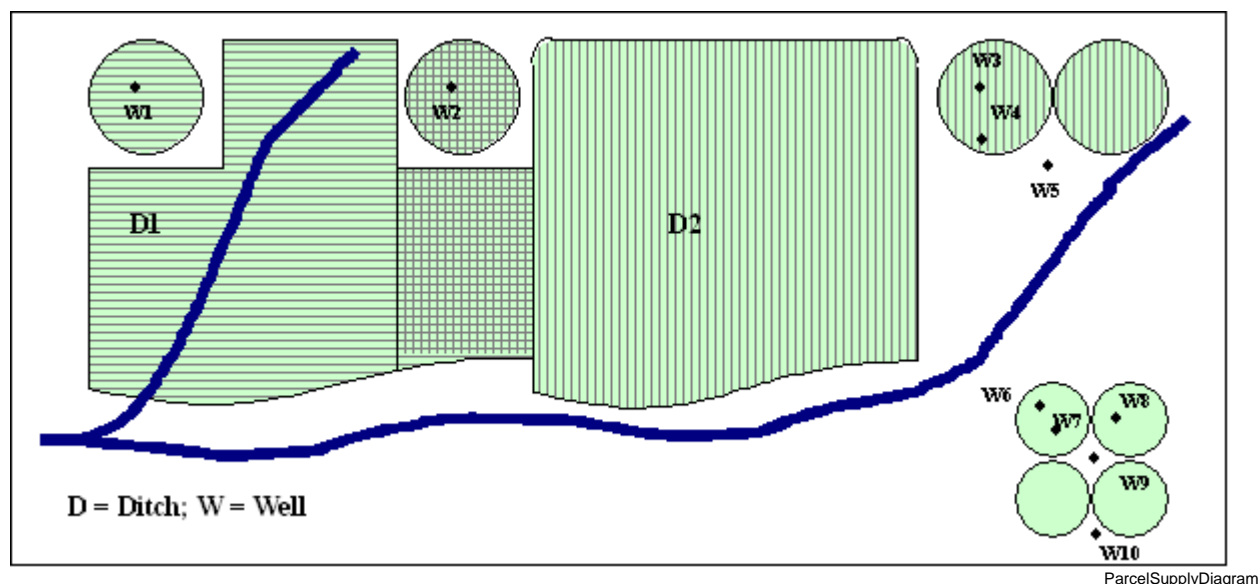
The `setIrrigationPracticeTSFromHydroBase()` command uses data in HydroBase to set the following irrigation practice time series (yearly) information for CU locations:

For each year of parcel data that is available, sprinkler acres for each location are set as the total of the parcel areas that are irrigated using the `SPRINKLER` irrigation type. Parcels associated with ditches have an area that is adjusted by the ditch coverage percent. All parcels associated with a CU location are used to determine the total sprinkler acres for the CU location. The sprinkler acres can be different for each year that parcels are available in HydroBase. Additional years of sprinkler data can be added using the `readIrrigationPracticeTSSprinklerAreaFromList()` command. The resulting data points can be used to fill the remaining period (e.g., see `fillIrrigationPracticeTS*()` commands).

- The well water rights/permits associated with parcels are used to determine the earliest appropriation date associated with each parcel. The groundwater acres value for the parcel is then recognized from the earliest date forward in time, with the earlier period being set to zero. Note that HydroBase data does not allow for turning off wells during the period – the wells currently in HydroBase are assumed to be active from the appropriation/permit date until the current time. Parcels associated with ditches have an area that is adjusted by the ditch coverage percent. All parcels associated with a CU location are used to determine the total groundwater acres for the CU location. **If multiple years of parcel data are in HydroBase, the last year to be processed will be in effect after the command is run. Therefore, the well rights/permits will be associated with the most recent year of parcel data. Additional development is necessary to evaluate how to merge multiple years of data.**
- Maximum well pumping is NOT processed by this command. However, the water rights resulting from the previous step are kept in memory and can be used by the `setIrrigationPracticeTSMaxPumpingToRights()` command. Separate commands are used in order to separate processing logic and minimize the command parameters necessary for each step. This also emphasizes the fact that the maximum pumping depends on water rights from the previous step. Therefore, the issue with handling multiple years of parcel data, if resolved in the previous step, will also cascade to the maximum pumping data.

The processing is very similar to that of the `readWellRightsFromHydroBase()` command, except that a list of CU locations is processed instead a list of StateMod well stations.

The following figure illustrates possible water supply for parcels.



**Example Supply for Parcels**

In this example, two ditches (D1 and D2, each represented with different cross-hatching) provide surface water supply to the indicated parcels. In some cases, only one ditch provides supply. Between the ditches, both supply water to shared parcels. Wells can supplement surface water supply (parcels above the river) or can be the sole supplier of water (lower right) and wells do not need to be physically located on a parcel to provide supply to the parcel. For StateCU, well-only lands are identified by CU locations that are defined by a collection (aggregate/system) of parcels. For StateMod, well-only lands are well stations that do not have a related diversion station. In both cases, lands irrigated by surface water are identified with ditch identifiers and parcels are associated to the ditches in HydroBase. Typically, well-only lands are grouped and multiple wells provide supply to the collection of parcels. Processing logic is different for ditch and well-only lands.

A well (hole in the ground) in HydroBase can be either a structure with water rights, a well permit, or both. In HydroBase, the relationship between well structure and well permit has been determined in CDSS projects by using common well attributes (e.g., name) or by spatial proximity analysis using GIS tools. However, for general well data in HydroBase, there has been no explicit link to help identify when a well structure matched a well permit. Additionally, well permit records are difficult to interpret because of replacement wells. Typically, major wells do have water rights, although the rights may have been applied for after a matching well permit. Specific knowledge about the basin should be used when evaluating the data. The CDSS projects have attempted to uniquely identify holes in the ground such that subsequent data processing can treat the hole as a structure or permit, but not both (to avoid double-counting). Wells were first modeled in the Río Grande RGDSS project and changes to HydroBase are occurring to better store well data to avoid some of the issues mentioned above. When processing well rights with this command, the CDSS processed data for wells (holes in the ground), water rights, and permits are used (raw well right and well permit data are not used).

The steps used to process irrigation practice time series are described below. Note that “CU location” refers to the StateCU model identifier (which can be a collection of wells) and “well” refers to a hole in the ground that has physical characteristics, water rights, and/or well permits.

Process each CU location that matches the ID pattern:



Process each year of parcel data. (see Year parameter).

Initialize the groundwater acreage time series to zeros.

If the CU location is an aggregate or system (specified using parcel ID, year, and div, and indicating that the CU location has only groundwater supply):

Loop through each parcel that the well irrigates:

If the irrigation type for the parcel is SPRINKLER, increment the sprinkler acres for the CU location, for the parcel year.

Determine wells associated with parcels.

Loop through each well:

Use the DefineRightHow parameter value to determine how to define the right. If the value is EarliestDate:

- Use the earliest of the right's appropriation date and permit's permit date. Convert the date to an administration number. If no date is available, assign the administration number to the value corresponding to the DefaultAppropriationDate parameter value or 99999.99999 as a final default.
- Assign the decree as the well yield, converted from GPM to CFS, multiplied by the percent of the well that irrigates the parcel.

If the value of DefineRightHow is RightIfAvailable:

- If a water right is available, use the appropriation date (and corresponding administration number) for the water right. If no date is available for the water right (this should not happen), assign the administration number to the value corresponding to the DefaultAppropriationDate parameter value or 99999.99999 as a final default.
- Assign the decree as the well yield, converted from GPM to CFS, multiplied by the percent of the well that irrigates the parcel. In this case the yield may have been previously converted from the water right CFS value.

Determine the earliest appropriation date for the rights. Increment the groundwater acres for the CU location by the parcel area, for the period of the earliest year to the end of the output period. Note that this calculation uses the individual rights, not the aggregate, because a relationship to parcel is required.

Else if the CU location is associated with a diversion station (indicating that well pumping supplements the diversion station surface water supply):

If the CU location is a collection (aggregate or system):

- Use the procedure described below to complete data processing.

Else if the CU location is explicitly modeled:

- Use the following procedure, treating the single diversion station as if it were the only diversion structure part in an aggregate/system.

Loop through each of the diversion structures associated with the CU location:

Determine the parcels that are irrigated by the diversion structure. Note that the following logic is similar to that for well-only lands above, except that the percent of the parcel served by the ditch is factored in.

Loop through each parcel that the diversion station irrigates:

If the irrigation type for the parcel is SPRINKLER, increment the sprinkler acres for the CU location, for the parcel year.

Determine wells associated with parcels. Loop through each well:

Use the `DefineRightHow` parameter value to determine how to define the right.

If the value is `EarliestDate`:

- Use the earliest of the right's appropriation date and permit's permit date. Convert the date to an administration number. If no date is available for the water right, assign the administration number to the value corresponding to the `DefaultAppropriationDate` parameter value or `99999.99999` as a final default.
- Assign the decree as the well yield, converted from GPM to CFS, multiplied by the percent of the well that irrigates the parcel AND the percent of the parcel that is irrigated by the ditch.

If the value of `DefineRightHow` is `RightIfAvailable`:

- If a water right is available, use the appropriation date (and corresponding administration number) for the water right. If no date is available for the water right (this should not happen), assign the administration number to the value corresponding to the `DefaultAppropriationDate` parameter value or `99999.99999` as a final default.
- Assign the decree as the well yield, converted from GPM to CFS, multiplied by the percent of the well that irrigates the parcel AND the percent of the parcel that is irrigated by the ditch. In this case the yield may have been previously converted from the water right CFS value.

Determine the earliest appropriation date for the rights. Increment the groundwater acres for the CU location by the parcel area, for the period of the earliest year to the end of the output period. Note that this calculation uses the individual rights, not the aggregate, because a relationship to parcel is required.

Else if the CU location is a well and is explicitly modeled

This case is not yet supported by StateDMI and has not been used in the past.

If aggregating rights (water rights classes are specified and the station is an aggregate or system), the following steps occur (well systems use steps 1-2 and are then explicitly added):

1. Water rights for each part of the aggregate are read from HydroBase as described above, reporting errors as necessary.
2. The rights are added to a list and are sorted by administration number. This ensures that the cumulative list of rights is listed in order of administration number.
3. Water rights are defined for each class (see the `AdminNumClasses` parameter description below), initializing the decree to zero.
4. For each class, the following sums are calculated: `sum(decree*AdminNum)` and `sum(decree)`, where the administration number is determined from the appropriation date derived from the original HydroBase administration number (it will not have a remainder).
5. The final administration number for the class is determined (it will not have a remainder):  

$$\text{int}(\text{sum}(\text{decree} * \text{AdminNum}) / \text{sum}(\text{decree}))$$

Water rights from HydroBase that are less than the decree minimum are ignored and during final output, water rights with a decree of 0.00 (the StateMod file format) are ignored. The name of the final right will include either water right (WDID and name) or permit information (number, suffix, and replacement),

depending on the input that was used. In the above process, status messages and warnings are printed to the log file as appropriate. For example, the following information is listed: the number of parcels for a CU location, the number of wells for the parcel, and the number of rights/permits for the well.

The following dialog is used to edit the command and illustrates the syntax of the command. Note that the input is very similar to the `readWellRightsFromHydroBase()` command because water rights are needed during processing.

**Edit setIrrigationPracticeTSFromHydroBase() Command**

THIS COMMAND IS OBSOLETE AND IS USED ONLY FOR PHASE 4 RIO GRANDE WORK - INSTEAD, SEE THE `readIrrigationPracticeTSFromHydroBase()` COMMAND.  
 This command sets irrigation practice time series data, using data from HydroBase.  
 Water rights are determined from derived well right and permit data, which have been matched with wells and parcels.  
 Derived data can be used as is, or well rights can be requested to obtain specific rights and add alternate point/exchange values.  
 The data are used to define groundwater acres, sprinkler acres, and maximum monthly well pumping.  
 Use the administration number classes to aggregate rights (e.g., to match StateMod rights data).  
 Well aggregates and wells associated with diversion stations require the year and division for parcels.  
 Specify administration number classes as administration numbers separated by commas.

CU location ID:	<input type="text" value="*"/>	Specify the CU locations to read (use * for wildcard).
Right ID format:	<input type="text" value=""/>	Indicate format for right identifiers (blank=StationIDW.NN).
Admin. number classes:	<input type="text" value="1,20000.00000,25000.00000,30000.00000,35000.00000,40000.00000,45000.00000"/>	
Input start (year):	<input type="text" value="1998"/>	Starting year to read data (blank for full period).
Input end (year):	<input type="text" value="1998"/>	Ending year to read data (blank for full period).
Water Division (Div):	<input type="text" value="3"/>	Specify the water division for the parcels.
Default appropriation date:	<input type="text" value="1932-01-01"/>	Use if date is not available from right or permit.
Define right how?:	<input type="text" value="RightIfAvailable"/>	Indicate how to define right (default is EarliestDate).
Read well rights?:	<input type="text" value="True"/>	Read well rights (default=True, False=use processed data).
Use Apex?:	<input type="text" value="False"/>	Used when ReadWellRights=True. Add APEX amount to right amount (default=True).
Command:	<pre>setIrrigationPracticeTSFromHydroBase(ID="*",AdminNumClasses="10000.00000,20000.00000,25000.00000,30000.00000,35000.00000,40000.00000,45000.00000,99999.99999",InputStart=1998,InputEnd=1998,Div=3,DefaultAppropriationDate="1932-01-01",DefineRightHow=RightIfAvailable,ReadWellRights=True,UseApex=False)</pre>	

OK Cancel

setIrrigationPracticeFromHydroBase

### setIrrigationPracticeTSFromHydroBase() Command Editor

The command syntax is as follows:

```
setIrrigationPracticeTSFromHydroBase(param=value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20 *).	None – must be specified.
AdminNum Classes	A list of administration numbers, separated by spaces or commas, to define the breaks for aggregate water rights, for well aggregates. For example, if the class breaks are 10000.000, 20000.00000, and 99999.99999, the first group will contain water rights with administration numbers <= 10000.00000, the second will contain water rights with administration number > 10000.00000 and <= 20000.00000, and the third will contain water rights with administration number > 20000.00000 and <= 99999.99999.	If not specified, well aggregates will be treated as well systems, with all water rights explicitly included in output.
InputStart	The starting calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems.  A single year or blank can be specified.	All years in HydroBase will be processed, with the most recent year being used for final groundwater acres and water rights output.
InputEnd	The ending calendar year to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems.  A single year or blank can be specified.	All years in HydroBase will be processed, with the most recent year being used for final groundwater acres and water rights output.
Div	A water division to use for parcel data, needed to determine relationships between diversion stations/parcels/wells and for well aggregate/systems.	None – must be specified.
Default Appropriation Date	Some right/permit data do not have a date in data records. For example, very old well permits may not have a date. In these cases a default date can be assigned to be used as the appropriation date in the well water right. The appropriation date will be converted to a State of Colorado administration number in StateMod water rights.	The administration number is set to 99999.99999.

Parameter	Description	Default
DefineRightHow	Wells (holes in the ground) are matched with water rights, well permits, and occasionally “estimated” wells necessary because a water right or permit could not be found. In some cases a right and permit will both exist for a well, each with their own dates. This parameter indicates how to define the right in these cases. A value of <code>EarliestDate</code> will use the earliest date determined from the right’s appropriation date and the permit’s permit date. A value of <code>RightIfAvailable</code> will always use the water right appropriation date, if available.	<code>EarliestDate</code>
ReadWellRights	This parameter is only used when <code>DefineRightHow=RightIfAvailable</code> , and indicates whether individual water rights should be read from HydroBase. The following values are recognized: <ul style="list-style-type: none"> <li>• <code>True</code> – the net amounts data are read, which may result in multiple well water rights for a well WDID. See also the <code>UseApex</code> parameter.</li> <li>• <code>False</code> – then a single processed water right will be returned, which is the sum of net amount rights, using the oldest appropriation date found for the rights (APEX is not considered).</li> </ul>	<code>True</code>
UseApex	Indicate whether to use alternate point/exchange values when processing rights. The following values are recognized: <ul style="list-style-type: none"> <li>• <code>True</code> – the APEX values corresponding to well rights are added to the net amount right values, resulting in a larger decree being considered for some rights.</li> <li>• <code>False</code> – the APEX values are not added to net amount rights.</li> </ul> Because net amount rights usually either have a decreed rate or an APEX amount, using <code>True</code> will generally result in more water rights, where the resulting right amount is either the decree or APEX.	<code>False</code>

This page is intentionally blank.

---

# Command Reference:

## SetIrrigationPracticeTSFromList()

**Set irrigation practice time series data from information in a delimited file**

**StateCU Command**

Version 3.09.01, 2010-02-17

The `SetIrrigationPracticeTSFromList()` command sets irrigation practice data for existing CU Locations by reading information from a delimited file. New locations are not added. The command can be used to set values over a period of 1+ years as follows:

1. If the `SetStart` and `SetEnd` parameters are specified and the year column is not specified, then repeat the values from the file for each year in the set period. For example, this can be used to repeat efficiency values through the period. Or, it can be used to provide acreage data not in HydroBase, for a specific year (set `SetStart` and `SetEnd` to the same value).
2. If the year column is provided, use the year in the file to specify the year for the set. In this case, `SetStart` and `SetEnd` control the period of data that will be processed from the file.

HydroBase may not contain all irrigated lands data. For example, additional lands may have been identified after HydroBase was populated or acreage must be set for a model identifier that is not a structure WDID in HydroBase (e.g., out of state lands). In this case, the command can be used to provide additional data to supplement HydroBase.

It is typical that separate `SetIrrigationPracticeTSFromList()` commands are used for different columns of data in the irrigation practice file. For example, efficiencies may be set with one command and acreage with another command.

The information-only surface water total and groundwater total values will be updated to agree with the acreage parts. However, no cascading adjustments will occur (as performed by `FillIrrigationPracticeInterpolate()` and other commands).

The following dialog is used to edit the command and illustrates the syntax of the command for repeating values over the specified period, with the values being set as the command is processed (omitting the year would repeat the values in all years):

**Edit SetIrrigationPracticeTSFromList() Command**
✕

This command sets irrigation practice time series data from a delimited list file, using the CU Location ID to look up the location. Resets will be enforced as the command is processed and can only apply to main locations, not aggregate/system parts. Use the ReadIrrigationPracticeTSFromList() command to read data that are not in HydroBase (e.g., parts of aggregates/systems). A comma-delimited list file is used to supply data, with values being set one of the following ways:

- 1) If the set start and end years are specified and a year column is not specified, the file data values are applied to each year in the set period.
- 2) If a year column is specified, year and corresponding values are read from the list file (the set period then controls how many years are processed).

The previous irrigation practice data will be reset to new values.  
Blanks in column fields will result in no change to the data.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase

List file:  Browse

CU Location ID:  Required - CU Location(s) to fill (use \* for wildcard).

Set start (year):  Optional - starting year to set data.

Set end (year):  Optional - ending year to set data.

Year column:  Optional - column in file for year.

CU location ID column:  Optional - column in file for CU location ID.

Surface delivery maximum efficiency column:  Optional - column in file for surface delivery maximum efficiency (fraction).

Flood application efficiency maximum column:  Optional - column in file for flood irrigation maximum efficiency (fraction).

Sprinkler application efficiency maximum column:  Optional - column in file for sprinkler maximum efficiency (fraction).

Acres irrigated by surface water only, flood column:  Optional - column in file for acres irrigated by surface water only, flood.

Acres irrigated by surface water only, sprinkler column:  Optional - column in file for acres irrigated by surface water only, sprinkler.

Acres irrigated by groundwater, flood column:  Optional - column in file for acres irrigated by groundwater, flood.

Acres irrigated by groundwater, sprinkler column:  Optional - column in file for acres irrigated by groundwater, sprinkler.

Total irrigated acres column:  Optional - column in file for total irrigated acres.

Pumping maximum column:  Optional - column in file for maximum monthly pumping (ACFT).

Groundwater mode column:  Optional - column in file for groundwater mode (see StateCU documentation).

```
SetIrrigationPracticeTSFromList (ListFile="Sp2008L_Eff.csv", ID="*", SetStart=1950, SetEnd=2006, IDCol="1", SurfaceDelEffMaxCol="3")
```

Add Working Directory
Cancel
OK

SetIrrigationPracticeTSFromList

### SetIrrigationPracticeTSFromList() Command Editor – Repeat Values



The following dialog is used to edit the command and illustrates the syntax of the command for providing acreage data that are not in HydroBase, for a single year of data.

**Edit SetIrrigationPracticeTSFromList() Command**
✕

This command sets irrigation practice time series data from a delimited list file, using the CU Location ID to look up the location. Resets will be enforced as the command is processed and can only apply to main locations, not aggregate/system parts. Use the ReadIrrigationPracticeTSFromList() command to read data that are not in HydroBase (e.g., parts of aggregates/systems). A comma-delimited list file is used to supply data, with values being set one of the following ways:

- 1) If the set start and end years are specified and a year column is not specified, the file data values are applied to each year in the set period.
- 2) If a year column is specified, year and corresponding values are read from the list file (the set period then controls how many years are processed).

The previous irrigation practice data will be reset to new values.  
 Blanks in column fields will result in no change to the data.  
 It is recommended that the location of the file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase

List file:	<input type="text" value="NoGIS_1998.csv"/>	<input type="button" value="Browse"/>
CU Location ID:	<input type="text" value="*"/>	Required - CU Location(s) to fill (use * for wildcard).
Set start (year):	<input type="text" value="1998"/>	Optional - starting year to set data.
Set end (year):	<input type="text" value="1998"/>	Optional - ending year to set data.
Year column:	<input type="button" value="v"/>	Optional - column in file for year.
CU location ID column:	<input type="button" value="1"/>	Optional - column in file for CU location ID.
Surface delivery maximum efficiency column:	<input type="button" value="v"/>	Optional - column in file for surface delivery maximum efficiency (fraction).
Flood application efficiency maximum column:	<input type="button" value="v"/>	Optional - column in file for flood irrigation maximum efficiency (fraction).
Sprinkler application efficiency maximum column:	<input type="button" value="v"/>	Optional - column in file for sprinkler maximum efficiency (fraction).
Acres irrigated by surface water only, flood column:	<input type="button" value="v"/>	Optional - column in file for acres irrigated by surface water only, flood.
Acres irrigated by surface water only, sprinkler column:	<input type="button" value="v"/>	Optional - column in file for acres irrigated by surface water only, sprinkler.
Acres irrigated by groundwater, flood column:	<input type="button" value="v"/>	Optional - column in file for acres irrigated by groundwater, flood.
Acres irrigated by groundwater, sprinkler column:	<input type="button" value="v"/>	Optional - column in file for acres irrigated by groundwater, sprinkler.
Total irrigated acres column:	<input type="button" value="3"/>	Optional - column in file for total irrigated acres.
Pumping maximum column:	<input type="button" value="v"/>	Optional - column in file for maximum monthly pumping (ACFT).
Groundwater mode column:	<input type="button" value="v"/>	Optional - column in file for groundwater mode (see StateCU documentation).

Command:

```
SetIrrigationPracticeTSFromList (ListFile="NoGIS_1998.csv"
, ID="*", SetStart=1998, SetEnd=1998, IDCol="1", AcresTotalCol
="3 ")
```

SetIrrigationPracticeTSFromList2

### SetIrrigationPracticeTSFromList() Command Editor – Provide Parcel Data not in HydroBase

The command syntax is as follows:

```
SetIrrigationPracticeTSFromList (Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
ListFile	Path to the delimited list file to read.	None – must be specified.
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
SetStart	The first year to set data values.	If not specified, data are set for the full output period.
SetEnd	The last year to set data values.	If not specified, data are set for the full output period.
YearCol	The column number (1+) containing the year for data.	The file values are applied to each year in the data set.
IDCol	The column number (1+) containing the CU Location identifiers. These values are matched against CU Location identifiers in the existing irrigation practice data.	None – must be specified.
SurfaceDel EffMaxCol	The column number (1+) containing the surface water delivery efficiency maximum.	If not specified, the previous data values will remain.
FloodApp EffMaxCol	The column number (1+) containing the flood application efficiency maximum.	If not specified, the previous data values will remain.
SprinklerApp EffMaxCol	The column number (1+) containing the sprinkler application efficiency maximum.	If not specified, the previous data values will remain.
AcresSWFloodCol	The column number (1+) containing the surface water flood acres.	If not specified, the previous data values will remain.
AcresSWSprinkler Col	The column number (1+) containing the surface water sprinkler acres.	If not specified, the previous data values will remain.
AcresGWFloodCol	The column number (1+) containing the groundwater flood acres.	If not specified, the previous data values will remain.
AcresGWSprinkler Col	The column number (1+) containing the groundwater sprinkler acres.	If not specified, the previous data values will remain.
AcresTotalCol	The column number (1+) containing the total acres.	If not specified, the previous data values will remain.
PumpingMaxCol	The column number (1+) containing the monthly maximum pumping rate.	If not specified, the previous data values will remain.
GWModeCol	The column number (1+) containing the groundwater mode value.	If not specified, the previous data values will remain.

Data file lines starting with the # character are treated as comments. If the first line's values are surrounded by double quotes, the line is assumed to indicate column headings. An example list file for setting efficiencies is shown below (the year would be provided as a parameter and values would apply to all years):

```

200505,0.70,0.7,0.8
200511,0.82,0.7,0.8
200512,0.71,0.7,0.8
200513,0.73,0.7,0.8
20MS01,0.80,0.7,0.8
200517,0.71,0.7,0.8
200518,0.88,0.7,0.8
200528,0.71,0.7,0.8
... etc. ...

```

An example list file for specifying acreage data (not in HydroBase) is shown below (the year column would be specified as a parameter and values would apply to the year in the list file). Currently, supplemental acreage data can have only a single irrigation method and supply type.

```
# The following data provide acreage for structures that did not have GIS data
# and consequently no data in HydroBase. The data are specific to 1998 and are
# used to set the CDS and IPY acres. The crop is used to provide CDS data. The
# irrigation method and source are used to provide IPY data.
"ID", "Crop", "Acres", "IrrigationMethod", "SupplySource"
200500, GRASS_PASTURE, 0, Flood, Surface
200506, GRASS_PASTURE, 100, Flood, Surface
200507, GRASS_PASTURE, 50, Flood, Surface
200508, GRASS_PASTURE, 40, Flood, Surface
200522, GRASS_PASTURE, 40, Flood, Surface
200523, GRASS_PASTURE, 50, Flood, Surface
200526, GRASS_PASTURE, 40, Flood, Surface
200529, GRASS_PASTURE, 5, Flood, Surface
200530, GRASS_PASTURE, 42, Flood, Surface
200532, GRASS_PASTURE, 25, Flood, Surface
200533, GRASS_PASTURE, 40, Flood, Surface
... etc...
```

This page is intentionally blank.

---

# Command Reference:

## setIrrigationPracticeTSMaxPumpingToRights()

**Set the irrigation practice max pumping time series (yearly) to well rights**

**StateCU Command**

Version 02.14.00, 2007-07-03, Color, Acrobat Distiller

THIS COMMAND IS OBSOLETE – INSTEAD, USE THE `setIrrigationPracticeTSPumpingMaxUsingWellRights()` COMMAND. This older command was used for Phase 4 Río Grande work, and only works with one year of parcel data (e.g., 1998). However, an entirely new procedure has now been implemented, which can be applied to all basins. The new procedure relies on processing water rights into a StateMod water rights file and then using this file as input when processing parcels for the irrigation practice time series. Other commands have also been implemented to allow more control over acreage processing.

The `setIrrigationPracticeTSMaxPumpingToRights()` command sets irrigation practice maximum well pumping time series (yearly) values to the water rights that were in effect at the time of the well, based on the appropriation date corresponding to water right administration numbers. The functionality of this command is similar to the `limitDiversionHistoricalTSMonthlyToRights()` command; however, the maximum pumping is simply set to the water rights. For each CU location being processed that has water supply from one or more wells, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights with administration numbers greater than or equal to 90000.00000 can be assigned an appropriate date, which is then used to compute an administration number for the check. The water rights can be supplied from a StateMod well rights file or from a list of rights in memory (e.g., as the result of the `setIrrigationPracticeTSFromHydroBase()` command). Water rights from a file may include the effects of set commands. For boundary purposes during the check, a zero flow condition is imposed at 1800-01-01 and carried forward until a right is found. A summary of the rights is printed to the log file.

If necessary, place set commands after the `setIrrigationPracticeTSMaxPumpingToRights()` command so that the set commands will not be impacted by the `setIrrigationPracticeTSMaxPumpingToRights()` command.

The water rights switch in the StateMod rights is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to limit the data).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to YYYY-01-01 during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - If the `UseOnOffDate` parameter is `True`, the appropriation date for the water right is set to (YYYY+1)-01-01 and the decree is set to negative during the limit process.
  - If the `UseOnOffDate` parameter is `False`, the appropriation date from the administration number is used and the decree is set to negative during the limit process.

If the administration number cannot be converted to an appropriation date, then the water right `OnOff` switch can be set to a year for each water right and `UseOnOffDate=True` should be specified.

If the sum of the water rights decrees is less than zero, it is reset to zero.

A summary of the logic is as follows:

For each CU location:

1. Determine the water rights for the CU location. If no rights are available, skip the remaining steps.
2. Determine the irrigation practice time series (yearly). If no time series is available, skip the remaining steps.
3. Process the water rights for the CU location.
  - a. Convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored. See the explanation above for how the water rights switch is handled.
  - b. Sort the rights according to the Julian day value for the appropriation date.
  - c. If the CU location has a free water right (those with administration numbers greater than or equal to 90000.00000): If the CU location has a senior water right, convert the free water right appropriation date to that of the senior water right (therefore the free water right is in effect since the time of the senior right). If the CU location has no senior water right (it has only free water right[s]), use the appropriation date corresponding to the `FreeWaterAppropriationDate` parameter described below.
  - d. Add a bounding zero decree for 1800-01-01 for the early period of the step function.
  - e. Generate a step function of sorted dates and decrees using the information described above. These values will be in CFS. Because appropriation dates are used, the sort order may be different from that of the numerical administration number.
  - f. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month, to determine a maximum pumping ACFT per month. Because of the conversion from CSFS to ACFT, monthly values in the step function will vary.
  - g. Using the monthly maximum values (January through December), determine the maximum monthly pumping for a year. The step-function will then use dates with a yearly precision because the value in the irrigation practice time series is the maximum monthly pumping in each year.
4. Set the yearly maximum pumping time series to the step function, where the step function is defined by a list of dates and decrees, determined from the previous step. The full period will be set.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit setIrrigationPracticeTSMMaxPumpingToRights() Command**

THIS COMMAND IS OBSOLETE AND IS USED ONLY FOR PHASE 4 RIO GRANDE WORK - INSTEAD, SEE THE setIrrigationPracticeTSPumpingMaxUsingWellRights() COMMAND.  
 This command sets irrigation practice maximum monthly pumping time series to water rights for each CU Location.  
 Water rights are specified by reading a StateMod well rights file, or use rights in memory from previous commands.  
 It is recommended that the location of the rights file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\java\_142\StateDMI

StateMod rights file:

CU location ID:  Specify the locations to process (use \* for wildcard)

Free water appropriation date:  Specify the appropriation date for admin numbers >= 90000

Use OnOff date?:  Get date from OnOff when YYYY, -YYYY (blank=False).

Number of days in month:  Use to normalize maximum pumping value.

Set flag:  1-character flag to use for reset values (optional).

Command:

setIrrigationPracticeTSMMaxPumpingToRights

### setIrrigationPracticeTSMMaxPumpingToRights() Command Editor

The command syntax is as follows:

```
setIrrigationPracticeTSMMaxPumpingToRights(param=value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the StateMod well rights file, surrounded by double quotes. The rights in the file are read and are used to set the CU location maximum pumping time series. The rights are assumed to be sorted by structure.  If in-memory rights resulting from the <code>readIrrigationPracticeTSFromHydroBase()</code> command are used (InputFile is blank), these rights may not exactly match those read from a StateMod well rights file. The rights file may include the effects of set commands.	Use StateMod well rights in memory, from previous commands.
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
FreeWater Appropriation Date	A date to be used for the free water rights found in the rights file. Free water rights are typically inserted to represent very junior rights. Rights having an administration number greater than or equal to 90000.00000 are assumed to be free water rights and will use the specified free water appropriation date when constraining the time series.	The date corresponding to an administration number of 0, which is Dec 31, 1849.
UseOnOffDate	If False, the appropriation date is always computed from the administration number. If True and the value of the OnOff switch for a right is YYYY or -YYYY, assign the appropriation date using the switch value (see notes earlier in the command description).	False
NumberOfDays InMonth	The number of days in a month. This is used when a constant value is needed.	Use the number of days in the month corresponding to the water right/permit date.
SetFlag	If specified as a single character, data flags will be enabled for the time series and each set value will be tagged with the specified character. The flag can then be used later to label graphs, etc. The flag will be appended to existing flags if necessary. This parameter is passed to the same features as used in the <code>limit*ToRights()</code> commands.	No flag is assigned.



---

# Command Reference: SetIrrigationPracticeTSPumpingMaxUsingWell Rights()

**Set the irrigation practice pumping maximum time series (yearly) to well rights**

## StateCU Command

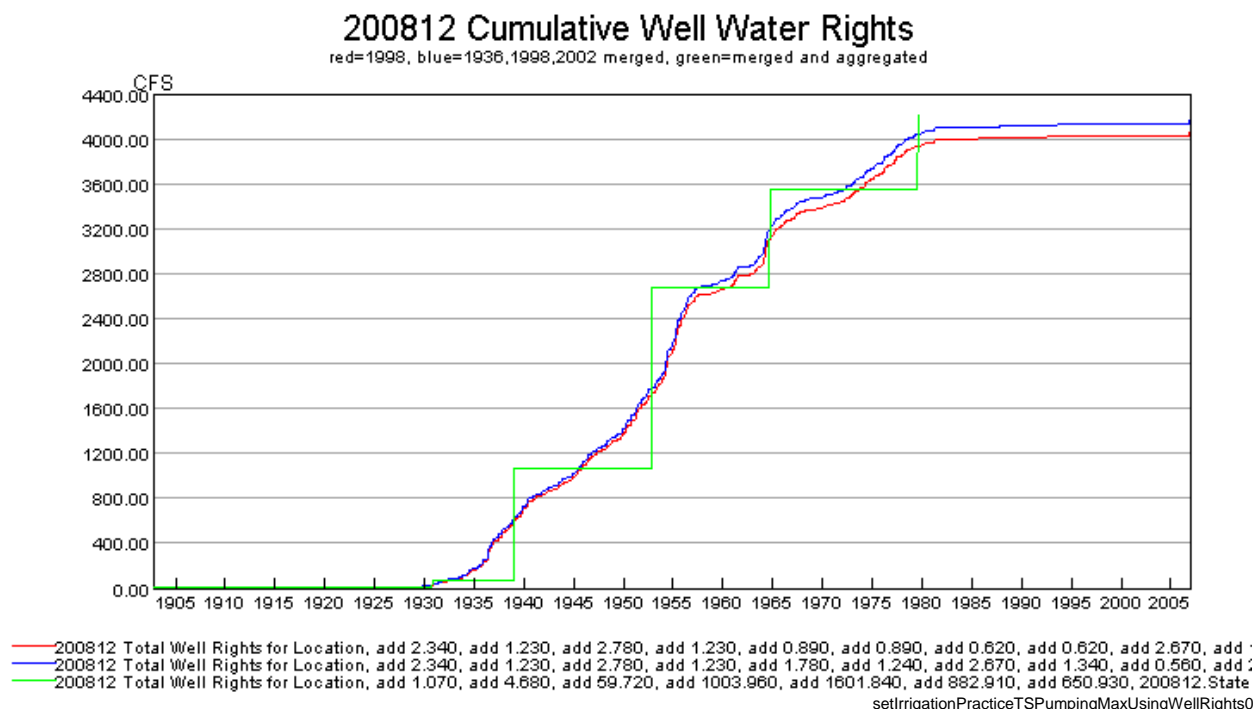
Version 3.09.01, 2010-02-01

The `SetIrrigationPracticeTSPumpingMaxUsingWellRights()` command sets irrigation practice well pumping maximum time series (yearly) values to the water rights that were in effect at each year in the period, based on the appropriation date corresponding to water right administration numbers. The functionality of this command is similar to the `LimitDiversionHistoricalTSMonthlyToRights()` command; however, the maximum pumping is simply set to the water rights. For each CU location being processed that has water supply from one or more wells, the cumulative rights are determined at each point in time, creating a step-function in CFS units. Very junior water rights are currently handled similar to other rights; however, a “free water” concept may be implemented in the future. The water rights are expected to have been processed with a previous command, for example `ReadWellRightsFromStateMod()`. In cases where multiple years of irrigated lands data are available, it is typical to have merged the water rights from multiple years using the `MergeWellRights()` command. Water rights from a file may include the effects of set commands. A zero flow condition is imposed at the start of the period (when no rights apply) and carried forward until a right is found.

The water rights on/off switch for each `StateMod` right is handled as follows:

- If the switch is zero, the water right is ignored in processing (it is not used to increment the decrees in the time series).
- If the switch is 1, no adjustments are done to the appropriation date for the water right.
- If the switch is +YYYY (indicating that the right should turn on in the given year):
  - If the switch is > the year from the appropriation date, set the right year to the switch. This ensures that the right is not turned on earlier than it was appropriated.
- If the switch is -YYYY (indicating that the right should turn off after the given year):
  - This case is not currently handled (the right is ignored as if the switch were zero) because standard procedures result in rights that are increasing over time. Additional enhancements are needed for this case, for example to ensure that the right is present with a positive switch in the early period.

An example of the resulting time series of decrees is shown in the following figure (this figure was generated by using TSTool to read three well rights files and graphing the rights at location 200812). Merged rights, since they represent more than one year of well/parcel matching, will typically result in slightly higher values. Aggregated rights will result in “blocky” decree time series.



A summary of the logic is as follows:

1. For each location, create a time series of decrees from the water rights, with the result having a monthly time step (since the pumping maximum is AF/M):
  - a. Determine the water rights for the CU location. If no rights are available, set the water right time series to zero and skip the remaining steps.
  - b. Initialize the decree time series to zero for the period.
  - c. For each right, convert the administration number to appropriation date. Use the same code as the Administration Number Calculator tool in StateView. The prior adjudication date associated with the administration number is ignored.
  - d. Check the on/off switch. See the explanation above for how the water rights switch is handled.
  - e. Add the decree value from the appropriation date (year and month) to the end of the output period.
2. Determine the irrigation practice pumping maximum time series:
  - a. If no irrigation practice time series is available, skip the remaining steps.
  - b. Loop through each month in the period and get the decree value from the step 1 above. Because the decrees are in CFS, convert to ACFT, considering the number of days in each month, to determine a maximum pumping ACFT per month:

$$Pumping = Decree * 1.9835 * NumberOfDaysInMonth$$

Because of the conversion from CSFS to ACFT, monthly values in the step function will vary unless the NumberOfDaysInMonth parameter is specified.

- c. Using the monthly maximum values (January through December), determine and set the maximum monthly pumping for a year. The step-function will then use dates with a yearly precision because the value in the irrigation practice time series is the maximum monthly pumping in each year.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetIrrigationPracticeTSPumpingMaxUsingWellRights() Command**

This command sets irrigation practice maximum monthly pumping time series to water rights for each CU Location. Water rights are specified by reading a StateMod well rights file with a previous command. The CU Location ID can contain a \* wildcard pattern to match one or more time series. The set period can optionally be specified. Only years in the output period can be set.

CU Location ID:  Required - CU locations to process (use \* for wildcard).

Include surface water supply?:  Optional - include locations with surface water supply? (default=true).

Include groundwater only supply?:  Optional - include locations with only groundwater supply? (default=true).

Set start (year):  Optional - start year as 4-digits or blank to set all.

Set end (year):  Optional - end year as 4-digits or blank to set all.

Free water administration number:  Optional - administration number >= which is free water (default=90000.00000).

Free water appropriation date:  Optional - appropriation date to use when free water (default=senior right at loc.)

Number of days in month:  Optional - use to convert CFS to AF/M (default=actual days/month).

Parcel data year:  Optional - 4-digit year for parcel data, if well rights are to be filtered.

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetIrrigationPracticeTSPumpingMaxUsingWellRights (ID="*", IncludeSurfaceWaterSupply=True, IncludeGroundwaterOnlySupply="True", NumberOfDaysInMonth=30.4)
```

OK Cancel

**SetIrrigationPracticeTSPumpingMaxUsingWellRights() Command Editor**

The command syntax is as follows:

```
SetIrrigationPracticeTSPumpingMaxUsingWellRights (Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single CU location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Include SurfaceWater Supply	Indicate whether locations with surface water supply should be included in processing. These locations are determined as being any other than groundwater only locations. This parameter is included to facilitate evaluating the overall approach.	True

Parameter	Description	Default
IncludeGroundwaterOnlySupply	Indicate whether locations with only groundwater supply should be included in processing. These locations are determined as being systems or aggregates specified by a list of parcels. This parameter is included to facilitate evaluating the overall approach.	True
SetStart	The starting year to set pumping maximum to water rights. This is typically blank.	OutputStart set by the SetOutputPeriod() command.
SetEnd	The ending year to set pumping maximum to water rights. This is typically blank.	OutputEnd set by the SetOutputPeriod() command.
FreeWaterMethod	<p>This parameter has not been added but may be added in the future, to control how “free water rights” (those with very junior administration numbers, such as 90000.00000) are handled. In general, free water rights should not apply to well rights and this parameter may never be implemented. Possible values are:</p> <ul style="list-style-type: none"> <li>AlwaysOn –Free water rights are always on for the full period.</li> <li>AsSpecified – use the administration number for the water right as specified. Typically this will result in the right only being in effect in the future and having no impact on the modeling period for this command.</li> <li>UseSeniorRightAppropriationDate – use the appropriation date for the senior water right for the location. Consequently, the water right is active for the full period that other water rights are active.</li> </ul>	AsSpecified
FreeWaterAdministrationNumber	<p>This parameter is currently not used since FreeWaterMethod=AsSpecified is the default. The administration number &gt;= to which the right is considered a “free water” right, typically 90000.00000 or higher.</p>	None.
FreeWaterAppropriationDate	<p>This parameter is currently not used since FreeWaterMethod=AsSpecified is the default. A date to be used for the free water rights found in the rights file, when no other date can be determined (e.g., no senior water for FreeWaterMethod=UseSeniorRightAppropriationDate).</p>	None.
NumberOfDaysInMonth	The number of days in a month when converting decree CFS to AF/M (acre-feet/month). This is used when a constant value is needed. For example, the StateCU model uses 30.4 days per month.	Use the number of days in the specific month.

Parameter	Description	Default
ParcelYear	The year of parcel/well matching data to use for water rights. This can be used if the StateMod well rights file was written with parcel year, for example with WriteWellRightsToStateMod(..., WriteDataComments=True,...). This is useful if evaluating the differences between rights determined with different years of parcel/well matching data, and rights from merged years.	Blank – use all water rights.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how to process the irrigation practice time series file where groundwater supply is used:

```
# Sp2008L_DDH.StateDMI
StartLog(LogFile="SP_IPY.log")
SetOutputPeriod(OutputStart="01/1950",OutputEnd="12/2006")
# Step 1 - Read CU Locations from list
ReadCULocationsFromList(ListFile="..\Sp2008L_StructList.csv",IDCol=1)
# Step 2 - Read SW aggregates, GW aggregates, and divsystems
#
SetDiversionAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn)
SetDiversionSystemFromList(ListFile="..\Sp2008L_DivSys_CDS.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
#
SetWellSystemFromList(ListFile="..\SP_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\SP_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="")
#
# Step 4 - Set conveyance efficiencies from file for key and sw aggregate structures - NOT in HydroBase
SetIrrigationPracticeTSFromList(ListFile="Sp2008L_Eff.csv",ID="",
    SetStart=1950,SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="3")
#
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
SetIrrigationPracticeTS(ID="",SetStart=1950,SetEnd=2006,FloodAppEffMax=.6,SprinklerAppEffMax=.8,GWMode=2)
#
# Step 6 - Read well rights file and Set Max pumping (use merged *.wer file)
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
SetIrrigationPracticeTSPumpingMaxUsingWellRights(ID="",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",NumberOfDaysInMonth=30.4)
# Step 7 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="",Div="1")
#
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="Sp2008L.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="")
#
```

```

# Step 9 - Estimate 1950 ground water acreage based on active wells as defined in the non-merged *.wer file
#
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L_NotMerged.wer",Append=False)
FillIrrigationPracticeTSAcreageUsingWellRights(ID="*",IncludeSurfaceWaterSupply=True,
    IncludeGroundwaterOnlySupply="True",FillStart=1950,FillEnd=1955,ParcelYear=1956)
#
# Step 10 - Fill Interpolate Acreage Type (SW and GW) 1956-2006
# Step 11a - estimate total GW and total SW
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-GroundWater",FillStart="1956",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-GroundWater",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-GroundWater",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-GroundWater",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 11b - set sprinkler to zero in early period
SetIrrigationPracticeTS(ID="*",SetStart=1950,SetEnd=1969,AcresSWSprinkler=0,AcresGWSprinkler=0)
#
# Step 11c - fill remaining irrigation method values
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1969",FillEnd="1976")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1976",FillEnd="1987")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="1987",FillEnd="2001")
FillIrrigationPracticeTSInterpolate(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2001",FillEnd="2005")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWaterSprinkler",FillStart="2005",FillEnd="2006",FillDirection="Forward")
#
# Step 12 - Set Acreage = 0 for structures that are in diversion systems, so acreage is not double counted
SetIrrigationPracticeTS(ID="0100503_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100507_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100687",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="0200834",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400511_D",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 13 - Set Acreage = 0, 1950-2006
SetIrrigationPracticeTS(ID="0100501",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100513",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
SetIrrigationPracticeTS(ID="0100829",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
SetIrrigationPracticeTS(ID="6400519",SetStart=1950,SetEnd=2006,AcresSWFlood=0,AcresSWSprinkler=0,
    AcresGWFlood=0,AcresGWSprinkler=0,PumpingMax=0,AcresTotal=0)
#
# Step 14 - Write final ipy file
#
WriteIrrigationPracticeTSToStateCU(OutputFile="Sp2008L.ipy",WriteHow=OverwriteFile)

```

---

# Command Reference: SetIrrigationPracticeTSSprinklerAcreageFrom List()

**Set irrigation practice time series sprinkler acreage time series values using a list file**

## StateCU Command

Version 3.09.01, 2010-02-01

The `SetIrrigationPracticeTSSprinklerAcreageFromList()` command sets irrigation practice time series sprinkler acreage data for a CU Location, using data from a list file, and adjusts other acreage terms accordingly to maintain the total acreage. This command is typically applied after all other data read and filling occurs, in order to utilize sprinkler acreage data that has been obtained for historical years. For example, the command is used in the Río Grande because user supplied sprinkler data are available, but may not be applied in the South Platte, where more years of irrigated lands data are in HydroBase. The list file typically contains sprinkler acreage by model location for the full period and may have been interpolated between observations and repeated on the ends of the period.

### Prerequisites:

1. This command should be executed after the irrigation practice time series are read from HydroBase (see `ReadIrrigationPracticeTSFromHydroBase()`).
2. Total acreage has been set to the crop pattern time series total (see `SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()`). The total acres are needed for checks.
3. The groundwater acreage should also have been filled using well rights before the first year of observations using `FillIrrigationPracticeTSAcreageUsingWellRights()`.
4. The surface water acreage should have been filled during the early period using `FillIrrigationPracticeTSInterpolate()`.
5. The end of the period should have acreage filled using `FillIrrigationPracticeTSRepeat()`.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetIrrigationPracticeTSSprinklerAcreageFromList() Command**

This command sets irrigation practice sprinkler acreage time series data from a list file, using the CU Location ID to look up the location. The command was developed for the case where sprinkler acreage information may be known in years between full irrigated land assessment. A comma-delimited list file is used to supply data. The previous irrigation practice data will be reset to new values and the acreage parts will be adjusted to match the total (see documentation). It is recommended that the location of the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase

List file:

CU Location ID:  Required - CU Location(s) to fill (use \* for wildcard).

Set start (year):  Optional - starting year to set data.

Set end (year):  Optional - ending year to set data.

CU location ID column:  Required - column in file for CU location ID.

Year column:  Optional - column in file for year.

Acres irrigated by sprinkler:  Optional - column in file for acres irrigated by sprinkler.

Command: 

```
SetIrrigationPracticeTSSprinklerAcreageFromList (ListFile="sprink_acreage_2007.csv", ID="*", YearCol=2, IDCol="1", AcresSprinklerCol="3")
```

SetIrrigationPracticeTSSprinklerAcreageFromList

### SetIrrigationPracticeTSSprinklerAcreageFromList() Command Editor

The sprinkler list file is processed one record at a time. The following check is done after setting the sprinkler acreage for a location and year:

1. Set the groundwater sprinkler acreage (GWsprinkler) to the minimum of the list file sprinkler acreage (ListFile) and the previous groundwater sprinkler value (GWprev).
2. Set the surface water sprinkler acreage (SWsprinkler) to  $\min((\text{ListFile} - \text{GWsprinkler}), \text{SWprev})$ . Then ensures that a negative number does not result.
3. Set the groundwater flood acreage (GWflood) to the previous groundwater total minus the groundwater sprinkler (GWsprinkler) acreage. This may result in a zero value based on previous adjustments.
4. Set the surface water flood acreage (SWflood) to the previous surface water total minus surface water sprinkler acres. This may result in a zero value based on previous adjustments.



The command syntax is as follows:

```
SetIrrigationPracticeTSSprinklerAcreageFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	Name of comma-delimited file containing sprinkler acreage for locations over time.	None – must be specified.
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
SetStart	The first year to set data.	If not specified, set the full period, using all available data from the list file.
SetEnd	The last year to set data.	If not specified, set the full period, using all available data from the list file.
IDCol	The column (1+) in the list file containing the location ID.	None – must be specified.
YearCol	The column (1+) in the list file containing the year.	None – must be specified.
AcresSprinklerCol	The column (1+) in the list file containing the sprinkler acres for the location and year.	None – must be specified.

This page is intentionally blank.

---

## Command Reference:

### SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()

**Set the irrigation practice time series total acreage to the crop pattern time series for each CU Location, and adjust irrigation practice acreage components as necessary**

#### StateCU Command

Version 3.09.01, 2010-02-01

The `SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage()` command sets the total acreage data in the irrigation practice time series (yearly) to the total for the crop pattern time series (yearly). The crop pattern time series should have been previously filled so that every year in the study period has observed or estimated values. Subsequent processing of irrigation practice acreage with other commands will ensure that acreage components (e.g., acres for irrigation sprinkler/flood method and ground/surface water source) add up to the total acres. To use this command, irrigation practice and crop pattern time series must be available in memory from previous commands (see the `ReadIrrigationPracticeTSFromHydroBase()` and `ReadCropPatternTSFromStateCU()` command). This command should be used after irrigation practice acreage time series are read from HydroBase, but before other filling of other acreage values occurs. This ensures that the total acreage controls when estimating acreage terms in the irrigation practice time series.

Currently, the command performs NO adjustments to any other acreage data in the irrigation practice time series. For this reason, this command should be used before any other irrigation practice acreage filling occurs. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage() Command**

This command sets irrigation practice total acreage time series values to the crop pattern total acreage time series values for each CU Location. The crop pattern time series should be read with a previous command. This command does not adjust other acreage values; however, the total will be used by other commands. The CU Location ID can contain a \* wildcard pattern to match one or more time series. The set period can optionally be specified. Only years in the output period can be set.

CU Location ID:  Required - locations to process (use \* for wildcard).

Set start (year):  Optional - start year as 4-digits (default=set all).

Set end (year):  Optional - end year as 4-digits (default=set all).

If not found:  Optional - indicate action if no match is found.

Command:

SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage

#### SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage() Command Editor

The command syntax is as follows:

```
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(
Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*). All matched locations in irrigation practice time series will be processed.	None – must be specified.
SetStart	The starting year to set the data.	Starting year set with SetOutputPeriod(), or the start of the time series.
SetEnd	The ending year to set the data.	Ending year set with SetOutputPeriod(), or the end of the time series.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how this command can be used:

```
# Step 1 - Set output period and read CU locations from structure file
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,NameCol=2,PartIDsCol=3,
    PartsListedHow=InRow)
# Step 3 - Create form for *.ipy file
CreateIrrigationPracticeTSForCULocations(ID="*")
# Step 5 - set max flood and surface water efficiencies and GWmode - NOT in HydroBase
# Set Max SW Eff = 1.0
SetIrrigationPracticeTS(ID="*",SurfaceDelEffMax=1.0,FloodAppEffMax=.60,
    SprinklerAppEffMax=.80,PumpingMax=0,GWMode=2)
SetIrrigationPracticeTSFromList(ListFile="cmstrlist.csv",ID="*",SetStart=1950,
    SetEnd=2006,IDCol="1",SurfaceDelEffMaxCol="7",FloodAppEffMaxCol="8",SprinklerAppEffMaxCol="9")
# Step 6 - Read category acreage from HydroBase
ReadIrrigationPracticeTSFromHydroBase(ID="*",Year="1993,2000",Div="5")
# Step 8 - Read total acreage from *.cds file and Set total for *.ipy file
ReadCropPatternTSFromStateCU(InputFile="..\StateCU\cm2006.cds")
SetIrrigationPracticeTSTotalAcreageToCropPatternTSTotalAcreage(ID="*")
# Step 9 - Fill all land use acreage
# Fill groundwater acreage first
# Fill surface water sprinkler and flood 1950-2006
# Fill ground water sprinkler and flood 1950-2006
# Step 9a - estimate total GW and total SW
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1950",FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="1993",FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
GroundWater",FillStart="2000",FillEnd="2006",FillDirection="Forward")
# Step 9b - fill remaining irrigation method values
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-SurfaceWaterOnlySprinkler",
    FillStart="1950",FillEnd="1993",FillDirection="Backward")
```

```
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="1993",
    FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-
SurfaceWaterOnlySprinkler",FillStart="2000",
    FillEnd="2006",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="1950",
    FillEnd="1993",FillDirection="Backward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="1993",
    FillEnd="1999",FillDirection="Forward")
FillIrrigationPracticeTSRepeat(ID="*",DataType="CropArea-GroundWaterSprinkler",FillStart="2000",
    FillEnd="2006",FillDirection="Forward")
# Step 10 - Write final ipy file
WriteIrrigationPracticeTSToStateCU(OutputFile="..\StateCU\cm2006.ipy")
# Check the results
CheckIrrigationPracticeTS(ID="*")
WriteCheckFile(OutputFile="cm2006.ipy.StateDMI.check.html")
```

This page is intentionally blank.

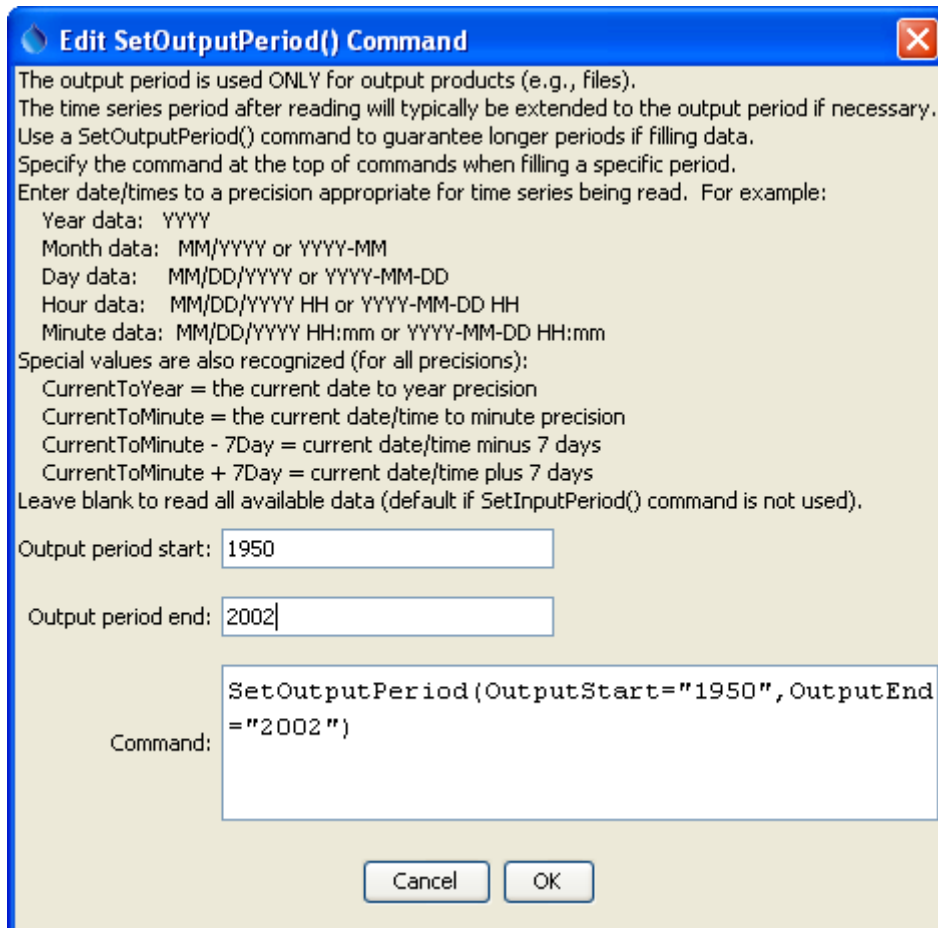
# Command Reference: SetOutputPeriod()

## Set the Output Period for Time Series

### General Command

Version 3.08.02, 2010-01-07

The `SetOutputPeriod()` command sets the output period for time series. Specifying the output period is necessary when creating model files or filling an extended period (time series will not automatically be extended by fill commands). The following dialog is used to edit this command and illustrates the syntax of the command. Note that the output period should always use calendar month and year, even if other than calendar year are used for output (see `SetOutputYearType()`).



**Edit SetOutputPeriod() Command**

The output period is used ONLY for output products (e.g., files).  
The time series period after reading will typically be extended to the output period if necessary.  
Use a `SetOutputPeriod()` command to guarantee longer periods if filling data.  
Specify the command at the top of commands when filling a specific period.  
Enter date/times to a precision appropriate for time series being read. For example:  
Year data: YYYY  
Month data: MM/YYYY or YYYY-MM  
Day data: MM/DD/YYYY or YYYY-MM-DD  
Hour data: MM/DD/YYYY HH or YYYY-MM-DD HH  
Minute data: MM/DD/YYYY HH:mm or YYYY-MM-DD HH:mm  
Special values are also recognized (for all precisions):  
CurrentToYear = the current date to year precision  
CurrentToMinute = the current date/time to minute precision  
CurrentToMinute - 7Day = current date/time minus 7 days  
CurrentToMinute + 7Day = current date/time plus 7 days  
Leave blank to read all available data (default if `SetInputPeriod()` command is not used).

Output period start:

Output period end:

Command: `SetOutputPeriod(OutputStart="1950", OutputEnd="2002")`

**SetOutputPeriod() Command Editor**

SetOutputPeriod

The command syntax is as follows:

```
SetOutputPeriod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputStart	The output period start, in a standard date/time format. The precision of this value should match that of data because it is used to iterate through the data. For example, if monthly data are being processed, specify the OutputStart using year and month.	None – must be specified.
OutputEnd	The output period end, in a standard date/time format. The precision of this value should match that of data because it is used to iterate through the data. For example, if monthly data are being processed, specify the OutputStart using year and month.	None – must be specified.



---

# Command Reference: SetOutputYearType()

## Set the output year type for time series and other data

### General Command

Version 3.08.02, 2010-01-07

The `SetOutputYearType()` command sets the output year type for time series and other time-dependent data (e.g., the order of monthly efficiencies in the StateMod diversion stations file depends on the year type). The output period used with `SetOutputPeriod()` should always use calendar month and year, even if other than calendar year are used for the output year type. The following dialog is used to edit this command and illustrates the syntax of the command.

**Edit SetOutputYearType() Command**

This command sets the global output year type, which is recognized by some output commands (e.g., for model and summary output). The following output year types are available:

- Calendar year (default): January to December.
- Water year: October (year - 1) to September (year) (e.g., water year 1970 is Oct 1969 to Sep 1970).
- NovToOct: November (year - 1) to October(year) (e.g., year 1970 is Nov 1969 to Oct 1970).

Output year type: Calendar Required - global output year type.

Command: `SetOutputYearType (OutputYearType=Calendar)`

Cancel OK

SetOutputYearType

### SetOutputYearType() Command

The command syntax is as follows:

```
SetOutputYearType( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
OutputYearType	The output year type, one of:  Calendar – each year is Jan – Dec. NovToOct – each year is Nov of the previous year to Oct of the current year Water – each water year is Oct of the previous year to Sep of the current year.	None – must be specified.

# Command Reference: SetPenmanMonteith()

## Set Penman-Monteith crop coefficients data

### StateCU Command

Version 3.10.00, 2010-04-02

The `SetPenmanMonteith()` command sets data in existing Penman-Monteith crop coefficients or adds a new crop type with crop coefficients. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetPenmanMonteith() Command

This command edits Penman-Monteith crop coefficient data for a specified crop type (name).  
For ALFALFA, specify 33 values (for 0,10,...,90,100 % of stage, repeat for 3 growth stages).  
For GRASS\_PASTURE, specify 11 values (for 0,10,...,90,100 % of stage, for 1 growth stage).  
For all other crops, specify 22 values (for 0,10,...,90,100 % of stage, repeat for 2 growth stages).  
If the crop type does not contain a \* wildcard pattern and does not match existing data,  
a new curve will be added if the "If not found" option is set to Add.  
Single values in a curve cannot be set - the entire curve must be set.

Crop Type:  Required - crop type (use \* for wildcard).

Coefficients: 

.100, .101, .102, .103, .104, .105, .106, .107  
.108, .109, .110, .200, .201, .202, .203, .204, .205, .206, .207, .208, .209, .210, .300, .301, .302, .303, .304, .305, .306, .307, .308, .309, .310

Required - separate by commas.

If not found:  Optional - action if no match is found (default=Warn).

Command: 

SetPenmanMonteith(CropType="ALFALFA",Coefficients=".100, .101, .102, .103, .104, .105, .106, .107, .108, .109, .110, .200, .201, .202, .203, .204, .205, .206, .207, .208, .209, .210, .300, .301, .302, .303, .304, .305, .306, .307, .308, .309, .310", IfNotFound=Add)

SetPenmanMonteith

### SetPenmanMonteith() Command Editor

The command syntax is as follows:

```
SetPenmanMonteith( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
CropType	A crop type to match or a pattern using wildcards (e.g., ALFALFA*).	None – must be specified.
Coefficients	A list of coefficients, surrounded by double quotes.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the crop if not found using the provided information</li> <li>• Fail – generate a failure message if the crop is not found</li> <li>• Ignore – ignore (don't add and don't generate a message) if the crop is not found</li> <li>• Warn – generate a warning message if the crop is not found</li> </ul>	Warn

# Command Reference: SetReservoirAggregate ()

## Set reservoir aggregate parts

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetReservoirAggregate()` command sets reservoir aggregate part identifier data for a reservoir. Aggregate reservoirs are defined as a combination of other reservoirs and may be used, for example, to aggregate stock ponds or other small reservoirs. This command should be specified before commands that need aggregate information during processing (e.g., those that read data from HydroBase). The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetReservoirAggregate() Command**

This command sets a Reservoir Aggregate location's information.  
Each Aggregate is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with `SetReservoirAggregate()` when water rights will be aggregated into classes.  
A "System" is used with `SetReservoirSystem()` when individual water rights will be maintained.  
For example, multiple nearby or related reservoirs may be grouped as a single identifier.  
When grouping reservoirs, specify the reservoir station IDs for parts.  
Separate the part IDs by spaces or commas.

Reservoir Aggregate ID:  Required - specify the Reservoir Aggregate ID.

Part IDs:  Required - up to 12 characters for each ID.

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
SetReservoirAggregate ( ID="24ARW01", PartIDs="243579,243580" )
```

SetReservoirAggregate

### SetReservoirAggregate() Command Editor

The command syntax is as follows:

```
SetReservoirAggregate (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The reservoir identifier to associate with the aggregate part identifiers.	None – must be specified.
PartIDs	The list of part identifiers to comprise the aggregate.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

# Command Reference: SetReservoirAggregateFromList()

Set reservoir aggregate parts from data in a list file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetReservoirAggregateFromList()` command reads reservoir aggregate part identifier data from a list file and saves the information for the reservoir. Aggregate reservoirs are defined as a combination of other reservoirs and may be used, for example, to aggregate stock ponds or other small reservoirs. Using a list file to define the aggregate allows the aggregate list to be shared between different commands files, minimizing errors. This command should be specified before commands that need aggregate information during processing (e.g., those that read data from HydroBase). The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetReservoirAggregateFromList() Command

This command sets a Reservoir Aggregate location's information from a list file.  
Each Aggregate is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with SetReservoirAggregateFromList() when water rights will be aggregated into classes.  
A "System" is used with SetReservoirSystemFromList() when individual water rights will be maintained.  
For example, multiple nearby or related reservoirs may be grouped as a single identifier.  
When grouping reservoirs, specify the reservoir station IDs for the parts in the list file.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetReservoirAggregate

List file: rg2007-aggres.csv

Browse

ID column: 1

Required - column for the Reservoir Aggregate IDs.

Name column:

Optional - column for the Reservoir Aggregate name.

Part IDs column: 2

Required - first/only column for the part IDs.

Parts listed how: InRow

Required - are part IDs listed in row or column?

Part IDs column (max):

Optional - maximum column for part IDs if in row (default is use all).

If not found:

Optional - indicate action if no ID match is found (default=Warn).

Command:

SetReservoirAggregateFromList (ListFile="rg2007-aggres.csv", IDCol=1, PartIDSCol=2, PartsListedHow=InRow)

Add Working Directory

Cancel

OK

SetReservoirAggregateFromList

**SetReservoirAggregateFromList() Command Editor**

The command syntax is as follows:

```
SetReservoirAggregateFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
IDCol	The column number (1+) containing the aggregate reservoir identifiers.	None – must be specified.
NameCol	The column number (1+) containing the aggregate reservoir name.	None – optional (name will be initialized to blank).
PartIDsCol	The column number (1+) for the first column having part identifiers.	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers.	Use all available columns.
PartsListedHow	If InRow, it is expected that all parts defining an aggregate are listed in the same row. If InColumn, it is expected that the parts defining an aggregate are listed one per row, with multiple rows defining the full aggregate (PartIDsColMax is ignored in this case).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

An example list file is shown below:

```
#
# Aggregate Reservoirs
20ARW01,203531,203533,203534,203535,203537,203540,203542,203543,203544,203545,203546
21ARW01,213584,213585,213586,213587,213589,217001
22ARW01,223301,223302,223303,223304,223305,223578,223580,223581,223583,223584
22ARW02,223575
24ARW01,243579,243580
25ARW01,250728,250729,250730,250731,253500,253501,253502,253503,253504,253505,253506
26ARW01,260721,260722,260723,260724,260725,263300,263581,263583,263584,263585,263586
27ARW01,273301,273303,273304,273305,273306,273307,273308,273309,273310,273311,273312
...
```



# Command Reference: SetReservoirRight()

## Set reservoir right data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetReservoirRight()` command sets data in existing reservoir rights or adds a new reservoir right. If a new right is added, it is added in alphabetical order according to the right identifier. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetReservoirRight() Command

This command sets (edits) data in reservoir right(s), using the reservoir right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
If the right ID does not contain a \* wildcard pattern and does not match an ID,  
the right will be added if the "If not found" option is set to Add.  
If the right ID does not contain a \* wildcard pattern and does match an ID,  
the right will be reset if the "If found" option is set to Set.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="503668.01"/>	Required - specify the right(s) to set (use * for wildcard).
Name:	<input type="text" value="WOLFORD_MOUNTAIN_RES"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text" value="ID"/>	Optional - station ID or "ID" to match first part of right ID.
Administration number:	<input type="text"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text"/>	Optional - decree amount, AF.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
Account served by right:	<input type="button" value="-2 - Fill first 2 accounts"/>	Optional - account(s) served by right.
Right type:	<input type="button" value="1 - Standard"/>	Optional - right type.
Fill type:	<input type="button" value="1 - First fill"/>	Optional - fill type.
Operational rightID:	<input type="text"/>	Optional - specify only if right type is -1
If not found:	<input type="button" value="Warn"/>	Optional - indicate action if no match is found (default=Warn).
If found:	<input type="button" value="Set"/>	Optional - indicate action if match is found (default=Warn).
Command:	<pre>SetReservoirRight (ID="503668.01",Name="WOLFORD_MOUNTAIN_RES" ,StationID="ID",OnOff=1,AccountDist="-2",RightType=1,FillType=1,IfNotFound=Warn,IfFound=Set)</pre>	

SetReservoirRight

### SetReservoirRight() Command Editor

The command syntax is as follows:

```
SetReservoirRight (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single reservoir right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching reservoir rights.	If not specified, the original value will remain.
StationID	The reservoir station identifier to be assigned for all matching reservoir rights.	If not specified, the original value will remain.
Administration Number	The administration number to be assigned for all matching reservoir rights.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching reservoir rights.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching reservoir rights, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
AccountDist	The account distribution option to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
RightType	The reservoir right type to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
FillType	The reservoir right fill type to be assigned for all matching reservoir rights (see StateMod documentation).	If not specified, the original value will remain.
OpRightID	The out-of-priority associated operational right (see StateMod documentation).	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the reservoir right if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn
IfFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Set – set the reservoir right if the ID is matched</li> <li>• Fail – generate a failure message if the ID is matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is matched</li> <li>• Warn – generate a warning message if the ID is matched</li> </ul>	Warn

---

# Command Reference: SetReservoirStation()

## Set reservoir station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetReservoirStation()` command sets data in existing reservoir stations or adds a new reservoir station. Because there are a large number of parameters, it may be desirable to use several commands for the same reservoir. Only one reservoir account can be assigned per command – an account identifier of 1 will clear all accounts before new accounts are defined. Bounding zero and high-end records are not automatically added for the content/area/seepage data – modelers must specify the bounds to prevent StateMod errors (the `FillReservoirStationsFromHydroBase()` command will provide bounding values).

The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetReservoirStation() Command	
<p>This command edits data in reservoir station(s), using the reservoir station ID to look up the location.            The reservoir station ID can contain a * wildcard pattern to match one or more locations.            If the reservoir station ID does not contain a * wildcard pattern and does not match an ID,            the location will be added if the "If not found" option is set to Add.            Use blanks in the any field to indicate no change to the existing value.            Only one account can be edited with each command (use additional commands for account 2+).</p>	
Reservoir station ID:	<input type="text" value="363543"/>
Name:	<input type="text"/>
River node ID:	<input type="text"/>
On/Off:	<input type="button" value="3 - On, do store above reservoir targets"/>
One fill rule:	<input type="button" value="4 - April"/>
Daily ID:	<input type="text" value="5"/>
Content (minimum):	<input type="text" value="0"/>
Content (maximum):	<input type="text" value="154645"/>
Release (maximum):	<input type="text" value="4010"/>
Dead storage:	<input type="text" value="0"/>
<p>Accounts (must use one command for each account - use multiple commands with only this section complete if necessary)</p>	
Account ID:	<input type="text" value="1"/>
Account name:	<input type="text" value="Hist_Users"/>
Maximum storage:	<input type="text" value="66000"/>
Initial storage:	<input type="text" value="0"/>
Evaporation distribution:	<input type="button" value="0 - Prorate evaporation based on current storage"/>
One fill rule calculation:	<input type="button" value="1 - Ownership is tied to first fill right(s)"/>
Evaporation stations:	<input type="text" value="10008,100"/>
Precipitation stations:	<input type="text"/>
Content/Area/Seepage:	<div style="border: 1px solid black; height: 100px;"></div>
If not found:	<input type="button" value="Warn"/>
Command:	<pre>SetReservoirStation( ID="363543", OnOff=3, OneFillRule=4, DailyID="5", ContentMin=0, ContentMax=154645, ReleaseMax=4010, DeadStorage=0, AccountID=1, AccountName="Hist_Users", AccountMax=66000, AccountInitial=0, AccountEvap=0, AccountOneFill=1, EvapStations="10008,100", IfNotFound=Warn)</pre>
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

SetReservoirStation

### SetReservoirStation() Command Editor

The command syntax is as follows:

```
SetReservoirStation(Parameter=Value,...)
```

**Command Parameters**

Parameter	Description	Default
ID	A single reservoir station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching reservoir stations.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching reservoir stations. Specify ID to assign to the reservoir station identifier.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching reservoir stations, either 1 for on or 0 for off.	If not specified, the original value will remain.
OneFillRule	The date for one fill rule administration (see the StateMod documentation) to be assigned for all matching reservoir stations.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching reservoir stations.	If not specified, the original value will remain.
ContentMin	The reservoir minimum content, ACFT.	If not specified, the original value will remain.
ContentMax	The reservoir maximum content, ACFT.	If not specified, the original value will remain.
ReleaseMax	The reservoir maximum release, CFS.	If not specified, the original value will remain.
DeadStorage	The reservoir dead storage, ACFT.	If not specified, the original value will remain.
AccountID	A reservoir account identifier, a number 1+. Reservoir accounts in the StateMod reservoir station are identified only by the account name. This AccountID lets the software know the order of the accounts. If the AccountID is specified as 1, all the accounts are deleted and a new list of accounts is started. Therefore, specify account information in sequential order.	Must be specified when providing account information.
AccountName	A reservoir account name.	If not specified, the original value will remain.
AccountMax	The account maximum content, ACFT.	If not specified, the original value will remain.
AccountInitial	The account initial content, ACFT.	If not specified, the original value will remain.

Parameter	Description	Default
AccountEvap	The account evaporation distribution – see the StateMod documentation.	If not specified, the original value will remain.
AccountOneFill	The account information for one fill calculations – see the StateMod documentation.	If not specified, the original value will remain.
EvapStations	A list of evaporation stations and weights (%) for the reservoir station, using the format: ID, % ; ID, %	If not specified, the original value will remain.
PrecipStations	A list of precipitation stations and weights (%) for the reservoir station, using the format: ID, % ; ID, %	If not specified, the original value will remain.
ContentAreaSeepage	Content/area/seepage values, using the format: Content, Area, Seepage; Content, Area, Seepage.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the reservoir station if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example illustrates how to set multiple accounts for one reservoir (note that more information is set in the first command whereas only account information is set in subsequent commands):

```
# GREEN MOUNTAIN RESERVIOR Characteristics
SetReservoirStation(ID="363543",OnOff=3,OneFillRule=4,DailyID="5",ContentMin=0,
  ContentMax=154645,ReleaseMax=4010,DeadStorage=0,AccountID=1,
  AccountName="Hist_Users",AccountMax=66000,AccountInitial=0,AccountEvap=0,
  AccountOneFill=1,EvapStations="10008,100",IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=2,AccountName="CBT_Pool",AccountMax=52000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=3,AccountName="Contract",AccountMax=20000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=4,AccountName="Silt_Proj",AccountMax=5000,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=5,AccountName="Inactive",AccountMax=11645,
  AccountInitial=0,AccountEvap=0,AccountOneFill=1,IfNotFound=Warn)
SetReservoirStation(ID="363543",AccountID=6,AccountName="SurplusFish",AccountMax=66000,
  AccountInitial=0,AccountEvap=0,IfNotFound=Warn)
```

# Command Reference: SetRiverNetworkNode()

## Set river network node data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetRiverNetworkNode()` command sets data in existing river network nodes or adds a new river network node. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetRiverNetworkNode() Command**

This command sets (edits) data in a river network node, using the river network node ID to look up the location. The river network node ID can contain a \* wildcard pattern to match one or more locations. If the river network node ID does not contain a \* wildcard pattern and does not match an ID, the river network node will be added if the "If not found" parameter is set to Add. Use blanks in the any field to indicate no change to the existing value.

ID:	<input type="text" value="CULEBRANF"/>	Required - river network node(s) to fill (use * for wildcard).
Name:	<input type="text" value="Culebra Natural Flow Upper"/>	Optional - up to 24 characters for StateMod.
Downstream river node ID:	<input type="text"/>	Optional - up to 12 characters.
Maximum recharge limit:	<input type="text"/>	Optional - maximum recharge limit (CFS) if modeling groundwater.
If not found:	<input type="button" value="Warn"/> ▼	Optional - indicate action if no match is found (default=Warn)
Command:	<pre>SetRiverNetworkNode (ID="CULEBRANF",Name="Culebra Natural Flow Upper",IfNotFound=Warn)</pre>	
<div><input type="button" value="OK"/> <input type="button" value="Cancel"/></div>		

SetRiverNetworkNode

### SetRiverNetworkNode() Command Editor

The command syntax is as follows:

```
SetRiverNetworkNode( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single river network node identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching river network nodes.	If not specified, the original value will remain.
DownstreamRiverNodeID	The downstream river node identifier to be assigned for all matching river network nodes.	If not specified, the original value will remain.
Comment	The comment to be assigned for all matching river network nodes.	If not specified, the original value will remain.
MaxRechargeLimit	The maximum recharge limit, CFS, for groundwater modeling, assigned for all matching river network nodes.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the river network node if the ID is not matched and is not a wildcard (<b>note that nodes that are upstream and downstream of the addition are NOT automatically changed</b>)</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



# Command Reference: SetStreamEstimateCoefficients()

Set stream estimate coefficients data

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetStreamEstimateCoefficients()` command sets data in existing stream estimate coefficients – the previous values will be overwritten. If base or gain data are specified, the original values will be replaced (not appended). The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetStreamEstimateCoefficients() Command**

This command sets (edits) data in stream estimate coefficients, using the station ID to look up the location. This command should be used after stream estimate coefficients are calculated. The station ID can contain a \* wildcard pattern to match one or more locations. If the station ID does not contain a \* wildcard pattern and does not match an ID, the station will be added if the "If not found" option is set to Add. Use blanks in the any field to indicate no change to the existing value.

Station ID:  Required - specify the station(s) to fill (use \* for wildcard)

Proration factor:  Optional - proration factor to set.

Base data:  Optional - base flow coeff, ID, ...repeat...

Gain data:  Optional - gain flow coeff, ID, ...repeat...

If not found:  Optional - action if no match is found.

Command: 

```
SetStreamEstimateCoefficients (ID="364512", ProrationFactor=1.000, IfNotFound=Warn)
```

SetStreamEstimateCoefficients

**SetStreamEstimateCoefficients() Command Editor**

The command syntax is as follows:

`SetStreamEstimateCoefficients (Parameter=Value, ...)`

## Command Parameters

Parameter	Description	Default
ID	A single stream estimate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Proration Factor	The proration factor for all matching stream estimate stations.	If not specified, the original value will

Parameter	Description	Default
		remain.
BaseData	The base flow coefficient and station ID pairs to be assigned for all matching stream estimate stations. Repeat for as many pairs as necessary, separated by commas.	If not specified, the original value will remain.
GainData	The gain flow coefficient and station ID pairs to be assigned for all matching stream estimate stations. Repeat for as many pairs as necessary, separated by commas.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the stream estimate coefficients if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how a StateMod stream estimate coefficients file can be created:

```

StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#          this baseflow nodes are generally on smaller non-gaged tribs and have
#          different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
...similar commands omitted...
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
...similar commands omitted...
#
# Step 5 - create streamflow estimate coefficient file
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")

```

---

# Command Reference:

## SetStreamEstimateCoefficientsPFGage()

**Set stream estimate coefficients to use a specific gage for proration factor calculations**

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `SetStreamEstimateCoefficientsPFGage()` command indicates that the proration factor for a specified station/node should be calculated using only the area\*precipitation value for the specified stream gage, rather than the next downstream node. The station/node is then treated as if it were a stream gage node for other natural flow calculations (as carried out by the `CalculateStreamEstimateCoefficients()` command). These commands should be specified before the `CalculateStreamEstimateCoefficients()` command.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetStreamEstimateCoefficientsPFGage() Command**

This command sets the a substitute gage for use when calculating the stream estimate coefficients proration factor. The data indicate stations (nodes) where the proration factor should be calculated using only the gage instead of the next downstream node. The node is then treated as if it were a gage for other stream estimate coefficient calculations. This command should be used before stream estimate coefficients are calculated with the `CalculateStreamEstimateCoefficients()` command. The information is saved in a separate list, not as a part of the stream estimate stations or coefficients data - not matching an ID will set the information by default. The station ID **should not** contain a \* wildcard pattern - provide a specific identifier.

Station ID:  Required - specific stream estimate station to match.

Gage ID:  Required - specific proration factor gage to use.

Command:

**SetStreamEstimateCoefficientsPFGage() Command Editor**

The command syntax is as follows:

`SetStreamEstimateCoefficientsPFGage (Parameter=Value, ...)`

### Command Parameters

Parameter	Description	Default
ID	A single stream estimate station identifier to match.	None – must be specified.
GageID	A stream gage station identifier to use, instead of the downstream gage.	None – must be specified.

The following command file illustrates how a StateMod stream estimate coefficients file can be created:

```

StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#           this baseflow nodes are generally on smaller non-gaged tribs and have
#           different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="360801",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="362002",GageID="09054000")
SetStreamEstimateCoefficientsPFGage(ID="360829",GageID="09047500")
..similar commands omitted...
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374641",ProrationFactor=0.200,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374648",ProrationFactor=0.350,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="380880",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="381594",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="384617",ProrationFactor=0.700,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510639",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514603",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514620",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510728",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530555",ProrationFactor=0.180,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530678",ProrationFactor=0.230,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="531082",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="954683",ProrationFactor=0.400,IfNotFound=Warn)
#
# Step 5 - create streamflow estimate coefficient file
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")

```

---

# Command Reference: SetStreamEstimateStation()

Set stream estimate station data

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetStreamEstimateStation()` command sets data in existing stream estimate stations or adds a new stream estimate station. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetStreamEstimateStation() Command**

This command sets (edits) data in stream estimate stations, using the station ID to look up the location. The station ID can contain a \* wildcard pattern to match one or more locations. If the station ID does not contain a \* wildcard pattern and does not match an ID, the station will be added if the "If not found" option is set to Add. Use blanks in the any field to indicate no change to the existing value.

Station ID:	<input type="text" value="CBP_Prod"/>	Required - specify the station(s) to fill (use * for wildcard)
Name:	<input type="text" value="CBPPRODUCTION"/>	Optional - up to 24 characters for StateMod.
River node ID:	<input type="text"/>	Optional.
Daily ID:	<input type="text" value="0"/>	Optional - corresponding daily data ID.
If not found:	<input type="button" value="v"/>	Optional - indicate action if no match is found.

Command:

```
SetStreamEstimateStation(ID="CBP_Prod",Name="CBPPRODUCTION",DailyID="0")
```

OK Cancel

SetStreamEstimateStation

**SetStreamEstimateStation() Command Editor**

The command syntax is as follows:

```
SetStreamEstimateStation(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single stream estimate station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching stream estimate stations.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching stream estimate stations.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching stream estimate stations.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the stream estimate station if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: SetStreamGageStation()

## Set stream gage station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetStreamGageStation ( )` command sets data in existing stream gage stations or adds a new stream gage station. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetStreamGageStation() Command**

This command sets (edits) data in stream gage stations, using the station ID to look up the location. The station ID can contain a \* wildcard pattern to match one or more locations. If the station ID does not contain a \* wildcard pattern and does not match an ID, the station will be added if the "If not found" option is set to Add. Use blanks in the any field to indicate no change to the existing value.

Station ID:  Required - specify the station(s) to fill (use \* for wildcard)

Name:  Optional - up to 24 characters for StateMod.

River node ID:  Optional.

Daily ID:  Optional - corresponding daily data ID.

If not found:  Optional - indicate action if no match is found.

Command:

SetStreamGageStation

### SetStreamGageStation() Command Editor

The command syntax is as follows:

```
SetStreamGageStation (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single stream gage station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching stream gage stations.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for all matching stream gage stations.	If not specified, the original value will remain.

Parameter	Description	Default
DailyID	The daily identifier to be assigned for all matching stream gage stations.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the stream gage station if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example command file illustrates the commands used to read stream gage stations from the network and create a StateMod file:

```

StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase,
#           fill in missing names from the network file
#
FillStreamGageStationsFromHydroBase(ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
...many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")

```



---

# Command Reference: SetWarningLevel()

## Set Level for Warning Messages

### General Command

Version 3.08.02, 2010-01-07

The `SetWarningLevel()` command is used to set warning levels for the screen and log file. The following dialog is used to edit this command and illustrates the command syntax. The default is warning level 1 to the screen and 2 to the log file.

**Edit SetWarningLevel() command**

Set the warning level for screen and/or log file warning messages.  
Setting the warning level to a higher number prints more warning information.  
Warning information is used for troubleshooting.  
Warning levels can be increased before and decreased after specific commands to troubleshoot the commands.

Screen warning level:  0=none, 100=all, blank=no change.

Log file warning level:  0=none, 100=all, blank=no change.

Command:

```
SetWarningLevel (ScreenLevel=1,LogFileLevel=2)
```

Cancel OK

SetWarningLevel

### SetWarningLevel() Command Editor

Warning messages are useful during troubleshooting. A general guideline is that a warning level of 1 prints important messages that a user should see, 2 prints warnings that by default are printed to the log file but are not displayed in the user interface, and 100 prints very low-level messages about input/output. Intermediate values will result in more or less output.

This command is useful for troubleshooting and can be specified multiple times to increase warning information for a specific command, if necessary.

This page is intentionally blank.

---

# Command Reference: SetWellAggregate ( )

## Set well aggregate parts

### StateCU and StateMod Command

Version 3.09.00, 2010-01-21

The `SetWellAggregate( )` command sets well aggregate part identifier data for a well (a CU Location that corresponds to a location with well supply, or StateMod well station). Well aggregates are specified using a list of part identifiers as follows:

- Part type is `Ditch` – the collection includes wells that are associated with a list of ditches, identified using ditch water district identifiers (WDIDs). The list of ditches is used for the full period.
- Part type is `Parcel` – the collection includes wells that are associated with a list of parcels. The division and year must be specified in the command because well to parcel relationships are determined for specific years.
- Part type is `Well` – the collection includes wells identified by well WDID (permit receipt number is not supported).

To facilitate processing, it is often best to use list files to specific aggregates (see `SetWellAggregateFromList( )`). Aggregates by convention have their water rights grouped into classes – to represent all water rights at a location, use a system (see the similar `System` commands). See also the **StateDMI Introduction** chapter, which provides additional information about aggregates and other modeling conventions. Aggregate information should be specified after well locations are defined and before their use in other processing, such as reading data from `HydroBase`.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellAggregate() Command**
✕

This command sets a Well Aggregate location's information.  
 Each Aggregate is a location where individual parts are combined into a single feature.  
 An "Aggregate" is used with SetWellAggregate() when water rights will be aggregated into classes.  
 A "System" is used with SetWellSystem() when individual water rights will be maintained.  
 For example, well-only parcels may be grouped as a single identifier.  
 Wells associated with ditches are grouped by specifying ditch identifiers and apply for the full period.  
 When grouping wells using parcels, specify parcel identifiers for the parts. Indicate the year and water division for the parcel data.  
 When grouping wells using well identifiers, specify a list of WDIDs or P:receipt for permits.  
 Separate the part IDs by spaces or commas.

Well Aggregate ID:	0102522_AuW	Required - specify the Well Aggregate ID.
Aggregate part type:	Ditch ▼	Required - the type of features being aggregated.
Year:		Required for part type + __command._Parcel + - year for the parcels.
Water division (Div):		Required for part type Parcel - water division for the parcels.
Part IDs:	108090, 108093, 108095, 108092, 0108091	Required - up to 12 characters for each ID.
If not found:	▼	Optional - indicate action if no ID match is found (default=Warn).
Command:	SetWellAggregate ( ID="0102522_AuW", PartType=Ditch, PartIDs="108090, 108093, 108095, 108092, 0108091" )	

OK
Cancel

SetWellAggregate

### SetWellAggregate() Command Editor

The command syntax is as follows:

```
SetWellAggregate(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	The well identifier to associate with the collection of individual wells.	None – must be specified.
PartType	Indicate the type of features being aggregated and specified by PartIDs, one of: <ul style="list-style-type: none"> <li>Ditch – the PartIDs (ditch WDIDs) indicate ditch service areas supplemented by wells.</li> <li>Parcel – the PartIDs (parcel numbers from GIS processing) indicate parcels irrigated by wells, with no surface water supply.</li> <li>Well – the PartIDs indicate wells (WDIDs), with no surface water supply.</li> </ul>	None – must be specified.
Year	The year defining the parcels.	Required when PartType is Parcel because parcel identifiers from well matching are specific to the data year.
Div	Water division for the parcels in the aggregate.	Required when PartType is Parcel because parcels require the division.
PartIDs	The list of part identifiers to comprise the aggregate. See the PartType description above.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

This page is intentionally blank.

---

# Command Reference: SetWellAggregateFromList()

Set well aggregate parts from data in a list file

**StateCU and StateMod Command**

Version 3.09.00, 2010-01-21

The `SetWellAggregateFromList()` command sets well aggregate part identifier data for a well (a CU Location that corresponds to a location with well supply, or StateMod well station). Well aggregates are specified using a list of part identifiers as follows:

- Part type is `Ditch` – the collection includes wells that are associated with a list of ditches, identified using ditch water district identifiers (WDIDs). The list of ditches is used for the full period.
- Part type is `Parcel` – the collection includes wells that are associated with a list of parcels. The division and year must be specified in the command because well to parcel relationships are determined for specific years.
- Part type is `Well` – the collection includes wells identified by well WDID (permit receipt number is not supported).

To facilitate processing, the list of parts is specified in a delimited list file. Aggregates by convention have their water rights grouped into classes – to represent all water rights at a location, use a system (see the similar `System` commands). See also the StateDMI **Introduction** chapter, which provides additional information about aggregates and other modeling conventions. Aggregate information should be specified after well locations are defined and before their use in other processing, such as reading data from HydroBase.

The `SetWellSystemFromList()` command is often used instead of the `SetWellAggregateFromList()` command if specific well rights are referred in augmentation plans (therefore the examples shown below are contrived).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellAggregateFromList() Command**

This command sets a Well Aggregate location's information from a list file.  
 This command can be used with StateMod Well stations and StateCU CU Locations.  
 Each Aggregate is a location where individual parts are combined into a single feature.  
 An "Aggregate" is used with SetWellAggregateFromList() when water rights will be aggregated into classes.  
 A "System" is used with SetWellSystemFromList() when individual water rights will be maintained.  
 For example, well-only parcels may be grouped as a single identifier.  
 Wells associated with ditches are grouped by specifying ditch identifiers for the parts.  
 When grouping wells using parcels, specify parcel identifiers for the parts and indicate the year and water division for the parcel data.  
 When grouping wells using well identifiers, specify a list of well WDIDs or P:receipt for permits.  
 Columns should be delimited by commas.  
 It is recommended that the location of the file be specified using a path relative to the working directory.  
 The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetWellAggregateFromList

List file:

Aggregate part type:  Required - the type of features being aggregated.

Year:  Required if part type is Ditch or Parcel - year for the parcels.

Water division (Div):  Required if part type is Ditch or Parcel - water division for the parcels.

ID column:  Required - column for the Well Aggregate IDs.

Name column:  Optional - column for the Well Aggregate name.

Part IDs column:  Required - first/only column for the part IDs.

Parts listed how:  Required - are part IDs listed in row or column?

Part IDs column (max):  Optional - maximum column for part IDs if in row (default is use all).

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv", PartType=Ditch, IDCol=1, NameCol=2, PartIDsCol=3, PartsListedHow=InColumn)
```

SetWellAggregateFromList

**SetWellAggregateFromList() Command Editor**

The command syntax is as follows:

```
SetWellAggregateFromList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
PartType	Indicate the type of features being aggregated and specified by PartIDs, one of: <ul style="list-style-type: none"> <li>Ditch – the PartIDs (ditch WDIDs) indicate ditch service areas supplemented by wells.</li> </ul>	None – must be specified.



Parameter	Description	Default
	<ul style="list-style-type: none"> <li>Parcel – the PartIDs (parcel numbers from GIS processing) indicate parcels irrigated by wells, with no surface water supply.</li> <li>Well – the PartIDs indicate wells (WDIDs), with no surface water supply.</li> </ul>	
Year	The year defining the parcels.	Required when PartType is Parcel because parcel identifiers from well matching are specific to the data year.
Div	Water division for the parcels in the aggregate.	Required when PartType is Parcel because parcels require the division.
IDCol	The column number (1+) containing the aggregate well identifiers.	None – must be specified.
NameCol	The column number (1+) containing the aggregate well name.	None – optional (name will remain as before).
PartIDsCol	The column number (1+) for the first column having part identifiers.	None – must be specified.
PartsListedHow	If InRow, it is expected that all parts defining an aggregate are listed in the same row. If InColumn, it is expected that the parts defining an aggregate are listed one per row, with multiple rows defining the full aggregate (PartIDsColMax is ignored in this case).	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers. Use if extra columns on the right need to be excluded from the list.	Use all available non-blank columns starting with PartIDsCol.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the aggregate identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the aggregate identifier is not matched</li> <li>Warn – generate a warning message if the aggregate identifier is not matched</li> </ul>	Warn

The following example illustrates a list file is used with PartType=Parcel and PartsListedHow=InColumn:

```
"UZONES" , "PARCEL"  
20URF0,16831  
20URF0,16832  
20URF0,16834  
...  
20URF0,18606  
20URF24,10295  
20URF24,10318  
...
```

The following example illustrates a list file is used with PartType=Ditch and PartsListedHow=InColumn, with the name being provided in column 2:

```
# Aggregate_ID/Agg_Name/WDID  
01_ADP037,South Platte River below Kersey Co North 2,0100643  
01_ADP037,South Platte River below Kersey Co North 2,0100644  
01_ADP037,South Platte River below Kersey Co North 2,0100835  
01_ADP037,South Platte River below Kersey Co North 2,0104486
```

# Command Reference: SetWellDemandTSMonthly()

## Set well demand time series (monthly) data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetWellDemandTSMonthly()` command sets the well demand time series (monthly) for a specific well, by reading another time series. If data already exist, the previous time series is discarded. The period of the time series is set to the output period. This command is useful if data cannot be calculated in an automated fashion (e.g., municipal demands may need to be specified manually). The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetWellDemandTSMonthly() Command

This command sets a well demand time series (monthly) by reading the data from a file or HydroBase. The well station identifier is used to match the time series that is read. Time series identifiers follow the conventions used by TSTool and other CDSS software. For example, for a StateMod file:  
ID..WellDem.Month~StateMod~..\path\to\file  
For example, for a DateValue file:  
ID..WellDem.Month~DateValue~..\path\to\file  
For example, for HydroBase (use TSTool to determine the identifier):  
ID.DWR.DivTotal.Month~HydroBase  
It is recommended files be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork  
If the period that is read is shorter than the output period, it is extended to the output period with missing data.

Well station ID:  Required - stations to process.

Time series ID:

<= zero values in average?:  Optional - are values <= zero used in averages? (default=True; used later in filling)

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetWellDemandTSMonthly ( ID="20ANWR", TSID="20ANWR..DivTotal.MONTH~StateMod~alamCBT.stm", IfNotFound=Add)
```

SetWellDemandTSMonthly

### SetWellDemandTSMonthly() Command Editor

The command syntax is as follows:

```
SetWellDemandTSMonthly( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
TSID	The full time series identifier, which is used to locate and read the time series. Currently time series from StateMod and DateValue files, and HydroBase are recognized. See the TSTool input type appendices for the formats of these files. Other input types can be enabled if necessary.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing historical averages.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference:

## SetWellDemandTSMonthlyConstant()

Set well demand time series (monthly) data to a constant value

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SetWellDemandTSMonthlyConstant()` command sets well demand time series (monthly) data to a constant value. The output period can be set or will default to that defined by the most recent `SetOutputPeriod()` command. If a matching time series is not found, it can be added to the list of time series (at the end). The values that are set are treated the same as observations from HydroBase. To ensure that set values remain, use the `SetWellDemandTSMonthlyConstant()` command after other commands that may modify the time series.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellDemandTSMonthlyConstant() Command**

This command sets monthly well demand time series data to a constant.  
The original data limits are recomputed as if the data are historical data. The time series will be created if it does not exist and IfNotFound=Add.

Well station ID:  Required - identifier of station to process.

Constant:  Required - constant value to use.

Set start:  Optional - start date (default=output period).

Set end:  Optional - end date (default=output period).

Recalculate limits:  Recalculate original data limits after set (default=True).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetWellDemandTSMonthlyConstant ( ID="2008001" )
```

SetWellDemandTSMonthlyConstant

**SetWellDemandTSMonthlyConstant() Command Editor**

The command syntax is as follows:

```
SetWellDemandTSMonthlyConstant(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Constant	A constant well demand value.	None – must be specified.
SetStart	The start of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period start.
SetEnd	The end of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period end.
RecalcLimits	If True, then the constant values will be treated as observations and the historical averages will be recalculated with the values. False will result in the time series being set but the previous averages remaining. The averages are used with fill commands.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the time series if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

---

# Command Reference: SetWellHistoricalPumpingTSMonthly()

Set well historical pumping time series (monthly) data

StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The SetWellHistoricalPumpingTSMonthly() command sets the well historical pumping time series (monthly) for a specific well, by reading another time series. If data already exist, the previous time series is discarded. The period of the time series is set to the output period. This command is useful if data cannot be calculated in an automated fashion (e.g., municipal pumping may need to be specified manually). The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellHistoricalPumpingTSMonthly() Command**

This command sets a well historical pumping time series (monthly) by reading the data from a file or HydroBase.  
The well station identifier is used to match the time series that is read.  
Time series identifiers follow the conventions used by TSTool and other CDSS software.  
For example, for a StateMod file:  
ID..PumpingHist.Month~StateMod~..\path\to\file  
For example, for a DateValue file:  
ID..PumpingHist.Month~DateValue~..\path\to\file  
For example, for HydroBase (use TSTool to determine the identifier):  
ID.DWR.DivTotal.Month~HydroBase  
It is recommended files be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellHistoricalPumpingTSMonthlyFromStateMod  
If the period that is read is shorter than the output period, it is extended to the output period with missing data.

Well station ID:  Required - stations to process.

Time series ID:

<= zero values in average?:  Optional - are values <= zero used in averages? (default=True; used later in filling)

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetWellHistoricalPumpingTSMonthly(ID="20DELNOR", TSID="20DELNOR..PumpingHist.MONTH~StateMod~..\Tsfiles_2007\20DELNOR.stm", IfNotFound=Add)
```

SetWellHistoricalPumpingTSMonthly

**SetWellHistoricalPumpingTSMonthly() Command Editor**

The command syntax is as follows:

```
SetWellHistoricalPumpingTSMonthly( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
TSID	The full time series identifier, which is used to locate and read the time series. Currently time series from StateMod and DateValue files, and HydroBase are recognized. See the TSTool input type appendices for the formats of these files. Other input types can be enabled if necessary.	None – must be specified.
LEZeroInAverage	Indicates whether values $\leq 0$ should be considered when computing historical averages.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Add – add the time series if the ID is not matched and is not a wildcard</li> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn



---

# Command Reference: SetWellHistoricalPumpingTSMonthlyConstant()

Set well historical pumping time series (monthly) data to a constant value

## StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The `SetWellHistoricalPumpingTSMonthlyConstant()` command sets well historical pumping time series (monthly) data to a constant value. The output period can be set or will default to that defined by the most recent `SetOutputPeriod()` command. If a matching time series is not found, it can be added to the list of time series (at the end).

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellHistoricalPumpingTSMonthlyConstant() Command**

This command sets monthly well historical pumping time series data to a constant.  
The original data limits are recomputed as if the data are historical data. The time series will be created if it does not exist and IfNotFound=Add.

Well station ID:  Required - identifier of station to process.

Constant:  Required - constant value to use.

Set start:  Optional - start date (default=output period).

Set end:  Optional - end date (default=output period).

Recalculate limits:  Recalculate original data limits after set (default=True).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetWellHistoricalPumpingTSMonthlyConstant ( ID="Well1", IfNotFound=Add)
```

SetWellHistoricalPumpingTSMonthlyConstant

## SetWellHistoricalPumpingTSMonthlyConstant() Command Editor

The command syntax is as follows:

```
SetWellHistoricalPumpingTSMonthlyConstant( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Constant	A constant historical value to set.	None – must be specified.
SetStart	The start of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period start.
SetEnd	The end of the period for the set, in a standard date/time format for monthly data (e.g., YYYY-MM or MM/YYYY).	The output period end.
RecalcLimits	If True, then the time series limits will be recalculated as if the provided values are observations. If False, the limits from before the set will remain.	True
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the time series if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

# Command Reference: SetWellRight()

## Set well right data

### StateCU and StateMod Command

Version 3.09.00, 2010-01-28

The `SetWellRight()` command sets data in existing well rights or adds a new well right. If a new right is added, it is added in alphabetical order according to the right identifier. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetWellRight() Command

This command sets (edits) data in well right(s), using the well right ID to look up the right.  
The right ID can contain a \* wildcard pattern to match one or more rights.  
If the right ID does not contain a \* wildcard pattern and does not match an ID,  
the right will be added if the "If not found" option is set to Add.  
If the right ID does not contain a \* wildcard pattern and does match an ID,  
the right will be reset if the "If found" option is set to Set.  
Use blanks in the any field to indicate no change to the existing value.

Right ID:	<input type="text" value="20MS06W.99"/>	Required - specify the right(s) to set (use * for wildcard).
Name:	<input type="text" value="Mumm_Well"/>	Optional - up to 24 characters for StateMod.
Station ID:	<input type="text" value="20MS06"/>	Optional - station ID or "ID" to match first part of right ID.
Administration number:	<input type="text" value="90000.00000"/>	Optional - administration number (priority, smaller is more senior).
Decree amount:	<input type="text" value="6.38"/>	Optional - decree amount, CFS.
On/Off:	<input type="button" value="1 - On"/>	Optional - indicate on/off, YYYY to start, -YYYY to end in year.
If not found:	<input type="button" value="Add"/>	Optional - indicate action if no match is found (default=Warn).
If found:	<input type="button" value="Warn"/>	Optional - indicate action if match is found (default=Warn).
Command:	<pre>SetWellRight (ID="20MS06W.99",Name="Mumm_Well",StationID="20MS06",AdministrationNumber=90000.00000,Decree=6.38,OnOff=1,IfNotFound=Add,IfFound=Warn)</pre>	

SetWellRight

SetWellRight() Command Editor

The command syntax is as follows:

```
SetWellRight (Parameter=Value...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well right identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching well rights.	If not specified, the original value will remain.
StationID	The well station identifier to be assigned for all matching well rights.	If not specified, the original value will remain.
AdministrationNumber	The administration number to be assigned for all matching well rights.	If not specified, the original value will remain.
Decree	The water right decree to be assigned for all matching well rights.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching well rights, either 1 for on or 0 for off, a positive 4-digit year to turn the right on starting in the year, or a negative 4-digit year to turn the right off starting in the year.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the water right if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn
IfFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Set – set the water right data</li> <li>• Fail – generate a failure message if the ID is matched</li> <li>• Ignore – ignore (don't set and don't generate a message) if the ID is matched</li> <li>• Warn – generate a warning message if the ID is matched</li> </ul>	Warn

# Command Reference: SetWellStation()

## Set well station data

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetWellStation()` command sets data in existing well stations or adds a new well station. The following dialog is used to edit the command and illustrates the syntax of the command.

Edit SetWellStation() Command

This command sets (edits) data in well station(s), using the well station ID to look up the location.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
If the station ID does not contain a \* wildcard pattern and does not match an ID, the location will be added if the "If not found" parameter is set to Add.  
Use blanks in the any field to indicate no change to the existing value.  
Monthly efficiencies should be separated by commas, with January first.  
Returns and depletions should be specified as triplets of location, percent, and return table ID:  
08123456,50.0,1;08234567,50.0,2

Well station ID:	0102522_AuW	Required - ID for stations to fill (use * for wildcard)
Name:	RIVERSIDE Aug Well	Optional - up to 24 characters for StateMod.
River node ID:	0102522_AuW	Optional - the river node identifier, or "ID" to use the station ID.
On/Off:	<input type="button" value="v"/>	Optional - is station on/off in data set?
Capacity:	999.	Optional - well capacity, CFS.
Daily ID:	4	Optional - the daily identifier, "ID", or StateMod flag).
Admin Num. Shift:	0 - Use water right priorities	Optional - shift for well station right administration number.
Diversion station ID:	NA	Optional - diversion station this well supplements (use "ID" to use the well station ID).
Demand type:	1 - Monthly total demand	Optional - monthly demand time series type.
Irrigated acres:	0.0	Optional - typically for the most recent year.
Use type:	5 - Other	Optional - water use type.
Demand source:	8 - Irrigated acres provided by user	Optional - water demand source.
Efficiency (Annual):	100.0	Optional - annual efficiency, percent (ignore if setting monthly).
Efficiencies (Monthly):		Optional - percent, annual is recomputed as average.
Returns (optional):	06759910,100.0,1	
Depletions (optional):	06759910,100.0,2	
If not found:	Add	Optional - indicate action if no match is found.
Command:	<pre>SetWellStation(ID="0102522_AuW",Name="RIVERSIDE Aug Well",RiverNodeID="0102522_AuW",Capacity=999.,DailyID="4",AdminNumShift=0,DiversionID="NA",DemandType=1,IrrigatedAcres=0.0,UseType=5,DemandSource=8,EffAnnual=100.0&gt;Returns="06759910,100.0,1",Depletions="06759910,100.0,2",IfNotFound=Add)</pre>	

SetWellStation

### SetWellStation() Command Editor

The command syntax is as follows:

```
SetWellStation(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
Name	The name to be assigned for all matching well stations.	If not specified, the original value will remain.
RiverNodeID	The river node identifier to be assigned for matching well stations. Specify ID to set to the well station identifier.	If not specified, the original value will remain.
OnOff	The on/off switch value to be assigned for all matching well stations, either 1 for on or 0 for off.	If not specified, the original value will remain.
Capacity	The well station capacity, CFS.	If not specified, the original value will remain.
DailyID	The daily identifier to be assigned for all matching well stations. Specify ID to set to the well station identifier.	If not specified, the original value will remain.
AdminNumShift	For all matching well stations, a shift to be applied to the administration number for well rights. See the “primary” flag in the StateMod well station documentation.	If not specified, the original value will remain.
DiversionID	For all matching well stations, the diversion station identifier associated with the well station. Typically, where well water supplements surface supply, one well station is assigned to the diversion station. Specify ID to assign to the well station identifier.	If not specified, the original value will remain.
DemandType	The demand type to be assigned for all matching well stations (see StateMod documentation).	If not specified, the original value will remain.
IrrigatedAcres	The irrigated acres to be assigned for all matching well stations.	If not specified, the original value will remain.
UseType	The use type to be assigned for all matching well stations (see StateMod documentation).	If not specified, the original value will remain.
DemandSource	The demand source to be assigned for all matching well stations (see StateMod documentation).	If not specified, the original value will remain.
EffAnnual	The annual efficiency (percent, 0 - 100) to be assigned for matching well stations. Monthly efficiencies will be set to the same value (but not used).	If not specified, the original value will remain.

Parameter	Description	Default
EffMonthly	The monthly efficiencies (percent, 0 – 100) to be assigned for all matching well stations, specified as 12 comma-separated values, January to December. The annual efficiency will be set to the average value. The order of the values in the output file will be according to the output year type set by <code>setOutputYearType()</code> , or calendar by default.	If not specified, the original value will remain.
Returns	The return flows to be assigned for all matching well stations. Specify as <code>StationID,Percent,DelayTableID; StationID,Percent,DelayTableID</code> etc.	If not specified, the original value will remain.
Depletions	The depletions to be assigned for all matching well stations. Specify as <code>StationID,Percent,DelayTableID; StationID,Percent,DelayTableID</code> etc.	If not specified, the original value will remain.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Add – add the well station if the ID is not matched and is not a wildcard</li> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

This page is intentionally blank.



---

# Command Reference: SetWellStationAreaToCropPatternTS ( )

**Set the well station area to the crop pattern time series maximum**

## StateMod Command

Version 3.09.01, 2010-02-01

The SetWellStationAreaToCropPatternTS ( ) command sets the well station area for each well station to the maximum crop pattern time series total area. The crop pattern time series must have been read or assigned with previous commands. If there is no crop pattern time series, the area will not be set.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationAreaToCropPatternTS() Command**

This command sets the well station area to the maximum irrigated area from the crop pattern time series. Crop pattern time series are specified by reading a StateCU crop pattern time series file with a previous command. The well station ID can contain a \* wildcard pattern to match one or more locations.

Well station ID: \* Required - well stations to process (use \* for wildcard).

If not found: Optional - indicate action if no match is found (default=Warn).

Command: SetWellStationAreaToCropPatternTS ( ID="\*" )

OK Cancel

SetWellStationAreaToCropPatternTS

### SetWellStationAreaToCropPatternTS() Command Editor

The command syntax is as follows:

```
SetWellStationAreaToCropPatternTS (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20 *).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file excerpt illustrates how the crop pattern time series can be used to set the irrigated area for the well stations:

```
ReadCropPatternTSFromStateCU (InputFile="..\Crops\Sp2008L.cds")
SetWellStationAreaToCropPatternTS (ID="*")
```

---

# Command Reference: SetWellStationCapacitiesFromTS()

**Set well station capacity data as maximum historical pumping**

**StateCU and StateMod Command**

Version 3.09.01, 2010-01-27

The SetWellStationCapacitiesFromTS() command sets well station capacities to the maximum historical pumping time series (monthly) value. The historical time series must have been previously read or calculated with other commands. Monthly ACFT values are converted to CFS units by applying the conversion:

$$\text{CFS} = \text{X ACFT} / (1.9835 * \text{DaysInMonth})$$

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationCapacitiesFromTS() Command**

This command sets the well station capacity to the largest value in the well historical pumping time series (monthly). This is necessary because the initial capacity value may be too small and will be a constraint in the simulation. The monthly ACFT value is converted to CFS using the number of days in the month. The capacity is reset only if the time series value is larger than the existing capacity.

Well station ID:  Required - stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:  

```
SetWellStationCapacitiesFromTS ( ID="*" )
```

SetWellStationCapacitiesFromTS

**SetWellStationCapacitiesFromTS() Command Editor**

The command syntax is as follows:

```
SetWellStationCapacitiesFromTS(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (do not generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn

---

# Command Reference: SetWellStationCapacityToWellRights ()

Set the well station capacity to well rights

## StateMod Command

Version 3.09.01, 2010-02-01

The `SetWellStationCapacityToWellRights()` command sets the well station capacity for each well station to the sum of the well rights corresponding to the station. The well rights must have been read or assigned with previous commands. If there are no well rights in the file, the capacity is set to zero.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationCapacityToWellRights() Command**

This command sets the well station capacity to the sum of the well water rights for the station. Water rights are specified by reading a StateMod well rights file with a previous command. The well station ID can contain a \* wildcard pattern to match one or more locations.

Well station ID:  Required - well stations to process (use \* for wildcard).

If not found:  Optional - indicate action if no match is found (default=Warn)

Command:

```
SetWellStationCapacityToWellRights ( ID="*" )
```

OK Cancel

SetWellStationCapacityToWellRights

### SetWellStationCapacityToWellRights() Command Editor

The command syntax is as follows:

```
SetWellStationCapacityToWellRights (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file excerpt illustrates how the well station capacities can be set to the sum of the water rights:

```
# Read Well rights from a StateMod well right file
ReadWellRightsFromStateMod(InputFile="..\Wells\Sp2008L.wer")
# Set capacity to total of water rights
SetWellStationCapacityToWellRights(ID="*")
```

---

# Command Reference:

## SetWellStationDelayTablesFromNetwork()

Set well station delay table data from the network

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetWellStationDelayTableFromNetwork()` command sets delay table data in existing well stations using network information. A default delay table is used to assign 100% of the returns to the downstream node in the network.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationDelayTablesFromNetwork() Command**

This command sets well station delay table data using network data.  
The network must be read with a previous command.  
The station ID can contain a \* wildcard pattern to match one or more locations.  
By default, 100% of the returns will be returned to the downstream node.  
Specify the delay table to use for the returns.  
Separate delay table files are used for monthly and daily data sets, although table identifiers can be made to agree.

Well station ID:  Required - well stations to fill (use \* for wildcard).

Default table:  Optional - default table to use with 100% of returns (default=1).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command: 

```
SetWellStationDelayTablesFromNetwork ( ID=" * ", DefaultTable=1 )
```

SetWellStationDelayTablesFromNetwork

### SetWellStationDelayTablesFromNetwork() Command Editor

The command syntax is as follows:

```
SetWellStationDelayTablesFromNetwork(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
ID	A single well station identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
DefaultTable	The default delay table to use when assigning the delay tables.	1
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"><li>• Fail – generate a failure message if the ID is not matched</li><li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li><li>• Warn – generate a warning message if the ID is not matched</li></ul>	Warn



---

# Command Reference: SetWellStationDelayTablesFromRTN()

Set well station delay table data from an RTN format file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The SetWellStationDelayTableFromRTN( ) command sets delay table data in existing well stations using information in an RTN format file, which is a format that has been used in CDSS StateMod modeling. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationDelayTablesFromRTN() Command**

This command reads and processes delay table information from an "RTN" format file.  
Delay (return flow) table data indicate the pattern by which unused water is returned to the system.  
The file may contain default efficiency information for well stations.  
This information can be used and can then be reset later when average efficiencies are estimated from time series.  
This file format has been used with CDSS modeling software and is provided for backward compatibility.  
A delimited list file format may be supported in the future.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork

Input file:

Set efficiency?:  Optional - if True, use default efficiency information in file (default=False).

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

**SetWellStationDelayTablesFromRTN() Command Editor**

SetWellStationDelayTablesFromRTN

The command syntax is as follows:

```
SetWellStationDelayTablesFromRTN( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the RTN file to process. Specify an absolute path or a path relative to the working directory.	None – must be specified.
SetEfficiency	Indicates whether the default efficiency value in the file should be used.	False
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

A sample RTN file is shown below:

200511	2	75	1
	200742	1	1
	200742	99	2
200742	2	75	1
	200787	1	1
	200787	99	2
200752	2	75	1
	20ADW07	1	1
	20ADW07	99	2

The first line contains the station identifier, number of return flow locations, default efficiency for the station, and the default delay table to use for the return. For the number of return flow locations, the following lines indicate the identifier for the station to receive the return, the percentage of the return to receive, and the delay table for the return.

---

# Command Reference: SetWellStationDepletionTablesFromRTN()

Set well station depletion table data from an RTN format file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The SetWellStationDepletionTableFromRTN( ) command sets depletion table data in existing well stations using information in an RTN format file, which is a format that has been used in CDSS StateMod modeling. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellStationDepletionTablesFromRTN() Command**

This command reads and processes depletion information from an "RTN" format file.  
Depletion table data indicate the pattern by which well pumping affects surface water locations.  
This file format has been used with CDSS modeling software and is provided for backward compatibility.  
A delimited list file format may be supported in the future.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork

Input file:

If not found:  Optional - indicate action if no match is found (default=Warn).

Command:

SetWellStationDepletionTablesFromRTN  
**SetWellStationDepletionTablesFromRTN() Command Editor**

The command syntax is as follows:

```
SetWellStationDepletionTablesFromRTN(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
InputFile	The name of the RTN file to process. Specify an absolute path or a path relative to the working directory.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

A sample RTN file is shown below:

200511	2	0	1
	200742	0	1
	200742	100	2
200742	2	0	1
	200787	0	1
	200787	100	2
200752	2	0	1
	20ADW07	0	1
	20ADW07	100	2

The first line contains the station identifier, number of depletion locations, default efficiency for the station (unused – included because file format is the same as the return flow file), and the default delay table to use for the depletion. For the number of depletion locations, the following lines indicate the identifier for the station to receive the depletion, the percentage of the depletion to receive, and the delay table for the depletion.

# Command Reference: SetWellStationsFromList()

## Set well station data from a list file

### StateMod Command

Version 3.09.01, 2010-02-01

The `SetWellStationFromList()` command sets data in existing well stations (it currently will not add a station – use `ReadWellStationsFromList()`). The following dialog is used to edit the command and illustrates the syntax of the command, in this case to set the well station average monthly efficiencies.

**Edit SetWellStationsFromList() Command**

This command edits/sets data in well stations, using the station ID to look up the location.  
Data are supplied by values in a delimited file.  
Use blanks in the any field to indicate no change to the existing value.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork

List file:

Well stations ID column:	<input type="text" value="1"/>	Required - column (1+) for identifier.
Name column:	<input type="text"/>	Optional - column (1+) for name.
River node ID column:	<input type="text"/>	Optional - column (1+) for river node ID.
On/Off column:	<input type="text"/>	Optional - column (1+) for on/off switch.
Capacity column:	<input type="text"/>	Optional - column (1+) for capacity.
Administration number shift column:	<input type="text"/>	Optional - column (1+) for admin. num. shift.
Diversion station ID:	<input type="text"/>	Optional - column (1+) for diversion station ID.
Daily ID column:	<input type="text"/>	Optional - column (1+) for daily ID.
Demand type column:	<input type="text"/>	Optional - column (1+) for demand type.
Irrigated acres column:	<input type="text"/>	Optional - column (1+) for irrigated acres.
Use type column:	<input type="text"/>	Optional - column (1+) for use type.
Demand source column:	<input type="text"/>	Optional - column (1+) for demand source.
Efficiency (annual) column:	<input type="text"/>	Optional - column (1+) for annual efficiency.
Efficiency (monthly) column:	<input type="text" value="2"/>	Optional - first column of 12, listed Jan...Dec.
Delimiter:	<input type="text" value="Space"/>	Optional - delimiter character(s) (default=",").
Merge delimiters:	<input checked="" type="checkbox"/>	Optional - treat consecutive delimiters as one (default=False).
If not found:	<input type="text" value="False"/>	Optional - indicate action if no ID match is found (default=Warn).

Command: `SetWellStationsFromList(ListFile="rg2004.wef", IDCol="1", EffMonthlyCol="2", Delim="Space", MergeDelim=True)`

SetWellStationsFromList

### SetWellStationsFromList() Command Editor

The command syntax is as follows:

```
SetWellStationsFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the delimited input file to read. Strings that include delimiter characters can be surrounded by double quotes in the list file. Lines starting with # are treated as comments.	None – must be specified.
IDCol	The column number (1+) containing the well station identifiers.	If not specified, the original value will remain.
NameCol	The column number (1+) containing the well station names.	If not specified, the original value will remain.
RiverNodeIDCol	The column number (1+) containing the river node identifiers.	If not specified, the original value will remain.
OnOffCol	The column number (1+) containing the on/off switch.	If not specified, the original value will remain.
CapacityCol	The column number (1+) containing the capacity.	If not specified, the original value will remain.
AdminNumShiftCol	The column number (1+) containing the administration number shift value.	If not specified, the original value will remain.
DiversionIDCol	The column number (1+) containing the associated diversion identifier.	If not specified, the original value will remain.
DailyIDCol	The column number (1+) containing the daily identifier.	If not specified, the original value will remain.
DemandTypeCol	The column number (1+) containing the demand type.	If not specified, the original value will remain.
IrrigatedAcresCol	The column number (1+) containing the irrigated acres.	If not specified, the original value will remain.
UseTypeCol	The column number (1+) containing the use type.	If not specified, the original value will remain.
DemandSourceCol	The column number (1+) containing the demand source.	If not specified, the original value will remain.
EffAnnualCol	The column number (1+) containing the annual efficiency. If the annual efficiency is specified, each monthly efficiency will be set to the annual value.	If not specified, the original value will remain.

Parameter	Description	Default
EffMonthlyCol	The column number (1+) containing the monthly efficiency for January. The efficiencies for other months should be specified in columns that follow. The annual efficiency is set to the average of the monthly efficiencies. The efficiencies in the list file must be listed January to December as percent (0 to 100). The order of the values in the StateMod well stations will be according to the output year type set by <code>setOutputYearType( )</code> , or calendar by default.	If not specified, the original values will remain.
Delim	The character(s) that delimits columns, or one of the literal words: <ul style="list-style-type: none"> <li>• Space</li> <li>• Tab</li> <li>• Whitespace – spaces and tabs.</li> </ul>	, (comma)
MergeDelim	If True, then treat consecutive delimiter characters as one delimiter. If False, separate columns will result.	False
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>• Fail – generate a failure message if the ID is not matched</li> <li>• Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>• Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following example illustrates how to create well stations from a list file and then set the efficiencies (in this case from a StateCU output file) from another list. The full data line is trimmed of whitespace before processing and data in columns are automatically trimmed of whitespace after parsing.

```
StartLog(LogFile="commands.StateDMI.log")
ReadWellStationsFromList(ListFile="rgdssall.csv",IDCol="1")
SetWellStationsFromList(ListFile="rg2004.wef",IDCol="1",
    EffMonthlyCol="2",Delim="Space",MergeDelim=True,IfNotFound=Warn)
WriteWellStationsToStateMod(OutputFile="rgdssall.dds")
```

The following is an example of the list file used with the above:

```
# Card 1 Control
# format: (Free)
# NOTE EFF1 IS JANUARY, EFF2 IS FEBRUARY, ETC.
#
# ID      cwelid:  Well ID
# Eff1    eff(1)   Efficiency in month 1
# Eff1    eff(2)   Efficiency in month 2
# ...     ....    ...
# Eff1    eff(12)  Efficiency in month 12
#
#
#1 ID      Eff1    Eff2    Eff3    Eff4    Eff5    Eff6'Eff7    Eff8    Eff9    Eff10    Eff11    Eff12
#-----eb-----eb-----eb-----eb-----eb-----eb-----'-----eb-----eb-----eb-----eb-----exb-----eb-----
#
# 200505      0.      0.      0.      0.      0.      0.      0.      0.      0.      0.      0.      0.      0. ALAMOSA D
# 200511      0.      0.      0.      0.      0.      0.      0.      80.     80.     80.     0.      0.      0. ANACONDA D
```



---

# Command Reference: SetWellSystem()

## Set well system parts

### StateCU and StateMod Command

Version 3.09.00, 2010-01-21

The `SetWellSystem()` command sets well system part identifier data for a well (a CU Location that corresponds to a location with well supply, or StateMod well station). Well systems are specified using a list of part identifiers as follows:

- Part type is `Ditch` – the collection includes wells that are associated with a list of ditches, identified using ditch water district identifiers (WDIDs). The list of ditches is used for the full period.
- Part type is `Parcel` – the collection includes wells that are associated with a list of parcels. The division and year must be specified in the command because well to parcel relationships are determined for specific years.
- Part type is `Well` – the collection includes wells identified by well WDID (permit receipt number is not supported).

To facilitate processing, it is often best to use list files to specific aggregates (see `SetWellSystemFromList()`). Systems by convention have their water rights fully represented in output – to aggregate water rights at a location, use an aggregate (see the similar `Aggregate` commands). See also the StateDMI **Introduction** chapter, which provides additional information about aggregates and other modeling conventions. System information should be specified after well locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellSystem() Command**
✕

This command sets a Well System location's information.  
 Each System is a location where individual parts are combined into a single feature.  
 An "Aggregate" is used with SetWellAggregate() when water rights will be aggregated into classes.  
 A "System" is used with SetWellSystem() when individual water rights will be maintained.  
 For example, well-only parcels may be grouped as a single identifier.  
 Wells associated with ditches are grouped by specifying ditch identifiers and apply for the full period.  
 When grouping wells using parcels, specify parcel identifiers for the parts. Indicate the year and water division for the parcel data.  
 When grouping wells using well identifiers, specify a list of WDIDs or P:receipt for permits.  
 Separate the part IDs by spaces or commas.

Well System ID:	0102522_AuW	Required - specify the Well System ID.
System part type:	Ditch ▼	Required - the type of features being aggregated.
Year:		Required for part type + __command._Parcel + - year for the parcels.
Water division (Div):		Required for part type Parcel - water division for the parcels.
Part IDs:	108090, 108093, 108095, 108092, 0108091	Required - up to 12 characters for each ID.
If not found:	▼	Optional - indicate action if no ID match is found (default=Warn).
Command:	<pre>SetWellSystem( ID="0102522_AuW", PartType=Ditch, PartIDs="108090, 108093, 108095, 108092, 0108091")</pre>	

OK
Cancel

SetWellSystem

**SetWellSystem() Command Editor**

The command syntax is as follows:

```
SetWellSystem (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ID	The well identifier to associate with the collection of individual wells.	None – must be specified.
PartType	Indicate the type of features being aggregated and specified by PartIDs, one of: <ul style="list-style-type: none"> <li>Ditch – the PartIDs (ditch WDIDs) indicate ditch service areas supplemented by wells.</li> <li>Parcel – the PartIDs (parcel numbers from GIS processing) indicate parcels irrigated by wells, with no surface water supply.</li> <li>Well – the PartIDs indicate wells (WDIDs), with no surface water supply.</li> </ul>	None – must be specified.
Year	The year defining the parcels.	Required when PartType is Parcel because parcel identifiers from well matching are specific to the data year.
Div	Water division for the parcels in the system.	Required when PartType is Parcel because parcels require the division.
PartIDs	The list of part identifiers to comprise the system.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the identifier is not matched</li> <li>Warn – generate a warning message if the identifier is not matched</li> </ul>	Warn

This page is intentionally blank.

---

# Command Reference: SetWellSystemFromList()

Set well system parts from data in a list file

**StateCU and StateMod Command**

Version 3.09.00, 2010-01-21

The `SetWellSystemFromList()` command sets well system part identifier data for a well (a CU Location that corresponds to a location with well supply, or StateMod well station). Well systems are specified using a list of part identifiers as follows:

- Part type is `Ditch` – the collection includes wells that are associated with a list of ditches, identified using ditch water district identifiers (WDIDs). The list of ditches is used for the full period.
- Part type is `Parcel` – the collection includes wells that are associated with a list of parcels. The division and year must be specified in the command because well to parcel relationships are determined for specific years.
- Part type is `Well` – the collection includes wells identified by well WDID (permit receipt number is not supported).

To facilitate processing, the list of parts is specified in a delimited list file. Systems by convention have their water rights fully represented in output – to aggregate water rights at a location, use an aggregate (see the similar `Aggregate` commands). See also the **StateDMI Introduction** chapter, which provides additional information about systems and other modeling conventions. System information should be specified after well locations are defined and before their use in other processing, such as reading data from HydroBase.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit SetWellSystemFromList() Command**
✕

This command sets a Well System location's information from a list file.  
Each System is a location where individual parts are combined into a single feature.  
An "Aggregate" is used with SetWellAggregateFromList() when water rights will be aggregated into classes.  
A "System" is used with SetWellSystemFromList() when individual water rights will be maintained.  
For example, well-only parcels may be grouped as a single identifier.  
Wells associated with ditches are grouped by specifying ditch identifiers for the parts.  
When grouping wells using parcels, specify parcel identifiers for the parts and indicate the year and water division for the parcel data.  
When grouping wells using well identifiers, specify a list of well WDIDs or P:receipt for permits.  
Columns should be delimited by commas.  
It is recommended that the location of the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\SetWellSystemFromList

List file:  Browse

System part type: Parcel

Year:

Water division (Div):

ID column: 1

Name column:

Part IDs column: 2

Parts listed how: InColumn

Part IDs column (max):

If not found:

Required - the type of features being aggregated.

Required if part type is Ditch or Parcel - year for the parcels.

Required if part type is Ditch or Parcel - water division for the parcels.

Required - column for the Well System IDs.

Optional - column for the Well System name.

Required - first/only column for the part IDs.

Required - are part IDs listed in row or column?

Optional - maximum column for part IDs if in row (default is use all).

Optional - indicate action if no ID match is found (default=Warn).

Command:

```
SetWellSystemFromList (ListFile="1956_01_GW.csv", Year=1956, Div=1, PartType=Parcel, IDCol=1, PartIDsCol=2, PartsListedHow=InColumn)
```

Add Working Directory
Cancel
OK

SetWellSystemFromList

### SetWellSystemFromList() Command Editor

The command syntax is as follows:

```
SetWellSystemFromList (Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
ListFile	The name of the input file to read, surrounded by double quotes.	None – must be specified.
PartType	Indicate the type of features being aggregated and specified by PartIDs, one of: <ul style="list-style-type: none"> <li>Ditch – the PartIDs (ditch WDIDs) indicate ditch service areas supplemented by wells.</li> <li>Parcel – the PartIDs (parcel</li> </ul>	None – must be specified.

Parameter	Description	Default
	<p>numbers from GIS processing) indicate parcels irrigated by wells, with no surface water supply.</p> <ul style="list-style-type: none"> <li>Well – the PartIDs indicate wells (WDIDs), with no surface water supply.</li> </ul>	
Year	The year defining the parcels.	Required when PartType is Parcel because parcel identifiers from well matching are specific to the data year.
Div	Water division for the parcels in the system.	Required when PartType is Parcel because parcels require the division.
IDCol	The column number (1+) containing the well system identifiers.	None – must be specified.
NameCol	The column number (1+) containing the well system name.	None – optional (name will remain as before).
PartIDsCol	The column number (1+) for the first column having part identifiers.	None – must be specified.
PartsListedHow	If InRow, it is expected that all parts defining a system are listed in the same row. If InColumn, it is expected that the parts defining a system are listed one per row, with multiple rows defining the full system (PartIDsColMax is ignored in this case).	None – must be specified.
PartIDsColMax	The column number (1+) for the last column having part identifiers. Use if extra columns on the right need to be excluded from the list.	Use all available non-blank columns starting with PartIDsCol.
IfNotFound	<p>Used for error handling, one of the following:</p> <ul style="list-style-type: none"> <li>Fail – generate a failure message if the aggregate identifier is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the aggregate identifier is not matched</li> <li>Warn – generate a warning message if the aggregate identifier is not matched</li> </ul>	Warn

The following example illustrates a list file is used with PartType=Parcel and PartsListedHow=InColumn:

```
"UZONES" , "PARCEL"  
20URF0,16831  
20URF0,16832  
20URF0,16834  
...  
20URF0,18606  
20URF24,10295  
20URF24,10318  
...
```

The following example illustrates a list file is used with PartType=Ditch and PartsListedHow=InColumn, with the name being provided in column 2:

```
# System_ID/Agg_Name/WDID  
01_ADP037,South Platte River below Kersey Co North 2,0100643  
01_ADP037,South Platte River below Kersey Co North 2,0100644  
01_ADP037,South Platte River below Kersey Co North 2,0100835  
01_ADP037,South Platte River below Kersey Co North 2,0104486
```



---

# Command Reference: SetWorkingDir()

## Set the working directory for the software

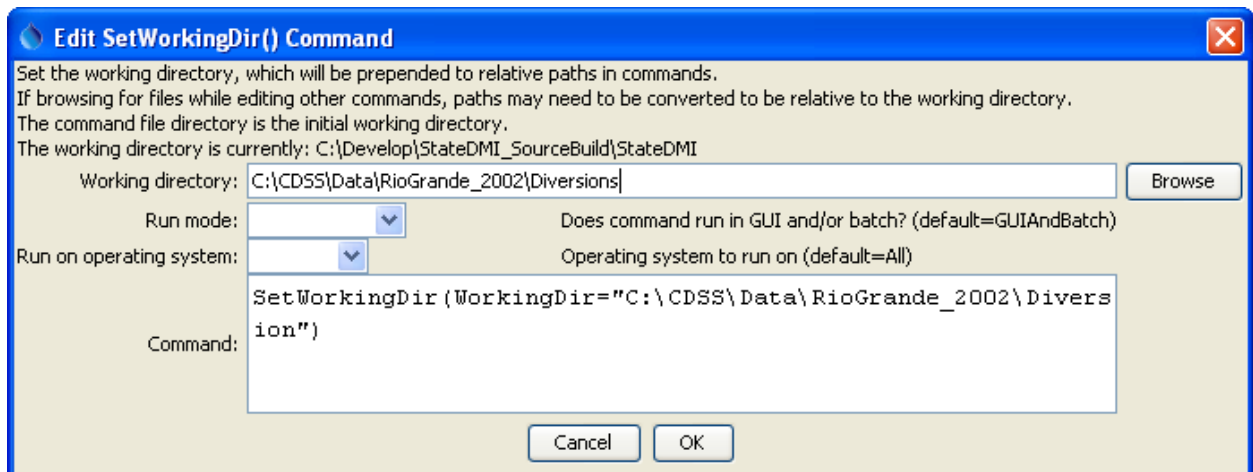
### General Command

Version 3.08.02, 2010-01-07

The `SetWorkingDir()` command is used to define the working directory for a set of commands. The working directory, when set properly, can greatly simplify commands files because relative file paths can be used for input and output. The working directory is normally set in one of the following ways, with the current setting being defined by the most recent item that has occurred:

1. The startup directory for the StateDMI program,
2. The directory where a commands file was opened,
3. The directory where a commands file was saved,
4. The directory specified by a `SetWorkingDir()` command,
5. The directory specified by **File...Set Working Directory**.

In most cases, a `SetWorkingDir()` command is not needed. However, for complicated command files, it may be necessary to change the working directory from one directory to another. Setting the working directory to an absolute path causes all relative paths for input and output files to be appended to the working directory. The following dialog is used to edit the command and illustrates the syntax of the command. The **Browse** button allows you to select the working directory.



**SetWorkingDir() Command Editor**

SetWorkingDir

The command syntax is as follows:

```
SetWorkingDir(WorkingDir)
```

#### Command Parameters

Parameter	Description	Default
WorkingDir	Working directory for the software, with which relative paths are converted into absolute paths.	None – must be specified.

---

# Command Reference: SortBlaneyCriddle()

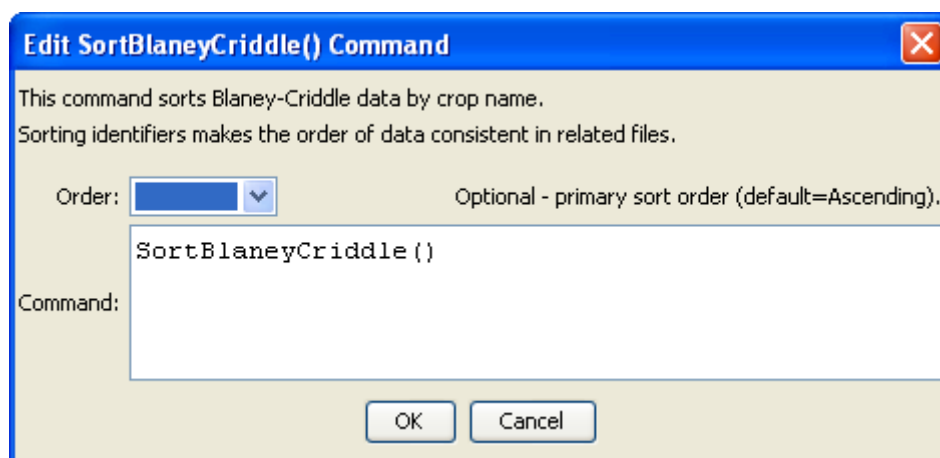
Sort Blaney-Criddle data

**StateCU Command**

Version 03.08.02, 2010-01-07

The `SortBlaneyCriddle()` command sorts the Blaney-Criddle crop coefficients using the crop name.

The following dialog is used to edit the command and illustrates the syntax of the command.



**SortBlaneyCriddle() Command Editor**

SortBlaneyCriddle

The command syntax is as follows:

```
SortBlaneyCriddle(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort for the crop name. Currently only Ascending is supported. The older Alphabetical will be converted automatically.	Ascending

---

# Command Reference: SortClimateStations()

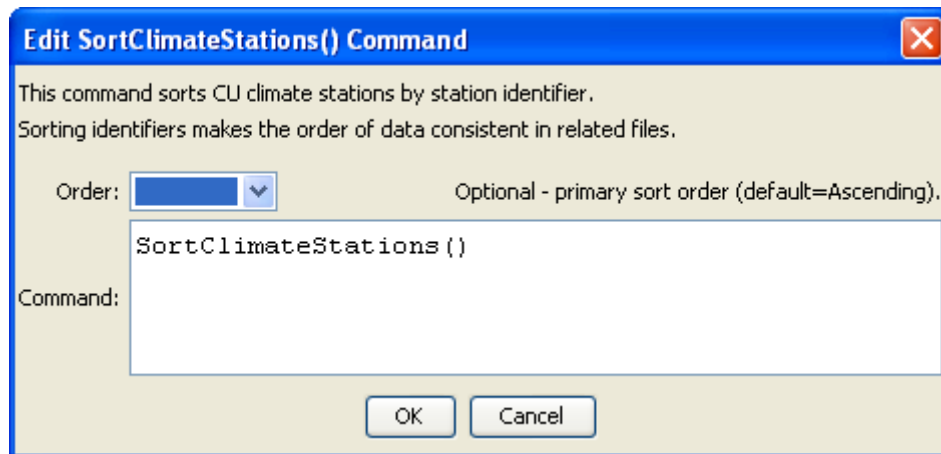
**Sort climate stations**

**StateCU Command**

Version 3.08.02, 2010-01-05

The `SortClimateStations()` command sorts the climate stations using the station identifiers.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortClimateStations

**SortClimateStations() Command Editor**

The command syntax is as follows:

```
SortClimateStations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the sort order. Currently only Ascending is supported.	Ascending

The following example command file illustrates how climate stations can be defined, sorted, and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
FillClimateStationsFromHydroBase(ID="*")
SetClimateStation(ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation(ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation(ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation(ID="0484",Region1="MOFFAT",IfNotFound=Add)
SortClimateStations()
WriteClimateStationsToStateCU(OutputFile="COclim2006.cli")
```

---

# Command Reference: SortCropCharacteristics()

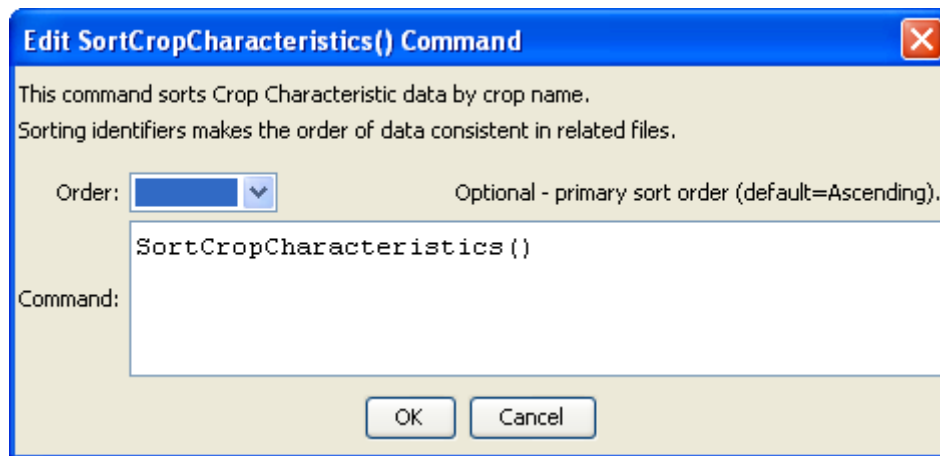
## Sort crop characteristics

### StateCU Command

Version 3.08.02, 2010-01-07

The `SortCropCharacteristics()` command sorts the crop characteristics using the crop name, and is typically used before writing output.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortCropCharacteristics

**SortCropCharacteristics() Command Editor**

The command syntax is as follows:

```
SortCropCharacteristics( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort for the crop name. Currently only Ascending is supported. The older Alphabetical will be converted automatically.	Ascending



---

# Command Reference: SortCropPatternTS()

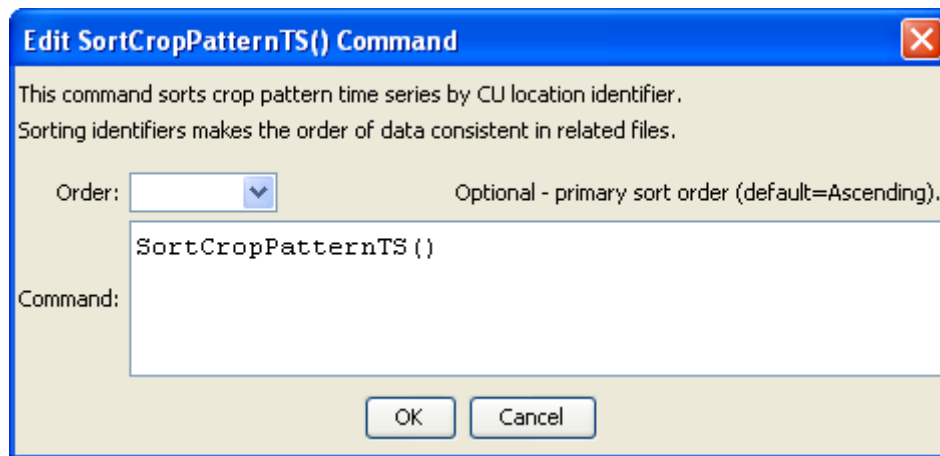
## Sort crop pattern time series

### StateCU Command

Version 3.09.01, 2010-02-01

The `SortCropPatternTS()` command sorts the crop pattern time series using the location identifier, and is typically used before writing output.

The following dialog is used to edit the command and illustrates the syntax of the command.



**SortCropPatternTS() Command Editor**

SortCropPatternTS

The command syntax is as follows:

```
SortCropPatternTS( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical will be converted automatically.	Ascending

---

# Command Reference: SortCULocations()

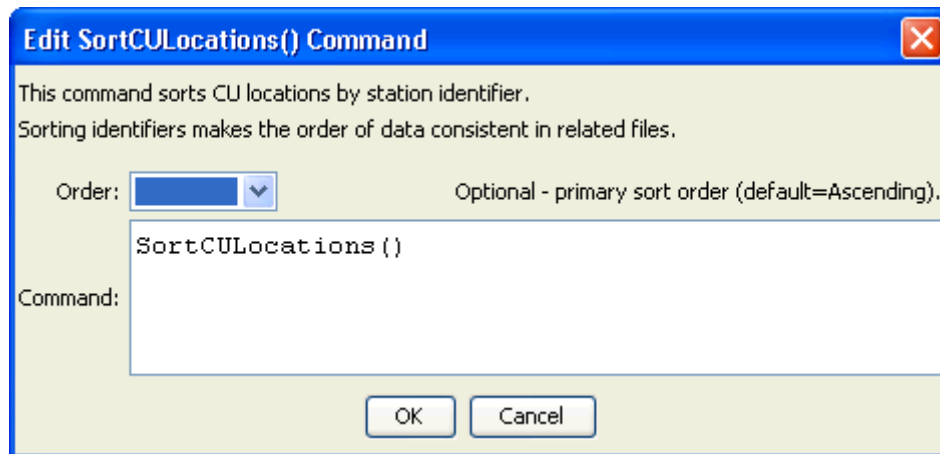
**Sort CU Locations**

**StateCU Command**

Version 3.09.00, 2010-01-10

The `SortCULocations()` command sorts the CU Locations.

The following dialog is used to edit the command and illustrates the syntax of the command.



**SortCULocations() Command Editor**

SortCULocations

The command syntax is as follows:

```
SortCULocations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the sort order for identifiers. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

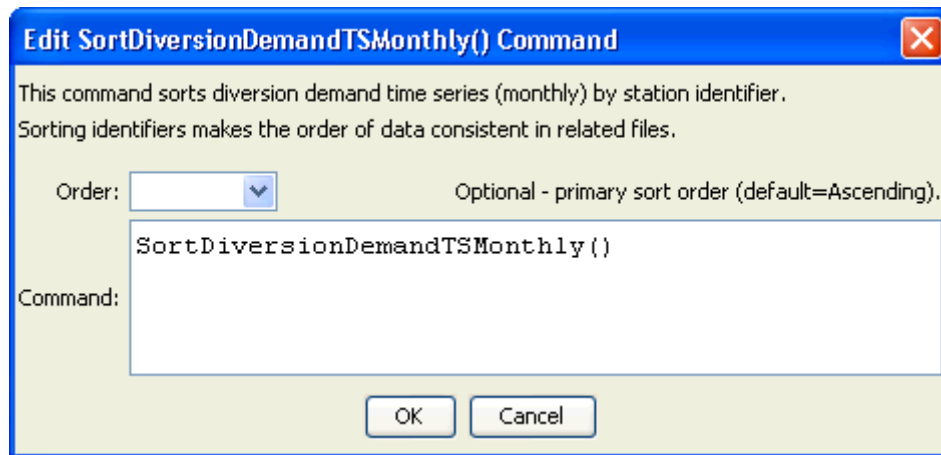
# Command Reference: SortDiversiionDemandTSMonthly()

## Sort diversion demand time series (monthly)

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `SortDiversiionDemandTSMonthly()` command sorts the diversion demand time series (monthly) in alphabetical order, using the time series identifier (typically by the location since the location will vary between time series). This command is useful if time series have been added during processing and therefore the time series order no longer agrees with the diversion station order.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortDiversiionDemandTSMonthly

**SortDiversiionDemandTSMonthly() Command Editor**

The command syntax is as follows:

```
SortDiversionDemandTSMonthly(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

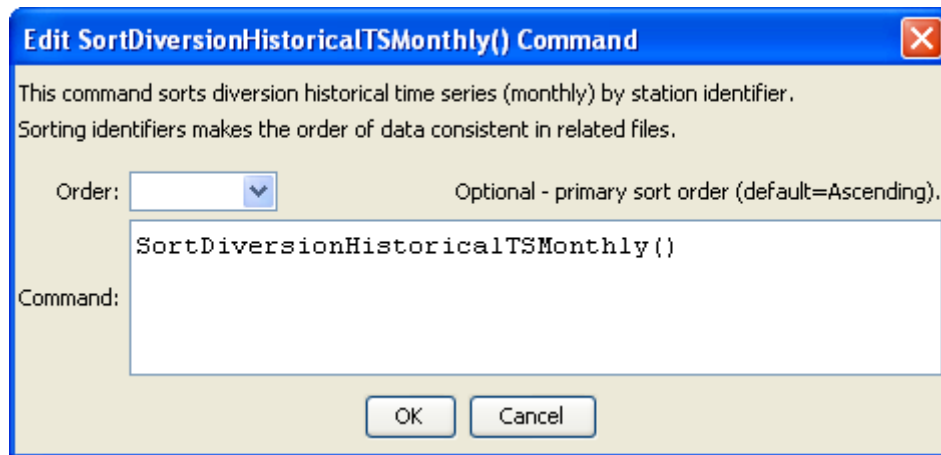
# Command Reference: SortDiversiOnHistoricalTSMonthly()

## Sort diversion historical time series (monthly)

**StateMod Command**  
Version 3.09.01, 2010-02-01

The SortDiversiOnHistoricalTSMonthly( ) command sorts the diversion historical time series (monthly) in alphabetical order, using the time series identifier (typically by the location since the location will vary between time series). This command is useful if time series have been added during processing and therefore the time series order no longer agrees with the diversion station order.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortDiversiOnHistoricalTSMonthly

**SortDiversiOnHistoricalTSMonthly() Command Editor**

The command syntax is as follows:

```
SortDiversionHistoricalTSMonthly( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending



---

# Command Reference: SortDiversiionRights()

## Sort diversion rights

### StateCU and StateMod Command

Version 3.09.01, 2010-01-26

The `SortDiversiionRights()` command sorts the diversion rights. This is useful to enforce consistency between files and simplify file comparison.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortDiversiionRights

**SortDiversiionRights() Command Editor**

The command syntax is as follows:

```
SortDiversiionRights(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

This page is intentionally blank.

---

# Command Reference: SortDiversionsStations()

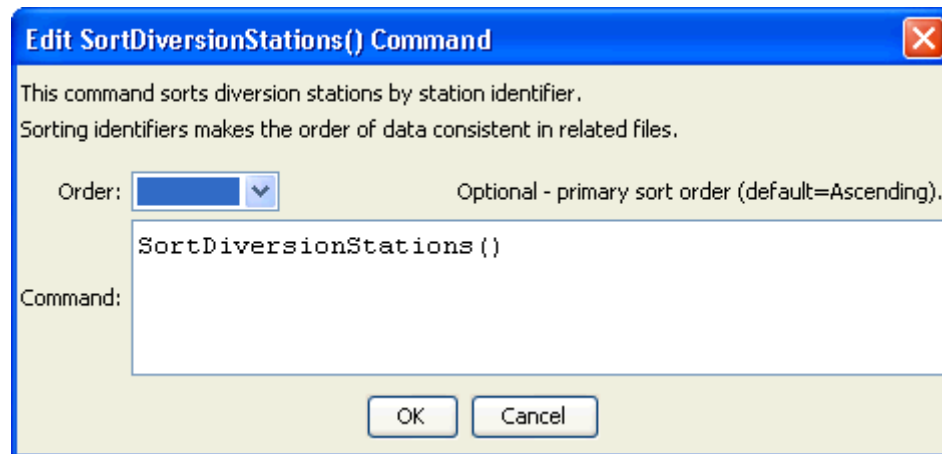
## Sort diversion stations

### StateMod Command

Version 3.09.01, 2010-02-01

The `SortDiversionsStations()` command sorts the diversion stations by station identifier.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortDiversionsStations

**SortDiversionsStations() Command Editor**

The command syntax is as follows:

```
SortDiversionStations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: SortInstreamFlowRights()

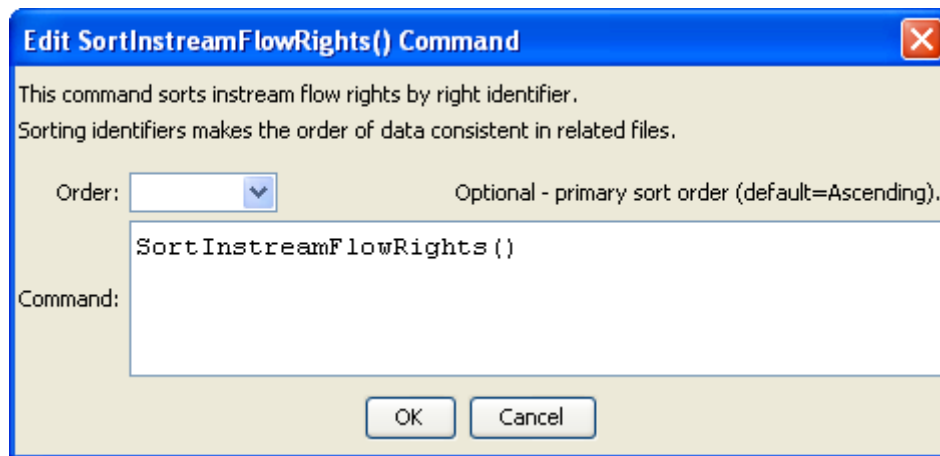
## Sort instream flow rights

### StateMod Command

Version 3.09.01, 2010-02-01

The `SortInstreamFlowRights()` command sorts the instream flow rights by right identifier. This is useful to enforce consistency between files and simplify file comparison.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortInstreamFlowRights

**SortInstreamFlowRights() Command Editor**

The command syntax is as follows:

```
SortInstreamFlowRights (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

This page is intentionally blank.

---

# Command Reference: SortInstreamFlowStations()

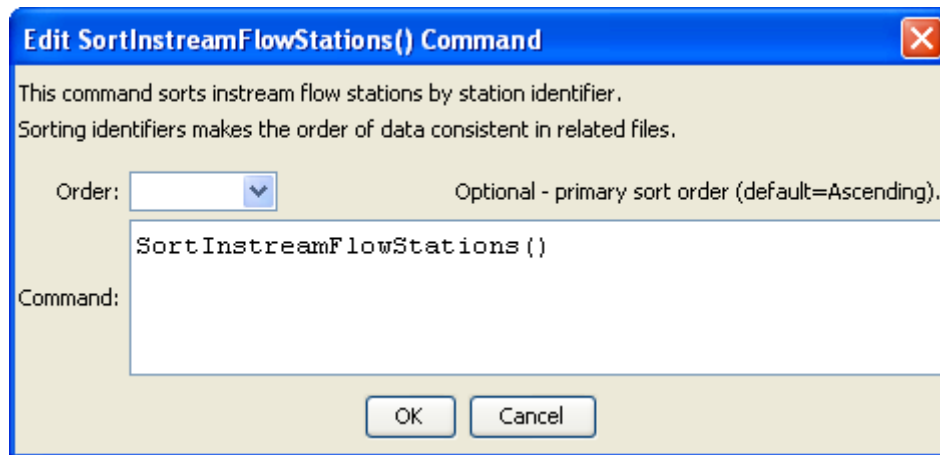
Sort instream flow stations

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SortInstreamFlowStations()` command sorts the instream flow stations.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortInstreamFlowStations

**SortInstreamFlowStations() Command Editor**

The command syntax is as follows:

```
SortInstreamFlowStations (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending



---

# Command Reference: SortIrrigationPracticeTS()

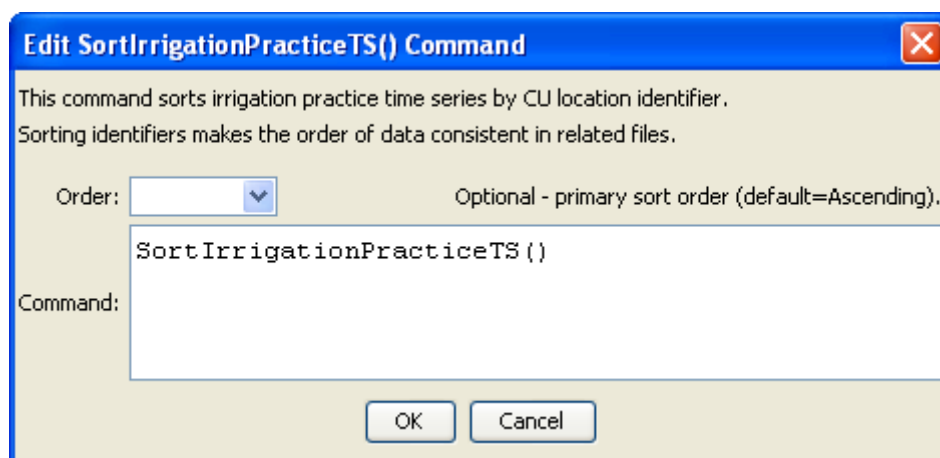
Sort irrigation practice time series

**StateCU Command**

Version 3.09.01, 2010-02-01

The `SortIrrigationPracticeTS()` command sorts the irrigation practice time series using the location identifier, and is typically used before writing output.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortIrrigationPracticeTS

**SortIrrigationPracticeTS() Command Editor**

The command syntax is as follows:

```
SortIrrigationPracticeTS(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical will be converted automatically.	Ascending

---

# Command Reference: SortPenmanMonteith()

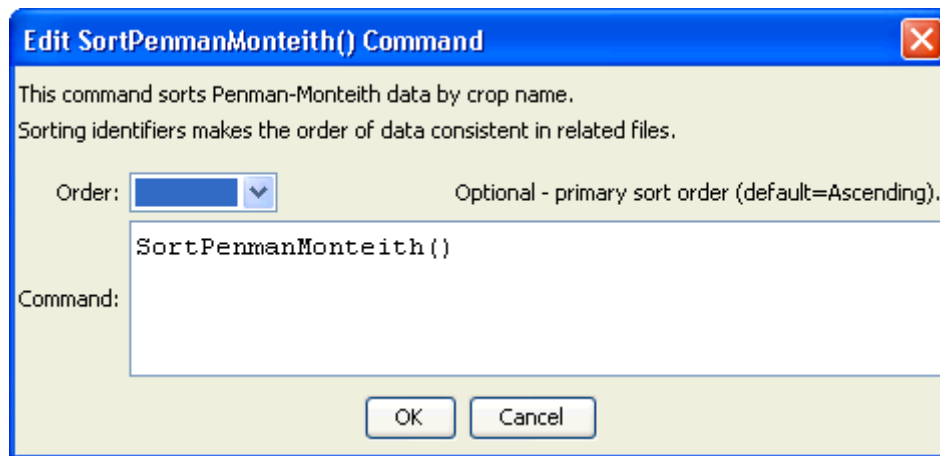
## Sort Penman-Monteith data

### StateCU Command

Version 03.10.00, 2010-04-02

The `SortPenmanMonteith()` command sorts the Penman-Monteith crop coefficients using the crop name.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortPenmanMonteith

**SortPenmanMonteith() Command Editor**

The command syntax is as follows:

```
SortPenmanMonteith(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort for the crop name. Currently only Ascending is supported. The older Alphabetical will be converted automatically.	Ascending

---

# Command Reference: SortReservoirRights()

**Sort reservoir rights**

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SortReservoirRights()` command sorts the reservoir rights. This is useful to enforce consistency between files and simplify file comparison.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortReservoirRights

**SortReservoirRights() Command Editor**

The command syntax is as follows:

```
SortReservoirRights(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: SortReservoirStations()

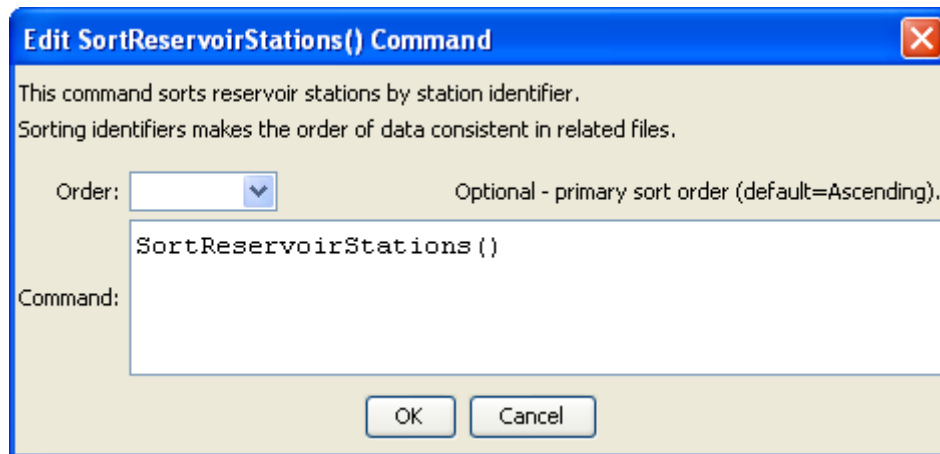
## Sort reservoir stations

### StateMod Command

Version 3.09.01, 2010-02-01

The `SortReservoirStations()` command sorts the reservoir stations by station identifier.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortReservoirStations

**SortReservoirStations() Command Editor**

The command syntax is as follows:

```
SortReservoirStations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending.



---

# Command Reference: SortStreamEstimateStations()

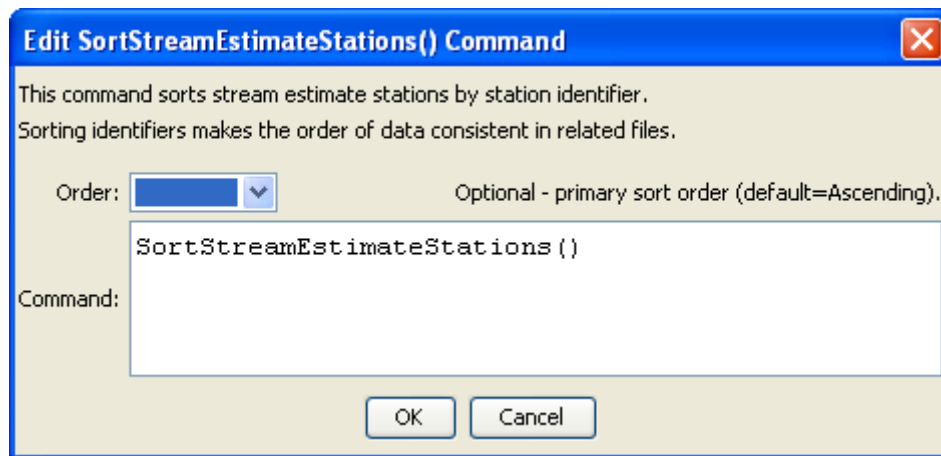
Sort stream estimate stations

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SortStreamEstimateStations()` command sorts the stream estimate stations by station identifier.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortStreamEstimateStations

**SortStreamEstimateStations() Command Editor**

The command syntax is as follows:

```
SortStreamEstimateStations (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: SortStreamGageStations()

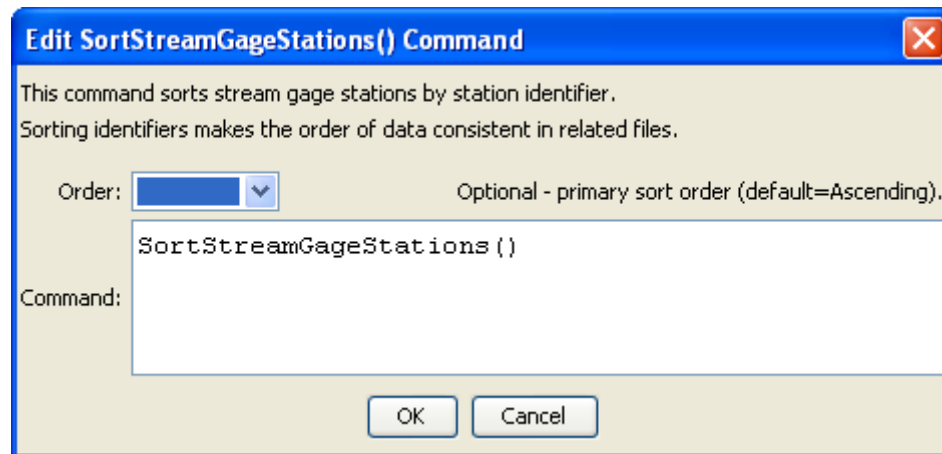
Sort stream gage stations

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SortStreamGageStations()` command sorts the stream gage stations by identifier.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortStreamGageStations

**SortStreamGageStations() Command Editor**

The command syntax is as follows:

```
SortStreamGageStations( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: SortWellDemandTSMonthly()

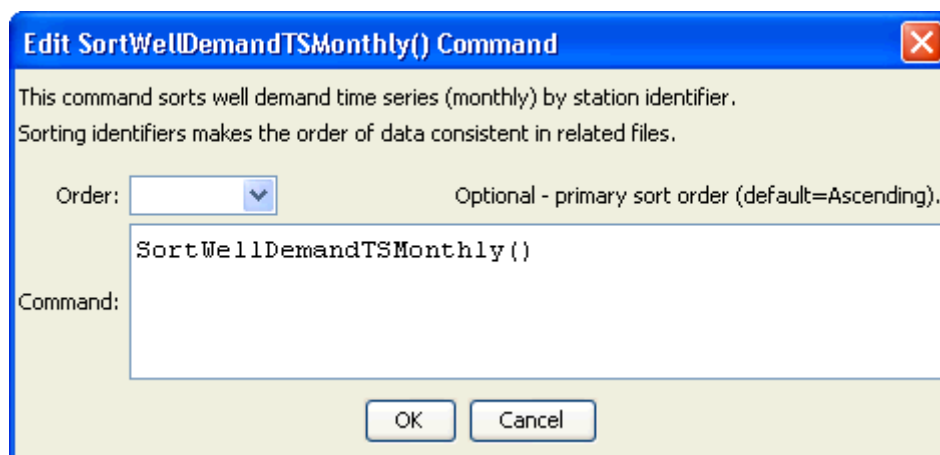
Sort well demand time series (monthly)

**StateMod Command**

Version 3.09.01, 2010-01-27

The `SortWellDemandTSMonthly()` command sorts the well demand time series (monthly) in alphabetical order, using the time series identifier (typically by the location since the location will vary between time series). This command is useful if time series have been added during processing and therefore the time series order no longer agrees with the well station order.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortWellDemandTSMonthly

**SortWellDemandTSMonthly () Command Editor**

The command syntax is as follows:

```
SortWellDemandTSMonthly( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: SortWellHistoricalPumpingTSMonthly()

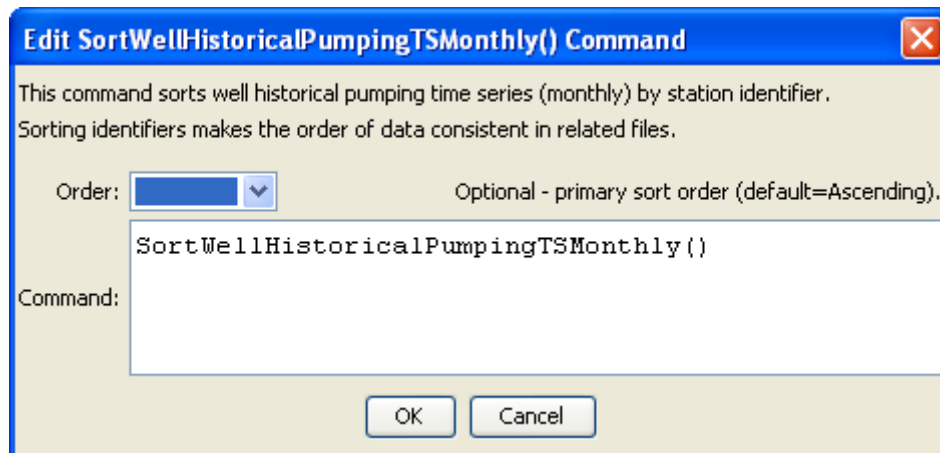
Sort well historical pumping time series (monthly)

StateCU and StateMod Command

Version 3.09.01, 2010-01-27

The SortWellHistoricalPumpingTSMonthly( ) command sorts the well historical pumping time series (monthly) in alphabetical order, using the time series identifier (typically by the location since the location will vary between time series). This command is useful if time series have been added during processing and therefore the time series order no longer agrees with the well station order.

The following dialog is used to edit the command and illustrates the syntax of the command.



SortWellHistoricalPumpingTSMonthly

SortWellHistoricalPumpingTSMonthly () Command Editor

The command syntax is as follows:

```
SortWellHistoricalPumpingTSMonthly( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending



---

# Command Reference: SortWellRights()

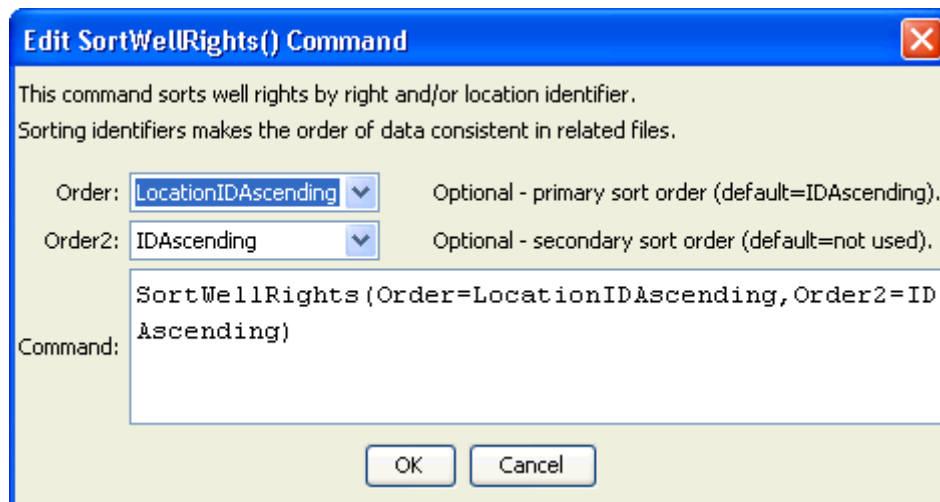
## Sort well rights

### StateMod Command

Version 3.09.00, 2010-01-24

The `SortWellRights()` command sorts the well rights. This is useful to enforce consistency between files and simplify file comparison.

The following dialog is used to edit the command and illustrates the syntax of the command.



**SortWellRights() Command Editor**

SortWellRights

The command syntax is as follows:

```
SortWellRights( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the primary sort order, one of: <ul style="list-style-type: none"><li>IDAscending</li><li>LocationIDAscending (typically specified)</li></ul>	IDAscending
Order2	Indicate the secondary sort order, one of: <ul style="list-style-type: none"><li>IDAscending (typically specified)</li><li>LocationIDAscending</li></ul>	None – must be specified. Typically IDAscending is specified.

---

# Command Reference: SortWellStations()

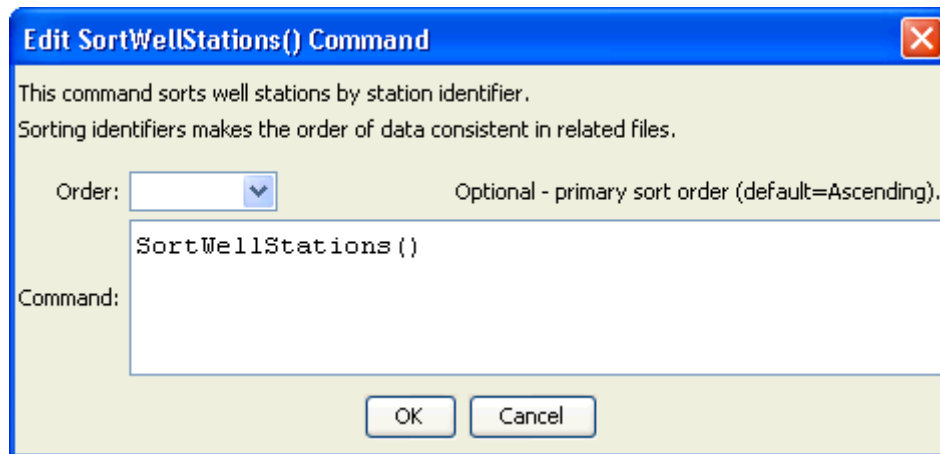
Sort well stations

**StateMod Command**

Version 3.09.01, 2010-02-01

The `SortWellStations()` command sorts the well stations.

The following dialog is used to edit the command and illustrates the syntax of the command.



**SortWellStations() Command Editor**

SortWellStations

The command syntax is as follows:

```
SortWellStations(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
Order	Indicate the order for the sort. Currently only Ascending is supported. The older Alphabetical is automatically converted to Ascending.	Ascending

---

# Command Reference: StartLog()

## (Re)start the log file

### General Command

Version 03.02.00, 2008-11-19

The `StartLog()` command (re)starts the log file. It is useful to insert this command as the first command in a command file, in order to persistently record the results of processing. A useful standard is to name the log file the same as the command file, with an additional `.log` extension, and this convention is enforced by default. A date or date/time can optionally be added to the log file name.

The following dialog is used to edit the command and illustrates the syntax for the command.

**Edit StartLog() command**

(Re)start the log file. This is useful when it is desirable to have a log file saved for a commands file.  
A blank log file name will restart the current file.  
The log file can be specified using a full or relative path (relative to the working directory).  
The working directory is: C:\Develop\TSTool\_SourceBuild\TSTool\test\regression\UserManualExamples\TestCases\CommandReference\StartLog  
The Browse button can be used to select an existing file to overwrite.  
Specifying a suffix for the file will insert the suffix before the "log" file extension.

Log file:

Suffix:  Suffix for log file (blank=none).

Command:

StartLog

**StartLog() Command Editor**

The command syntax is as follows:

```
StartLog(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
LogFile	The name of the log file to write surrounded by double quotes. The extension of <i>.log</i> will automatically be added, if not specified.	If not specified, the existing file will be restarted.
Suffix	<p>Indicates that a suffix will be added before the <i>.log</i> extension, one of:</p> <ul style="list-style-type: none"><li>▪ Date – add a date suffix of the form YYYYMMDD.</li><li>▪ DateTime – add a date/time suffix of the form YYYYMMDD_HHMMSS.</li></ul> <p>This is useful for automatically archiving logs corresponding to commands files, to allow checking the output at a later time. However, generating date/time stamped log files can increase the amount of disk space that is used.</p>	Do not add the suffix.

The following example command file illustrates how to open a log file at the start of processing:

```
StartLog(LogFile="Example_StartLog.log")  
# Remainder of comments and commands follow...
```

---

# Command Reference:

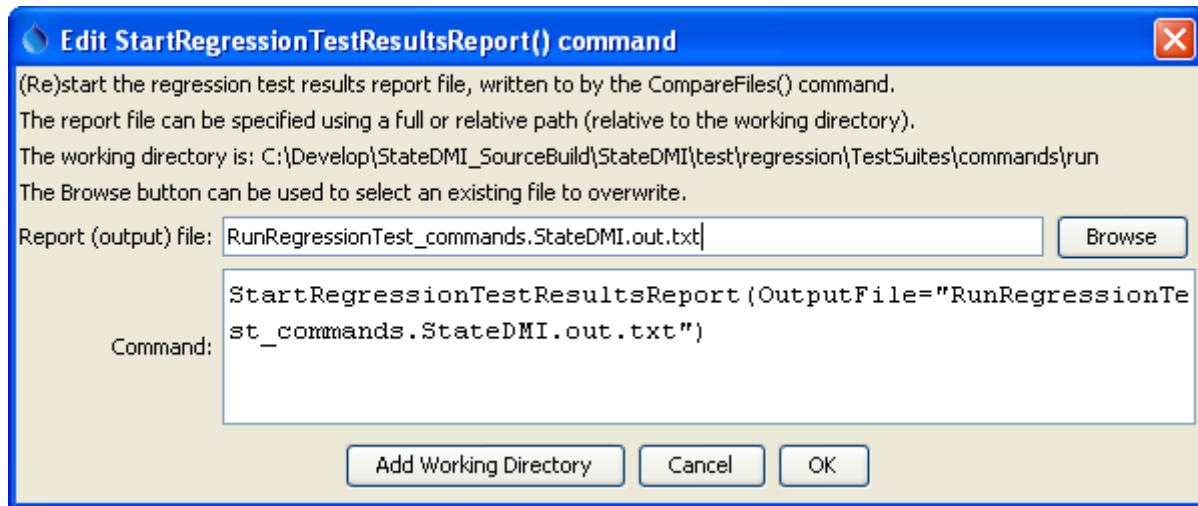
## StartRegressionTestResultsReport()

Start a report file to contain regression test results

**General Command**  
Version 3.08.02, 2010-01-06

The `StartRegressionTestResultsReport()` command starts a report file to be written to as regression tests are run. The `CreateRegressionTestCommandFile()` automatically inserts this command. The `CompareFiles()` and `CompareTimeSeries()` commands will write to this file if it is available.

The following dialog is used to edit the command and illustrates the syntax for the command.



StartRegressionTestResultsReport

**StartRegressionTestResultsReport() Command Editor**

The command syntax is as follows:

```
StartRegressionTestResultsReport (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the report file, enclosed in double quotes if the file contains spaces or other special characters. A path relative to the command file can be specified.	None – must be specified.

See the RunCommands ( ) documentation for how to set up a regression test. The following command file illustrates how to start the results report:

```
StartRegressionTestResultsReport(  
    OutputFile="RunRegressionTest_commands_general.StateDMI.out.txt")  
...  
RunCommands (InputFile="..\..\..\commands\ReadClimateStationsFromList\  
    Test_ReadClimateStationsFromList.StateDMI")  
...
```

Each of the above command files should produce expected time series results, without warnings. If any command file unexpectedly produces a warning, a warning will also be visible in StateDMI. The issue can then be evaluated to determine whether a software or configuration change is necessary.



# Command Reference: TranslateBlaneyCriddle()

Translate Blaney-Criddle crop types from one value to another

**StateCU Command**

Version 3.08.02, 2010-01-07

The `TranslateBlaneyCriddle()` command translates Blaney-Criddle data. In particular, it converts one crop type to another. Primary uses of the command are:

1. A data source may use one variant of the crop type (e.g., ORCHARD W/O COVER but the rest of a StateCU data set uses another type (e.g., ORCHARD\_WO\_COVER). In this case the command is used simply to change the spelling of a crop type.
2. The raw crop data may need to be adjusted to reflect differences in crops, for modeling purposes. For example, the original data may identify pasture (e.g., GRASS\_PASTURE) but for modeling the crop type is set to a different value (e.g., GRASS\_PASTURE\_HA) for high altitude coefficients. The following example illustrates a command of this type.

The following dialog is used to edit the command and illustrates the syntax of the command (for the second case listed above):

**Edit TranslateBlaneyCriddle() Command**

This command changes the crop type (name) in Blaney-Criddle coefficients to a new value. This is sometimes necessary because the crop types read from input may not agree with standard types. Crop types in raw data may also need to be changed to values appropriate for model files (e.g., for regional coefficients).

Old crop type:  Required - for example: "MEADOW"

New crop type:  Required - for example: "GRASS\_PASTURE"

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
TranslateBlaneyCriddle (OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE")
```

TranslateBlaneyCriddle

**TranslateBlaneyCriddle() Command Editor**

The command syntax is as follows:

```
TranslateBlaneyCriddle( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OldCropType	A single crop type identifier to match. This crop type will be replaced with the value for NewCropType.	None – must be specified.
NewCropType	The new crop type to use.	None – must be specified.

The following simple example illustrates how to translate a crop type:

```
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")  
TranslateBlaneyCriddle(OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE")
```

---

# Command Reference: TranslateCropCharacteristics()

**Translate crop characteristics crop types from one value to another**

**StateCU Command**

Version 3.08.02, 2010-01-07

The `TranslateCropCharacteristics()` command translates crop characteristics data. In particular, it converts one crop type to another. Primary uses of the command are:

1. A data source may use one variant of the crop type (e.g., ORCHARD W/O COVER but the rest of a StateCU data set uses another type (e.g., ORCHARD\_WO\_COVER). In this case the command is used simply to change the spelling of a crop type.
2. The raw crop data may need to be adjusted to reflect differences in crops, for modeling purposes. For example, the original data may identify pasture (e.g., GRASS\_PASTURE) but for modeling the crop type is set to a different value (e.g., GRASS\_PASTURE\_HA) for high altitude coefficients. The following example illustrates a command of this type.

The following dialog is used to edit the command and illustrates the syntax of the command (for the second case listed above):

**Edit TranslateCropCharacteristics() Command**

This command changes the crop type in crop characteristics to a new value.  
This is sometimes necessary because the crop types read from input may not agree with standard types.  
Crop types in raw data may also need to be changed to values appropriate for model files (e.g., for regional coefficients).

Old crop type:  Required - for example: "MEADOW"

New crop type:  Required - for example: "GRASS\_PASTURE"

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
TranslateCropCharacteristics (OldCropType="ALFALFA.TR21",NewCropType="ALFALFA")
```

TranslateCropCharacteristics

**TranslateCropCharacteristics() Command Editor**

The command syntax is as follows:

```
TranslateCropCharacteristics(Parameter=value,...)
```

#### Command Parameters

Parameter	Description	Default
OldCropType	A single crop type identifier to match. This crop type will be replaced with the value for NewCropType.	None – must be specified.
NewCropType	The new crop type to use.	None – must be specified.

The following simple example illustrates how to translate a crop type from the more specific name to a more generic name:

```
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")  
TranslateCropCharacteristics(OldCropType="ALFALFA.TR21",NewCropType="ALFALFA")
```

# Command Reference: TranslateCropPatternTS()

Translate crop pattern time series crop types from one value to another

**StateCU Command**

Version 3.09.01, 2010-02-01

The TranslateCropPatternTS ( ) command translates crop pattern time series data. In particular, it converts one crop type to another. Primary uses of the command are:

1. A data source may be using one variant of the crop type (e.g., ORCHARD W/O COVER but the rest of a StateCU data set uses another type (e.g., ORCHARD\_WO\_COVER). In this case the command is used simply to change the spelling of a crop type.
2. The raw crop data may need to be adjusted to reflect differences in crops, for modeling purposes. For example, the original data may identify pasture (e.g., ALFALFA) but for modeling the crop type is set to a different value (e.g., ALFALFA.CCRG) for high altitude coefficients. The following example illustrates a command of this type, using a list file to provide location identifiers at which crop types should be adjusted for the high-altitude crop coefficients (by translating to a different crop type).

If the new crop name is the same as an existing crop name, the time series will be combined to give new totals for the crop. The following dialog is used to edit the command and illustrates the syntax of the command (for the second case listed above):

**Edit TranslateCropPatternTS() Command**

This command changes the crop type in crop pattern time series to a new value.  
This may be necessary because the crop types read from input may not agree with standard types.  
Crop types from parcel data may also need to be changed to values appropriate for model files (e.g., for regional coefficients).  
The CU Locations to change can optionally be specified with an ID pattern OR list file (the default is to change all).

CU Location ID: \* Required if no list file - CU Location(s) to process (use \* for wildcard)

List file: Browse

ID column: Required for list file - column for the CU location IDs.

Old crop type: GRASS\_PASTURE Required - for example: "MEADOW".

New crop type: GRASS\_PASTURE.TR21 Required - for example: "GRASS\_PASTURE".

If not found: Optional - indicate action if no ID match is found (default=Warn).

Command: TranslateCropPatternTS ( ID="\*", OldCropType="GRASS\_PASTURE", NewCropType="GRASS\_PASTURE.TR21" )

Add Working Directory OK Cancel

**TranslateCropPatternTS() Command Editor**

TranslateCropPatternTS

The command syntax is as follows:

```
TranslateCropPatternTS ( Parameter=Value , ... )
```

## Command Parameters

Parameter	Description	Default
ID	A single CU Location identifier to match or a pattern using wildcards (e.g., 20*).	None – must be specified.
ListFile	The name of an input file to read, surrounded by double quotes.	If not specified, crop patterns for all locations will be processed.
IDCol	If ListFile is specified, this parameter specifies the column number (1+) containing the CU Location identifiers.	None – must be specified.
OldCropType	A single crop type identifier to match. This crop type will be replaced with the value for NewCropType.	None – must be specified.
NewCropType	The new crop type to use.	None – must be specified.
IfNotFound	Used for error handling, one of the following: <ul style="list-style-type: none"> <li>Fail – generate a failure message if the ID is not matched</li> <li>Ignore – ignore (don't add and don't generate a message) if the ID is not matched</li> <li>Warn – generate a warning message if the ID is not matched</li> </ul>	Warn

The following command file illustrates how to create a crop pattern time series file:

```
# Step 1 - Set output period and read CU locations
SetOutputPeriod(OutputStart="1950",OutputEnd="2006")
ReadCULocationsFromStateCU(InputFile="..\StateCU\cm2006.str")
# Step 2 - Read SW aggregates
SetDiversionSystemFromList(ListFile="colorado_divsys.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
SetDiversionAggregateFromList(ListFile="colorado_agg.csv",IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow)
# Step 3 - Create *.cds file form and read acreage/crops from HydroBase
CreateCropPatternTSForCULocations(ID="*",Units="ACRE")
ReadCropPatternTSFromHydroBase(ID="*")
# Step 4 - Need to translate crops out of HB to include TR21 suffix
# Translate all crops from HB to include .TR21 suffix
TranslateCropPatternTS(ID="*",OldCropType="GRASS_PASTURE",NewCropType="GRASS_PASTURE.TR21")
TranslateCropPatternTS(ID="*",OldCropType="CORN_GRAIN",NewCropType="CORN_GRAIN.TR21")
TranslateCropPatternTS(ID="*",OldCropType="ALFALFA",NewCropType="ALFALFA.TR21")
...similar commands omitted...
# Step 5 - Translate crop names
# use high-altitude coefficients for structures with more than 50% of
# irrigated acreage above 6500 feet
TranslateCropPatternTS(ListFile="cm2005_HA.lst",IDCol=1,
    OldCropType="GRASS_PASTURE.TR21",NewCropType="GRASS_PASTURE.DWHA")
# Step 6 - Fill Acreage
# Fill SW structure acreage backward from 1999 to 1950
# Fill acreage forward for all structures from 2000 to 2006
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1950,FillEnd=1993,FillDirection=Backward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=1993,FillEnd=1999,FillDirection=Forward)
FillCropPatternTSRepeat(ID="*",CropType="*",FillStart=2000,FillEnd=2006,FillDirection=Forward)
# Step 7 - Write final *.cds file
WriteCropPatternTSToStateCU(OutputFile="..\StateCU\cm2006.cds",
    WriteCropArea=True,WriteHow=OverwriteFile)
# Check the results
CheckCropPatternTS(ID="*")
WriteCheckFile(OutputFile="cm2006.cds.StateDMI.check.html")
```

---

# Command Reference: TranslatePenmanMonteith()

**Translate Penman-Monteith crop types from one value to another**

**StateCU Command**  
Version 3.10.00, 2010-04-02

The `TranslatePenmanMonteith()` command translates Penman-Monteith data. In particular, it converts one crop type to another. Primary uses of the command are:

1. A data source may use one variant of the crop type (e.g., ORCHARD W/O COVER but the rest of a StateCU data set uses another type (e.g., ORCHARD\_WO\_COVER). In this case the command is used simply to change the spelling of a crop type.
2. The raw crop data may need to be adjusted to reflect differences in crops, for modeling purposes. For example, the original data may identify pasture (e.g., GRASS\_PASTURE) but for modeling the crop type is set to a different value (e.g., GRASS\_PASTURE.ASCE) for ASCE standardized coefficients. The following example illustrates a command of this type.

The following dialog is used to edit the command and illustrates the syntax of the command (for the second case listed above):

**Edit TranslatePenmanMonteith() Command**

This command changes the crop type (name) in Penman-Monteith coefficients to a new value. This is sometimes necessary because the crop types read from input may not agree with standard types. Crop types in raw data may also need to be changed to values appropriate for model files (e.g., for regional coefficients).

Old crop type:  Required - for example: "MEADOW"

New crop type:  Required - for example: "GRASS\_PASTURE"

If not found:  Optional - indicate action if no ID match is found (default=Warn).

Command: 

```
TranslatePenmanMonteith(OldCropType="ALFALFA",NewCropType="ALFALFA.ASCE")
```

TranslatePenmanMonteith

**TranslatePenmanMonteith() Command Editor**

The command syntax is as follows:

```
TranslatePenmanMonteith( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
OldCropType	A single crop type identifier to match. This crop type will be replaced with the value for NewCropType.	None – must be specified.
NewCropType	The new crop type to use.	None – must be specified.

The following simple example illustrates how to translate a crop type:

```
ReadPenmanMonteithFromHydroBase( PenmanMonteithMethod= "PENMAN-MONTEITH_ALFALFA" )  
TranslatePenmanMonteith(OldCropType= "ALFALFA" ,NewCropType= "ALFALFA.ASCE" )
```



---

# Command Reference: WriteBlaneyCriddleToList()

**Write Blaney-Criddle crop coefficients data to a delimited file**

**StateCU Command**  
Version 3.08.02, 2010-01-07

The `WriteBlaneyCriddleToList()` command writes Blaney-Criddle crop coefficients data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteBlaneyCriddleToList() Command**

This command writes the StateCU Blaney-Criddle data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteBlaneyCriddleToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteBlaneyCriddleToList

**WriteBlaneyCriddleToList() Command Editor**

The command syntax is as follows:

```
WriteBlaneyCriddleToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to process crop characteristics data from HydroBase:

```
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")  
WriteBlaneyCriddleToList(OutputFile="test.lst")
```

---

# Command Reference: WriteBlaneyCriddleToStateCU()

**Write Blaney-Criddle crop coefficients data to a StateCU file**

## StateCU Command

Version 3.08.02, 2010-01-07

The WriteBlaneyCriddleToStateCU() command writes Blaney-Criddle crop coefficients that have been defined to a StateCU Blaney-Criddle crop coefficients file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteBlaneyCriddleToStateCU() Command**

This command writes the StateCU Blaney-Criddle data to a StateCU Blaney-Criddle file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadBlaneyCriddleFromHydroBase

Output file:

Version:  Optional - indicate StateCU version for format (default=latest).

Precision:  Optional - digits after decimal (default=3).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteBlaneyCriddleToStateCU (OutputFile="rg2007.kbc")
```

WriteBlaneyCriddleToStateCU

### WriteBlaneyCriddleToStateCU() Command Editor

The command syntax is as follows:

```
WriteBlaneyCriddleToStateCU(Parameter=value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Version	The StateCU version, indicating the version of the file format to write.	Write the most current version format.
Precision	The number of digits after the decimal for curve values, used for backward compatibility with older file versions.	3
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example command file illustrates how to read Blaney-Criddle coefficients from HydroBase, sort the data, create a StateCU file, and check the results:

```
StartLog(LogFile="Crops_KBC.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Blaney-Criddle coefficients File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-23 SAM, RTi               Update for Rio Grande Phase 5.
#
# Step 1 - read data from HydroBase
#
# Read the general Blaney-Criddle coefficients first and then override with Rio Grande
# data.
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_TR-21")
ReadBlaneyCriddleFromHydroBase(BlaneyCriddleMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 3 - write the file
#
SortBlaneyCriddle(Order=Ascending)
WriteBlaneyCriddleToStateCU(OutputFile="rg2007.kbc")
#
# Check the results
#
CheckBlaneyCriddle(ID="*")
WriteCheckFile(OutputFile="rg2007.kbc.check.html")
```

---

# Command Reference: WriteCheckFile()

## Write a check file containing a summary of data/processing problems

Version 3.09.00, 2010-01-14

The `WriteCheckFile()` command summarizes the results of command processing warning/failure messages in a “check file”. This file is useful for reviewing results and for quality control. The check file is essentially a persistent record of any problems that occurred during processing, whereas a full log file contains a sequential list of processing. Multiple check commands can be used as appropriate and one or more check files can be written.

The following dialog is used to edit the command and illustrates the syntax for the command, in this case applied to climate stations data.

**Edit WriteCheckFile() Command**

This command writes command warning/failure messages to a check file, as a summary of data/processing problems. Use Check\*() commands prior to this command to perform checks on specific data object types. Specify an "html" extension for the output file to generate an HTML report, or "csv" to create a comma-separated value file. The HTML file will contain navigable information whereas the CSV file will only contain a list of warning/failure messages.

Check (output) file:

Title:  Optional - title for output file.

Command: 

```
WriteCheckFile (OutputFile="COclim2006.cli.check.html", Title="Climate Stations Checks")
```

WriteCheckFile

**WriteCheckFile() Command Editor**

The command syntax is as follows:

```
WriteCheckFile(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	<p>The name of the check file to create, enclosed in double quotes if the file contains spaces or other special characters. A path relative to the command file containing this command can be specified.</p> <p>Specify a filename with <i>.html</i> extension to generate an HTML file or <i>.csv</i> to generate a comma-separated value file suitable for use with Excel. The HTML file will contain more information and include navigation links.</p>	None – must be specified.
Title	A title that will be shown in the output file. This is recommended to provide context for results because the default title uses the command file name.	Auto-generated and includes command file name.

---

# Command Reference: WriteClimateStationsToList()

Write climate station data to a delimited file

**StateCU Command**  
Version 3.08.02, 2010-01-05

The WriteClimateStationsToList() command writes climate stations data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteClimateStationsToList() Command**

This command writes the StateCU climate station data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteClimateStationsToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteClimateStationsToList

**WriteClimateStationsToList() Command Editor**

The command syntax is as follows:

```
WriteClimateStationsToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to process climate data from HydroBase, starting with a list of station identifiers, and creating a full list of climate station data:

```
ReadClimateStationsFromList(ListFile="idonly.csv",IDCol="2")
FillClimateStationsFromHydroBase(ID="*")
WriteClimateStationsToList(OutputFile="COclim2006.csv")
```



---

# Command Reference: WriteClimateStationsToStateCU()

Write climate station data to a StateCU file

**StateCU Command**  
Version 3.08.02, 2010-01-05

The WriteClimateStationsToStateCU() command writes climate stations that have been defined to a StateCU climate stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteClimateStationsToStateCU() Command**

This command writes the StateCU climate station data to a StateCU climate stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteClimateStationsToStateCU

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteClimateStationsToStateCU

**WriteClimateStationsToStateCU() Command Editor**

The command syntax is as follows:

```
WriteClimateStationsToStateCU( Parameter=Value ,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example command file illustrates how climate stations can be defined and written to a StateCU file:

```
ReadClimateStationsFromList(ListFile="climsta.lst",IDCol=1)
FillClimateStationsFromHydroBase(ID="*")
SetClimateStation(ID="3016",Region2="14080106",IfNotFound=Warn)
SetClimateStation(ID="1018",Region2="14040106",IfNotFound=Warn)
SetClimateStation(ID="1928",Elevation=6440,IfNotFound=Warn)
SetClimateStation(ID="0484",Region1="MOFFAT",IfNotFound=Add)
WriteClimateStationsToStateCU(OutputFile="COclim2006.cli")
```

---

# Command Reference: WriteCropCharacteristicsToList()

Write crop characteristics data to a delimited file

**StateCU Command**  
Version 3.08.02, 2010-01-07

The `WriteCropCharacteristicsToList()` command writes crop characteristics data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCropCharacteristicsToList() Command**

This command writes the StateCU crop characteristics data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteCropCharacteristicsToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteCropCharacteristicsToList

**WriteCropCharacteristicsToList() Command Editor**

The command syntax is as follows:

```
WriteCropCharacteristicsToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to process crop characteristics data from HydroBase:

```
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")  
WriteCropCharacteristicsToList(OutputFile="test.csv")
```

# Command Reference: WriteCropCharacteristicsToStateCU()

Write crop characteristics data to a StateCU file

**StateCU Command**  
Version 3.08.02, 2010-01-07

The WriteCropCharacteristicsToStateCU( ) command writes crop characteristics data to a StateCU climate crop characteristics file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCropCharacteristicsToStateCU() Command**

This command writes the StateCU crop characteristics data to a StateCU crop characteristics file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCropCharacteristicsFromHydroBase

Output file:

Version:

Automatically adjust?:

Write how:

Optional - indicate StateCU version for format (default=latest).  
Optional - remove trailing ".xxxx" when Version=10 (default=False).  
Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteCropCharacteristicsToStateCU

## WriteCropCharacteristicsToStateCU() Command Editor

The command syntax is as follows:

WriteCropCharacteristicsToStateCU(Parameter=Value,...)

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Version	Indicate whether output should be formatted for a specific version of StateCU file.	Write the most current version.
AutoAdjust	Automatically adjust the crop names by removing trailing .XXX characters (the period and any trailing characters). This may be needed because current modeling procedures use a longer crop name (e.g.,	False

Parameter	Description	Default
	ALFALFA.TR21) whereas older procedures simply used ALFALFA. The conversion is necessary to allow comparison with older files.	
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example illustrates how to create a StateCU crop characteristics file with data from HydroBase:

```

StartLog(LogFile="Crops_CCH.StateDMI.log")
#
# StateDMI commands to create the Rio Grande Crop Characteristics File
#
# History:
#
# 2004-03-16 Steven A. Malers, RTi   Initial version using StateDMI.
# 2007-04-22 SAM, RTi               Use new directory structure, current
#                                   software and HydroBase.
#
# Step 1 - read data from HydroBase
#
# Read the general TR-21 characteristics first and then override with Rio Grande
# data.
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_TR-21")
ReadCropCharacteristicsFromHydroBase(CUMethod="BLANEY-CRIDDLE_RIO_GRANDE")
#
# Step 2 - adjust crop characteristics if needed
#   No resets are needed.
#
# Step 3 - write the file
#
WriteCropCharacteristicsToStateCU(OutputFile="rg2007.cch")
#
# Check the results
#
CheckCropCharacteristics(ID="*")
WriteCheckFile(OutputFile="rg2007.cch.check.html")

```

---

# Command Reference: WriteCropPatternTSToDateValue()

Write crop pattern time series data to a DateValue file

**StateCU Command**  
Version 3.09.01, 2010-02-01

The `WriteCropPatternTSToDateValue()` command writes crop pattern time series to a DateValue time series file. This file can be used with TSTool, a spreadsheet, or other software. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCropPatternTSToDateValue() Command**

This command writes crop pattern time series to a DateValue time series file, compatible with the TSTool software.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCropPatternTSFromHydroBase

DateValue file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

**WriteCropPatternTSToDateValue() Command Editor**

WriteCropPatternTSToDateValue

The command syntax is as follows:

```
WriteCropPatternTSToDateValue( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteCropPatternTSToStateCU()

Write crop pattern time series data to a StateCU file

## StateCU Command

Version 3.09.01, 2010-02-01

The WriteCropPatternTSToStateCU( ) command writes crop pattern time series to the StateCU crop pattern file. A number of parameters are available to control the format and content of output.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCropPatternTSToStateCU() Command**

This command writes crop pattern time series to a StateCU crop pattern time series file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadCropPatternTSFromHydroBase  
The default is to write the total for each location and fraction for each crop.  
Specifying a value of True for WriteCropArea will write the fraction and corresponding area.  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateCU crop pattern file:

Output start (year):  Optional - output start year as 4-digits (default=output all).

Output end (year):  Optional - output end year as 4-digits (default=output all).

Write crop area:  Optional - writes crop area in addition to fraction of total (default=True).

Write only total:  Optional - write only total, not each crop (default=False).

Version:  Optional - StateCU version for format (default=most current).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteCropPatternTSToStateCU

### WriteCropPatternTSToStateCU() Command Editor

The command syntax is as follows:

```
WriteCropPatternTSToStateCU(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	The starting year for output.	Write the full period.
OutputEnd	The ending year for output.	Write the full period.
WriteCropArea	If specified as True, the crop area for each crop will be written in addition to the percentage of the total area. This is being phased in as a feature of StateCU and this parameter may be removed in the future.	True
WriteOnlyTotal	If specified as True, only the total for the location will be written. This is useful if it is desired to generate an annual total time series file.	False
Version	Indicate the StateCU version file format. An older version format may need to be written when modifying older data sets or comparing current and previous data sets.	Write the must current StateCU version's format.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteCULocationsToList()

Write CU Locations data to a delimited file

**StateCU Command**

Version 3.09.00, 2010-01-24

The WriteCULocationsToList ( ) command writes CU Locations data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCULocationsToList() Command**

This command writes the StateCU CU locations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteCULocationsToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

**WriteCULocationsToList() Command Editor**

WriteCULocationsToList

The command syntax is as follows:

```
WriteCULocationsToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to process CU Locations data from HydroBase, starting with a list of station identifiers, and creating a full list of data:

```
ReadCULocationsFromList(ListFile="list.csv")  
FillCULocationsFromHydroBase(ID="*",CULocType="Structure",Region1Type="County",Region2Type="HUC")  
WriteCULocationsToList(OutputFile="test2.lst")
```

---

# Command Reference: WriteCULocationsToStateCU()

Write CU Location data to a StateCU file

**StateCU Command**  
Version 3.09.00, 2010-01-24

The WriteCULocationsToStateCU( ) command writes CU Locations that have been defined to a StateCU structure file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteCULocationsToStateCU() Command**

This command writes the StateCU CU locations data to StateCU locations (structure) file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillCULocationClimateStationWeights

Output file:

Version:  Optional - indicate StateCU version for format (default=latest).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteCULocationsToStateCU(OutputFile="cm2006.str", WriteHow=OverwriteFile)
```

WriteCULocationsToStateCU

**WriteCULocationsToStateCU() Command Editor**

The command syntax is as follows:

```
WriteCULocationsToStateCU(Parameter=Value...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Version	Indicate the StateCU version, to allow writing file formats for older versions of StateCU.	Write the format for the most current known StateCU version.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteDelayTablesDailyToList()

Write delay tables (daily) data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-04

The `WriteDelayTablesDailyToList()` command writes daily delay tables that have been defined to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDelayTablesDailyToList() Command**

This command writes delay table (daily) data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteDelayTablesDailyToList (OutputFile="cm2005.dld.csv")
```

**WriteDelayTablesDailyToList() Command Editor**

WriteDelayTablesDailyToList

The command syntax is as follows:

```
WriteDelayTablesDailyToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Delimiter	The character used to delimit columns in the file.	, (comma)
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteDelayTablesDailyToStateMod()

Write delay tables (daily) data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The WriteDelayTablesDailyToStateMod( ) command writes daily delay tables that have been defined to a StateMod delay tables file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDelayTablesDailyToStateMod\_Command() Command**

This command writes delay tables (daily) to a StateMod delay tables file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Precision:  Optional - number of digits after decimal (default=2).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command: 

```
WriteDelayTablesDailyToStateMod_Command(OutputFile="cm2005.dld")
```

WriteDelayTablesDailyToStateMod

### WriteDelayTablesDailyToStateMod() Command Editor

The command syntax is as follows:

```
WriteDelayTablesDailyToStateMod( Parameter=Value , ... )
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Precision	The number of digits after the decimal point for data values.	2
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteDelayTablesMonthlyToList()

Write delay tables (monthly) data to a delimited file

**StateMod Command**

Version 3.09.01, 2010-02-04

The `WriteDelayTablesMonthlyToList()` command writes monthly delay tables that have been defined to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDelayTablesMonthlyToList() Command**

This command writes delay table (monthly) data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateRiverNetworkFromNetwork

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteDelayTablesMonthlyToList

**WriteDelayTablesMonthlyToList() Command Editor**

The command syntax is as follows:

```
WriteDelayTablesMonthlyToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Delimiter	The character used to delimit columns in the file.	, (comma)
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteDelayTablesMonthlyToStateMod()

Write delay tables (monthly) data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The WriteDelayTablesMonthlyToStateMod( ) command writes monthly delay tables that have been defined to a StateMod delay tables file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDelayTablesMonthlyToStateMod() Command**

This command writes delay tables (monthly) to a StateMod delay tables file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateRiverNetworkFromNetwork  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Precision:  Optional - number of digits after decimal (default=2).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command: 

```
WriteDelayTablesMonthlyToStateMod(OutputFile="cm2005.dly")
```

WriteDelayTablesMonthlyToStateMod

## WriteDelayTablesMonthlyToStateMod() Command Editor

The command syntax is as follows:

```
WriteDelayTablesMonthlyToStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Precision	The number of digits after the decimal point for data values.	2
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteDiversiionDemandTSMonthlyToStateMod()

Write diversion demand time series (monthly) to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The WriteDiversiionDemandTSMonthlyToStateMod() command writes diversion demand time series (monthly) to a StateMod diversion demand time series file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiionDemandTSMonthlyToStateMod() Command**

This command writes monthly diversion demand time series data to a StateMod file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteDiversiionDemandTSMonthlyToStateMod  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Output start:  Optional - start of output (default=write all).

Output end:  Optional - end of output (default=write all).

Output year type:  Optional - year type (default=Calendar).

Precision:  Optional - number of digits after decimal (default=0).

Missing value:  Optional - missing value indicator (default=-999).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteDiversiionDemandTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005C.ddm")
```

WriteDiversiionDemandTSMonthlyToStateMod

## WriteDiversiionDemandTSMonthlyToStateMod() Command Editor

The command syntax is as follows:

```
WriteDiversionDemandTSMonthlyToStateMod( Parameter=Value , ... )
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	The start date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputEnd	The end date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputYearType	The output year type to write, one of: <ul style="list-style-type: none"> <li>Calendar – January to December.</li> <li>NovToOct – November to October.</li> <li>Water – October to September.</li> </ul>	Calendar, or the value set by the previous SetOutputYearType( ) command.
Precision	The number of digits after the decimal to write.	2
MissingValue	The value to write for missing data.	-999
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteDiversiOnHistoricalTSMonthlyToStateMod()

Write diversion historical time series (monthly) to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The WriteDiversiOnHistoricalTSMonthlyToStateMod( ) command writes diversion historical time series (monthly) to a StateMod diversion historical time series file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiOnHistoricalTSMonthlyToStateMod() Command**

This command writes monthly diversion historical time series data to a StateMod file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversiOnHistoricalTSMonthlyFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Output start:  Optional - start of output (default=write all).

Output end:  Optional - end of output (default=write all).

Output year type:  Optional - year type (default=Calendar).

Precision:  Optional - number of digits after decimal (default=0).

Missing value:  Optional - missing value indicator (default=-999).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command: 

```
WriteDiversiOnHistoricalTSMonthlyToStateMod(OutputFile="..\StateMod\cm2005.ddh")
```

WriteDiversiOnHistoricalTSMonthlyToStateMod

## WriteDiversiOnHistoricalTSMonthlyToStateMod() Command Editor

The command syntax is as follows:

```
WriteDiversiionHistoricalTSMonthlyToStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	The start date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputEnd	The end date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputYearType	The output year type to write, one of: <ul style="list-style-type: none"> <li>Calendar – January to December.</li> <li>NovToOct – November to October.</li> <li>Water – October to September.</li> </ul>	Calendar, or the value set by the previous SetOutputYearType( ) command.
Precision	The number of digits after the decimal to write.	2
MissingValue	The value to write for missing data.	-999
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteDiversiionRightsToList()

Write diversion rights data to a delimited file

StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The WriteDiversiionRightsToList ( ) command writes diversion rights data to a delimited file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiionRightsToList() Command**

This command writes diversion rights data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillDiversiionRight

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteDiversiionRightsToList

**WriteDiversiionRightsToList() Command Editor**

The command syntax is as follows:

```
WriteDiversionRightsToList(param=value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of diversion rights from a list of stations:

```
ReadDiversionStationsFromList(ListFile="test.csv")
ReadDiversionRightsFromHydroBase(ID="*")
WriteDiversionRightsToList(OutputFile="rights.csv")
```

# Command Reference: WriteDiversiionRightsToStateMod()

Write diversion rights data to a StateMod file

StateCU and StateMod Command

Version 3.09.00, 2010-01-26

The WriteDiversiionRightsToStateMod( ) command writes diversion rights that have been defined to a StateMod diversion rights file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiionRightsToStateMod() Command**

This command writes diversion rights data to a StateMod diversion rights file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadDiversiionRightsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteDiversiionRightsToStateMod

**WriteDiversiionRightsToStateMod() Command Editor**

The command syntax is as follows:

WriteDiversiionRightsToStateMod(Parameter=Value,...)

## Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

This page is intentionally blank.

---

# Command Reference: WriteDiversiOnStationsToLiSt()

Write diversion station data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteDiversiOnStationsToLiSt()` command writes diversion stations data to a delimited file. In addition to the main station file, files with suffixes *\_Collections* and *\_ReturnFlows* are written, containing secondary station information.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiOnStationsToLiSt() Command**

This command writes diversion stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillDiversiOnStationsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteDiversiOnStationsToLiSt (OutputFile="cm2005.csv")
```

WriteDiversiOnStationsToLiSt

**WriteDiversiOnStationsToLiSt() Command Editor**

The command syntax is as follows:

```
WriteDiversionStationsToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of diversion stations from a network file:

```
ReadDiversionStationsFromNetwork ( InputFile="cm2005.net" )  
WriteDiversionStationsToList (OutputFile="cm2005.csv" )
```



---

# Command Reference: WriteDiversiionStationsToStateMod()

Write diversion stations data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The WriteDiversiionStationsToStateMod( ) command writes diversion stations that have been defined to a StateMod diversion stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteDiversiionStationsToStateMod() Command**

This command writes diversion stations data to a StateMod diversion stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillDiversiionStationsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteDiversiionStationsToStateMod

### WriteDiversiionStationsToStateMod() Command Editor

The command syntax is as follows:

```
WriteDiversionStationsToStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference:

## WriteInstreamFlowDemandTSAverageMonthlyToStateMod()

Write instream flow demand time series (average monthly) data to a StateMod file

**StateMod Command**  
Version 3.09.01, 2010-02-02

The `WriteInstreamFlowDemandTSAverageMonthlyToStateMod()` command writes instream flow demand time series (average monthly) that have been defined to a StateMod instream flow demand time series (average monthly) file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteInstreamFlowDemandTSAverageMonthlyToStateMod() Command**

This command writes instream flow demand time series (average monthly) data to a StateMod time series file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteInstreamFlowDemandTSAverageMonthlyToStateMod  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Output year type:  Optional - year type (default=Calendar).

Precision:  Optional - number of digits after decimal (default=0).

Missing value:  Optional - missing value indicator (default=-999).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command: 

```
WriteInstreamFlowDemandTSAverageMonthlyToStateMod (OutputFile="..\StateMod\cm2005.ifa")
```

**WriteInstreamFlowDemandTSAverageMonthlyToStateMod() Command Editor**

The command syntax is as follows:

```
WriteInstreamFlowDemandTSAverageMonthlyToStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputYearType	The output year type for the StateMod file.	Calendar
Precision	The number of digits after the decimal point for output values.	0
MissingValue	The value to use in output for missing data.	-999
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteInstreamFlowRightsToList()

Write instream flow rights data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteInstreamFlowRightsToList()` command writes instream flow rights data to a delimited file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteInstreamFlowRightsToList() Command**

This command writes instream flow rights data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadInstreamFlowRightsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteInstreamFlowRightsToList (OutputFile="cm2005.ifr.csv")
```

WriteInstreamFlowRightsToList

**WriteInstreamFlowRightsToList() Command Editor**

The command syntax is as follows:

```
WriteInstreamFlowRightsToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of instream flow rights from a list of stations:

```
ReadInstreamFlowStationsFromList(ListFile="cm2005.ifs.csv")
ReadInstreamFlowRightsFromHydroBase(ID="*")
WriteInstreamFlowRightsToList(OutputFile="cm2005.ifr.csv")
```

---

# Command Reference: WriteInstreamFlowRightsToStateMod()

Write instream flow rights data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteInstreamFlowRightsToStateMod()` command writes instream flow rights that have been defined to a StateMod instream flow rights file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteInstreamFlowRightsToStateMod() Command**

This command writes instream flow rights data to a StateMod instream flow rights file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadInstreamFlowRightsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteInstreamFlowRightsToStateMod(OutputFile="..\STATEMOD\cm2005.ifr")
```

WriteInstreamFlowRightsToStateMod

## WriteInstreamFlowRightsToStateMod() Command Editor

The command syntax is as follows:

```
WriteInstreamFlowRightsToStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteInstreamFlowStationsToList()

Write instream flow station data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteInstreamFlowStationsToList()` command writes instream flow stations data to a delimited file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteInstreamFlowStationsToList() Command**

This command writes instream flow stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillInstreamFlowStationsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteInstreamFlowStationsToList (OutputFile="cm2005.ifs.csv")
```

WriteInstreamFlowStationsToList

**WriteInstreamFlowStationsToList() Command Editor**

The command syntax is as follows:

```
WriteInstreamFlowStationsToList( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of instream flow stations from a network file:

```
ReadInstreamFlowStationsFromNetwork( InputFile="cm2005.net" )  
WriteInstreamFlowStationsToList( OutputFile="cm2005.ifs.csv" )
```

---

# Command Reference: WriteInstreamFlowStationsToStateMod()

Write instream flow stations data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteInstreamFlowStationsToStateMod()` command writes instream flow stations that have been defined to a StateMod instream flow stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteInstreamFlowStationsToStateMod() Command**

This command writes instream flow stations data to a StateMod instream flow stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillInstreamFlowStationsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteInstreamFlowStationsToStateMod

## WriteInstreamFlowStationsToStateMod() Command Editor

The command syntax is as follows:

```
WriteInstreamFlowStationsToStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteIrrigationPracticeTSToDateValue()

Write irrigation practice time series data to a DateValue file

## StateCU Command

Version 01.17.00, 2005-01-20, Color, Acrobat Distiller

The WriteIrrigationPracticeTSToDateValue( ) command writes irrigation practice time series to a DateValue time series file. This file can be used with TSTool, a spreadsheet, or other software. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteIrrigationPracticeTSToDateValue() Command**

This command writes irrigation practice time series data to a DateValue time series file, compatible with the TSTool software. It is recommended that the file be specified using a path relative to the working directory. The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase

DateValue file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteIrrigationPracticeTSToDateValue

## WriteIrrigationPracticeTSToDateValue() Command Editor

The command syntax is as follows:

```
WriteIrrigationPracticeTSToDateValue(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteIrrigationPracticeTSToStateCU()

Write irrigation practice time series data to a StateCU file

**StateCU Command**  
Version 3.09.01, 2010-02-01

The WriteIrrigationPracticeTSToStateCU( ) command writes irrigation practice time series to a StateCU crop pattern file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteIrrigationPracticeTSToStateCU() Command**

This command writes available irrigation practice time series data to a StateCU irrigation practice time series file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadIrrigationPracticeTSFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateCU irrigation practice file:

Output start (year):  Optional - output start year as 4-digits (default=output all).

Output end (year):  Optional - end year as 4-digits (default=output all).

Precision for area:  Optional - default=1 to minimize roundoff errors to .1 acre.

Version:  Optional - default=most current.

One location per file?:  Optional - separate files are useful for verification (default=False).

Check data?:  Optional - check data for integrity (default=True, use False for intermediate output).

Write how:

Command:

WriteIrrigationPracticeTSToStateCU

**WriteIrrigationPracticeTSToStateCU() Command Editor**

The command syntax is as follows:

```
WriteIrrigationPracticeTSToStateCU(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	Starting year for output.	Write all years.
OutputEnd	Ending year for output.	Write all years.
PrecisionForArea	The number of digits after the decimal point for area values.	1
Version	Indicate the StateCU version, to control the file format. It is sometimes necessary to write an older version to compare data sets or update an old data set.	Write the most current format.
OneLocationPerFile	Useful for troubleshooting and verification. If True, then each location is written to a separate file.	False
CheckData	Check the data for integrity (do values add up). Set to False if processing preliminary data.	True
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteNetworkToStateMod()

Write generalized network data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteNetworkToStateMod()` command writes the generalized network to a StateMod XML network file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteNetworkToStateMod() Command**

This command writes the generalized network data to a StateMod file (XML format).  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateNetworkFromRiverNetwork  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteNetworkToStateMod

### WriteNetworkToStateMod() Command Editor

The command syntax is as follows:

```
WriteNetworkToStateMod( Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example command file illustrates how the command might be used:

```
# Create a generalized XML network from individual StateMod files
# Read the network, which contains upstream to downstream connectivity but does
# not indicate node types
ReadRiverNetworkFromStateMod(InputFile=cm2005.rin)
# Read the stations, which imply the node types
ReadRiverStreamGageStationsFromStateMod(InputFile=cm2005.ris)
ReadRiverDiversionStationsFromStateMod(InputFile=cm2005.dds)
ReadRiverReservoirStationsFromStateMod(InputFile=cm2005.res)
ReadRiverInstreamFlowStationsFromStateMod(InputFile=cm2005.ifs)
ReadRiverWellStationsFromStateMod(InputFile=cm2005.wes)
# To be developed...
#ReadRiverPlanStationsFromStateMod()
ReadRiverStreamEstimateStationsFromStateMod(InputFile=cm2005.ris)
# Now create the generalized network, using the connectivity and node types
CreateNetworkFromRiverNetwork()
# Fill in node names and locations from HydroBase, if any is still missing
FillNetworkFromHydroBase()
# Write the generalized network
WriteNetworkToStateMod(OutputFile="cm2005.net")
# Check for errors (the following is not yet implemented)
#CheckNetwork()
WriteCheckFile(OutputFile="cm2005.net.check.html")
```

---

# Command Reference: WritePenmanMonteithToList()

**Write Penman-Monteith crop coefficients data to a delimited file**

**StateCU Command**  
Version 3.10.00, 2010-04-02

The `WritePenmanMonteithToList()` command writes Penman-Monteith crop coefficients data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WritePenmanMonteithToList() Command**

This command writes the StateCU Penman-Monteith data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\commands\WritePenmanMonteithToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

**WritePenmanMonteithToList() Command Editor**

WritePenmanMonteithToList

The command syntax is as follows:

```
WritePenmanMonteithToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to process crop characteristics data from HydroBase:

```
ReadPenmanMonteithFromHydroBase(PenmanMonteithMethod="PENMAN-MONTEITH_ALFALFA")  
WritePenmanMonteithToList(OutputFile="test.csv")
```

---

# Command Reference: WritePenmanMonteithToStateCU()

**Write Penman-Monteith crop coefficients data to a StateCU file**

**StateCU Command**  
Version 3.10.00, 2010-04-02

The `WritePenmanMonteithToStateCU()` command writes Penman-Monteith crop coefficients that have been defined to a StateCU Penman-Monteith crop coefficients file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WritePenmanMonteithToStateCU() Command**

This command writes the StateCU Penman-Monteith data to a StateCU Penman-Monteith file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadPenmanMonteithFromHydroBase

Output file:

Precision:  Optional - digits after decimal (default=3).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WritePenmanMonteithToStateCU(OutputFile="rg2007.kpm")
```

WritePenmanMonteithToStateCU

**WritePenmanMonteithToStateCU() Command Editor**

The command syntax is as follows:

```
WritePenmanMonteithToStateCU(Parameter=value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
Precision	The number of digits after the decimal for curve values, used for backward compatibility with older file versions.	3
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example command file illustrates how to read Penman-Monteith coefficients from HydroBase, sort the data, create a StateCU file, and check the results:

```
StartLog(LogFile="Crops_KPM.StateDMI.log")
#
# StateDMI commands to create the Penman-Monteith crop coefficients file
#
# Step 1 - read data from HydroBase
#
# Read the general ASCE standardized coefficients
ReadPenmanMonteithFromHydroBase(PenmanMonteithMethod="PENMAN-MONTEITH_ALFALFA")
#
# Step 3 - write the file
#
SortPenmanMonteith()
WritePenmanMonteithToStateCU(OutputFile="rg2007.kpm")
#
# Check the results
#
CheckPenmanMonteith(ID="*")
WriteCheckFile(OutputFile="Crops_KPM.StateDMI.check.html")
```

---

# Command Reference: WriteReservoirRightsToList()

Write reservoir rights data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteReservoirRightsToList()` command writes reservoir rights data to a delimited file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteReservoirRightsToList() Command**

This command writes reservoir rights data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadReservoirRightsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteReservoirRightsToList

**WriteReservoirRightsToList() Command Editor**

The command syntax is as follows:

```
WriteReservoirRightsToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of reservoir rights from a list of stations:

```
ReadReservoirStationsFromList(ListFile="cm2005.res.csv",IDCol=1)
ReadReservoirRightsFromHydroBase(ID="*")
WriteReservoirRightsToList(OutputFile="cm2005.rer.csv")
```



---

# Command Reference: WriteReservoirRightsToStateMod()

Write reservoir rights data to a StateMod file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The WriteReservoirRightsToStateMod( ) command writes reservoir rights that have been defined to a StateMod reservoir rights file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteReservoirRightsToStateMod() Command**

This command writes reservoir rights data to a StateMod reservoir rights file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadReservoirRightsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteReservoirRightsToStateMod

**WriteReservoirRightsToStateMod() Command Editor**

The command syntax is as follows:

```
WriteReservoirRightsToStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteReservoirStationsToList()

Write reservoir station data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteReservoirStationsToList()` command writes reservoir stations data to a delimited file. In addition to the main station file, files with suffixes *\_Collections*, *\_Accounts*, *\_ContentAreaSeepage*, *\_EvapStations*, and *\_PrecipStations* are written, containing secondary station information.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteReservoirStationsToList() Command**

This command writes reservoir stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillReservoirStationsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  
`WriteReservoirStationsToList (OutputFile="rg2007.csv")`

WriteReservoirStationsToList

**WriteReservoirStationsToList() Command Editor**

The command syntax is as follows:

```
WriteReservoirStationsToList( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of diversion stations from a network file:

```
ReadReservoirStationsFromNetwork( InputFile="rg2007.net" )  
WriteReservoirStationsToList( OutputFile="rg2007.csv" )
```

---

# Command Reference: WriteReservoirStationsToStateMod()

Write reservoir stations data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteReservoirStationsToStateMod()` command writes reservoir stations that have been defined to a StateMod reservoir stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteReservoirStationsToStateMod() Command**

This command writes reservoir stations data to a StateMod reservoir stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillReservoirStationsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteReservoirStationsToStateMod

### WriteReservoirStationsToStateMod() Command Editor

The command syntax is as follows:

```
WriteReservoirStationsToStateMod( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteRiverNetworkToList()

Write river network data to a delimited file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteRiverNetworkToList()` command writes river network data to a delimited file. It is often more useful to write lists of individual station types. Consequently, see commands like `WriteDiversionStationsToList()`.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteRiverNetworkToList() Command**

This command writes the StateMod river network data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateRiverNetworkFromNetwork

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteRiverNetworkToList (OutputFile="cm2005.rin.csv")
```

WriteRiverNetworkToList

### WriteRiverNetworkToList() Command Editor

The command syntax is as follows:

```
WriteRiverNetworkToList( Parameter=Value, ...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of river network nodes from a network file:

```
ReadNetworkFromStateMod( InputFile="cm2005.net" )  
CreateRiverNetworkFromNetwork( )  
WriteRiverNetworkToList( OutputFile="cm2005.rin.csv" )
```



# Command Reference: WriteRiverNetworkToStateMod()

Write StateMod river network data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteRiverNetworkToStateMod()` command writes the river network to a StateMod river network file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteRiverNetworkToStateMod() Command**

This command writes the StateMod river network data to a StateMod river network file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CreateRiverNetworkFromNetwork  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteRiverNetworkToStateMod

**WriteRiverNetworkToStateMod() Command Editor**

The command syntax is as follows:

```
WriteRiverNetworkToStateMod(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following command file illustrates how a StateMod river network file can be created from the generalized network file:

```
StartLog(LogFile="rin.commands.StateDMI.log")
# rin.commands.StateDMI
#
# creates the river network file for the Colorado River monthly/daily models
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadNetworkFromStateMod(InputFile="cm2005.net")
CreateRiverNetworkFromNetwork()
#
# Step 2 - get node (diversion, stream stations, reservoirs, instream flows)
#           names from HydroBase
#
FillRiverNetworkFromHydroBase(ID="*",NameFormat=StationName_NodeType)
#
# Step 3 - read missing node names from network file
#
FillRiverNetworkFromNetwork(ID="*",NameFormat="StationName_NodeType",
    CommentFormat="StationID")
#
# Step 4 - create StateMod river network file
#
WriteRiverNetworkToStateMod(OutputFile="..\StateMod\cm2005.rin")
#
# Check the results
CheckRiverNetwork(ID="*")
WriteCheckFile(OutputFile="rin.commands.StateDMI.check.html")
```

---

# Command Reference: WriteStreamEstimateCoefficientsToList()

Write stream estimate coefficients data to a delimited file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The `WriteStreamEstimateCoefficientsToList()` command writes stream estimate coefficient data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamEstimateCoefficientsToList() Command**

This command writes stream estimate coefficients data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CalculateStreamEstimateCoefficients

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteStreamEstimateCoefficientsToList (OutputFile="cm2005.rib.csv")
```

WriteStreamEstimateCoefficientsToList

**WriteStreamEstimateCoefficientsToList() Command Editor**

The command syntax is as follows:

```
WriteStreamEstimateCoefficientsToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

# Command Reference: WriteStreamEstimateCoefficientsToStateMod()

Write stream estimate coefficients data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteStreamEstimateCoefficientsToStateMod()` command writes stream estimate coefficients that have been defined to a StateMod stream estimate coefficients file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamEstimateCoefficientsToStateMod() Command**

This command writes stream estimate coefficients data to a StateMod stream estimate coefficients file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CalculateStreamEstimateCoefficients  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteStreamEstimateCoefficientsToStateMod

## WriteStreamEstimateCoefficientsToStateMod() Command Editor

The command syntax is as follows:

```
WriteStreamEstimateCoefficientsToStateMod(Parameter=Value,...)
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following command file illustrates how a StateMod stream estimate coefficients file can be created:

```

StartLog(LogFile="rib.commands.StateDMI.log")
# rib.commands.StateDMI
#
# Creates the Stream Estimate Station Coefficient Data file
#
# Step 1 - read river nodes from the network file and create file framework
#
ReadStreamEstimateStationsFromNetwork(InputFile="..\Network\cm2005.net")
#
# Step 2 - set preferred gages for "neighboring" gage approach
#           this baseflow nodes are generally on smaller non-gaged tribs and have
#           different flow characteristics than next downstream gages
#
SetStreamEstimateCoefficientsPFGage(ID="360645",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="360801",GageID="09055300")
SetStreamEstimateCoefficientsPFGage(ID="362002",GageID="09054000")
SetStreamEstimateCoefficientsPFGage(ID="360829",GageID="09047500")
..similar commands omitted...
#
# Step 3 - calculate stream coefficients
CalculateStreamEstimateCoefficients()
#
# Step 4 - set proration factors directly
#
SetStreamEstimateCoefficients(ID="364512",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374641",ProrationFactor=0.200,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="374648",ProrationFactor=0.350,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="380880",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="381594",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="384617",ProrationFactor=0.700,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510639",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514603",ProrationFactor=0.800,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="514620",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="510728",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530555",ProrationFactor=0.180,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="530678",ProrationFactor=0.230,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="531082",ProrationFactor=1.000,IfNotFound=Warn)
SetStreamEstimateCoefficients(ID="954683",ProrationFactor=0.400,IfNotFound=Warn)
#
# Step 5 - create streamflow estimate coefficient file
#
WriteStreamEstimateCoefficientsToStateMod(OutputFile="..\StateMOD\cm2005.rib")
#
# Check the results
CheckStreamEstimateCoefficients(ID="*")
WriteCheckFile(OutputFile="rib.commands.StateDMI.check.html")

```

---

# Command Reference: WriteStreamEstimateStationsToList()

Write stream estimate station data to a delimited file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `WriteStreamEstimateStationsToList()` command writes stream estimate stations data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamEstimateStationsToList() Command**

This command writes stream estimate stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamEstimateStationsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:

WriteStreamEstimateStationsToList

**WriteStreamEstimateStationsToList() Command Editor**

The command syntax is as follows:

```
WriteStreamEstimateStationsToList( Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of stream estimate stations from a network file:

```
ReadStreamEstimateStationsFromNetwork( InputFile="rgTW.net" )  
WriteStreamEstimateStationsToList( OutputFile="rgTW.ses.csv" )
```



---

# Command Reference: WriteStreamEstimateStationsToStateMod()

Write stream estimate stations data to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteStreamEstimateStationsToStateMod()` command writes stream estimate stations that have been defined to a StateMod stream estimate stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamEstimateStationsToStateMod() Command**

This command writes stream estimate stations data to a StateMod stream estimate stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamEstimateStationsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteStreamEstimateStationsToStateMod

## WriteStreamEstimateStationsToStateMod() Command Editor

The command syntax is as follows:

```
WriteStreamEstimateStationsToStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteStreamGageStationsToList()

Write stream gage station data to a delimited file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `WriteStreamGageStationsToList()` command writes stream gage stations data to a delimited file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamGageStationsToList() Command**

This command writes stream gage stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamGageStationsFromHydroBase

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteStreamGageStationsToList (OutputFile="cm2005.ris.csv")
```

WriteStreamGageStationsToList

**WriteStreamGageStationsToList() Command Editor**

The command syntax is as follows:

```
WriteStreamGageStationsToList( Parameter=Value ,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of stream gage stations from a network file:

```
ReadStreamGageStationsFromNetwork( InputFile="cm2005.net" )  
WriteStreamGageStationsToList( OutputFile="cm2005.ris.csv" )
```

# Command Reference: WriteStreamGageStationsToStateMod()

Write stream gage stations data to a StateMod file

**StateMod Command**  
Version 3.09.01, 2010-02-01

The WriteStreamGageStationsToStateMod( ) command writes stream gage stations that have been defined to a StateMod stream gage stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteStreamGageStationsToStateMod() Command**

This command writes stream gage stations data to a StateMod stream gage stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillStreamGageStationsFromHydroBase  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
```

WriteStreamGageStationsToStateMod

**WriteStreamGageStationsToStateMod() Command Editor**

The command syntax is as follows:

```
WriteStreamGageStationsToStateMod(Parameter=Value,...)
```

## Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

The following example command file illustrates the commands used to read stream gage stations from the network and create a StateMod file:

```

StartLog(LogFile="ris.commands.StateDMI.log")
# ris.commands.StateDMI
#
# StateDMI command file to create streamflow station file for the Colorado River
#
# Step 1 - read streamgages and baseflows ids from the network file
#
ReadStreamGageStationsFromNetwork(InputFile="..\Network\cm2005.net",
    IncludeStreamEstimateStations="True")
#
# Step 2 - read baseflow nodes names from HydroBase,
#           fill in missing names from the network file
#
FillStreamGageStationsFromHydroBase(ID="*",NameFormat=StationName,CheckStructures=True)
FillStreamGageStationsFromNetwork(ID="*",NameFormat="StationName")
#
# Step 3 - set streamgage station to use to disaggregate monthly baseflows to daily
#
# add set daily pattern gages for WD 36
SetStreamGageStation(ID="36*",DailyID="09047500",IfNotFound=Warn)
...many similar commands omitted...
#
# Step 4 - create streamflow station file
#
WriteStreamGageStationsToStateMod(OutputFile="..\StateMod\cm2005.ris")
#
# Check the results
CheckStreamGageStations(ID="*")
WriteCheckFile(OutputFile="ris.commands.StateDMI.check.html")

```

---

# Command Reference: WriteWellDemandTSMonthlyToStateMod()

Write well demand time series (monthly) to a StateMod file

## StateMod Command

Version 3.09.01, 2010-02-01

The `WriteWellDemandTSMonthlyToStateMod()` command writes well demand time series (monthly) to a StateMod well demand time series file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellDemandTSMonthlyToStateMod() Command**

This command writes monthly well demand time series data to a StateMod file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Output start:  Optional - start of output (default=write all).

Output end:  Optional - end of output (default=write all).

Output year type:  Optional - year type (default=Calendar).

Precision:  Optional - number of digits after decimal (default=2).

Missing value:  Optional - missing value indicator (default=-999).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command: 

```
WriteWellDemandTSMonthlyToStateMod(OutputFile="rgTW.wem")
```

WriteWellDemandTSMonthlyToStateMod

## WriteWellDemandTSMonthlyToStateMod() Command Editor

The command syntax is as follows:

```
WriteWellDemandTSMonthlyToStateMod( Parameter=Value, ... )
```

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	The start date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputEnd	The end date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputYearType	The output year type to write, one of: <ul style="list-style-type: none"> <li>Calendar – January to December.</li> <li>NovToOct – November to October.</li> <li>Water – October to September.</li> </ul>	Calendar, or the value set by the previous SetOutputYearType( ) command.
Precision	The number of digits after the decimal to write.	2
MissingValue	The value to write for missing data.	-999
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile



---

# Command Reference: WriteWellHistoricalPumpingTSMonthlyToStateMod()

Write well historical pumping time series (monthly) to a StateMod file

## StateCU and StateMod Command

Version 3.09.01, 2010-01-27

The `WriteWellHistoricalPumpingTSMonthlyToStateMod()` command writes well historical pumping time series (monthly) to a StateMod time series file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellHistoricalPumpingTSMonthlyToStateMod() Command**

This command writes well historical pumping time series (monthly) data to a StateMod file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Output start:  Optional - start of output (default=write all).

Output end:  Optional - end of output (default=write all).

Output year type:  Optional - year type (default=Calendar).

Precision:  Optional - number of digits after decimal (default=2).

Missing value:  Optional - missing value indicator (default=-999).

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:  

```
WriteWellHistoricalPumpingTSMonthlyToStateMod(OutputFile="rg2007.weh")
```

WriteWellHistoricalPumpingTSMonthlyToStateMod

## WriteWellHistoricalPumpingTSMonthlyToStateMod() Command Editor

The command syntax is as follows:

```
WriteWellHistoricalPumpingTSMonthlyToStateMod( Parameter=Value, ... )
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
OutputStart	The start date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputEnd	The end date to write, using format YYYY-MM or MM/YYYY.	Full period will be written.
OutputYearType	The output year type to write, one of: <ul style="list-style-type: none"> <li>Calendar – January to December.</li> <li>NovToOct – November to October.</li> <li>Water – October to September.</li> </ul>	Calendar, or the value set by the previous SetOutputYearType( ) command.
Precision	The number of digits after the decimal to write.	2
MissingValue	The value to write for missing data.	-999
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Command Reference: WriteWellRightsToList()

Write well rights data to a delimited file

**StateCU and StateMod Command**

Version 3.09.00, 2010-01-25

The `WriteWellRightsToList()` command writes well rights data to a delimited file.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellRightsToList() Command**

This command writes well rights data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\WriteWellRightsToList

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteWellRightsToList (OutputFile="Sp2008L.wer.csv")
```

WriteWellRightsToList

**WriteWellRightsToList() Command Editor**

The command syntax is as follows:

```
WriteWellRightsToList(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of well rights from a list of stations:

```
ReadWellStationsFromList(ListFile="test.lst")
ReadWellRightsFromHydroBase(ID="*")
WriteWellRightsToList(OutputFile="rights.lst")
```

# Command Reference: WriteWellRightsToStateMod()

Write well rights data to a StateMod file

## StateMod Command

Version 3.09.00, 2010-01-25

The WriteWellRightsToStateMod( ) command writes well rights to a StateMod well rights file. The current in-memory rights are written. See also the MergeWellRights( ) and AggregateWellRights( ) commands.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellRightsToStateMod() Command**

This command writes well rights data to a StateMod well rights file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\ReadWellStationsFromNetwork  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Write data comments?:  Optional - write parcel year, well matching class, parcel ID (default=False)

Command:  
`WriteWellRightsToStateMod (OutputFile="Sp2008L.wer", WriteDataComments=False, WriteHow=OverwriteFile)`

WriteWellRightsToStateMod

WriteWellRightsToStateMod() Command Editor

The command syntax is as follows:

WriteWellRightsToStateMod(Parameter=Value,...)

### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
WriteDataComments	Write comments to the right of normal data, including the parcel year, parcel/well matching class, and parcel ID. This information is necessary to fill irrigation practice and crop pattern time series with well water rights. Typically, a “_NotMerged.wer” well right file is written and then merged and possibly aggregated rights files are written.	False

An excerpt from a well rights file with data comments is shown below:

#>	ID	Name	Struct	Admin #	Decree	On/Off	PYr--Cls--PID
#>-----eb-----eb-----eb-----eb-----eb-----eb-----exb--exb--exb-----e							
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1936 1 3107
2005001	W0006	WELL NO 01	200812	38836.00000	1.23	1956	1936 1 3107
2005001	W0006	WELL NO 01	200812	31592.00000	2.34	1936	1998 2 11016
2005001	W0006	WELL NO 01	200812	31592.00000	1.15	1936	2002 5 20902
2005001	W0006	WELL NO 01	200812	38836.00000	0.61	1956	2002 5 20902
...							

The following example command file illustrates how well rights can be defined, sorted, checked, and written to a StateMod file:

```
# Well Rights File (*.wer)
StartLog(LogFile="Sp2008L_WER.log")
#
# Step 1 - Read all structures
ReadWellStationsFromNetwork(InputFile="..\Network\Sp2008L.net")
#
# Step 2 - define diversion and d&w aggregates and demand systems
SetWellAggregateFromList(ListFile="..\Sp2008L_SWAgg.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InColumn,IfNotFound=Warn)
SetWellSystemFromList(ListFile="..\Sp2008L_DivSys_DDH.csv",PartType=Ditch,IDCol=1,
    NameCol=2,PartIDsCol=3,PartsListedHow=InRow,IfNotFound=Warn)
#
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow)
#
# Step 3- Set Well aggregates (GW Only lands)
# rrb Same as provided by LRE as Sp_GWAgg_xxxx.csv except non WD 01 and 64 removed
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1956.csv",Year=1956,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1976.csv",Year=1976,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_1987.csv",Year=1987,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2001.csv",Year=2001,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
SetWellSystemFromList(ListFile="..\Sp2008L_GWAgg_2005.csv",Year=2005,Div=1,
    PartType=Parcel,IDCol=1,PartIDsCol=2,PartsListedHow=InColumn)
#
# Step 4 - Read Augmentation and Recharge Well Aggregate Parts
SetWellAggregateFromList(ListFile="Sp2008L_AugRchWell_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=25,IfNotFound=Ignore)
SetWellAggregateFromList(ListFile="Sp2008L_AlternatePoint_Aggregates.csv",PartType=Well,
    IDCol=1,PartIDsCol=2,PartsListedHow=InRow,PartIDsColMax=1,IfNotFound=Ignore)
#
# Step 5 - Read rights from HydroBase
ReadWellRightsFromHydroBase(ID="*",IDFormat="HydroBaseID",Year="1956,1976,1987,2001,2005",
    Div="1",DefaultAppropriationDate="1950-01-01",DefineRightHow=RightIfAvailable,
    ReadWellRights=True,UseApex=True,OnOffDefault=AppropriationDate)
#
# Step 6 - Sort and Write
# Write Data Comments="True" provides output used for subsequent cds & ipy acreage filling
# Write Data Comments="False" provides merged file used for setting ipy max pumping
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L_NotMerged.wer",WriteDataComments=True)
MergeWellRights(OutputFile="..\StateMod\Historic\Sp2008L.wer")
SortWellRights(Order=LocationIDAscending,Order2=IDAscending)
#
WriteWellRightsToStateMod(OutputFile="Sp2008L.wer",WriteDataComments=False,WriteHow=OverwriteFile)
# Check the well rights
CheckWellRights(ID="*")
WriteCheckFile(OutputFile="Sp2008L.wer.check.html",Title="Well Rights Check File")
```

---

# Command Reference: WriteWellStationsToList()

Write well station data to a delimited file

**StateMod Command**

Version 3.09.01, 2010-02-01

The `WriteWellStationsToList()` command writes well stations data to a delimited file. In addition to the main station file, files with suffixes *\_Collections*, *\_Depletions*, and *\_ReturnFlows* are written, containing secondary station information.

The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellStationsToList() Command**

This command writes well stations data to a delimited list file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is:  
C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\FillWellStationsFromNetwork

Output file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Field delimiter:  Optional - delimiter between columns (default=comma).

Command:  

```
WriteWellStationsToList (OutputFile="rgtw.wes.csv")
```

WriteWellStationsToList

**WriteWellStationsToList() Command Editor**

The command syntax is as follows:

```
WriteWellStationsToList (Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile
Delimiter	The delimiter character to use between columns.	Comma

The following example illustrates how to create a list of well stations from a network file:

```
ReadWellStationsFromNetwork (InputFile="rgtw.net")  
WriteWellStationsToList (OutputFile="rgtw.wes.csv")
```



---

# Command Reference: WriteWellStationsToStateMod()

Write well stations data to a StateMod file

## StateCU StateMod Command

Version 3.09.01, 2010-01-27

The WriteWellStationsToStateMod() command writes well stations that have been defined to a StateMod well stations file. The following dialog is used to edit the command and illustrates the syntax of the command.

**Edit WriteWellStationsToStateMod() Command**

This command writes well stations data to a StateMod well stations file.  
It is recommended that the file be specified using a path relative to the working directory.  
The working directory is: C:\Develop\StateDMI\_SourceBuild\StateDMI\test\regression\UserManualRef\CheckWellHistoricalPumpingTSMonthly  
The default value for "Write how" is OverwriteFile, which will create a new file, overwriting an old file if it exists.

StateMod file:

Write how:  Optional - indicate whether to overwrite/update (default=OverwriteFile).

Command:

WriteWellStationsToStateMod

**WriteWellStationsToStateMod() Command Editor**

The command syntax is as follows:

```
WriteWellStationsToStateMod(Parameter=Value,...)
```

#### Command Parameters

Parameter	Description	Default
OutputFile	The name of the output file to write, surrounded by double quotes.	None – must be specified.
WriteHow	OverwriteFile if the file should be overwritten or UpdateFile if the file should be updated, resulting in the previous header being carried forward.	OverwriteFile

---

# Appendix: StateDMI Installation and Configuration

Version 03.09.01, 2010-02-15

## 1. Overview

This appendix describes how to install StateDMI in the CDSS (Colorado's Decision Support Systems) environment. CDSS consists of the HydroBase database, modeling, and data viewing/editing software. StateDMI can be used within this system to process data from the HydroBase database, CDSS model files, and other files.

## 2. Installing StateDMI as Part of CDSS

Standard locations of StateDMI software files are as follows. Files are normally installed on Windows on the C: drive but can be installed in a shared location on a server.

<code>\CDSS\StateDMI-Version</code>	Top-level install directory.
<code>bin\</code>	Software directory for <i>StateDMI.bat</i> file and Java JAR files.
<code>batik*.jar</code>	Scalable Vector Graphics (SVG) output packages.
<code>Blowfish*.jar</code>	Used for encryption/security.
<code>cdss.domain*.jar</code>	CDSS components.
<code>HydroBaseDMI*.jar</code>	State of Colorado HydroBase database interface package.
<code>jcommon.jar, jfreechart.jar</code>	Plotting package.
<code>jsr173_1.0_api.jar,</code>	XML support.
<code>libXMLJava.jar</code>	
<code>jython.jar</code>	Jython support.
<code>sqljdbc4.jar</code>	Microsoft SQL Server packages.
<code>RTi_Common*.jar</code>	Riverside Technology, inc. supporting packages.
<code>SatmonSysDMI*.jar</code>	State of Colorado Satellite Monitoring System package.
<code>shellcon.exe</code>	Executable program used to read from the Windows registry (e.g., to determine the default web browser and list available ODBC data source names). – PHASING OUT.
<code>StateDMI.exe</code>	Executable program to run StateDMI using the JRE software.
<code>StateDMI.l4j.ini</code>	Configuration file for <i>StateDMI.exe</i> launcher.
<code>StateDMI*.jar</code>	StateDMI program components.
<code>StateMod*.jar</code>	State of Colorado's StateMod and StateCU model packages.
<code>TSCCommandProcessor*.jar</code>	Time series command processor package.

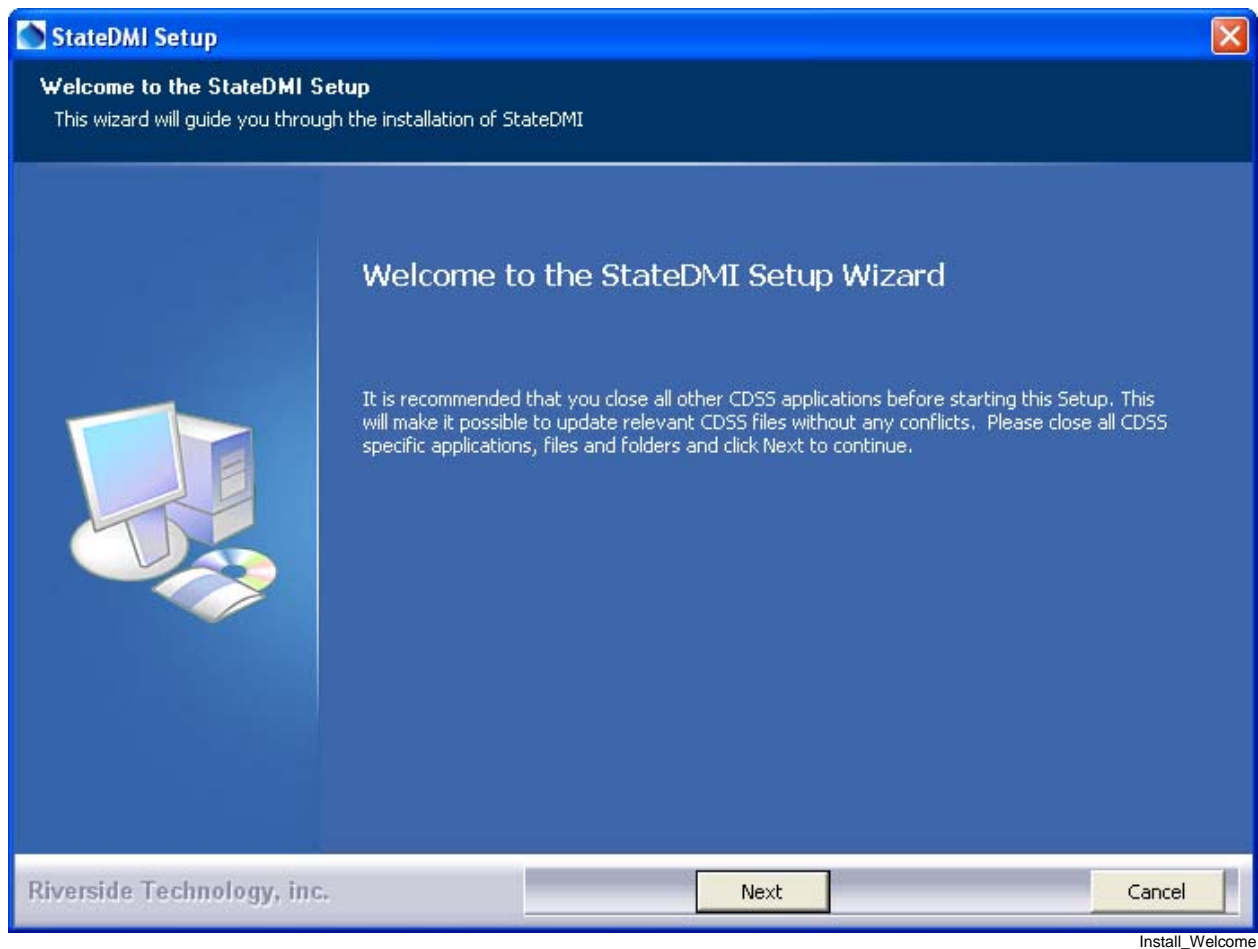
<i>doc\StateDMI\UserManual\</i>	Main documentation directory for StateDMI.
<i>StateDMI.pdf</i>	StateDMI documentation as PDF.
<i>jre*\</i>	Java Runtime Environment (JRE) used by StateDMI.
<i>logs\</i>	Directory for StateDMI log files (should be writable).
<i>system\</i>	Directory for system files.
<i>CDSS.cfg</i>	CDSS configuration file for HydroBase database connection.
<i>DATAUNIT</i>	Data units file.
<i>StateDMI.cfg</i>	Configuration file to modify StateDMI defaults (under development).

### 3. Installing StateDMI

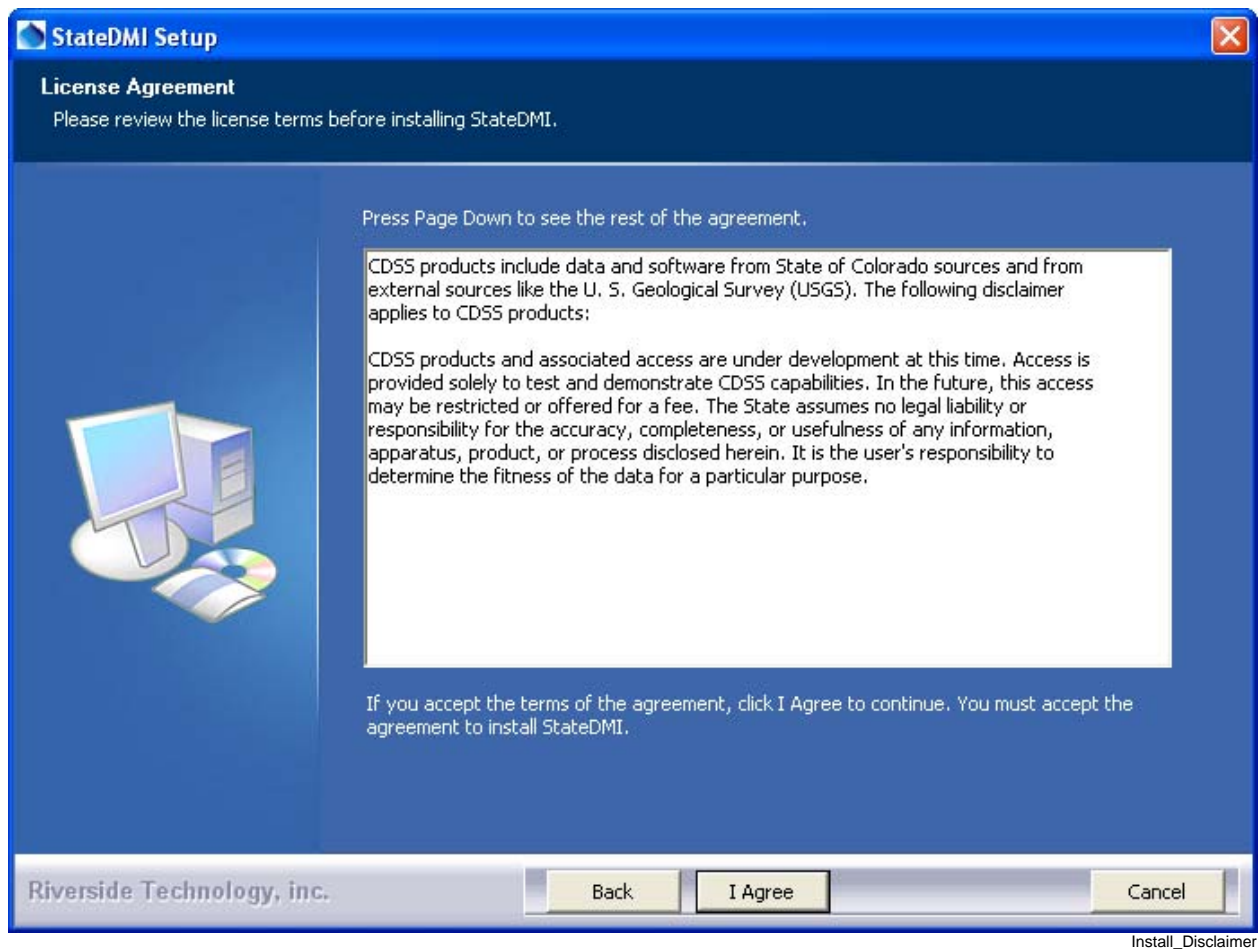
Use the following instructions to install StateDMI using the *StateDMI\_CDSS\_Version\_Setup.exe* installer program, for example if StateDMI software was downloaded from the CDSS web site (<http://cdss.state.co.us>):

1. Run the *StateDMI\_CDSS\_Version\_Setup.exe* file by selecting from Windows Explorer, the **Start...Run...** menu, or from a command shell. The setup filename will include a version number (e.g., *StateDMI\_CDSS\_03.09.01\_Setup.exe*).

You must be logged into the computer using an account with administrator privileges. If you have administrative privileges, the following welcome will be displayed, and the installation can continue:

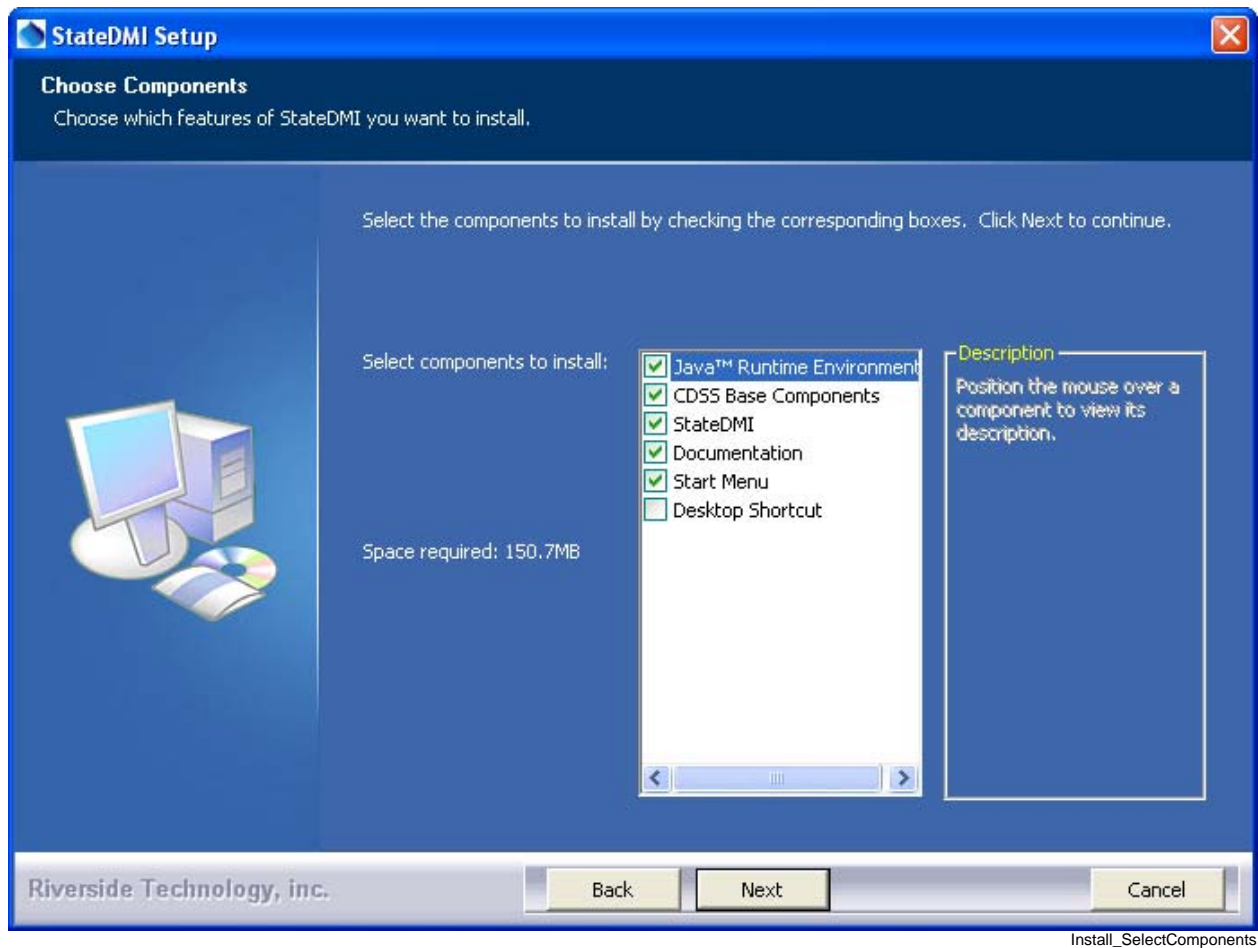


Press **Next** to continue with the installation.



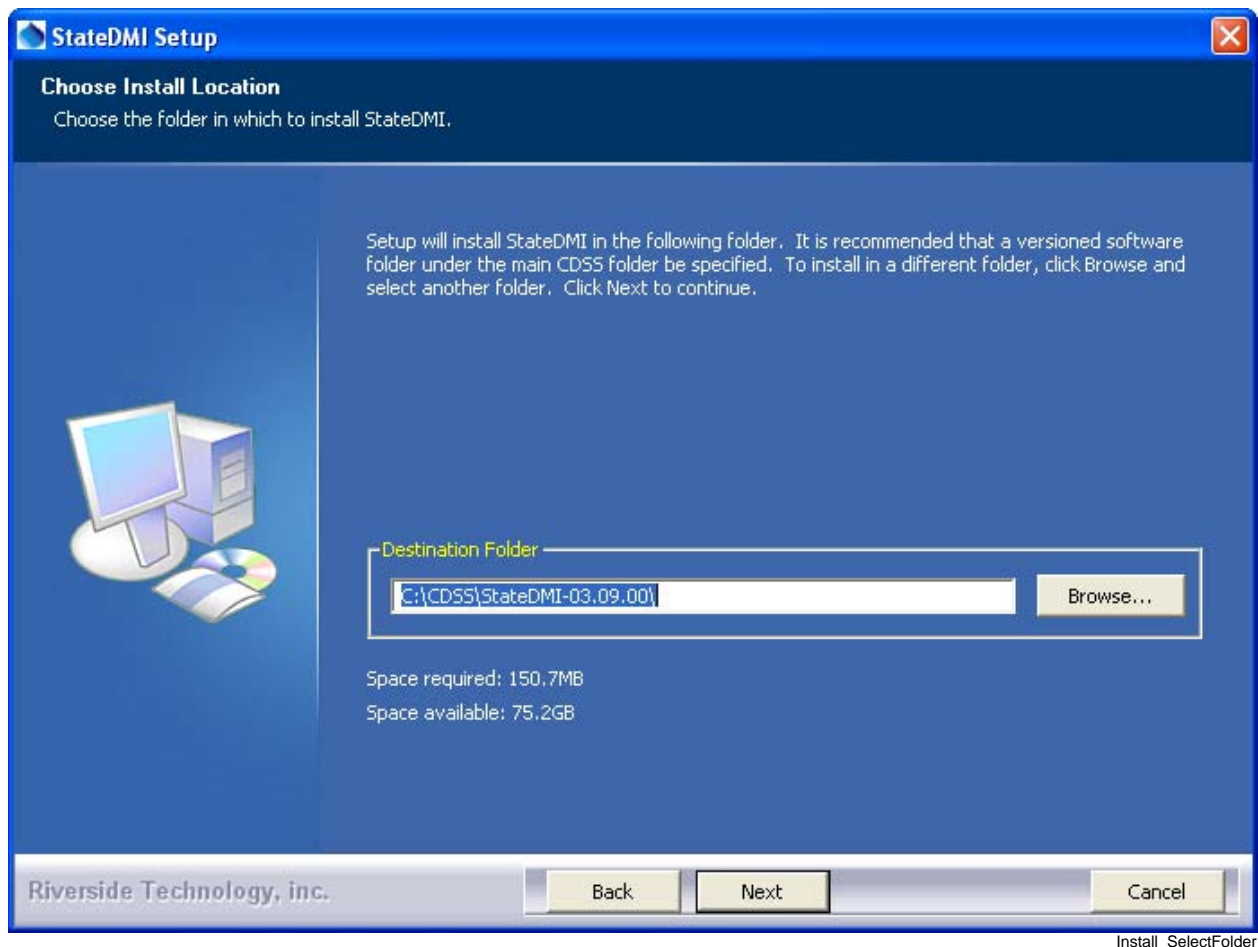
StateDMI is distributed with CDSS with no license restrictions. However the disclaimer must be acknowledged. Press **I Agree** to continue with the installation.

2. Several components can be selected for the install as shown in the following dialog. Position the mouse over a component to see its description.



Select the components to install and press **Next**.

3. The following dialog is then shown and is used to select the installation location for StateDMI. Multiple versions of StateDMI can be installed and there are no dependencies between the versions. It is recommended that the default install location shown is used.

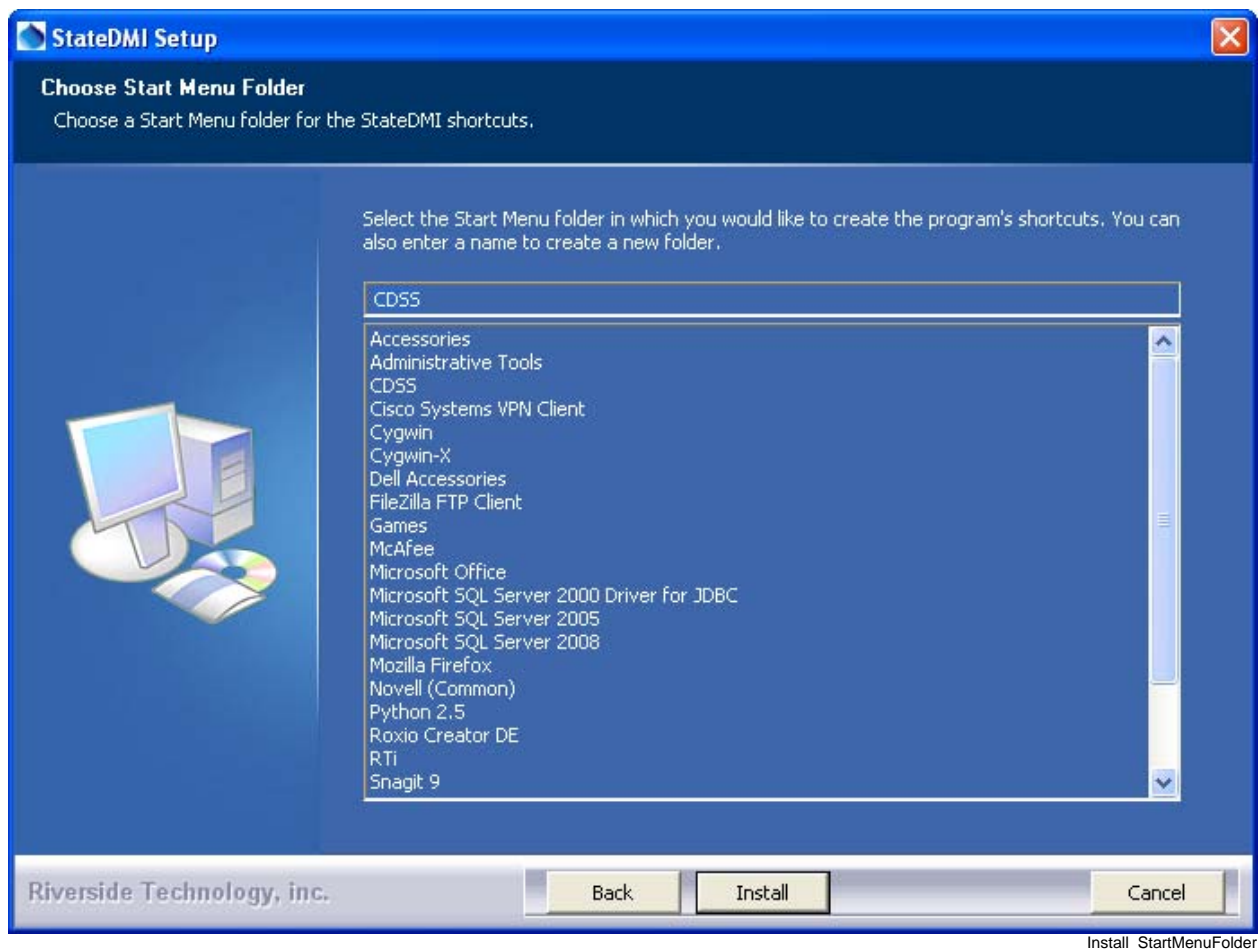


After selecting the install location, press **Next**.

Note that this location will be saved as a Windows registry setting (*HKEY\_LOCAL\_MACHINE\Software\State of Colorado\StateDMI-Version\Path*) to allow future updates to check for and default to the same install location, and to allow the standard software uninstall procedure to work correctly.

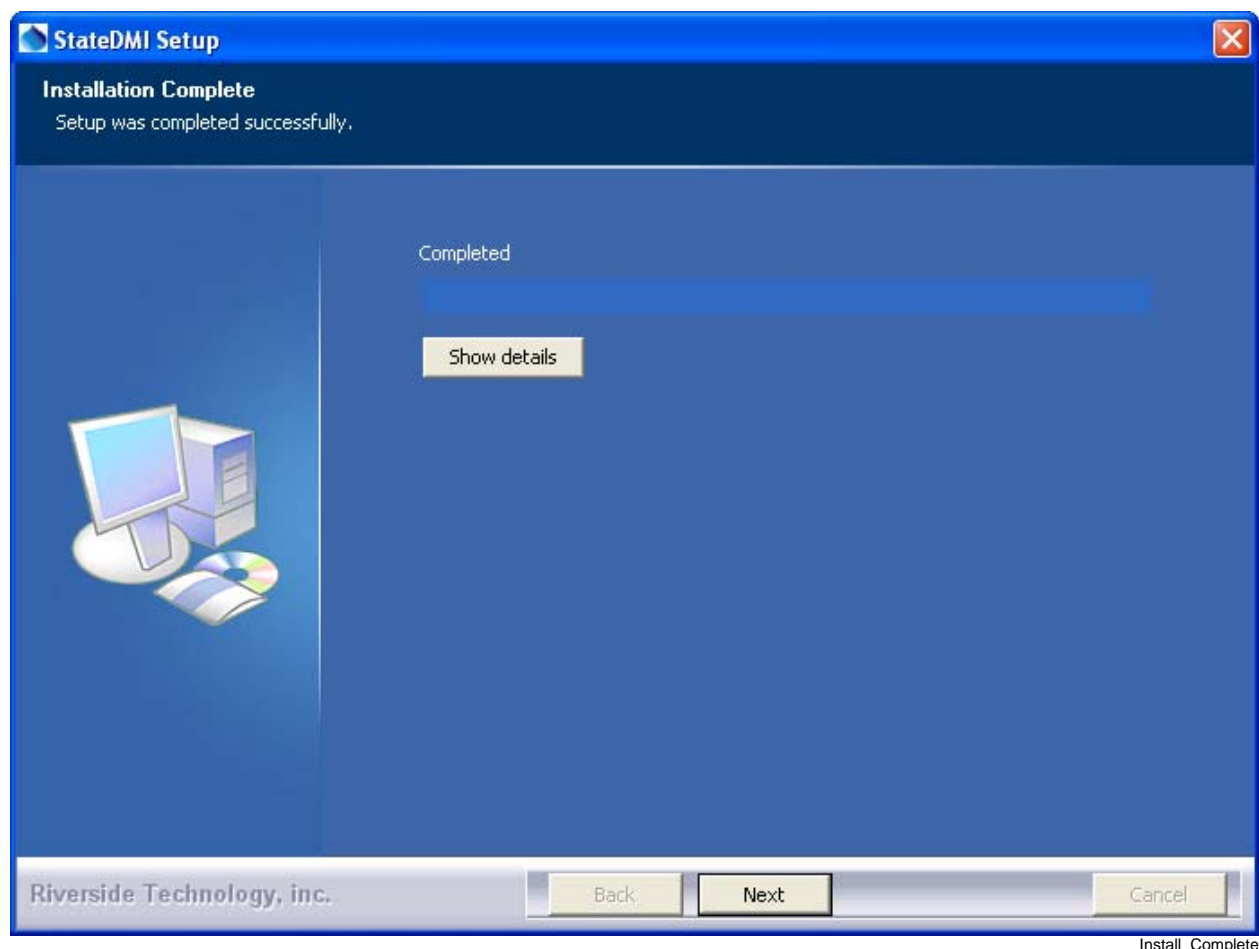


4. The following dialog will be shown to select the menu for the software:



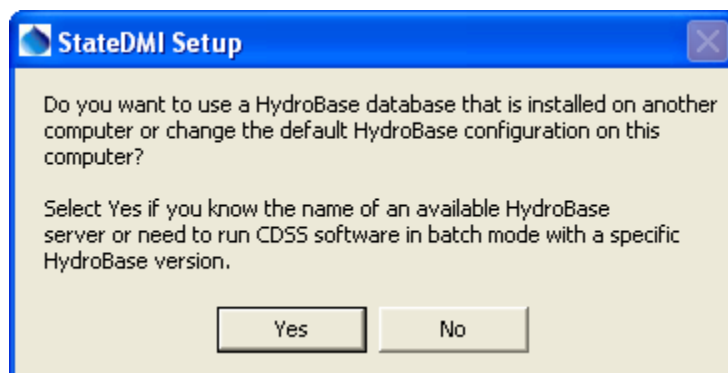
After selecting the folder, press ***Install***.

5. The following dialog will show the progress of the installation:



Press **Show details** to see the files that were installed or press **Next** to continue.

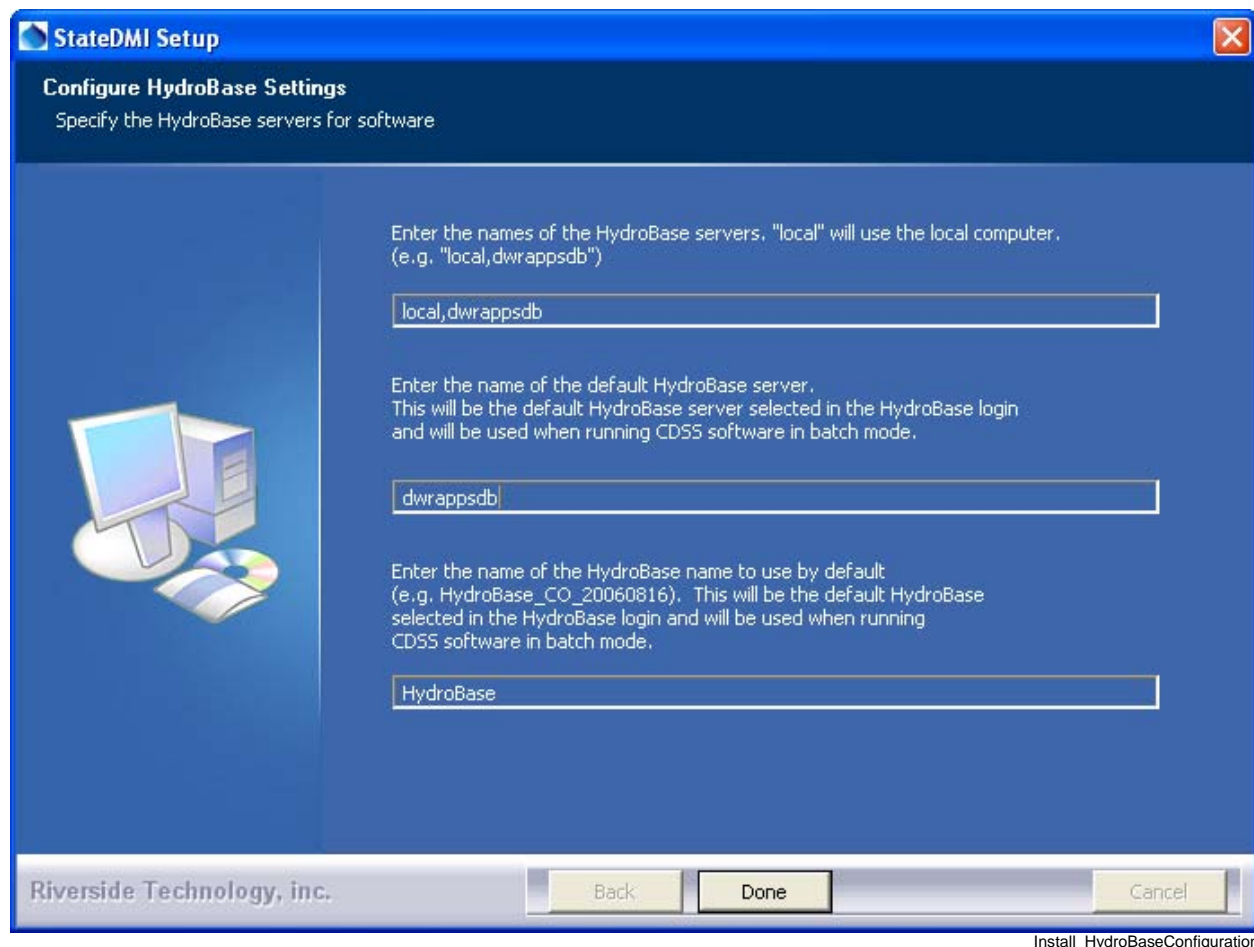
6. If the CDSS Base Components were selected for install (the default), the following dialog will be displayed:



StateDMI and other CDSS software can utilize HydroBase running on the local computer as well as other computers. Press **Yes** if HydroBase has been installed on another computer in the network

environment and may be used by the software (then continue to the next step). Otherwise, press **No** (skip to step 8).

7. The following dialog allows additional HydroBase servers to be specified for use by CDSS software (the example below configures CDSS software to list the *dwrappsdb* HydroBase server in choices and defaults to HydroBase on the local computer):



After entering the name of a HydroBase server and the default server to use, press **Done**.

8. The following dialog will then be shown asking whether the StateDMI software should be run:



Press **Yes** to run the software or **No** to exit the installation procedure.

### 3.1 StateDMI Configuration Files

StateDMI requires minimal configuration after installation. This section describes StateDMI configuration files that can be customized for a system.

#### 3.1.1 Data Units File

The *system\DATAUNIT* file under the main installation directory contains data unit information that defines conversions and output precision. In most cases the default file can be used but additional units may need to be added for a user's needs (in this case please notify the developers so the units can be added to the default file distributed with installations). Currently, the data units file is the only source for units information – in the future units may be determined from the various input sources.

#### 3.1.2 CDSS Configuration File

By default, StateDMI will automatically look for HydroBase databases on the current (local) machine and the State servers. State server databases are typically only accessible to State of Colorado computers. If SQL Server HydroBase versions have been installed on a different machine, the *\CDSS\StateDMI-Version\system\CDSS.cfg* file can be used to indicate the database servers. An example of the configuration file is as follows:

```
[HydroBase]

ServerNames="ServerName,local"
DefaultServerName="ServerName"
DefaultDatabaseName="HydroBase_CO_20080730"
```

The CDSS configuration properties are described in the following table:

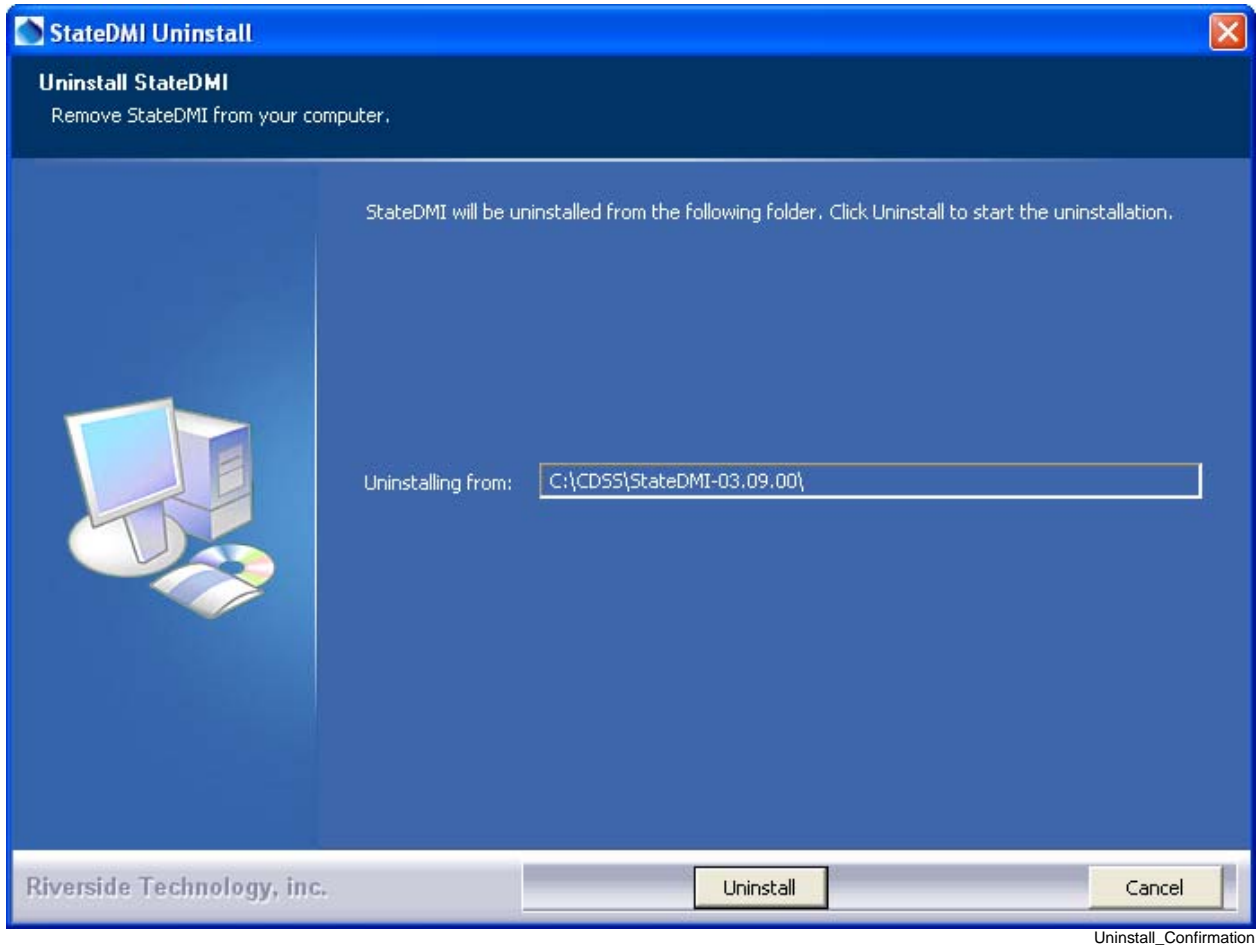
**CDSS HydroBase Database Configuration Properties**

Property	Description	Default
ServerNames	A comma-separated list of server names to list in the HydroBase login dialog.	The state server is listed.
Default ServerName	The default HydroBase server name to use. This allows the HydroBase login dialog to preselect a default that applies to most users in the system.	greenmtn. state.co.us
Default DatabaseName	The default HydroBase database name to use. This allows the HydroBase login dialog to preselect a default that applies to most users in the system.	
Database Engine	Reserved for internal use.	
DatabaseName	The database name to use for the initial connection. This overrides the default server.	
Database Server	The server name to use for the initial connection. This overrides the default server.	
SystemLogin	Reserved for internal use.	
SystemPassword	Reserved for internal use.	
UserLogin	Reserved for internal use.	

---

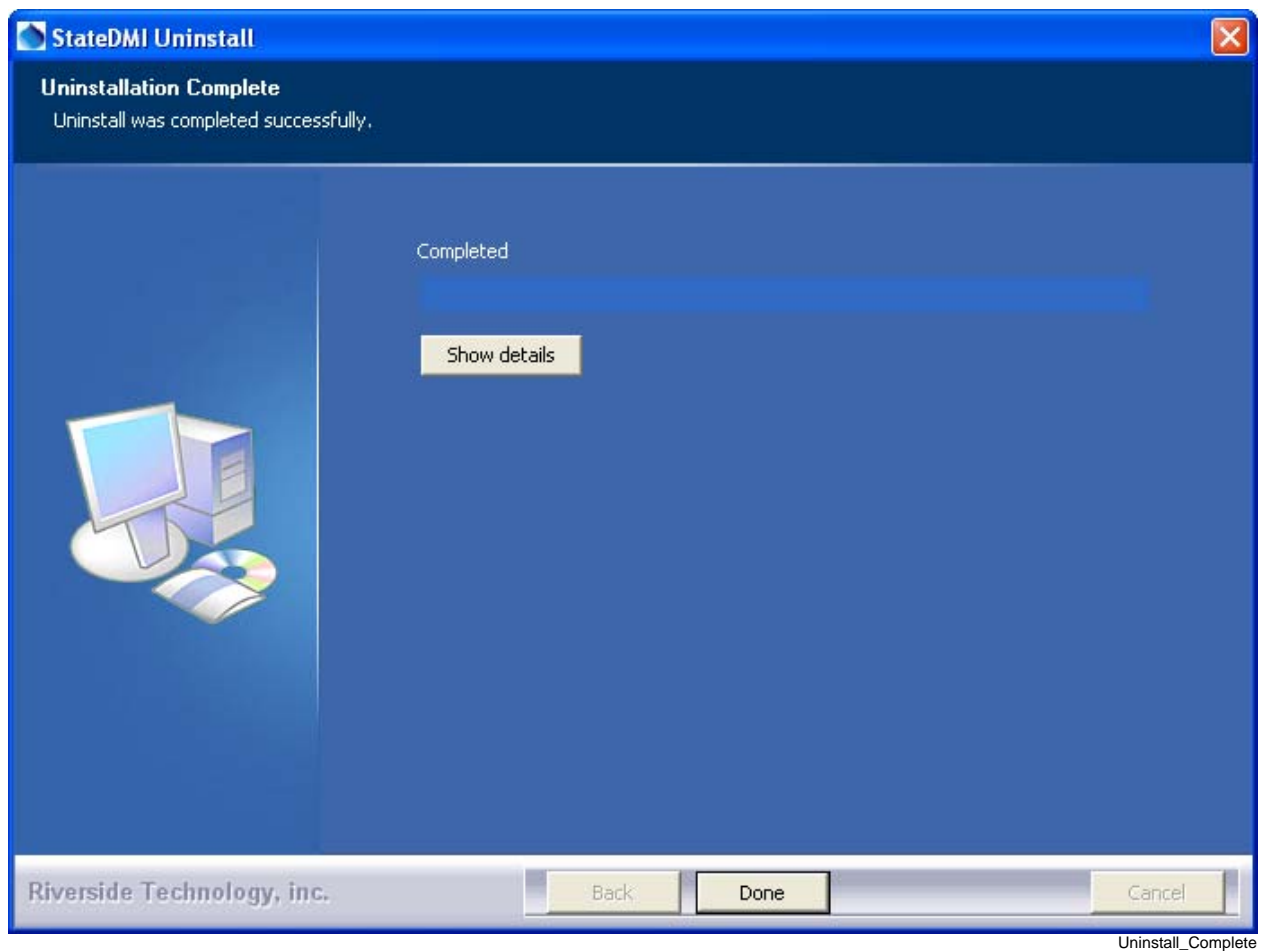
## 4. Uninstalling StateDMI Software

To uninstall StateDMI software, select **CDSS...Uninstall...StateDMI-Version** from the **Start** menu and confirm the uninstall. User data will remain installed.



Press **Uninstall** to uninstall the software.

The following dialog shows the status of the uninstall.



Press **Show details** to see the list of files that were removed. Press **Done** to exit the uninstall.

---

# Appendix: StateDMI Release Notes

Version 03.11.01, 2010-08-11

This appendix provides information about changes that have occurred in StateDMI versions.

## StateDMI Version History

The following table summarizes the StateDMI release history. See the following section for more detailed information about each version. Recent release note items are categorized as follows:

**Bug Fix** – A bug has been fixed. Users should evaluate whether their work is impacted.

**Known Limitation** – A known limitation has been documented and may impact the user. The limitation will be addressed in a future release.

**Change** – An existing feature has been changed.

**Remove** – A feature has been removed.

**New Feature** – A new feature has been added, with functionality that was not previously available.

### StateDMI Version History Summary (most current at top)

StateDMI Version	Version Information	Release Date
3.11.00 – 3.11.01	Minor maintenance releases. Windows 7 installer.	2010-08-11
3.10.00	Add StateCU Penman-Monteith crop coefficients commands. Include training materials with software.	2010-05-11
3.09.00 – 3.09.02	Update to use Java 1.6 release 18 to increase performance. Increase performance on slow commands. Update all documentation. Several bug fixes.	2010-02-18
3.04.00 – 3.08.02	Several changes have been made based on feedback for the 3.03.00+ releases, and enhancements related to South Platte processing. All StateCU and StateMod components now have check commands. Version 3.08.00 is meant to be evaluated by modelers in production work.	2009-09-29
3.03.00	All commands have been updated to the new error handling design, similar to TSTool. During this effort, all commands were cleaned up, including updating command editors to indicate required and optional commands. Many commands were updated to impose consistent behavior throughout commands. Additional warnings may now be generated to force users to be more explicit in handling issues in processing. In particular, identifiers must be specified (no * default) and IfNotFound=Add is required to add new data in set commands. Note: only the release notes in the documentation have been updated – a complete documentation update will be included in an upcoming release.	2009-02-16
2.18.00	Improve irrigation practice (IPY) file processing to remove additional NaN and missing values in results.	2007-10-18

StateDMI Version	Version Information	Release Date
2.17.00	Crop pattern time series acreage values are now explicitly written and the total/fraction information is for information only. This is compatible with changes in StateCU 12.19.	2007-10-17
2.16.00	Major changes in processing the irrigation practice (IPY) file. In particular, each command results in a full estimate of all acreage values.	2007-09-09
2.14.00	Implemented many changes to implement new well data processing. Refer to the documentation for well stations and rights, crop pattern time series, and irrigation practice time series for more information.	2007-07-11
2.02.00 – 2.13.00	Versions released leading up to 2.14, which has features for data processing approach for well data.	
2.01.00	Update crop names to 30-characters in StateCU files, update IPY file format to StateCU version 12, implement initial CIU filling features for diversions.	2007-03-02
2.00.00	First version using new installer.	2006-11-03
1.22.00	Update handling of well rights identifiers.	2006-10-24
1.21.00	Read well rights from net amounts table, add APEX to decree for wells.	2006-10-09
1.20.05	Add additional sort commands to menus; remove code that resets groundwater to sprinkler if groundwater is less.	2006-07-07
1.20.04	Add list output files to viewing choices in main window.	2006-06-13
1.20.03	Add read*FromList() commands to more menus.	2006-06-12
1.20.02	Add translate commands for crop characteristics and Blaney-Criddle files, improve well processing.	2006-04-30
1.20.01	Continue to improve well processing.	2006-04-24
1.20.00	Improve well processing.	2006-04-10
1.18.10	Update to allow wells and D&Ws to be ignored when processing well demand time series list.	2006-01-30
1.18.09	Implemented several fixes in network editor.	2005-12-23
1.18.08	Add commands to fill diversion, instream flow, reservoir, and well stations from the network.	2005-12-06
1.18.07	Update filling crop patterns with AgStats to better handle case where location does not grow all crops to be filled.	2005-11-22
1.18.06	Maintenance update.	2005-11-21
1.18.05	Add more complete headers to crop pattern and irrigation practice time series, fill CU locations with AWC and elevation.	2005-11-02
1.18.04	Change so that when processing well demands and efficiencies, only process stations where idvcomw=1.	2005-10-18
1.18.03	Maintenance update.	2005-10-13
1.18.02	Use NA as default diversion station ID for well stations.	2005-10-12
1.18.01	Change command dialogs to use scrollable area and allow double click on command to start an edit.	2005-10-10
1.18.00	Add commands for well pumping time series.	2005-10-03
1.17.21	Add list output files to results file list.	2005-09-27
1.17.20	Fix several bugs in processing crop commands.	2005-08-18
1.17.19	Fix two bugs handling diversion time series for aggregates.	2005-08-11



StateDMI Version	Version Information	Release Date
1.17.18	Enhance crop pattern and irrigation practice acreage synchronization.	2005-07-27
1.17.17	Updates to help with comparison of old data sets.	2005-07-13
1.17.16	Fix a bug in processing crop pattern time series.	2005-07-08
1.17.15	Add option to control processing well rights.	2005-06-28
1.17.14	Add features to facilitate comparing StateCU data sets.	2005-06-03
1.17.13	Enable new messaging and log file viewer and commands to write list files.	2005-04-05
1.17.12	Add warnings for obsolete commands.	2005-03-25
1.17.11	StateCU IPY features are enabled. New log file viewer is enabled. Enable StateCU and StateMod results viewers regardless of menus that are active.	2005-03-24
1.17.10	Enable right switch as appropriation date year and resolve related issues in aggregate rights, limiting to rights.	2005-03-14
1.17.09	When reading stream gage stations from the network, make the default output order the network order.	2005-02-14
1.17.08	Handle the water rights switch when limiting diversion and demand time series to rights.	2005-02-10
1.17.07	Add command to limit diversion demands to rights.	2005-02-09
1.17.06	Additional fixes for processing diversion historical and demand time series.	2005-02-07
1.17.05	Fix problem limiting diversion time series to rights. Expand fill capabilities for aggregate/system diversion time series. Several other small fixes.	2005-02-03
1.17.04	Reservoir accounts are no longer automatically adjusted if dead storage is specified, setting station ID using "ID" in rights now works.	2005-02-01
1.17.03	Fix problem where creating instream flow demand average monthly time series needed the output year type.	2005-01-31
1.17.02	Fix problem where diversion historical time series without monthly data in HydroBase were causing a premature end to filling with historical average.	2005-01-27
1.17.01	Fix problem with historical diversion time series being missing when not found in HydroBase.	2005-01-26
1.17.00	Several enhancements and fixes for processing diversion station historical time series (monthly). First release with tabular output displays.	2005-01-26
1.16.03	Fix output year type bug. Change default when writing files to overwrite.	2005-01-13
1.16.02	Fix so that conversion of makenet network retains label positions. Other minor updates.	2005-01-12
1.16.01	Fix case where station IDs that look like WDID but are not were causing HydroBase query errors. Other minor updates.	2005-12-17
1.16.00	Fix a number of errors in the setReservoirStation( ) command dialog.	2004-11-11
1.15.02	Official release for all StateCU and StateDMI features, with complete documentation.	2004-10-01
1.15.01	Includes StateMod well files, lacking some documentation.	2004-09-30

StateDMI Version	Version Information	Release Date
1.15.00	Includes all StateMod files except wells.	2004-09-16
1.14.00	Additional changes to all features except wells.	2004-08-22
1.13.00	Finalize features for network files, delay tables, instream flow, reservoirs, and diversions.	2004-07-12
1.12.00	Implement menus for all StateMod data components and implement station data features.	2004-06-01
1.11.00	Add <code>fillCULocationClimateStationWeights()</code> and update <code>translateCropPatternTS()</code> to take a list file.	2004-05-08
1.10.00	Minor changes to StateCU features based on user feedback. Begin implementing StateMod file features.	2004-04-10
1.09.00	Includes StateCU data file features.	2004-04-08
1.05.00b – 1.08.00b	Beta version for discussions with the State. Many changes in response to StateCU changes.	2004-01-13 – 2004-04-01
1.01.00 – 1.04.00	Internal versions for development.	
1.00.00	Initial version with framework.	2002-09-10

## Known Limitations

- **Known Limitation** Printing the network or saving as an image may not work. If necessary, use a screen capture tool to capture and print an image. Printing will be enhanced in an upcoming release.
- **Known Limitation** There are no commands for StateMod plan stations; however, plan stations can be represented in the network.

## Changes in Versions 3.11.00 – 3.11.01

- **Bug Fix** [03.11.00] Fix bug where the `ReadWellRightsFromHydroBase(DefineRightHow)` parameter was generating an error if left blank (work-around is to explicitly specify the parameter).
- **Change** [3.11.01] Update installer to support Vista and Windows 7 installations.
- **New Feature** [03.11.00] Include **CDSS Overview** presentation with software under *doc* folder.

## Changes in Version 3.10.00

- **Bug Fix** [03.10.00] Fix bug where **Paste** menus were not enabled on first copy/cut action.
- **New Feature** [03.10.00] Support has been added for StateCU Penman-Monteith crop coefficient processing, similar to Blaney-Criddle commands.
- **New Feature** [03.10.00] Training materials are now included with the installation in the `doc\Training` folder. Several slideshows are included, with supporting examples.

## Changes in Versions 3.09.00 – 03.09.02

- **Bug Fix** [03.09.02] Commands read from a command file that have invalid parameters were not always generating a visible warning for the user – this has been fixed.

- **Bug Fix** [03.09.02] The `SetCropPatternTSFromList(...ProcessWhen=Now..)` command, when used with a file that had multiple crops for a location, would only set the acreage for the last crop listed (all others were set to zero in a year). The command has been updated to properly handle multiple crops at a location.
- **Bug Fix** [3.09.01] The `SortWellRights()` command editor was not correctly displaying the `Order2` parameter upon re-edit – this has been fixed.
- **Bug Fix** [3.09.01] Fix bug where the `SetIrrigationPracticeTSFromList()` command was swapping the columns for surface water acres flood and groundwater acres flood. This command has also been updated to compute the groundwater total and surface water total values. The totals are provided for information only (not used by StateCU).
- **Bug Fix** [3.09.01] StateMod time series files could not be read if they did not have at least one comment at the top – this has been fixed.
- **Bug Fix** [3.09.00] The `Set*AggregateFromList()` and `Set*SystemFromList()` commands now will set the station name if the `NameCol` is specified when `PartsListedHow=InColumn`. The set will only occur if the original value is blank and the first non-blank value in the list file will be used. Previously the name could only be set if the parts were provided in a single row.
- **Bug Fix** [3.09.00] The `MergeWellRights()` command would result in no rights if all the original rights were explicit (not determined from parcel matching). This is an extreme case that normally would not be encountered.
- **Bug Fix** [3.09.00] The `ReadWellFromHydroBase()` command was always setting the date for well permits to the default date – this has been fixed. The bug was introduced in version 3.04.00.
- **Change** [3.09.01] All documentation has been updated to current software features.
- **Change** [3.09.00] The `ReadIrrigationPracticeTSFromHydroBase()` command has been updated to include the `Optimization` parameter, with the default now being to use more memory to increase performance.
- **Change** [3.09.01] Writing StateMod time series files will now write the total as the sum of the monthly or daily values as printed in the file (previously was computed as the in-memory total, which results in a different value). The total will also now be shown as a missing value more often due to more checks on the other values (previously may have been shown as zero). Output from commands that write time series may be slightly different; however, this column is not used by models and therefore results will not change.
- **Change** [3.09.00] The Java Runtime Environment (JRE) version 1.6 was updated to release 18, providing increased performance.
- **Change** [3.09.00] The `ReadWellRightsFromHydroBase()` command has been updated to include the `Optimization` parameter, with the default now being to use more memory to increase performance.
- **Change** [3.09.00] The `SetDiversionAggregateFromList()` and `SetDiversionSystemFromList()` commands now CANNOT be used to specify collection information for StateMod well stations. Instead, use the `SetWellAggregateFromList()` and `SetWellSystemFromList()` commands. This change allows error handling to be more robust and focuses well processing on well data.
- **New Feature** [3.09.00] Add the Administration Number Calculator tool to help users convert between appropriation dates and administration numbers.

## Changes in Versions 3.04.00 – 3.08.02

- **Bug Fix** [3.08.02] The `CalculateStreamEstimateCoefficients()` command was ignoring information from the `SetStreamEstimateCoefficientsPFGage()` commands – this bug was introduced in the 3.+ version and has been fixed.
- **Bug Fix** [3.08.02] The `SetStreamGageStation()` command was generating an error about the ID not being matched even when it was – this has been fixed.
- **Bug Fix** [3.08.02] Commands that set aggregate/system information from a list file were trying to match an empty ID from a blank line in the list file – this generated warnings and has been fixed.
- **Bug Fix** [3.08.00] The `FillWellStationsFromDiversionStations()` command was using the wrong station to fill data – this would be evident from incorrect well station names.
- **Bug Fix** [3.08.00] The `ReadWellHistoricalPumpingTSFromStateCU()` and `ReadWellHistoricalPumpingTSFromStateMod()` commands were not accessing the list of well rights for checks for whether a station is a diversion or well – some time series were not being read.
- **Bug Fix** [3.07.00] The `FillCropPatternTSConstant()` command would only allow integer constants – this has been fixed to also allow floating point values.
- **Bug Fix** [3.06.00] The network editor annotation dialog would indicate an invalid Y coordinate for networks that were very wide compared to the height – this has been fixed.
- **Bug Fix** [3.04.00] The `SetRiverNetworkNode()` and `FillRiverNetworkNode()` commands were not functioning properly, resulting in values not getting set – this has been fixed.
- **Bug Fix** [3.04.00] The `Read*FromNetwork()` commands were not properly handling the case where no network file was specified – this has been fixed.
- **Bug Fix** [3.04.00] The `ReadWellRightsFromHydroBase()` command was ignoring the Year parameter and was trying to read all years in HydroBase – this has been fixed.
- **Change** [3.08.02] The `CompareFiles()` command `WarnIfDifferent` parameter has been changed to `IfDifferent`, to allow for more parameter values and be similar to other commands. The `WarnIfSame` parameter has similarly been changed to `IfSame`. Old commands are automatically updated.
- **Change** [3.08.00] The `Set*TSConstant()` commands have been updated to by default reset the original data limits, based on typical use of the command. The `RecalcLimits` parameter can be used to skip this computation. This change may have some impact on data that is filled with the `Fill*HistMonthAverage()` commands.
- **Change** [3.07.00] The Java Runtime Environment (JRE) was updated from version 1.4.2 to version 1.6, providing increased performance and allowing upgrades in other areas. In particular, a new SQL Server database driver is now being used to allow an update to SQL Server 2008 for HydroBase.
- **Change** [3.06.00] The `MergeWellRights()` command has been updated to include the `SumDecreases` parameter to merge multiple rights that are otherwise the same.
- **Change** [3.06.00] Some previous output check features have been disabled in preparation of finalizing the new check design (see `CheckWellRights()`). Check commands have been added for all StateCU components.

- **Change** [3.05.00] The `MergeWellRights()` command now explicitly passes through well rights that have no parcel year, in order to retain rights from explicit well (or well collections) that do not utilize parcel relationships.
- **Change** [3.04.00] The `Set*AggregateFromList()`, `Set*SystemFromList()` and `SetDiversionMultiStructFromList()` commands now warn about missing files when the commands are loaded.
- **Change** [3.04.00] Fill and set commands for StateMod diversion stations, instream flow stations, river network nodes, reservoir stations, stream gage and estimate stations, well stations, and water rights have been updated to trim whitespace from parameters because extra whitespace included in quoted values was causing identifiers to not be matched properly and values to be formatted incorrectly.
- **Remove** [3.06.00] The StateCU commands related to delay tables have been removed since these files are no longer used with StateCU.
- **New Feature** [3.08.00] Add check commands for all StateMod components. Check commands are now in place for all components produced by StateDMI. Future releases may add additional specific checks to these commands.
- **New Feature** [3.08.00] Add the `ReadInstreamFlowDemandTSAverageMonthly()` command.
- **New Feature** [3.07.00] Add the `SortCropPatternTS()` command.
- **New Feature** [3.04.00] The progress indicator for commands is now active and has been enabled for the `ReadWellRightsFromHydroBase()` command. Additional commands will be updated in future releases to show progress within the command, in particular for longer-running commands.
- **New Feature** [3.04.00] The **Problems** tab has been added in the results area to summarize the warning/failure messages from all commands. The `WriteCheckFile()` command also has been added to format the messages to a file. The `CheckWellRights()` command has been added to check well rights and generate warning/failure messages. Additional check commands will be implemented in upcoming releases using this design.

## Changes in Versions 3.00.00 to 3.03.00

- **Bug Fix** [3.08.00] The `ReadIrrigationPracticeTSFromHydroBase()` and `ReadWellRightsFromHydroBase()` commands were allowing WDIDs to be provided that were not associated with the water division provided by the `Div` parameter – a warning has now been added and offending locations will not be processed to ensure that only locations in the specified division are processed by the command. Some data sets may need to be updated to ensure that separate commands are used to process data in different divisions.
- **Bug Fix** [3.03.00] The `ReadRiverNetworkFromStateMod()` command was not reading the comment and groundwater maximum recharge correctly – this has been fixed.
- **Bug Fix** [3.03.00] The `Set*TSConstant()` commands were not recognizing the set period when adding a new time series (the output period was always used for the set) – this has been fixed.
- **Bug Fix** [3.03.00] Well historical pumping time series (monthly) commands were included under diversion data rather than well data.
- **Bug Fix** [3.02.00] Fix the `ReadWellStationsFromStateMod()` command – variable efficiencies were being read such that the first month was used for all months.

- **Bug Fix** [3.02.00] Fix the `WriteDelayTables*ToStateMod()` command – the table identifier and number of values in the table were not being written correctly. Also add the `Precision` parameter to allow more flexibility and simplify software testing.
- **Bug Fix** [3.02.00] Fix the `SetWellStation()` and `SetDiversionStation()` commands – previously using “ID” in the river node field would not automatically use the station identifier.
- **Bug Fix** [3.02.00] The `Fill/Set ReservoirStation()` command was setting the on/off switch instead of the one fill rule value when new stations were added. – this has been fixed. Setting values in existing reservoirs did not have the problem.
- **Change** [3.03.00] The `Set*TSConstant()` commands now set the original data limits to that of the set data when new time series are added (averages are computed by including values  $\leq 0$ ) – this allows filling with average or pattern with a later command.
- **Change** [3.03.00] The `FillNetworkFromHydroBase()` command has been updated to automatically project geographic coordinates to UTM if only geographic are available in the database.
- **Change** [3.02.00] The `ReadDelayTablesFromStateMod()` command has been renamed `ReadDelayTablesMonthlyFromStateMod()` and `ReadDelayTablesDailyFromStateMod()` in order to minimize confusion about command functionality related to various data components.
- **Change** [3.02.00] The `SetInstreamFlowDemandTSAverageMonthlyConstant()` and `SetInstreamFlowDemandTSAverageMonthlyFromRights()` commands will now by default warn if the requested time series is not found. Specify the `IfNotFound=Add` parameter to request that the time series be added in this situation. This allows for increased error handling and quality control.
- **Change** [3.02.00] The `WriteDelayTablesToStateMod()` command has been renamed `WriteDelayTablesMonthlyToStateMod()` and `WriteDelayTablesDailyToStateMod()` in order to minimize confusion about command functionality related to various data components.
- **Change** [3.02.00] Change the `FillStreamEstimateStationsFromHydroBase()` command to optionally check for structure information in `HydroBase` – previously only station information could be filled.
- **Change** [3.02.00] Commands that generate list files now include header comments with a short description of the file. The command file and other information are also now included in comments.
- **Change** [3.02.00] Most set and fill commands now use the `IfNotFound` parameter, to give more control of error handling. The `Ignore` value for the parameter has been added to allow warnings to be ignored, if such a case is expected as possible. Some commands may generate warnings – set the parameter appropriately to remove the warning.
- **Change** [3.02.00] The main interface has been simplified to be more similar to `TSTool`. Features to manage full datasets have been disabled but may be enabled in the future.
- **Change** [3.02.00] Where appropriate, menu items have been prefixed with “1: “, “2: “, etc. to indicate that related commands are generally used in a sequence. For example, it may be necessary to read a water rights file in order to set time series using the rights.
- **Remove** [3.03.00] The `FillWellStationsFromHydroBase()` command has been removed – its capabilities are included in other fill commands, and the change removes redundant processing and allows for better error handling.
- **New Feature** [3.03.00] Enable the `LimitWellDemandTSMonthlyToRights()` command.



- **New Feature** [3.03.00] Add the `SortDiversionDemandTSMonthly()` command.
- **New Feature** [3.03.00] Add the `SortWellDemandTSMonthly()` command.
- **New Feature** [3.03.00] Add the `FillRiverNetworkNode()` command.
- **New Feature** [3.03.00] Add the `SetWellDemandTSMonthlyConstant()` command.
- **New Feature** [3.02.00] Commands have been updated to use new command status error handling, similar to the TSTool software. Problems (if any are detected) and corresponding recommendations are noted for each command.

## Changes in Version 2.18.00

- Additional changes in IPY processing to correct for some NaN and -999 values that were being generated. Additional care has been taken to set values to zero in some cases, resulting in more complete computation of other acreage terms.

## Changes in Version 2.17.00

- There is a major change in this release in how the crop pattern time series acreages are written to the CDS file. Previously, the acreage was written as a total and a fraction by crop. When the file was read (e.g., when processing the irrigation practice [IPY] file), this resulted in only three significant digits of precision and the resulting acreage by crop would not match that of the raw values in HydroBase, the GIS layers, the total in the CDS file, or the total computed in the IPY file (based on supply source). The legacy approach was maintained for a long time to allow comparison of model results but it became increasingly difficult to perform quality control on data as it moved through the system. The new approach writes the actual acreage for each crop. The fraction is still displayed but is for information only and is not used in computations. StateCU version 12.19 or later can be used to read the acreage column (prior versions used the total and fraction). It is important that the CDS and IPY files are generated with the same version of StateDMI (2.17.00 or later for both, or versions earlier than 2.17.00 for both).
- The IPY acreage values are now written by default to a precision of .1 and can be controlled with the `PrecisionForArea` parameter. This is necessary to minimize errors in round-off and warnings about acreage totals, in particular because acreage are categorized by supply type (surface and ground water) and irrigation method (flood and sprinkler) and fractions when rounded to integers were difficult to automatically prorate, especially with the processing described in version 2.16.00 notes below.
- The `setIrrigationPracticeFromList()` command can be used instead of the `readIrrigationPracticeFromList()` command. This sets the IPY values at the time the command is executed, instead of providing parcels that are later read when processing HydroBase parcels. For example, using the read command on an aggregate provides aggregate part data to be considered when reading data from HydroBase (which may supply data for other parts of the aggregate). Using the set command will set the values as the command is processed.

## Changes in Version 2.16.00

- The incremental releases leading up to and culminating with this release have implemented major changes in the processing of the irrigation practice (IPY) file. In particular, dependencies between commands that process IPY acreage have been removed. For each command, the acreage numbers are computed using the currently available information, with general order and importance of data being total acreage, then groundwater acreage (when groundwater data are available), then surface

water acreage. This recognizes that the total acreage from the CDS file should control and that groundwater acreage estimates (e.g., from center pivot irrigation and field data) are the most reliable. Each command initiates a cascade of computations in order to compute IPY acreages as completely as possible. For example, setting groundwater acreage does not try to adjust the total but will try to compute the remaining surface water acres (total – groundwater), and then if possible the acreage by irrigation method (flood or sprinkler). Consequently, as commands are executed to process the data, values will be converted from missing (-999) to specified or computed values. In troubleshooting data processing, commands can be incrementally uncommented to evaluate the results of each step. Log messages may indicate that some computation could not be done (e.g., groundwater total acres are set but split between flood and sprinkler cannot be done).

- The `writeIrrigationPracticeToStateCU()` command has been updated to include a `OneLocationPerFile` parameter. This is useful during troubleshooting because by default the IPY file is printed in blocks of years. By printing one location per file, the full period for a structure can be reviewed. This option is particularly useful if write commands are used after each major step of processing, in order to see the impacts of a command on results.

## Changes in Version 2.14.00

- Review all current procedures for the well rights, crop pattern time series, and irrigation practice time series, update all command reference documentation, and make minor software changes based on review.
- Fix bug in `readWellRightsFromHydroBase()` command – the `DefineRightHow` parameter was always being set to `EarliestDate`. The impacts of this bug should be minor, based on a previous review of different parameter combinations.
- Change so that when specifying aggregate/systems using a list file, if the list file specifies a location that is not found in the data set, the user will be warned.
- Add ability to read the associated diversion ID when reading a well station list file – this allows well right aggregation to properly handle different location types.
- The `fillWellStationsFromHydroBase()` command is being phased out. Instead, use `fillWellStationsFromDiversionStations()`, `setWellStationAreaFromCropPatternTS()`, and `setWellStationCapacityFromWellRights()` commands.
- Update the **Command** menu to have three levels, to improve usability and allow further consolidation of StateCU and StateMod commands.

## Changes in Version 2.02.00 – 2.13.00

- These versions were made with features to explore implementing a new modeling approach and were finalized in version 2.14 – see the notes for that version.

## Changes in Version 2.01.00

- Add `setRiverNetworkNode()` command to set river node network information, mainly to change the node name.
- Update the CCH, CDS, and KBC files to default to new 30-character crop names. The previous file versions can still be read using a `Version=10` parameter.
- Update the IPY file format to by default use the new format with more columns for acreage. The processing logic to fill the values is not yet in place. Therefore, the `Version=10` parameter should be used when writing the IPY file, until the next release.



- Add preliminary features to fill diversion records with “currently in use” (CIU) information when using the `readDiversionHistoricalTSMonthlyFromHydroBase()` command – features will be finalized after further testing.
- Implement improvements in the installer to better handle configuration of the HydroBase settings.

### Changes in Version 2.00.00

- First version using the new installer, to facilitate distribution and installation of the software.
- Remove need for well water rights to be sorted in a particular order to be processed for the StateCU IPY file. The max pumping values in the IPY file will generally have a higher maximum.

### Changes in Version 1.22.00

- Change `readWellRightsFromHydroBase()` IDFormat parameter dialog note and fix to make sure that identifiers are still being formatted properly for the previous release.

### Changes in Version 1.21.00

- Adjust reading well rights to reread from the database rather than relying on the “wells” table. This results in slower run times and potentially more water rights in output files. The South Platte and Rio Grande modeling approaches are different and use different command parameters when reading well rights.
- Add APEX amounts to the net amount decrees. This results in larger decrees in model files.

### Changes in Version 1.20.05

- Fixed problem where some sort commands were not available from menus.
- In `synchronizeIrrigationPracticeAndCropPatternTS()`, remove code that resets groundwater acreage to sprinkler acreage if groundwater is less – it is unneeded based on modeling conventions.

### Changes in Version 1.20.04

- When writing list files, add the files to the list of output files available in the GUI.

### Changes in Version 1.20.03

- Add `read*FromList()` commands to all menus that through oversight did not have them added previously.
- Fix problem with “see check file” dialog being shown before editing commands.

### Changes in Version 1.20.02

- Continue to improve well processing.
- Always create the check file for well stations and rights.
- Add the `translateBlaneyCriddle()` and `translateCropCharacteristics()` commands to change crop name to facilitate modeling.

### Changes in Version 1.20.01

- Continue to improve well processing.

### Changes in Version 1.20.00

- Improve well processing based on user feedback.

### Changes in Version 1.18.10

- Update the `readWellDemandTSMonthlyFromStateMod()` command to allow ignoring wells or D&Ws, to facilitate processing subsets of the data set.

### Changes in Version 1.18.09

- Implemented several fixes in the network editor.

### Changes in Version 1.18.08

- To allow filling station names from the network file, add the following commands:  
`fillDiversionStationsFromNetwork()`,  
`fillInstreamFlowStationsFromNetwork()`,  
`fillReservoirStationsFromNetwork()`, `fillWellStationsFromNetwork()`.
- Add elevation to the `readCULocationsFromList()` command.

### Changes in Version 1.18.07

- Add AWC to the `readCULocationsFromList()` command.
- Fix the `fillCropPatternTSProrateAgStats()` command so that all county crops are used even if a location does not have a crop type.

### Changes in Version 1.18.06

- Add the `readDiversionDemandTSMonthlyFromStateMod()` command.
- Improve packaging of image files with Jar files to resolve issues with icons not displaying in the network editor.

### Changes in Version 1.18.05

- Add the `setDiversionStationsFromList()` and `setWellStationsFromList()` commands.
- Update `fillCULocation()` and `setCULocation()` to include elevation and AWC.
- Update the StateCU CDS and IPY file headers to include more complete headers, as expected by StateMod.
- Add the `mergeListFileColumns()` command.

### Changes in Version 1.18.04

- Implement minor changes to well processing based on user feedback.
- When processing well demand time series to calculate average efficiencies or to estimate demands using average efficiencies, only process well stations where `idvcomw=1`.

### Changes in Version 1.18.03

- Implement changes to better support product-oriented file management.

### Changes in Version 1.18.02

- For well stations, default the associated well to NA rather than N/A.

### Changes in Version 1.18.01

- Change command dialogs to use scrollable text areas instead of text fields of a fixed size. This allows longer commands to be fully viewed.
- Double-clicking on a command now displays the editor for the command.

### Changes in Version 1.18.00

- Add commands for well pumping time series (historical monthly).
- Reverse the **Run All Commands** and **Run Selected Commands** buttons to agree with the TSTool order.
- Add graphical buttons at the top of the main window to facilitate opening and saving commands files.
- Add a complete menu for well historical time series monthly (previously only a subset of commands was included).

### Changes in Version 1.17.21

- Update to include `write*ToList()` output files in the results file list.
- Add the efficiency report that is created when processing demands to the output results file list.

### Changes in Version 1.17.20

- For well-only aggregates, do not put a W in the water right ID. D&W nodes still have the W, as per the Watright software.
- Fix a bug where the last year filling crop pattern time series was not getting normalized to basin statistics.
- Fix a bug in the `setCropPatternTS()` command overriding an existing pattern causes erroneous output.
- When using a time series that is read from an external file, reset the period to the output period so that the time series can be filled.
- Fix a bug where when filling time series with a constant, the start and end dates were not being handled properly.
- When filling diversion time series with diversion comments, read the comments after setting the period of record.
- Add the `setDiversionDemandTSMonthlyConstant()` command.

### Changes in Version 1.17.19

- Fix bug where missing file with the `readAgStatsTSFromDateValue()` command was not being handled gracefully.
- Add `IgnoreUseType` parameter to the `readDiversionRightsFromHydroBase()` command, to address double counting of some rights in HydroBase.

- Change so that if an aggregate/system diversion part has missing capacity, the total capacity is not incremented for the part (which has a large default value).
- Fix so that an aggregate/system historical diversion is handled properly, even if the first part has no data in the database.

### Changes in Version 1.17.18

- Fix a bug in the `writeCropPatternTSToStateCU()` command where the `WriteCropArea` parameter was not defaulting properly.
- Implement new parameters in the `synchronizeIrrigationPracticeAndCropPatternTS()` command to allow more options in synchronizing acreage.
- Update the `setIrrigationPracticeTSMaxPumpingToRights()` command to have the `NumberOfDaysInMonth` parameter, to be consistent with `StateCU` conventions.
- Update the `fillCropPatternTSProrateAgStats()` command to include the `NormalizeTotals` parameter, to allow acreage to be prorated to the totals for a group of crop types.

### Changes in Version 1.17.17

- Update the `readCropPatternTSFromHydroBase()` command to truncate parcel acreage to .2 to compare to work done by Leonard Rice. This feature is available only in test mode.
- Update the `setIrrigationPracticeTSFromList()` command so that data other than efficiencies can be set.

### Changes in Version 1.17.16

- Fix bug where the `setCropPatternTS()` command results were not getting refreshed after the initial processing, resulting in zeros in the output for totals.
- Add a tool to print surface-only diversions to the log file. This is useful for finding diversion stations that are not D&W model nodes.

### Changes in Version 1.17.15

- Add `DefineRightHow=LatestDate` when processing well rights.

### Changes in Version 1.17.14

- Change `writeCropPatternTSToStateCU()` command to optionally write only the total acreage by location, to facilitate comparison with previous data sets, and to use the output period, if specified.
- Add the file version to the `readCropPatternTSFromStateCU()` and `readIrrigationPracticeTSFromStateCU()` commands, to facilitate comparison with previous data sets.
- Add the `openHydroBase()` command.
- Add the `ReadStart` and `ReadEnd` parameters to the `readDiversionHistoricalTS*FromHydroBase()` commands.
- Fix bug where reading historical diversion time series was initializing the first part in an aggregate and then adding the part again.
- Enable flags for filling diversions with historical average, pattern, constant, and limiting to rights.

## Changes in Version 1.17.13

- Change the log file warning level to 3 to reflect application warnings being level 1, command warnings being level 2, and important low-level warnings being level 3.
- Finalize results displays for reservoirs, wells, instream flow and network data.
- Begin phasing in stored procedures to production version.
- Add `write*ToList()` commands.
- Add the `readReservoirRightsFromStateMod()`, `readWellRightsFromStateMod()` commands.
- Add the `readStreamEstimateCoefficientsFromStateMod()` command.
- Add the `readDelayTablesFromStateCU()` and `readCULocationDelayTableAssignmentsFromStateCU()` commands.
- Add `sortBlaneyCriddle()` command and add precision to `writeBlaneyCriddleToStateCU()` to facilitate comparison with previous data sets.
- Add version to `writeCULocationsToStateCU()` to facilitate comparison with previous data sets.
- Convert commands to messaging that is integrated with the log file viewer.
- Update the `setIrrigationPracticeTSSprinklerAreaFromList()` command to allow using the area in the list file, to facilitate comparison with previous data sets.
- Fix bug where `sortReservoirStations()` was not being recognized.
- Add ability to open new model networks.
- Add the `startLog()` command.
- Fix a bug where the dialog for the commands file was not being initialized to a recently accessed directory.

## Changes in Version 1.17.12

- Add warnings for obsolete commands.
- Change message levels to minimize console output.

## Changes in Version 1.17.11

- The StateCU IPY file can now be processed. See specific changes below.
- Irrigation practice time series groundwater and sprinkler acreage can now be read from HydroBase using the `setIrrigationPracticeTSFromHydroBase()` command.
- The `setIrrigationPracticeTSMaxPumpingToRights()` command will now use water rights from the `setIrrigationPracticeTSFromHydroBase()` command, or read a StateMod well rights file.
- The `setIrrigationPracticeTSSprinklerAreaFromList()` command has been enabled to process snapshots of sprinkler parcels from a list file and HydroBase.
- A new log file viewer has been enabled. The old Notepad default viewer is still available but the new viewer provides a summary of level 1 and 2 warning messages and allows navigation in the large log file. Additional enhancements will be enabled in future releases in order to simplify the interpretation of messages. In particular, additional attention is focusing on the classification of warnings and errors.
- The results of a commands run were previously tied to whether StateCU or StateMod menus were activated. Output components for both models are now listed to simplify access to results. The prototype displays are being finalized.

## Changes in Version 1.17.10

- Add the `synchronizeIrrigationPracticeAndCropPatternTS()` command for processing the irrigation practice time series.
- Add `writeCropPatternTSToStateCU()` to the irrigation practice time series commands to update the file after synchronization.
- The irrigation practice commands to assign the maximum pumping, groundwater acreage, and sprinkler acreage are not yet functional.
- Add the `sortCULocations()` command to sort the data before writing.
- Add the `sortDiversionRights()`, `sortReservoirRights()`, `sortInstreamFlowRights()`, and `sortWellRights()` commands to sort right data before writing.
- Update the `readWellRightsFromHydroBase()` command to have the `DefineRightHow` and `DefaultAppropriationDate` parameters to control how StateMod rights are created from well rights and permits.
- Update the `readDiversionRightsFromHydroBase()`, `readReservoirRightsFromHydroBase()`, `readInstreamFlowRightsFromHydroBase()`, and `readWellRightsFromHydroBase()` commands to include the `OnOffDefault` parameter to allow the right switch to be set to the appropriation date year.
- Update the above commands that aggregate rights to set aggregate rights to the integer value for the decree-weighted appropriation date. Previously the fractional remainder was not cleared and the resulting administration numbers could give erroneous appropriation dates (e.g., when used with the `limitDiversionHistoricalTSMonthlyToRights()` or similar commands).
- Update the `limitDiversionHistoricalTSMonthlyToRights()` and `limitDiversionDemandTSMonthlyToRights()` commands to have the `UseOnOffDate` parameter, allowing the appropriation date to be determined from the administration number or the `OnOff` switch (when a year).
- When processing diversion and well rights, ignore water rights that have units other than C or CFS. Previously only C was checked but there is apparently a change in HydroBase.
- Implement initial enhancements to the log file viewer, which provides a summary of warning messages and provides navigation tools for the log file.

## Changes in Version 1.17.09

- When reading stream gage stations from the network, make the default output order the network order, which is expected by StateMod. Previously, stream gages were listed first and then other baseflow nodes.
- Rearrange the order of the diversion demand time series (monthly) menus to reflect typical use.

## Changes in Version 1.17.08

- Handle the water rights switch in the StateMod diversion rights file when using the `limitDiversionHistoricalTSMonthlyToRights()` and `limitDiversionDemandTSMonthlyToRights()` commands.

## Changes in Version 1.17.07

- Add `limitDiversionDemandTSMonthlyToRights()`.

## Changes in Version 1.17.06

- The first time series part in an aggregate/system was not being filled in the `readDiversionHistoricalTSMonthlyFromHydroBase()` command. This has been fixed.
- The efficiency report from the `calculateDiversionStationEfficiencies()` command is now listed in the output files and can be displayed.
- The list of stations to ignore in the `limitDiversionHistoricalTSMonthlyToRights()` command was not being processed correctly, resulting in an error. This has been fixed.

## Changes in Version 1.17.05

- The `limitDiversionHistoricalTSMonthlyToRights()` command was not triggering a save of the original time series, as needed. This has been fixed. The documentation for the command was also significantly expanded.
- In `calculateDiversionDemandTSMonthly()`, change so that if the efficiency and IWR is zero, set the demand to zero. Previously it was set to missing. This will also impact well demands.
- Add the `IncludeCollections` parameter to the `fillDiversionHistoricalTSMonthlyAverage()` and `fillDiversionHistoricalTSMonthlyPattern()` commands to allow diversion aggregate and system stations to be ignored in processing (because they can also be filled during the read).
- Enhance `readDiversionHistoricalTSMonthlyFromHydroBase()` to allow filling of aggregate/system parts before aggregation.
- The fill period from command parameters was not being considered when filling time series with a pattern – this has been fixed.
- The `setDiversionHistoricalTSConstant()` and other similar commands were not using the `SetStart` when specified by the user.
- The `fillDiversionStation()`, `setDiversionStation()`, `fillWellStation()`, and `setWellStation()` commands were not properly transferring efficiencies specified in calendar year to water year data in station files.

## Changes in Version 1.17.04

- Previously, if the dead storage value for a reservoir was specified, the reservoir accounts were adjusted down by this amount and the dead storage was always written as zero. This was a workaround for a limitation in StateMod. The dead storage value is now written as specified and accounts are not adjusted.
- Setting or filling rights by specifying a `StationID` of “ID” was not previously working. The software will now set the station ID to the first part of the right (the part before “.”).
- Added a warning when processing diversion demand time series (monthly) if no diversion stations have been read.

## Changes in Version 1.17.03

- Fix bug in `setInstreamFlowDemandTSAverageMonthlyFromRights()` where the period for the time series was incorrectly being taken from the `setOutputPeriod()` command. It now uses the `setOutputYearType()` command information.
- Fix bug where the `calculateStreamEstimateCoefficients()` command was generating an error about the command not being recognized. This was a spurious message that was removed.

- Change so that reading stations from the network results in the river node identifier being set to the station identifier. Previously the river node identifier was set to missing and required an additional fill or set command to assign the value.

### Changes in Version 1.17.02

- Fix bug filling `fillDiversionsHistoricalTSMonthlyAverage()` where missing original data was causing the command to end. Time series with no original data are now not filled with historical averages and the processing is allowed to continue through other time series.
- Begin simplifying `readSprinklerParcelsFromList()` – additional enhancements need to be completed before the command can be used in production.

### Changes in Version 1.17.01

- The `readDiversionsHistoricalTSMonthlyFromHydroBase()` command was not allocating memory for blank time series – therefore subsequent filling was ignored and output was missing.

### Changes in Version 1.17.00

- Introduce tabular displays for output components, available at the bottom of the main interface. These displays can be used to review data while fine-tuning commands. Additional enhancements to these displays will occur – this is an initial release of these features.
- Rework the size of the display panels in the main interface to provide more display area for commands. The other panel areas are still retained but may be removed or hidden in the future.
- Add `sortDiversionsHistoricalTSMonthly()` command to facilitate maintaining consistency with the diversion station file.
- Update the `setHistoricalDiversionsHistoricalTSMonthly()` command to allow reading from HydroBase, for cases where a diversion's time series may actually be stored under a different identifier (e.g., a stream gage). Also save a backup copy of the time series after reading, for use with the `limitDiversionsHistoricalTSMonthlyToRights()` command.
- Fix the `limitDiversionsHistoricalTSMonthlyToRights()` command – previously the list of rights was accumulating as diversion stations were processed, instead of just using the rights for the specific diversion station.
- Fix an error in the `setReservoirStation()` command editor dialog – the account name was being discarded when re-editing an old command.
- Fix inconsistencies in the Select All and Deselect All commands menus – previously the behavior was not correct.

### Changes in Version 1.16.03

- The default for commands that write files is now to overwrite files. The previous default of updating the file was resulting in long file headers.
- Setting the output year type with `setOutputYearType()` was not being recognized. This impacted both output of time series and processing of data like diversion station efficiencies.
- Add `setDiversionsHistoricalTSMonthlyConstant()` – this eliminates the need for replacement files in some cases.

### Changes in Version 1.16.02



- The `setDiversionStationDelayTablesFromRTN()` command editor dialog was changing the spelling of the command after edits – this has been corrected.
- For station collections (aggregates and systems), the station name in the list file was not being used to set data for the stations – this has been corrected.
- When reading an old Makenet network file into StateDMI, the label positions were being reversed – this has been fixed. It may be necessary to reconvert networks to retain the original label positions.

### Changes in Version 1.16.01

- Diversion, reservoir, well, and instream flow station identifiers that looked like WDIDs but which were not (e.g., 990001) were resulting in HydroBase queries, which caused an error. Additional error handling has been enabled.
- The `setDiversionStationDelayTablesFromRTN()` command editor dialog was changing the spelling of the command after edits – this has been corrected.

### Changes in Version 1.16.00

- Added to troubleshooting to explain errors caused by Ctrl-M characters in commands.
- Added a popup menu choice for commands to find a command using a line number – this facilitates debugging the commands.
- Updated the command editor dialog and documentation for the `setReservoirStation()` command to clarify the meaning of `AccountID`. The `AccountName` parameter was also mistakenly being set to the `AccountID`.
- Fix limitation where the position of the legend in the network was not being saved after the legend was interactively moved.
- Fix bug where after adding a new node in the network, the node cannot be selected for further changes.
- Display the page margins on the network diagram by default.
- Clarify the documentation for system and aggregate commands to indicate that the commands should be specified before reading data from HydroBase.

### Changes in Version 1.15.02

- Completed documentation for all data files.
- Updated documentation to discuss conventions for station identifiers.
- Updated documentation to incorporate general information from old **StateMod Appendix B** procedures manual.
- Updated documentation to include well demand commands.
- Updated network data documentation to describe use of the network up front versus list files.
- Fixed several well demand command editor dialogs were not displaying correctly.
- Fixed problem where fill commands for time series were not updating the time series to the files.
- Enable viewing results in text editor.

### Changes in Version 1.15.01

- Includes all well file commands.

### Changes in Version 1.15.00

- All StateMod files are supported except for wells. Well features are preliminary.

- Change StateCU IPY file format to match previous version. The precision for some data that are in more than one file (e.g., area) is once again inconsistent.

This page, when printed, can be used for a spine in a binder.

# **Colorado's Decision Support Systems (CDSS)**

---

## **StateDMI**

