

# **Programs for Simplifying the Analysis of Geographic Information in U.S. Geological Survey Ground-Water Models**

By Richard B. Winston

---

U.S. GEOLOGICAL SURVEY  
Open-File Report 01-392

Reston, Virginia  
2001

U.S. DEPARTMENT OF THE INTERIOR  
GALE A. NORTON, Secretary

U.S. GEOLOGICAL SURVEY  
Charles G. Groat, Director

Although the computer program described in this report has been tested and used by the U.S. Geological Survey (USGS), no warranty, expressed or implied, is made by the USGS as to the accuracy of the functioning of the program and related material. The code may be updated and revised periodically.

Any use of trade, product, or firm names in this publication is for descriptive purposes only and does not imply endorsement by the U.S. Government.

---

For additional information  
write to:  
Office of Ground Water  
U.S. Geological Survey  
411 National Center  
Reston, VA 20192

Copies of this report can be  
purchased from:  
U.S. Geological Survey  
Branch of Information Services  
Box 25286  
Denver, Colorado 80225-0286

## Contents

	Page
Abstract .....	1
Introduction .....	1
Installation instructions .....	18
Navigating .....	18
Acknowledgments .....	19
Interpolation Methods .....	19
Commands.....	20
Edit .....	20
Edit Contours.....	20
DeclutterContours .....	21
Join Contours.....	22
Edit Grid.....	23
Edit Data.....	26
Create Parameters in Multiple Layers.....	27
Set Multiple Parameters .....	27
Import.....	28
Import Gridded Data .....	28
Import Points from Spreadsheet.....	29
Import Contours from Spreadsheet .....	29
Sample DEM Data .....	30
Copy Tri Mesh/Copy Quad Mesh.....	31
Convert.....	31
Contours To Data .....	31
Data to Contours.....	31
Reverse Contours on Clipboard .....	31
Mesh Objects To Contours.....	32
Mesh To Contours.....	32
Hidden Commands.....	32
Set Parameter Locks.....	32
Delete Multiple Layers.....	33
Functions .....	33
Utility_CheckVersion.....	33
EvalRealAtXY, EvalIntegerAtXY, EvalBooleanAtXY, and EvalStringAtXY .....	33
Rotated X, and Rotated Y .....	34
GetMyDirectory .....	34
ReadFileValue functions .....	35
RF_Get_Value_From_File.....	35
RF_Clear_Files.....	35
RF_Save_Files .....	35
RF_CheckVersion .....	35
Conversion functions.....	36
Hidden Functions .....	36
OKCancel Functions .....	36
IsOK .....	36

Ok_Add_Radio_Choice .....	36
Ok_Get_Radio_Choice .....	37
Ok_Radio_Free .....	37
Ok_UserFloat and Ok_UserInteger .....	37
Ok_CheckVersion .....	37
ProgressBar Functions.....	38
ProgressBarInitialize .....	38
ProgressBarFree .....	38
ProgressBarMax .....	38
ProgressBarAdvance .....	38
ProgressBarSetMessage .....	38
ProgressBarAddLine .....	38
ProgressBarSaveToFile.....	39
ProgressBarCheckVersion .....	39
JoinFiles Functions.....	39
Join_Files .....	39
Delete_File .....	39
Rename_File.....	39
Split_File.....	39
Int2Str.....	39
JF_CopyLines.....	40
JF_CheckVersion .....	40
List functions.....	40
L_Initialize .....	40
L_CreateNewList .....	40
L_SetListSize .....	41
L_GetListSize.....	41
L_FreeAList .....	41
L_AddToList.....	41
L_GetFromList.....	41
L_SetListItem.....	41
L_DeleteListItem .....	42
L_SortList.....	42
L_EliminateDuplicates.....	42
L_IndexOf.....	42
L_UnsortedIndexOf .....	42
L_CreateNew3DList .....	42
L_FreeA3DList .....	43
L_GetFrom3Dlist and L_GetFromOneBased3DList.....	43
L_Set3DlistItem and L_SetOneBased3DListItem .....	43
L_ResetA3DList.....	43
L_Add3DLists.....	43
L_Subtract3DLists .....	44
L_Multiply3DLists.....	44
L_Divide3DLists.....	44
L_Multiply3DByConstant.....	44

L_Invert3DListMembers.....	44
L_IsSingPrecUniform .....	44
L_GetErrorCount .....	45
L_FreeAllLists .....	45
L_CheckVersion.....	45
BlockList functions .....	45
BL_InitializeGridInformation .....	46
BL_AddVertexLayer.....	46
BL_ReInitializeVertexList.....	46
BL_GetCountOfCellLists.....	46
BL_GetCountOfACellList .....	46
BL_GetCellRow and BL_GetCellColumn.....	46
BL_GetVertexCount .....	47
BL_GetVertexXPos and BL_GetVertexYPos .....	47
BL_SegmentCount .....	47
BL_SegmentFirstX and BL_SegmentFirstY .....	47
BL_SegmentSecondX and BL_SegmentSecondY.....	47
BL_SegmentLengthX and BL_SegmentLengthY .....	47
BL_SegmentLength .....	47
BL_SumSegmentsX and BL_SumSegmentsY .....	47
BL_SumSegmentLengths.....	47
BL_GetCountOfCombinedCellList .....	48
BL_GetCellRowFromCombinedList and BL_GetCellColumnFromCombinedList .....	48
BL_GetCountOfCrossRowLists and BL_GetCountOfCrossColumnLists.....	48
BL_GetCountOfACrossRowList and BL_GetCountOfACrossColumnList .....	48
BL_GetCrossRowRow and BL_GetCrossColumnRow.....	48
BL_GetCrossRowColumn and BL_GetCrossColumnColumn.....	48
BL_GetCrossRowNeighborColumn and BL_GetCrossColumnNeighborRow .....	48
BL_GetCrossRowCompositeY and BL_GetCrossColumnCompositeX.....	49
BL_GetSumCrossRowCompositeY and BL_GetSumCrossColumnCompositeX....	50
BL_GetCrossRowCompositeLength and BL_GetCrossColumnCompositeLength .....	50
BL_GetRowBoundary and BL_GetColumnBoundary .....	50
BL_PointInsideContour .....	50
BL_GetRowNodePosition and BL_GetColumnNodePosition .....	50
BL_GetRowBoundaryCount and BL_GetColumnBoundaryCount.....	51
BL_GetRowNodeCount and BL_GetColumnNodeCount .....	51
BL_GetCellArea.....	51
BL_FractionOfLine.....	51
BL_FreeVertexList.....	51
BL_FreeAllBlockLists .....	51
BL_GetErrorCount.....	51
BL_CheckVersion.....	51
Conclusions .....	52
References .....	52
Appendix 1: Custom Components Used in PIEs .....	54
TRbwZoomBox.....	54

TRbwDynamicCursor .....	55
TArgusDataEntry .....	55
TRbwQuadTree.....	55
TRbwOctTree.....	55
Installation.....	55
Appendix 2: Example Export Template for ProgressBar PIE.....	57
Appendix 3: Example Export Template for List PIE.....	58
Appendix 4: Example Export Template for BlockList PIE .....	61
Appendix 5: Export Templates Using Rotated X and Rotated Y .....	64
Appendix 6: DEM2Image.exe.....	67

## Tables

	Page
Table 1. Commands added to Argus ONE in Utility.dll .....	3
Table 2. Hidden commands added to Argus ONE in Utility.dll .....	3
Table 3. Functions added to Argus ONE .....	4
Table 4. Hidden functions added to Argus ONE .....	10

## Figures

	Page
Figure 1. Effect of using the Declutter Contours command .....	21
Figure 2. If used improperly, the Declutter Contours command can result in contours that overlap or that no longer show correct geometric relationships.....	22
Figure 3. Effect of the Join Contours command .....	23
Figure 4. The cells intercepted by the contour (heavy black line) are shown in dark gray. Those cells which are their neighbors as determined by BL_GetCrossRowNeighborColumn and BL_GetCrossColumnNeighborRow are shown in light gray except for those which are also intercepted by the contour. The heavy dashed line separates each cell intercepted by the contour from its neighbor or neighbors.....	49
Figure 5. For the cell indicated by the dark gray square, the length returned by BL_GetCrossRowCompositeY is indicated by the double-headed arrow. To determine this, the function retrieved data not only from the dark gray cell but also from the light gray cells.....	50
Figure 6. The result of BL_FractionOfLine for the cell indicated by the dark square would be 0.10 because 10 percent of the contour lies within the cell.....	51

# **Programs for Simplifying the Analysis of Geographic Information in U.S. Geological Survey Ground-Water Models**

**By Richard B. Winston**

## **Abstract**

This report describes a number of programs developed at the U.S. Geological Survey (USGS) to enhance existing USGS graphical user-interfaces for ground-water modeling programs. The programs are in the form of dynamic-link libraries that add interpolation methods, commands, and functions to Argus ONE™ through the use of Plug-In Extensions (PIEs). The interpolation methods, commands, and functions are not specific to any particular type of model but instead may find application under a variety of circumstances. The new interpolation methods provide results much more quickly than the existing interpolation methods in Argus ONE while still giving reasonable results. Some of them also incorporate anisotropy. This may be particularly valuable for cross sectional models. The commands are useful for copying, editing, converting, or importing information. They include commands for creating or modifying grids, editing the precise locations of nodes along contours, importing data as either contours or data points, setting the expressions for multiple parameters, and importing Digital Elevation Models (DEMs). The functions provide convenient methods for evaluating, converting, or storing information. Many of the functions are hidden because they could cause memory leaks if used improperly or provide specialized functions not likely to be of interest to most users. Nevertheless, these functions provide a convenient method of performing Geographic Information System (GIS) functions that would be difficult to do with Argus ONE alone.

## **Introduction**

Argus ONE™ is a Geographic Information System for numerical modeling developed by Argus Interware. It can serve as a platform for creating graphical user-interfaces for models (for example Voss and others (1997) and Winston (2000)). It can create finite-element meshes and finite-difference grids and assign properties to nodes, elements, or cells based on GIS analyses that it performs. It can write the input files for numerical models and graphically display the results of those models. The functionality of Argus ONE can be extended through the use of Plug-In Extensions (PIEs). A PIE can provide an interface for a particular model, add a command to the Argus ONE menu structure, add a GIS function that can be used by Argus ONE, import data into Argus ONE, export data from Argus ONE, or implement a method of interpolating among data.

This report describes a number of interpolation methods, commands, and functions that have been added to Argus ONE through the use of PIEs. The PIEs described in this report are not specific to any particular type of model but instead may find application under a variety of circumstances. The new interpolation methods are much faster than existing interpolation methods in Argus ONE and can also incorporate anisotropy. The commands listed in tables 1 and 2 are useful for copying, editing, converting, or importing information. The commands listed in table 2 are hidden; special steps must be taken before they appear in

the menu structure. The functions listed in tables 3 and 4 provide convenient methods for evaluating, converting, or storing information. The functions in table 4 are hidden and do not appear in the Argus ONE Expression editor unless special steps are taken to reveal them but still will be accepted by Argus ONE. The hidden commands and functions were hidden for three reasons. (1) The hidden commands circumvent protections built into Argus ONE against users inadvertently changing or deleting information. (2) Some of the hidden functions can not be used in the Argus ONE parameters because they either may allocate memory that must be released later by a call to another function or because they use data stored in a previous function call. (3) The hidden functions provide specialized capabilities that are unlikely to be useful to most users although they are used extensively in the export template for the MODFLOW GUI (Winston, 2000).

Some of the PIEs described in this report, such as the List and BlockList PIEs, have been used in the MODFLOW GUI (Winston, 2000) or SUTRA GUI (Voss and others, 1997) and are distributed with them. Others have not been released previously.

The purpose for creating these PIEs was to meet needs within the U.S. Geological Survey (USGS). The USGS is using Argus ONE as a graphical-user interface for its ground-water modeling programs. Argus ONE was chosen for this purpose because it has an open architecture that allows the USGS to customize the graphical-user interfaces with less effort than would be required if they were created from scratch. Argus ONE has numerous GIS functions that facilitate the conversion of graphical information to the numerical information required by ground-water modeling programs. In some cases, however, Argus ONE did not have certain capabilities that would be useful to USGS scientists using the graphical-user interfaces. For example, Argus ONE did not have a method for importing data from USGS digital elevation models. To remedy the situation, a PIE command (described in this report) was developed that could read USGS digital elevation models and convert the data into a format that could be used by Argus ONE. Most of the commands and functions described in this report were developed for similar reasons. In a few cases, however, commands or functions described in this report duplicate capabilities already present in Argus ONE. These commands and functions were a byproduct of developing other commands and functions not present in Argus ONE. In some cases, the new functions may be faster than the corresponding functions in Argus ONE. It would have been possible to use such functions only for internal calculations. However, making the functions available as independent functions required almost no effort. In specific cases, the new functions can facilitate better performance of the graphical user interface.

An example of a command that partially duplicates Argus ONE functionality but also adds new functionality is the **Edit Grid** command. In Argus ONE, the user can set the grid angle by typing a value of the grid angle, but there is no graphical way of specifying the grid angle. With the **Edit Grid** command, the user can specify the grid angle graphically. In Argus ONE, the user can set the position of an interior grid line by double-clicking on it or by dragging it to a new position. However, Argus ONE does not have a similar capability for the exterior grid lines. With the **Edit Grid** command, the user can set the positions of both interior and exterior grid lines.

The PIEs described in this report are written in Object Pascal using the Borland Delphi version 5 compiler. When published, the source code will be available at <http://water.usgs.gov/nrp/gwsoftware/>.



**Table 1. Commands added to Argus ONE in Utility.dll**

<b>Command</b>	<b>Purpose</b>
PIEs Edit Edit Contours	Editing the exact positions of nodes on contours
PIEs Edit DeclutterContours	Removing extra nodes from contours
PIEs Edit Join Contours	Joining together open contours whose ends match exactly and which have exactly the same contour values
PIEs Edit Edit Grid	Creating and editing grids
PIEs Edit Edit Data	Editing the locations and values of data points on a data layer
PIEs Edit Create Parameters in Multiple Layers	Creates identical copies of parameters in multiple layers
PIEs Edit Set Multiple Parameters	Sets the value of multiple parameters in multiple layers in one step
PIEs Import Import Gridded Data	Importing data points at the centers of grid blocks
PIEs Import Import Points from Spreadsheet	Importing point contours from a spreadsheet
PIEs Import Import Contours from Spreadsheet	Importing point, open, or closed contours from a spreadsheet
PIEs Import Sample DEM Data	Import Digital Elevation Models into Argus ONE
PIEs Import Copy Tri Mesh	Copying a finite-element mesh with triangular elements to another mesh layer
PIEs Import Copy Quad Mesh	Copying a finite-element mesh with quadrilateral elements to another mesh layer
PIEs Convert Contours To Data	Converting contours to data points on a data layer
PIEs Convert Data to Contours	Converting data points on a data layer to point contours
PIEs Convert Reverse Contours on Clipboard	Reversing the order in which nodes occur in a contour
PIEs Convert Mesh Objects To Contours	Creating contours along the edges of selected finite elements
PIEs Convert Mesh To Contours	Creating closed contours that duplicate the shapes of finite elements

**Table 2. Hidden commands added to Argus ONE in Utility.dll**

<b>Command</b>	<b>Purpose</b>
PIEs Set Parameter Locks	Hidden command that locks or unlocks multiple parameters in one step
PIEs Delete Multiple Layers	Hidden command that deletes multiple layers in one step

**Table 3. Functions added to Argus ONE**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
Utility_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	Utility.dll	Returns True if the version number of the Utility PIE is greater than or equal to the version number passed in the arguments
EvalRealAtXY(X, Y, "Expression_As_Quoted_String", ["Layer_Name_As_Quoted_String"])	Utility.dll	A real number based on the evaluation of "Expression_As_Quoted_String" evaluated at (X,Y)
EvalIntegerAtXY(X, Y, "Expression_As_Quoted_String", ["Layer_Name_As_Quoted_String"])	Utility.dll	An integer based on the evaluation of "Expression_As_Quoted_String" evaluated at (X,Y)
EvalBooleanAtXY(X, Y, "Expression_As_Quoted_String", ["Layer_Name_As_Quoted_String"])	Utility.dll	A Boolean based on the evaluation of "Expression_As_Quoted_String" evaluated at (X,Y)
EvalStringAtXY(X, Y, "Expression_As_Quoted_String", ["Layer_Name_As_Quoted_String"])	Utility.dll	A string based on the evaluation of "Expression_As_Quoted_String" evaluated at (X,Y)
Rotated X('X', 'Y', 'GridAngle')	Utility.dll	A row/column position converted to an X coordinate
Rotated Y('X', 'Y', 'GridAngle')	Utility.dll	A row/column position converted to an Y coordinate
GetMyDirectory()	GetMyDirectory.dll	The name of the directory in which the PIE is installed
RF_Get_Value_From_File(Key, Default_Value, [FileName])	ReadFileValue.dll	A value from a file associated with "Key" or if "Key" is not found, the Default_Value
RF_Clear_Files()	ReadFileValue.dll	Attempts to clear a file and returns True if the function succeeds
RF_Save_Files()	ReadFileValue.dll	Attempts to save a file and returns True if the function succeeds

**Table 3 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
RF_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	ReadFileValue.dll	Returns True if the version number of the ReadFile PIE is greater than or equal to the version number passed in the arguments
Sec2Day(Number)	Utility.dll	Converts number from seconds to days and returns the result
Day2Sec(Number)	Utility.dll	Converts Number from days to seconds and returns the result
K2F(Number)	Utility.dll	Converts Number from degrees Kelvin to degrees Fahrenheit and returns the result
F2K(Number)	Utility.dll	Converts Number from degrees Fahrenheit to degrees Kelvin and returns the result
J2BTU(Number)	Utility.dll	Converts Number from Joules to British Thermal Units (BTU) and returns the result
BTU2J(Number)	Utility.dll	Converts Number from BTU to Joules and returns the result
sq_m2sq_ft(Number)	Utility.dll	Converts Number from square meters to square feet and returns the result
sq_ft2sq_m(Number)	Utility.dll	Converts Number from square feet to square meters and returns the result
cu_m2cu_ft(Number)	Utility.dll	Converts Number from cubic meters to cubic feet and returns the result
cu_ft2cu_m(Number)	Utility.dll	Converts Number from cubic feet to cubic meters and returns the result
m_s2ft_day(Number)	Utility.dll	Converts Number from meter-seconds to feet-days and returns the result
ft_day2m_s(Number)	Utility.dll	Converts Number from feet-days to meter-seconds and returns the result

**Table 3 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
Pa2psi(Number)	Utility.dll	Converts Number from Pascals to pounds per square inch (psi) and returns the result
psi2Pa(Number)	Utility.dll	Converts Number from psi to Pascals and returns the result
m_per_s2ft_per_day(Number)	Utility.dll	Converts Number from meters per second to feet per day and returns the result
ft_per_day2m_per_s(Number)	Utility.dll	Converts Number from feet per day to meters per second and returns the result
sq_m_per_s2sq_ft_per_day(Number)	Utility.dll	Converts Number from square meters per second to square feet per day or from cubic meters per meter-second to cubic feet per foot-day and returns the result
sq_ft_per_day2sq_m_per_s(Number)	Utility.dll	Converts Number from square feet per day to square meters per second and returns the result
cu_m_per_s2cu_ft_per_day(Number)	Utility.dll	Converts Number from cubic meters per second to cubic feet per day and returns the result
cu_ft_per_day2cu_m_per_s(Number)	Utility.dll	Converts Number from cubic feet per day to cubic meters per second and returns the result
l_per_s2cu_ft_per_day(Number)	Utility.dll	Converts Number from liters per second to cubic feet per day and returns the result
cu_ft_per_day2l_per_s(Number)	Utility.dll	Converts Number from cubic feet per day to liters per second and returns the result
kg_per_s2lb_per_day(Number)	Utility.dll	Converts Number from kilograms per second to pounds per day and returns the result
lb_per_day2kg_per_s(Number)	Utility.dll	Converts Number from pounds per day to kilograms per second and returns the result
Pa_per_s2psi_per_day(Number)	Utility.dll	Converts Number from Pascals per second to psi per day and returns the result

**Table 3 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
psi_per_day2Pa_per_s(Number)	Utility.dll	Converts Number from psi per day to Pascals per second and returns the result
kg_per_cu_m2lb_per_cu_ft(Number)	Utility.dll	Converts Number from kilograms per cubic meter to pounds per cubic foot and returns the result
lb_per_cu_ft2kg_per_cu_m(Number)	Utility.dll	Converts Number from pounds per cubic foot to kilograms per cubic meter and returns the result
W_per_cu_m2BTU_per_hr_cu_ft (Number)	Utility.dll	Converts Number from Watts per cubic meter to BTU per hour-cubic foot and returns the result
BTU_per_hr_cu_ft2W_per_cu_m (Number)	Utility.dll	Converts Number from BTU per hour-cubic foot to Watts per cubic meter and returns the result
J_per_kg2BTU_per_lb(Number)	Utility.dll	Converts Number from Joules per kilogram to BTU per pound and returns the result
BTU_per_lb2J_per_kg(Number)	Utility.dll	Converts Number from BTU per pound to Joules per kilogram and returns the result
J_per_kg2ft_lb_f_per_lb_m(Number)	Utility.dll	Converts Number from Joules per kilogram to foot-pound force per pound mass and returns the result
ft_lb_f_per_lb_m2J_per_kg(Number)	Utility.dll	Converts Number from foot-pound force per pound mass to Joules per kilogram and returns the result
cu_m_per_kg2cu_ft_per_lb(Number)	Utility.dll	Converts Number from cubic meter per kilogram to cubic foot per pound and returns the result
cu_ft_per_lb2cu_m_per_kg(Number)	Utility.dll	Converts Number from cubic foot per pound to cubic meter per kilogram and returns the result
cu_m_per_sq_m_s2cu_ft_per_sq_ft_ day (Number)	Utility.dll	Converts Number from cubic meter per square meter-second to cubic foot per square foot-day and returns the result

**Table 3 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
cu_ft_per_sq_ft_day2cu_m_per_sq_m_s (Number)	Utility.dll	Converts Number from cubic foot per square foot-day to cubic meter per square meter-second and returns the result
W_per_sq_m2BTU_per_hr_sq_ft (Number)	Utility.dll	Converts Number from Watts per square meter to BTU per hour-square foot and returns the result
BTU_per_hr_sq_ft2W_per_sq_m (Number)	Utility.dll	Converts Number from BTU per hour-square foot to Watts per square meter and returns the result
kg_per_sq_m_s2lb_per_sq_ft_day (Number)	Utility.dll	Converts Number from kilogram per square meter-second to pound per square foot-day and returns the result
lb_per_sq_ft_day2kg_per_sq_m_s (Number)	Utility.dll	Converts Number from pound per square foot-day to kilogram per square meter-second and returns the result
kg_per_m_s2cP(Number)	Utility.dll	Converts Number from kilogram per meter-second to centipoises and returns the result
cP2kg_per_m_s(Number)	Utility.dll	Converts Number from centipoises to kilogram per meter-second and returns the result
J_per_kg_m2BTU_per_lb_ft (Number)	Utility.dll	Converts Number from Joules per kilogram-meter to BTU per pound-foot and returns the result
BTU_per_lb_ft2J_per_kg_m (Number)	Utility.dll	Converts Number from BTU per pound-foot to Joules per kilogram-meter and returns the result
W_per_m_deg_C2BTU_per_ft_hr_deg_F (Number)	Utility.dll	Converts Number from Watts per meter-degree Celsius to BTU per foot-hour-degree Fahrenheit and returns the result
BTU_per_ft_hr_deg_F2W_per_m_deg_C (Number)	Utility.dll	Converts Number from BTU per foot-hour-degree Fahrenheit to Watts per meter-degree Celsius and returns the result

**Table 3 (Continued)**

<b>Function</b>	<b>File</b>	<b>Result</b>
W_per_sq_m_deg_C2BTU_per_hr_sq_ft_deg_F (Number)	Utility.dll	Converts Number from Watts per square meter-degree Celsius to BTU per square foot-hour-degree Fahrenheit and returns the result
BTU_per_hr_sq_ft_deg_F2W_per_sq_m_deg_C (Number)	Utility.dll	Converts Number from BTU per square foot-hour-degree Fahrenheit to Watts per square meter-degree Celsius and returns the result
J_per_kg_deg_C2BTU_per_lb_deg_F (Number)	Utility.dll	Converts Number from Joules per kilogram-degree Celsius to BTU per pound-degree Fahrenheit and returns the result
BTU_per_lb_deg_F2J_per_kg_deg_C (Number)	Utility.dll	Converts Number from BTU per pound-degree Fahrenheit to Joules per kilogram-degree Celsius and returns the result
J_per_cu_m_deg_C2BTU_per_cu_ft_deg_F (Number)	Utility.dll	Converts Number from Joules per cubic meter-degree Celsius to BTU per cubic foot-degree Fahrenheit and returns the result
BTU_per_cu_ft_deg_F2J_per_cu_m_deg_C (Number)	Utility.dll	Converts Number from BTU per cubic foot-degree Fahrenheit to Joules per cubic meter-degree Celsius and returns the result
cu_m_per_s_m_Pa2cu_ft_per_day_lb_sq_in (Number)	Utility.dll	Converts Number from cubic meter per second-meter-Pascal to cubic foot per day-point-square inch and returns the result
cu_ft_per_day_lb_sq_in2cu_m_per_s_m_Pa (Number)	Utility.dll	Converts Number from cubic foot per day-point-square inch to cubic meter per second-meter-Pascal and returns the result
m_per_sq_sec2ft_per_sq_day (Number)	Utility.dll	Converts Number from meter per square second to foot per square day-point-square inch and returns the result
ft_per_sq_day2m_per_sq_sec (Number)	Utility.dll	Converts Number from foot per square day to meter per square second and returns the result

**Table 4. Hidden functions added to Argus ONE**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
IsOK(Message, [HideCancel])	OkCancel.dll	Displays a dialog box with a message and several buttons Returns True or False depending on which button the user pushes
ok_Add_Radio_Choice(Message, [Message], [Message], [Message], [Message])	OkCancel.dll	Adds up to five radio button to a dialog box and returns True if it succeeds
ok_Get_Radio_Choice(Message, [Choices_Height], [Width], [Question_Height])	OkCancel.dll	Returns the index of the radio button that the user selects with the first radio button given an index of zero
ok_Radio_Free()	OkCancel.dll	Returns True if the dialog box with the radio buttons has been destroyed
ok_UserFloat(Message, Response, [Minimum], [Maximum])	OkCancel.dll	Displays a dialog box on which a user may enter a real number, the result is the real number that the user enters
ok_UserInteger(Message, Response, [Minimum], [Maximum])	OkCancel.dll	Displays a dialog box on which a user may enter an integer, the result is the integer that the user enters
ok_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	OkCancel.dll	Returns True if the version number of the OkCancle PIE is greater than or equal to the version number passed in the arguments
ProgressBarInitialize(Number, [Show_Cancel])	ProgressBar.dll	Attempts to create a dialog box with a progress bar and returns True if it succeeds
ProgressBarFree()	ProgressBar.dll	Attempts to destroy the dialog box with the progress bar and returns True if it succeeds
ProgressBarMax(Number)	ProgressBar.dll	Attempts to set the maximum value of the progress bar and returns True if the Abort button has not been pressed and the function succeeds
ProgressBarAdvance()	ProgressBar.dll	Attempts to advance the progress bar by one and returns True if the Abort button has not been pressed and the function succeeds



**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
ProgressBarSetMessage(Message)	ProgressBar.dll	Attempts to display a message with the progress bar and returns True if the Abort button has not been pressed and the function succeeds
ProgressBarAddLine(Message)	ProgressBar.dll	Attempts to add a line to the memo-box beneath the progress bar and returns True if the Abort button has not been pressed and the function succeeds
ProgressBarSaveToFile(File_Name)	ProgressBar.dll	Attempts to save the lines in the memo box and returns the number of lines in the file it saves
ProgressBarCheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	ProgressBar.dll	Returns True if the version number of the ProgressBar PIE is greater than or equal to the version number passed in the arguments
Join_Files(First_File, Second_File, Result_File)	JoinFiles.dll	Attempts to concatenate two files and returns True if the function succeeds
Delete_File(File_Name)	JoinFiles.dll	Attempts to delete a file and returns True if the function succeeds
Rename_File(Old_File_Name, New_File_Name)	JoinFiles.dll	Attempts to rename a file and returns True if the function succeeds
Split_File('Input_File, First_File, Search_String, Second_File [, Search_String, Third_File] [, Search_String, Fourth_File]...[, Search_String, Thirtieth_File])	JoinFiles.dll	Attempts to split a file into 2 to 30 separate files and returns True if the function succeeds
Int2Str(Number)	JoinFiles.dll	Returns the base 36 representation of Number
JF_CopyLines(Old_File_Name, New_File_Name, Line_Count, [Is_Local_File])	JoinFiles.dll	Copies Line_Count from Old_File_Name to a new file named New_File_Name and returns True if it succeeds.
JF_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	JoinFiles.dll	Returns True if the version number of the JoinFile PIE is greater than or equal to the version number passed in the arguments
L_Initialize()	List.dll	True (obsolete function)

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
L__CreateNewList()	List.dll	Creates a list and returns the number of the list that was created
L_SetListSize(ListIndex, Size)	List.dll	Sets the size of a list and returns True if the function succeeds
L_GetListSize(ListIndex)	List.dll	Returns the number of items in the list
L_FreeAList(ListIndex)	List.dll	Attempts to remove all items from a list and returns True if the function succeeds
L_AddToList(ListIndex, Value)	List.dll	Adds a value to a list and returns the position of Value in the list if the function succeeds
L_GetFromList(ListIndex, Index)	List.dll	Returns the item indicated by ListIndex and Index
L_SetListItem(ListIndex, Index, Value)	List.dll	Attempts to set the value of an item in a list and returns True if the function succeeds
L_DeleteListItem(ListIndex, Index)	List.dll	Attempts to remove an item from a list and returns True if the function succeeds
L_SortList(ListIndex)	List.dll	Sorts a list in ascending order and returns True if the function succeeds
L_EliminateDuplicates(ListIndex)	List.dll	Eliminates duplicate values from a sorted list and returns True if the function succeeds
L_IndexOf(ListIndex, Value)	List.dll	Returns the position of Value in the sorted list of the position of the first copy of highest number less than Value in the sorted list if Value is not in the sorted list
L_UnsortedIndexOf(ListIndex, Value)	List.dll	Returns the position of the first copy of Value in the list Returns -1 if Value is not in the list
L_CreateNew3DList(Maximum_X, Maximum_Y, Maximum_Z)	List.dll	Creates a 3D list and returns the number of the 3D list that was created
L_FreeA3DList(ListIndex)	List.dll	Attempts to destroy a 3D list and returns True if the function succeeds

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
L_GetFrom3DList(ListIndex, X_Index, Y_Index, Z_Index)	List.dll	Returns the item indicated by ListIndex X_Index, Y_Index, and Z_Index
L_GetFromOneBased3DList(ListIndex, X_Index, Y_Index, Z_Index)	List.dll	Returns the item indicated by ListIndex X_Index, Y_Index, and Z_Index
L_Set3DListItem(ListIndex, X_Index, Y_Index, Z_Index, Value)	List.dll	Sets the value indicated by ListIndex X_Index, Y_Index, and Z_Index and returns True if the function succeeds
L_SetOneBased3DListItem(ListIndex, X_Index, Y_Index, Z_Index, Value)	List.dll	Sets the value indicated by ListIndex X_Index, Y_Index, and Z_Index and returns True if the function succeeds
L_ResetA3DList(ListIndex)	List.dll	Sets all members of the 3D list indicated by ListIndex to 0 and returns True if the function succeeds
L_Add3DLists(FirstListIndex, SecondListIndex, ResultListIndex)	List.dll	Adds the corresponding members of two 3D lists and returns True if the function succeeds
L_Subtract3DLists(FirstListIndex, SecondListIndex, ResultListIndex)	List.dll	Subtracts the corresponding members of one 3D list from those of another 3D list and returns True if the function succeeds
L_Multiply3DLists(FirstListIndex, SecondListIndex, ResultListIndex)	List.dll	Multiplies the corresponding members of two 3D lists and returns True if the function succeeds
L_Divide3DLists(FirstListIndex, SecondListIndex, ResultListIndex)	List.dll	Divides the corresponding members of one 3D list by those of another 3D list and returns True if the function succeeds
L_Multiply3DByConstant(ListIndex, ResultListIndex, Value)	List.dll	Multiplies the members of a 3D list by a constant and returns True if the function succeeds
L_Invert3DListMembers(ListIndex, ResultListIndex)	List.dll	Inverts the members of a 3D list and returns True if the function succeeds

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
L_IsSingPrecUniform(ListIndex)	List.dll	Returns True if all the members of a list are the same after being converted to single precision
L_GetErrorCount()	List.dll	Returns the number of errors
L_FreeAllLists()	List.dll	Attempts to destroy all lists and 3D lists and returns True if the function succeeds
L_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	List.dll	Returns True if the version number of the List PIE is greater than or equal to the version number passed in the arguments
BL_InitializeGridInformation (Grid_Layer_Name_as_String, [GridType])	BlockList.dll	Attempts to read grid information from Argus ONE and returns 1 if the function succeeds
BL_AddVertexLayer (Information_Layer_Name_as_String)	BlockList.dll	Attempts to read contour information from Argus ONE and returns True if the function succeeds
BL_ReInitializeVertexList()	BlockList.dll	Attempts to delete all contour information that has been read from Argus ONE and returns True if the function succeeds
BL_GetCountOfCellLists()	BlockList.dll	Returns the number of lists of cells currently in memory
BL_GetCountOfACellList(ListIndex)	BlockList.dll	Returns the number of cells in a list of cells
BL_GetCellRow(ListIndex, Index)	BlockList.dll	Returns the row number of a cell
BL_GetCellColumn(ListIndex, Index)	BlockList.dll	Returns the column number of a cell
BL_GetVertexCount(ListIndex, CellIndex)	BlockList.dll	Returns the number of vertices of a contour in a cell
BL_GetVertexXPos(ListIndex, CellIndex, VertexIndex)	BlockList.dll	Returns the X-coordinate of a vertex
BL_GetVertexYPos(ListIndex, CellIndex, VertexIndex)	BlockList.dll	Returns the Y-coordinate of a vertex
BL_SegmentCount(ListIndex, CellIndex)	BlockList.dll	Returns the number of segments of a contour in a cell
BL_SegmentFirstX(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the X-coordinate of the first vertex of a segment

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
BL_SegmentFirstY(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the Y-coordinate of the first vertex of a segment
BL_SegmentSecondX(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the X-coordinate of the second vertex of a segment
BL_SegmentSecondY(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the Y-coordinate of the second vertex of a segment
BL_SegmentLengthX(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the length of a segment in the X-direction
BL_SegmentLengthY(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the length of a segment in the Y-direction
BL_SegmentLength(ListIndex, CellIndex, SegmentIndex)	BlockList.dll	Returns the length of a segment
BL_SumSegmentsX(ListIndex, CellIndex)	BlockList.dll	Returns the sum of the lengths in the X-direction of all the segments in a cell
BL_SumSegmentsY(ListIndex, CellIndex)	BlockList.dll	Returns the sum of the lengths in the Y-direction of all the segments in a cell
BL_SumSegmentLengths(ListIndex, CellIndex)	BlockList.dll	Returns the sum of the lengths of all the segments in a cell
BL_GetCountOfCombinedCellList()	BlockList.dll	Returns the number of cells intersected by any contour
BL_GetCellRowFromCombinedList (CellIndex)	BlockList.dll	Returns the row number of a cell intersected by any contour
BL_GetCellColumnFromCombinedList (CellIndex)	BlockList.dll	Returns the column number of a cell intersected by any contour
BL_GetCountOfCrossRowLists()	BlockList.dll	Returns the number of lists of cells in which a contour crosses the Y-coordinate of the cell node
BL_GetCountOfCrossColumnLists()	BlockList.dll	Returns the number of cells in a list of cells in which a contour crosses the X-coordinate of the cell node
BL_GetCountOfACrossRowList (ListIndex)	BlockList.dll	Returns the number of cells in a list of cells in which a contour crosses the Y-coordinate of the cell node
BL_GetCountOfACrossColumnList (ListIndex)	BlockList.dll	Returns the number of cells in a list of cells in which a contour crosses the X-coordinate of the cell node

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
BL_GetCrossRowRow(ListIndex, CellIndex)	BlockList.dll	Returns the row number of a cell in a list of cells in which a contour crosses the Y-coordinate of the cell node
BL_GetCrossColumnRow(ListIndex, CellIndex)	BlockList.dll	Returns the row number of a cell in a list of cells in which a contour crosses the X-coordinate of the cell node
BL_GetCrossRowColumn(ListIndex, CellIndex)	BlockList.dll	Returns the column number of a cell in a list of cells in which a contour crosses the Y-coordinate of the cell node
BL_GetCrossColumnColumn(ListIndex, CellIndex)	BlockList.dll	Returns the column number of a cell in a list of cells in which a contour crosses the X-coordinate of the cell node
BL_GetCrossRowNeighborColumn(ListIndex, CellIndex)	BlockList.dll	Returns the column number of the neighbor of a cell in a list of cells in which a contour crosses the Y-coordinate of the cell node
BL_GetCrossColumnNeighborRow(ListIndex, CellIndex)	BlockList.dll	Returns the row number of the neighbor of a cell in a list of cells in which a contour crosses the X-coordinate of the cell node
BL_GetCrossRowCompositeY(ListIndex, CellIndex)	BlockList.dll	Returns a total difference in Y-coordinate of segments in one or more cells
BL_GetCrossColumnCompositeX(ListIndex, CellIndex)	BlockList.dll	Returns a total difference in X-coordinate of segments in one or more cells
BL_GetSumCrossRowCompositeY(ListIndex)	BlockList.dll	Returns the sum of the BL_GetCrossRowCompositeY's for all the cells in the list that cross the Y-coordinate of the cell node
BL_GetSumCrossColumnCompositeX(ListIndex)	BlockList.dll	Returns the sum of the BL_GetCrossColumnCompositeX's for all the cells in the list that cross the X-coordinate of the cell node

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
BL_GetCrossRowCompositeLength (ListIndex, CellIndex)	BlockList.dll	Returns the length of the contour associated with a cell in the list that crosses the Y-coordinate of the cell node
BL_GetCrossColumnCompositeLength (ListIndex, CellIndex)	BlockList.dll	Returns the length of the contour associated with a cell in the list that crosses the X-coordinate of the cell node
BL_GetRowBoundary(Row)	BlockList.dll	Returns the position of the Row boundary indicated by Row
BL_GetColumnBoundary(Column)	BlockList.dll	Returns the position of the Column boundary indicated by Column
BL_PointInsideContour (ListIndex, X, Y)	BlockList.dll	Returns True if (X, Y) is inside the contour indicated by ListIndex
BL_GetRowNodePosition(Row)	BlockList.dll	Returns the Y position of the node of the Row indicated by "Row"
BL_GetColumnNodePosition(Column)	BlockList.dll	Returns the X position of the node of the Column indicated by "Column"
BL_GetRowBoundaryCount()	BlockList.dll	Returns the number of row boundaries in the grid
BL_GetColumnBoundaryCount()	BlockList.dll	Returns the number of column boundaries in the grid
BL_GetRowNodeCount()	BlockList.dll	Returns the number of row nodes in the grid
BL_GetColumnNodeCount()	BlockList.dll	Returns the number of column nodes in the grid
BL_GetCellArea(Column, Row)	BlockList.dll	Returns the area of the cell indicated by Column, Row
BL_FractionOfLine(ListIndex, CellIndex)	BlockList.dll	Returns the fraction of the total length of the contour inside the cell indicated by CellIndex
BL_FreeVertexList()	BlockList.dll	Attempts to free all memory associated with a list of vertices and returns True if the function succeeds
BL_FreeAllBlockLists()	BlockList.dll	Attempts to free all memory allocated by the BlockList PIE and returns True if the function succeeds

**Table 4 (Continued)**

<b>Function(arguments)</b>	<b>File</b>	<b>Result</b>
BL_GetErrorCount()	BlockList.dll	Returns the number of errors
BL_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)	BlockList.dll	Returns True if the version number of the BlockList PIE is greater than or equal to the version number passed in the arguments






### ***Installation instructions***

All the PIEs described in this report may be installed by placing the dynamic link libraries (dll's) in the Argus Interware\ArgusPIE directory or a subdirectory of it. In cases where the PIE has a help system, the help system files (with the extensions .hlp and .cnt) must be placed in the same directory as the dll. Normal practice is to create a subdirectory in the ArgusPIE directory with the same name as the dll except without the dll extension and place the files there. For example, the Utility PIE would normally be installed as "Argus Interware/ArgusPIE/Utility/Utility.dll".

Some of the commands or functions described in this report are now implemented in the Utility.dll whereas previously they were implemented in several different dll's. The new versions replace and often improve upon the previous versions so the dll's containing the previous versions must be removed when installing the new versions. The dll's that have been replaced are EditContoursPie.dll, EditDataLayer.dll, GriddedImport.dll, JoinContoursPie.dll, MoreConversions.dll, EvalAtXY.dll, and RotateCells.dll. **If the conflicting dll's are not removed, the Utility.dll can not be used.**

Two of these PIEs (JoinContoursPie.dll and MoreConversions.dll) were written by the author prior to joining the USGS. Two of them (EditContoursPie.dll, RotateCells.dll) were documented as part of version 3 of the MODFLOW GUI (Winston, 1999). Three more (GriddedImport.dll, EvalAtXY.dll, and EditDataLayer.dll) were documented as part of version 4 of the MODFLOW GUI (Winston, 2000).

### ***Navigating***

A number of the dialog boxes described in this report display spatial data and allow the user to zoom in on or out of an area of interest or to pan to a different location. Standardized buttons have been used for these operations. To zoom out all the way, click the **Zoom Extents** button . To zoom in by a factor of two, click the **Zoom In** button . To zoom out by a factor of two, click the **Zoom Out** button . To zoom to a specified region, click the **Zoom** button  and select to region to zoom in on. To select the region, click the mouse in one corner of the region, hold down the mouse button while moving to the opposite corner, and release the mouse button. To pan, click the **Pan** button  and then hold down the mouse button while moving the mouse to a new position.



## **Acknowledgments**

I would like to thank Allen M. Shapiro and Bernard J. Stolp for their helpful reviews of this manuscript. I would also like to thank Leonard Konikow, Clifford Voss, Alden Provost, George Z. Hornberger, Allen Shapiro, and Martha Scholl for helpful suggestions.

## **Interpolation Methods**

Several new interpolation methods are included in Utility.dll. The new methods have two advantages over existing interpolation methods in Argus ONE.

1. They are much faster.
2. They can incorporate anisotropy.

To test the speed of the new interpolation methods, a test case was constructed with a grid containing approximately 500 cells. The grid had one parameter that was linked to a data layer containing approximately 900,000 points (from a digital elevation model). When the interpolation method used for the data layer was one of the new methods, coloring the grid the first time took about 7 seconds. If the Argus ONE window was minimized and then maximized again, recoloring the grid took about 1 second. If the default interpolation method for the data layer was used, coloring the grid took about 1 minute regardless of whether it was being colored for the first time or was being recolored. Other interpolation methods all took even longer.

There are 10 new interpretation methods as listed below.

QT\_Nearest  
QT\_Mean of 5 Nearest  
QT\_Mean of 20 Nearest  
QT\_Inv Dist Sq of 5 Nearest  
QT\_Inv Dist Sq of 20 Nearest  
QT\_Nearest (Anis = 100)  
QT\_Mean of 5 Nearest (Anis = 100)  
QT\_Mean of 20 Nearest (Anis = 100)  
QT\_Inv Dist Sq of 5 Nearest (Anis = 100)  
QT\_Inv Dist Sq of 20 Nearest (Anis = 100)

**QT\_Nearest** returns the value of the data point that is closest to the location for which a value is requested. This is exactly equivalent to the NN2D method (except faster). In the event that two locations are equally distant from the location for which a value is requested, an arbitrary choice between the values will be made.

**QT\_Mean of 5 Nearest** finds the five data points that are closest to the location for which a value is requested and returns their mean value. However, it is possible that there will be additional data points that are no further way from the desired location as the point that is the fifth most distant from it. If that is the case, all data points that are at the same distance as the fifth most distant data point will be included in the calculation.

**QT\_Mean of 20 Nearest** is the same as **QT\_Mean of 5 Nearest** except that it uses the 20 closest points rather than the five closest points.

**QT\_Inv Dist Sq of 5 Nearest** retrieves the five closest data points to the search location in the same way as **QT\_Mean of 5 Nearest** and then calculates a weighted mean of the data points. The weights applied to each data point are the inverses of the square of their distances from the search location. (In the event that one or more data points lies exactly at the search location, the inverse of the distance can not be calculated so the value that is returned is the mean of the values of all the data points that lie exactly at the search location.)

**QT\_Inv Dist Sq of 20 Nearest** is the same as **QT\_Inv Dist Sq of 5 Nearest** except that it uses the 20 closest points rather than the five closest points.

**QT\_Nearest (Anis = 100)** is the same as **QT\_Nearest** except that the Y coordinates of all the data points and the search location are multiplied by 100 before performing any operations. This would be useful for performing interpolations in cross sectional models where aquifer properties are typically much more continuous in the horizontal than in the vertical direction.

The remaining anisotropic methods are the same as their isotropic equivalents except that the Y coordinates are all multiplied by 100. For instance, **QT\_Mean of 5 Nearest (Anis = 100)** is the same as **QT\_Mean of 5 Nearest** except that the Y coordinates of all the data points and the search location are multiplied by 100 before performing any operations.

The reason for the “QT” in the name of all these interpolation methods is that the data are stored in a data structure known as a Quadtree (Stephens, 1998). The advantage of using a Quadtree is that it can make retrieving data of the sort required by interpolation methods much faster than would be required by a sequential search through the data.

## Commands

The commands described here are all implemented in the file Utility.dll. The commands appear in the Argus ONE menus. Commands are in **bold** type to help distinguish them from the surrounding text although other items may also in bold type. In the description of commands, a vertical bar (|) is used separate menu items from submenu items. For example, and instruction to select **PIEs|Edit** means to select the **PIEs** menu item and then to select the **Edit** submenu item from the **PIEs** menu.

### **Edit**

#### **Edit Contours**

The **Edit Contours** command is used to edit the positions of individual vertices in contours. To use it, select **PIEs|Edit|Edit Contours**. You will be prompted for a layer name. Select the layer for which you wish to edit a vertex position. All the contours from the layer will be read and displayed. Click on any vertex, and a dialog box will appear in which you can edit the vertex position. When you are done, click OK. The layer will be cleared and a new set of contours will be written with the new vertex positions.

Users of the **Edit Contours** command should be aware of the following characteristics of the command before using it.

1. The contour parameter values will be the same as they were originally; however, any parameters that were set using *Expressions* will now be specified in the contour itself. To go back to having the contour value being set by the expression for the parameter, edit the contour and delete the parameter value.

2. The command does not check for contours that are illegal in Argus ONE. A contour is illegal if it crosses another contour and "Allow Intersection" is not turned on for that layer. A contour is also illegal if it crosses itself even if "Allow Intersection" is turned on. Argus ONE will not accept illegal contours so illegal contours will be lost if you specify contours that are illegal. For this reason, it is best to back-up your file before using the Edit Contours command. If you anticipate wanting to import contours that touch, you should select "Allow Intersection" beforehand.

### DeclutterContours

Reducing the number of vertices in a layer can make the export process faster. If the vertices are much more closely spaced than the grid spacing, reducing the number of vertices may have little effect on the model inputs. To use this command, select **PIEs|Edit|Declutter Contours**. A dialog box will appear with a list of information layers. Select the Domain Outline or Information layer in which you wish to reduce the number of vertices. Another dialog box will appear in which you should enter the desired vertex spacing and an angle used to control which vertices are removed.

When you click OK the contours on the layer will be processed so that vertices that are closer together than the desired spacing will be eliminated if the "Delete nodes based on spacing" check box is checked. However, if the "Delete Node based on angle" check box is checked, nodes will only be deleted if the angle at the node exceeds the limit you specify. The effect of **Declutter Contours** is illustrated in Figure 1. Figure 1A shows contours with very closely spaced vertices many of which have been eliminated in Figure 1B by using the **Declutter Contours** command.

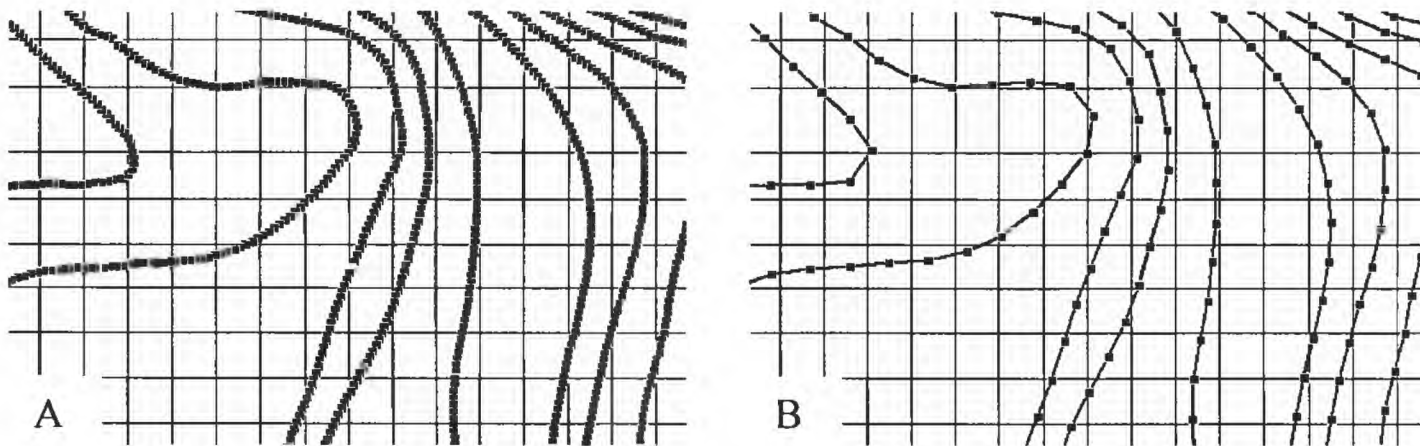
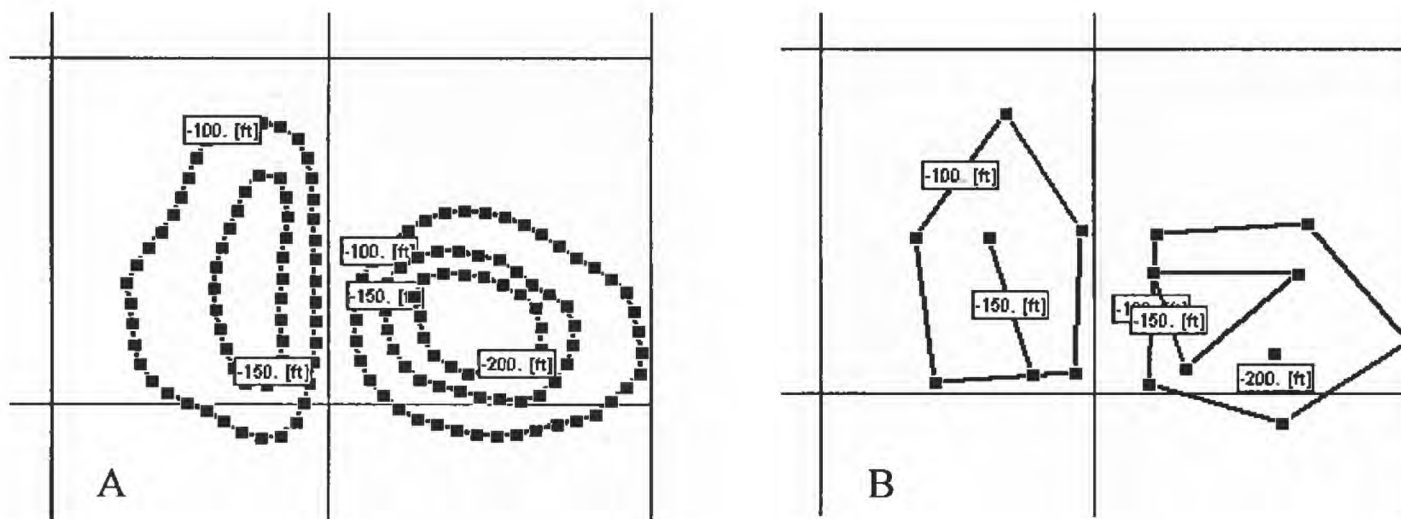


Figure 1. Effect of using the Declutter Contours command. (A) Before. (B) After.

A word of warning is in order about the **Declutter Contours** command. After removing vertices, some contours that previously did not overlap may overlap and the geometric relationships among contours may differ. For example, in Figure 2, the -200 ft contour has been reduced to a single point that is no longer inside the -150 ft contour as it was before. The user should check the contours for such problems after using this command.

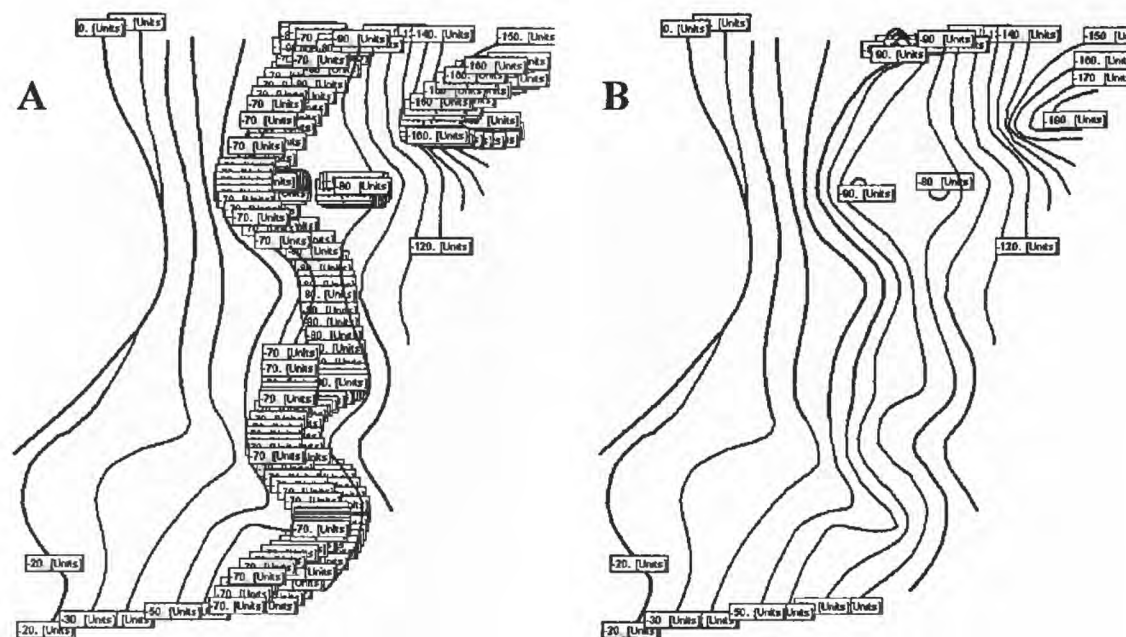


**Figure 2.** If used improperly, the Declutter Contours command can result in contours that overlap or that no longer show correct geometric relationships. (A) The contours before the Declutter Contours command was used. (B) The contours after the Declutter Contours command was used. Note that the -200 ft. contour is outside the -150 ft contour after the Declutter Contours command was used.

Both the **Join Contours** (to be described next) and **Declutter Contours** commands first clear the layer before writing to it. If there is an error when writing the data to the layer, this will result in loss of data. An error might occur in the **Declutter Contours** command if some of the contours cross themselves after some vertices are removed. Thus it is a good idea to save the Argus ONE file before running either of these PIEs. It usually would not cause an error if contours crossed each other because the **Declutter Contours** command automatically turns on "Allow Intersection" for the information layer that it is using. However, "Allow Intersection" is not allowed for Domain Outline layers so for them, it would cause an error.

### **Join Contours**

You may sometimes wish to join contours together to make it easier to manipulate them. To join contours together, select **PIEs|Edit|Join Contours**. A dialog box will appear listing the information layers. Select the layer in which you want to combine contours. The contours will be read from the layer. If any two contours have exactly the same starting or ending point and all their parameter values are the same, those contours will be combined into a single contour (Figure 3).



**Figure 3. Effect of the Join Contours command (A) Before the Join Contours command is applied, there are a large number of contours that meet at their ends and that have exactly the same contour values. Each contour has its own label. (B) After the Join Contours command is applied, these contours have been joined together.**

If two contours have starting or ending points that are even slightly different, the contours will not be combined. To ensure that such contours are combined, make sure the "Special|Allow Intersection" is checked. Then move the end of one of the contours close to the end of the other. The cursor will change to a hollow cross to indicate that it has detected another vertex. When you release the mouse, the vertex will be placed exactly over the position of the other vertex.

### **Edit Grid**


The **Edit Grid** command is used for creating and editing grids. The user can move or rotate the grid, or add, delete, or move rows and columns. The user can also subdivide rows and columns or specify the width or positions individual rows and columns. To ensure numerical stability and accuracy during model computation, it is a good idea to make sure that the ratio between the widths of adjacent rows and columns is less than or equal to a 1.5 (Anderson and Woessner, 1992). The **Edit Grid** command can automatically adjust an existing grid or create a new one that meets this criterion.

To start the Edit Grid command, select **PIEs|Edit|Edit Grid**. You will be prompted for the name of the grid layer whose grid you want to edit or you can choose to create a new grid layer. You will also be prompted for the type of grid (Block-Centered or Grid-Centered). If you decide to create a new grid layer, you can use either existing Domain Outline and Density layers or you can create new ones. Once you have selected or created the grid layer, the main dialog box will appear. It will have a copy of the grid (if there is one) on the grid layer you selected. You can edit the grid in this dialog box.

## Moving the Grid

To **move the grid**, you change the coordinates of the grid origin: the corner of the grid at column 1, row 1. There are two ways to change the coordinates of the grid origin.

1. Type the values of the X and Y coordinates of the grid origin in the appropriate edit boxes.

2. Click on the **arrow** button . Next, click inside the grid. The cursor will change to a hand pointer. While holding the mouse button down, move to the location where you would like the grid to be. A gray outline of the grid will move along with the mouse. When you release the mouse button, the coordinates of the grid origin will be moved to reflect the position you specified.


## Rotate the Grid

There are three ways to **rotate the grid**.



1. Type the angle of the grid into the edit box labeled Grid Angle.





2. Rotate the knob at the bottom of the dialog box.

3. Click on the **rotate** button . Next, click inside the grid. The cursor will change to a hand pointer. While holding the mouse button down, move to the location where you would like the grid to be. A gray outline of the grid will rotate along with the mouse. When you release the mouse button, the angle of the grid will be changed to reflect the position you specified.



## Moving Grid Lines

To **move grid lines**, click on the **move column**  or **move row**  button. Then move the mouse over the column or row line you wish to move. The cursor will change to an image similar to those on the buttons when the cursor is over a column or row. Click the mouse button and hold it down. As you move the mouse, an image of the column or row will follow the mouse. When you release the mouse, the column or row will be moved. You can also move grid lines by specifying column and row positions or by specifying column and row widths as described later in this section.


## Adding Grid Lines

To **add row or column grid lines**, click on the **add column**  or **add row**  button. Then move the mouse to where you wish to add a column or row. When you click the mouse button and release it, a new column or row will be added at the mouse position. As you move the mouse, an image of the new column or row boundary will follow the mouse. You can also add grid lines by specifying column and row positions or by specifying column and row widths as described later in this section.


## Changing the Width of Rows or Columns

To **change the width of rows or columns**, click on the **column width**  or **row height**  button. Then move the mouse over one of the columns or rows whose width you wish to change. Hold down the left mouse button and, if desired, move the mouse to another column or row. When you release the mouse button, a dialog box will appear where you can enter a new width for all the selected columns or rows. As you move the mouse with the button down, the selected columns or rows will be shown in gray. You can also change the width of rows or columns by specifying column and row positions or by specifying column and row widths as described later in this section.


## Deleting Grid Lines

To **delete grid lines**, click on the **delete** button . Then move the mouse over one of the grid lines that you wish to delete and click the mouse button. As you move the mouse with the button down, the selected columns or rows will be shown in gray and the cursor will change to an X. You can also delete grid lines by specifying column and row positions or by specifying column and row widths as described later in this section.


## Subdividing Rows or Columns

To **subdivide rows or columns**, click on the **subdivide** button . Then move the mouse over one of the columns or rows whose width you wish to change. Hold down the left mouse button and, if desired, move the mouse to another column or row. When you release the mouse button, a dialog box will appear where you can enter the number of columns and rows that you wish to subdivide the selected columns or rows into. If you enter a number greater than one for either and click the OK button, each selected columns and row will be subdivided into the number of columns and rows you specify. As you move the mouse with the button down, the selected columns or rows will be shown in gray. You can also subdivide rows or columns by specifying column and row positions or by specifying column and row widths as described below.

## Specifying Row and Column Positions


To **specify row and column positions**, click on the **Column/Row Positions** button . In the dialog box, you can increase or decrease the number of columns or rows and enter new positions for columns or rows. The positions don't need to be in order because they will be arranged automatically.

## Specifying Row and Column Widths


To **specify row and column widths**, click on the **Column/Row Widths** button . In the dialog box, you can increase or decrease the number of columns or rows and enter new widths for the columns or rows. You can drag rows in the table to rearrange them.

## Adjusting Row and Column Boundaries

According to Anderson and Woessner (1992), the maximum ratio between adjacent row or column widths should usually be less than or equal to 1.5. They also recommend that the maximum ratio between the length and width of individual cells should be less than 10. Numerical difficulties in solving the model equations arise less frequently in models with grids that meet these criteria.

To **adjust the row and column boundaries** so that the ratio between the size of adjacent rows and columns is less than a specified value, first, enter a number greater than 1 in the grid smoothing criterion edit box. This number represents the maximum desired ratio between the sizes of adjacent columns or rows. Then click the **smooth grid** button . The positions of the grid lines will be changed in an attempt to satisfy the criterion. If it doesn't succeed, the maximum ratio in the grid will be displayed in a warning message. Clicking the button again may reduce the ratio further.

## Creating Grids

If there isn't a grid already, you can add row or column grid lines to create it as described above. If you want to **create a grid** using information from the Domain Outline and Density layers, first set the grid angle and smoothing criterion to the desired value. Then click on the **Magic Wand** button . A dialog box will appear in which you may select the

names of the Domain Outline and Density layers and the type of grid to be created. When you have done this, the grid will be created.

When you click the **OK** button, your edited grid will be imported into Argus ONE. **However, all the cells in the grid will be active.** To deactivate the cells outside the domain outline, click with the Argus ONE "Magic Wand" button inside the domain outline and choose "Deactivate". When the operation has finished, all cells outside the Domain outline will be shaded to show they are inactive.

## Edit Data

To use this procedure, select **Files|Edit|Edit Data**.




You must have at least one parameter on the data layer before it can be edited. If you attempt to edit a data layer that has no parameters, you will get an error message about the problem.

If there are no data points on the data layer, you will get a warning message but you will still be able to edit the data layer. When you edit the data layer, you can: move data points, change the value or values of data points, add data points, and/or delete data points.



When you are finished editing the data points click on the **OK** button to accept the changes or the **Cancel** button to discard them. The data points appear both on the **Graphical** tab and the **Table** tab.

The **OK** and **Cancel** buttons will both close the dialog box but if the **OK** button is clicked the changes made in the dialog box will be kept. Otherwise they will be discarded. Other buttons on the Graphical tab are used for navigating.

You can move data points, change their values, add new data points or delete existing data points on both the **Graphical** and **Table** tab.

On the **Graphical** tab, you can do these tasks as follows. To move a data point or edit its values, make sure that the **Select point**  button is depressed and click on the data point. Then edit the X and Y coordinates in the edit boxes or edit the parameter values in the table. To add a data point, first make sure that the **Add point**  button is depressed and then click at the location where you would like to add a data point. Then edit the parameter values and X and Y coordinates and data values in the dialog box. To delete a data point, first make sure that the **Delete point**  button is depressed and then click on the data point you would like to delete.

On the **Table** tab, you can do these tasks as follows. To move a data point, locate the data point in the table of data points and edit the X and Y coordinates in the table. To change the values of a data point, locate the data point in the table of data points and edit the parameter values in the table. To add a data point, increase the number of data points by clicking on the up arrow next to the control that shows the number of data points. Then edit the parameter values and X and Y coordinates in the table. To delete a data point, locate the data point you would like to delete and drag it to the bottom of the table. Then decrease the number of data points by clicking on the down arrow next to the control that shows the number of data points.

If you have data in a spreadsheet you can copy them to the clipboard and paste them in the table on the **Table** tab by clicking on the **Paste from Clipboard**  button. You can also read the data from a file by clicking the **Read from file**  button. Several file formats can be used as described below.



If the file or clipboard contains data for more parameters than exist on the data layer, extra columns will be created in the table. You can drag the columns to new positions to determine which values will actually be used.

Data to be read from a file or pasted from the clipboard must either be in the format of an Argus ONE contour as described in the Argus ONE documentation (Argus Interware, Inc., 1997, p. 104 to 108) or they must have the following format.

1. Any line in the data whose first character is a "#" will be considered a comment and will be used to assign names to imported parameters.
2. Each line in the data will begin a new row in the table except for comment lines.
3. The data may be either in a tab-delimited format or a comma/space-delimited format.
4. In tab-delimited format, the data value that will appear in each cell in a row is separated from the data value in the next cell in the row by a tab character.
5. In comma/space-delimited format, the data value that will appear in each cell in a row is separated from the data value in the next cell in the row by one or more tab characters, commas, or spaces. If you wish to import strings that include spaces while still using the comma/space delimited format, enclose the strings in quotation marks.

Note: If you copy data from a commercial spreadsheet to the clipboard, the data will generally be in a tab-delimited format.

### **Create Parameters in Multiple Layers**

This command allows you to add identical parameters to multiple layers in one step. To activate this command, select **PIEs|Edit|Create Parameters in Multiple Layers**. A dialog box will appear on which are listed those layers that can have parameters. Make sure that the check box next to each layer is checked for every layer to which you wish to add the parameter. In the bottom half of the dialog box, enter the parameter name, type, units, and value (expression). If you click the OK button, a parameter of the type you defined will be added to each of layers you selected. However, for any data layer, the type of the parameter will always be a real number regardless of the type that you specify because that is the only type allowed in data layers.

### **Set Multiple Parameters**

This command allows you to set the expressions of multiple parameters in multiple layers in one step. To activate this command, select **PIEs|Edit|Set Multiple Parameters**. A dialog box will appear with a hierarchical arrangement of layers and their parameters. A parameter will not appear on the list if it has been locked so that its expression can not be modified. Make sure that the check box is checked next to every parameter whose expression you wish to change. Set the value (expression) for the parameters in the edit box and click the OK button to set the expressions. The expressions are not checked for correctness; the user is responsible for ensuring their validity. In the case of complicated expressions, it may be helpful to create the expression first using the Argus ONE expression editor. Then copy the expression to the clipboard and apply it to multiple parameters using this command.

## **Import**

### **Import Gridded Data**

**Import Gridded Data** is a command for Argus ONE that facilitates importing gridded data into Argus ONE. This capability is helpful for importing data from existing finite difference models into Argus ONE. A new data layer will be created with data points at the center of the cells in an existing grid in Argus ONE. Each data point will have one or more associated parameters. The PIE provides an easy method for creating these data points and for setting the values of the data points.

You must create a grid in Argus ONE before you can use the **Import Gridded Data** command because the command will attempt to create data points at the center of each of the cells in the grid. Consult the Argus ONE documentation for how to create a grid. Once you have created a grid, select the **PIEs|Import|Import gridded data** menu item. The main window is where the gridded data are prepared for importing into Argus ONE. If the checkbox labeled **Export each data set to a separate layer** is checked, each data set that is imported into Argus ONE will be imported as a separate layer. By default, they are all imported into the same layer.

In the edit box labeled **Layer Name** is the name of the data layer on which the data should be placed. If there is no layer by this name, it will be created. If it does exist, you will be given a choice of overwriting the existing data or selecting a new layer name that does not exist. This edit box is only enabled if all the data sets will be imported into a single layer.

The **Number of data sets** is the number of data values associated with each new data point. Each value will be stored in an Argus ONE parameter. You must also assign names to the Argus ONE **Parameter names**. Parameter names must be unique within the layer. However, if each data set will be imported into a separate layer, enter the layer names in the table rather than parameter names.

There may be some cells for which you do not wish to create data points. If these cells have one or more values that do not occur in the rest of the data, they can be ignored. Set the number of data values and then set the values to be ignored. If the data will all be imported to a single layer, only the first data set is checked in this way so the rest of the data sets must have valid data for all locations that are valid in the first data set.

When pasting data from the clipboard, the values for individual data points must be in one of the following formats:

- **Tab-delimited:** each row of the table starts on a new line. Each data value on the line is separated from the next by a single tab.
- **Comma, space-delimited:** each row of the table starts on a new line. Each data value on the line is separated by one or more commas, spaces, or tabs.

In either case, lines beginning with a "#" character will be considered comments and ignored. If you copy data from a spreadsheet to the clipboard and paste them to the table, it should work with either format. Data that are copied to the clipboard from spreadsheets are generally in tab-delimited format.

In some cases, the data you want to import may have multiple lines of data in the file that all belong on the same row of the model. If the **Multiple data lines per grid row** checkbox is checked, each row of the grid will be filled before going on to the next row of the grid.

After copying the desired data to the clipboard, click on **Paste from clipboard** button to paste them in the table. Be sure to use the right format for the data as described above.

You may also read data from one or more files each containing a single two-dimensional array that you want to import. Click the **Read data from files** button and select the files. If required, the number of data sets will be increased to accommodate the number of files you select. Each parameter will be assigned the name of the file from which its data were imported. (You can then edit those names if you wish.) The first file to be imported will be placed in the table on the tab that is active. As each file is read, the selected tab will be changed so that each file goes in its own table.

The table contains the data to be placed in the model. Note that the numbers of the rows and columns in the table will match the numbering of the rows and columns in the grid. For example, if in Argus ONE, the Grid direction is Negative X, the columns in Argus will be numbered from right to left instead of left to right. However, the columns in the table will still be numbered from left to right.

Clicking the OK button will create one or more data layers with the data you have entered.

### **Import Points from Spreadsheet**

**Import Points from Spreadsheet** allows you to import point contours into Argus ONE from a spreadsheet-like format. To use it, select **PIEs|Import|Import Points from Spreadsheet**. Then select the information layer into which you wish to import point contours. A dialog box will appear in which you can enter the X and Y coordinates of the point contours and all the parameter values. If you have a spreadsheet with this data, you can arrange the data in the spreadsheet in the same order as shown in the table and copy the data to the clipboard. Then paste the data by clicking the **Paste from clipboard** button. The data may be either in a tab-delimited format or it can be delimited by commas and/or spaces as described in the section entitled "Import Gridded Data". If the data are in a text file, you can also read them from that text file by clicking the **Read from file** button.

When the contours are about to be imported, the user will be prompted to either retain the existing contours or delete them. This option is provided because two or more point contours can exist at the same location and importing new point contours can conceal the presence of existing contours at the same location.

### **Import Contours from Spreadsheet**

**Import Contours from Spreadsheet** allows you to import contours into Argus ONE from a spreadsheet-like format. To use it, select **PIEs|Import|Import Contours from Spreadsheet**. Then select the information layer into which you wish to import point contours. A dialog box will appear in which you can enter the X and Y coordinates of the contours in one table and the parameter values in a second table. If you have a spreadsheet with this data, you can arrange the data in the spreadsheet in the same order as shown in the table and copy the data to the clipboard. Then paste the data by clicking the **Paste from clipboard** button. The data may be either in a tab-delimited format or they can be delimited by commas and/or spaces as described in the section entitled "Import Gridded Data". If the data are in a text file, you can also read them from that text file by clicking the **Read from file** button.

The number of contours to be imported and the maximum number of vertices in any of the contours are set in a pair of edit boxes in the lower left corner of the **Contour Coordinates** tab of the **Import Contours** dialog box. If data are pasted into the table, the number of contours and the maximum number of vertices per contour will be automatically updated to match the pasted data. Similarly, on the **Parameter Values** tab, the number of columns will be adjusted to accommodate the data pasted into the table. However, the Argus ONE layer into which the data are to be imported can only hold data for the parameters that already exist. Therefore, the user should either insure that sufficient parameters already exist in the layer to hold the data to be imported before starting to import it or may rearrange the columns in the table by dragging them to new positions so that the correct data are imported.

### **Sample DEM Data**

**Sample DEM data** reads Digital Elevation Models (DEMs) in the format described in the DEM data users guide (U.S. Department of the Interior, U.S. Geological Survey, 1992) (<http://rockyweb.cr.usgs.gov/nmpstds/demstds.html>). The 1:250K DEMs at <http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/ndcdb.html> are in this format. The DEMs are displayed as bitmaps in the Universal Transverse Mercator (UTM) projection. To use it, select **PIEs|Import|Sample DEM data**.

When you select **Sample DEM data**, you are first prompted for whether you wish to import just the outline of the DEM, or data relative to a grid or mesh. If you wish to import data relative to a grid, you must specify whether the grid is "Block-Centered" or "Grid-Centered".

When you click the **OK** button, another dialog box will appear. In it select **File|Open** to select a DEM file to read. If you choose to import just the outline of the DEM into Argus ONE, you will be prompted whether you wish to read just the outline of the DEM or the full DEM. As the file is read, it will be displayed in the dialog box.

The size of the bitmap showing the DEM is determined by the size of the display area when the DEM is read. If the size of the display area is changed, you can select **File|Refresh** to create a new bitmap at the current size. Although the user can zoom in and out, this is accomplished by stretching the bitmap rather than by redrawing it so zooming, by itself, does not change the size of the bitmap. To save a bitmap, select **File|Save as BMP** or **File|Save as JPEG**. (Argus ONE can read the BMP but not the JPEG files)

If the check box labeled **High elevations are red** is checked, elevations are color coded from red at the highest elevation to purple at the lowest elevation. If it is not checked, the color scale is reversed.

It is possible to read more than one DEM at a time. To do so, just select multiple DEM files when you select **File|Open**. However, if the DEMs come from zones that use different central meridians to calculate the UTM coordinates, you will be prompted to choose which central meridian you wish to use.

If you wish to emphasize the coastline, it is a good idea to leave the checkbox labeled **Make locations with elevations of 0 white** checked. However, for areas that have elevations below sea level, you may wish to uncheck this checkbox.

When determining the central meridian for calculating the UTM coordinates, all UTM zones are treated as being exactly 6 degrees wide. This is not correct for a few zones. The area where this is not true is from 0 to 42 degrees east north of 72 degrees north and from 0 to 12 degrees east between 56 and 64 degrees north. Because the author is unaware

of any available DEMs in this area in the format accepted by this program, no effort has been made to address this issue.

If you click the OK button, the data are imported into Argus ONE at blocks, nodes, or element centers. You may choose to use either a mean value for the block node or element, the closest DEM data value to the point of interest, the highest value in the area around the point of interest, or the lowest value in the area around the point of interest. For block-centered grids, such as the MODFLOW grid (Harbaugh, Banta, Hill, and McDonald, 2000), the data are imported at the center of the block. For grid-centered grids, such as the HST3D grid (Kipp, 1997), the data are imported at the grid nodes.

The ability to convert DEMs to colored bitmaps might be of interest to those who do not have Argus ONE. Because this aspect of the PIE does not rely on Argus ONE, a stand-alone program, DEM2Image.exe, was created (Appendix 6).

### **Copy Tri Mesh/Copy Quad Mesh**

To use these commands, select **PIEs|Import|Copy Tri Mesh** or **PIEs|Import|Copy Quad Mesh**. Then select a pair of Tri Mesh or Quad Mesh layers. The mesh on one of the layers will be copied to the other layer. Only the nodes and elements will be copied. Parameters and parameter values will not be copied.

### **Convert**

#### **Contours To Data**

To use this procedure, select **Files|Convert|Contours To Data**. You will then be prompted to select the name of an Information or Domain Outline layer. After you have selected it you will be prompted to select the name of a new or existing data layer.

The contours on the Information or Domain Outline layer will be copied to the data layer. A data point will be placed at the position of each vertex of the contours.

All the data points for a contour will have the same value even if the value of the original contour is set by an expression that varies along the length of the contour.

#### **Data to Contours**



**Data to Contours** converts data points on a data layer to point contours on an information layer. To use it, select **PIEs|Convert|Data to Contours**. Then select the data layer whose data points you wish to convert.

When the contours are about to be imported, the user will be prompted to either retain the existing contours or delete them. This option is provided because two or more point contours can exist at the same location and importing new point contours can conceal the presence of existing contours at the same location.

#### **Reverse Contours on Clipboard**

**Reverse Contours on Clipboard** reverses the order of the vertices in a contour. To use it, cut one or more contours to the clipboard. Then select **PIEs|Convert|Reverse Contours on Clipboard**. Select the layer in which you wish the contours to be placed and click the OK button. The same warnings that apply to **Edit Contours** also apply to **Reverse Contours on Clipboard**.

## **Mesh Objects To Contours**

**Mesh Objects To Contours** allows you to select nodes or edges of elements and import them into Argus ONE as contours on information layers. To use it, select **PIEs|Convert|Mesh Objects To Contours**. Then select the mesh layer from which you wish to import data followed by an information, map, or domain outline layer into which you wish to import data. A dialog box will appear in which you can select nodes or sides of elements. If the **arrow** button  is depressed, you can select nodes or sides by clicking on them with the mouse. If the **lasso** button  is depressed, you can select nodes by encircling them with the mouse button depressed. In either case, if the shift button is depressed when you click down with the mouse, the selected nodes or sides will be added to removed from the list of selected nodes or sides rather than replacing them.

When you click on the **OK** button, the contours will be created. Contours can not branch so if the selected items allow a contour to go in several directions at a node, an arbitrary decision will be made as to which way the contour should go.

## **Mesh To Contours**

**Mesh to Contours** converts each element in a triangular or quadrilateral finite-element mesh to a closed contour on an information layer. To use it, select **PIEs|Convert|Mesh to Contours**. **These contours should not be used to assign properties to nodes where two or more contours intersect!** Attempting to do so may produce unpredictable results because the system must determine arbitrarily which of the contours to use in assigning properties to such nodes.

## **Hidden Commands**

Some of the commands described in this report have been "hidden". They do not normally appear in the Argus ONE menu structure. This was done because these commands circumvent protections built into Argus ONE to prevent users from inadvertently changing or deleting information that usually should not be changed or deleted.

It is possible to make the hidden commands described in this report visible. To do so, create a text file in the same directory as the Utility.dll PIE whose command you want to reveal. Name the file Utility.ini. Make the first line of the file "Show". The next time you start Argus ONE, the hidden commands in that PIE will be visible in the menu structure. To hide the functions again, delete the Utility.ini file or change its first line to something other than "Show." "Show" is case insensitive.

## **Set Parameter Locks**

**Set Parameter Locks** is hidden because if an option for a parameter is locked, it generally means that users should not make ill-considered changes to it.

**Set Parameter Locks** can be used to lock or unlock multiple parameters at one time. Once the command has been revealed as described above, it can be activated by selecting **PIEs|Set Parameter Locks**. A dialog box will appear with a hierarchical arrangement of layers and parameters. The user should select the parameters for which something should be locked or unlocked. The checkboxes at the bottom of the dialog box will be unchecked for all parameter locks which are not set for all the selected parameters. Similarly, the checkboxes will be checked for all parameter locks which are set for all the selected

parameters. The checkboxes will be gray if some of the selected parameters have the related feature locked and others do not. After selecting the parameters to change, the user should check or uncheck the check boxes to make the desired changes to the parameters. When the user clicks the OK button all the selected parameters will have their locks changed as the user specifies using the checkboxes. If any of the checkboxes is in a gray state, that lock will not be changed for any of the selected parameters.

### **Delete Multiple Layers**

**Delete Multiple Layers** is hidden because it allows layers that have been locked to prevent deletion to be deleted easily. Because deleting locked layers without proper consideration can destroy large amounts of data and make a model unusable, this command is somewhat dangerous.

Once the **Delete Multiple Layers** command has been revealed as described at the beginning of this section, it can be activated by selecting **PIEs|Delete Multiple Layers**. A dialog box will appear with a series of check boxes with layer names next to them. The user must select the layers to delete by checking the check boxes. If the user then clicks the OK button, the user will be warned that the layers will be deleted and given a chance to change the decision to delete. If the user still chooses to delete the layers, they will be deleted.

## **Functions**

Functions in Argus ONE can be accessed through the expression editor. The Argus ONE documentation describes how to show the expression editor. When the expression editor is visible, there is a list of function categories followed by a list of layers. Under the list of function categories, one of the categories is **PIEs**. All of the functions described in this document are found under the **PIEs** category.

### **Utility\_CheckVersion**

**Utility\_CheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit)** returns True if the version number of the Utility dynamic link library is greater than or equal to the version number passed in the arguments. To check the version number of the dynamic link library right-click on **Utility.dll**, select "Properties", and go to the version information tab.

### **EvalRealAtXY, EvalIntegerAtXY, EvalBooleanAtXY, and EvalStringAtXY**

**EvalRealAtXY**, **EvalIntegerAtXY**, **EvalBooleanAtXY**, and **EvalStringAtXY** allow an expression to be evaluated at a specified X, Y location. The four functions return a real number, integer, Boolean, and string respectively. The expression is passed to the function as a quoted string. It is up to the user to make sure that the expression is correctly formatted and that the value returned by the function is compatible with the type of the parameter to which it is assigned. If desired, the user may specify the layer on which the expression should be evaluated. The formats of the commands are as follows:

```
EvalRealAtXY(X, Y, "Expression_As_Quoted_String",  
            ["Layer_Name_As_Quoted_String"])
```

```
EvalIntegerAtXY(X, Y, "Expression_As_Quoted_String",  
    ["Layer_Name_As_Quoted_String"])  
EvalBooleanAtXY(X, Y, "Expression_As_Quoted_String",  
    ["Layer_Name_As_Quoted_String"])  
EvalStringAtXY(X, Y, "Expression_As_Quoted_String",  
    ["Layer_Name_As_Quoted_String"])
```

### ***Rotated X, and Rotated Y***

**Rotated X**, and **Rotated Y** are functions that can take row or column positions provided by Argus ONE along with the grid angle and convert them to (X, Y) coordinates. This could be done entirely by an export template, but using a PIE can make the export template simpler and easier to understand and perhaps faster too.

**Rotated X** has three arguments: X, Y, and GridAngle. X is a column position. Y is a row position. GridAngle is the angle of the grid. **Rotated X** returns the X-coordinate of the point defined by the intersection of the row and column positions. **Rotated Y** is just like **Rotated X** except that it returns the Y-Coordinate.

The first export template in Appendix 5 uses **Rotated X** and **Rotated Y** to export the row and column numbers or each cell in a grid followed by the X and Y coordinates of the four corners of the cell. The template can be used to reproducing the grid in other programs. Users may modify the template to meet their specific requirements.

The second template in Appendix 5 uses **Rotated X** and **Rotated Y** to export a contour that surrounds a single cell. The contour can be used as a domain outline in models that define a subgrid in a single cell.

### ***GetMyDirectory***

GetMyDirectory.dll is a PIE that returns the name of the directory in which it is installed. The name of the function will be the name of the dll itself without the extension. Typically, two programs are installed in the same directory as GetMyDirectory.dll: SelectChar.exe, and WaitForMe.exe.

SelectChar.exe is a console program that simulates a keyboard event based on its command line arguments. In the following [Char] is used to represent any character and [N] represents any number. If a command line argument contains "Alt-[Char]", SelectChar.exe will simulate depressing the Alt key and the [Char] key simultaneously. If a command line argument contains "Ctrl-[Char]", SelectChar.exe will simulate depressing the Ctrl key and the [Char] key simultaneously. If a command line argument contains "Chr-[N]", SelectChar.exe will simulate depressing the key whose key code is [N]. For any other command line argument SelectChar.exe will simulate depressing the first character in the command line argument. If SelectChar.exe is called with no command line arguments, it will simulate a carriage return.

WaitForMe.exe is a program that can run until it is closed. It is only possible to run one copy of WaitForMe.exe at a time. If you attempt to run a second copy, the first copy will be activated instead. It can be closed by activating it and typing a carriage return. The purpose of this is to start the program before doing something else and then activate it again when that "something else" is done. You can then close it by running SelectChar.exe. An external program, such as UCODE, can be waiting for WaitForMe.exe to finish executing before continuing.



These three files make it possible for an external program such as UCODE (Poeter and Hill, 1998) to run MODFLOW or SUTRA through Argus ONE and wait until MODFLOW has finished executing before continuing. In the MODFLOW GUI 3 and Sutra PIE 1.0.3 or later, if you define a Global variable named "Calibrate" and set its value to True (= 1). Argus ONE will activate and then close WaitForMe.exe in the directory containing GetMyDirectory.dll after running MODFLOW. It will also allow the DOS window in which MODFLOW or Sutra ran to close. For this to work, GetMyDirectory.dll, SelectChar.exe, and WaitForMe.exe must all be in the same directory and must be in the ArgusPIE directory or a subdirectory of the ArgusPIE directory.

### ***ReadFileValue functions***

The functions in the ReadFileValue PIE have the function names shown below except that they may have a prefix such as "MODFLOW\_" or "SUTRA\_".

The functions in the ReadFileValue PIE are primarily intended to be used with an external program such as UCODE (Poeter and Hill, 1998) that is running Argus ONE. RF\_Clear\_Files and RF\_Save\_Files are called in the export templates for MODFLOW and SUTRA.

#### **RF\_Get\_Value\_From\_File**

In RF\_Get\_Value\_From\_File(Key, Default\_Value, [FileName]), "FileName" is an optional parameter. If "FileName" is not specified, "FileName" will be given the value of "Default.txt". The function checks if it has already opened "FileName". If it hasn't, it reads "FileName" into memory. If "FileName" does not exist, it creates a new version of the "FileName" in memory. It then checks for "Key" in "FileName". If any full line in "FileName" is equal to "Key", the next line is converted to a double-precision real number and returned as the result of RF\_Get\_Value\_From\_File. If none of the lines in "FileName" is equal to "Key", "Key" is added to the file followed by "Default Value" and the result of RF\_Get\_Value\_From\_File is "Default Value".

"FileName" can be either a fully qualified path name or just a file name. If it is just a file name, ReadFileValue.dll will look for "FileName" in the current directory. The file that ReadFileValue.dll will check must consist of alternating lines of Keys and numbers. The first line must be a key. Blank lines and comments are not allowed.

#### **RF\_Clear\_Files**

RF\_Clear\_Files() clears all files that the ReadFileValue.dll currently has in memory.

#### **RF\_Save\_Files**

RF\_Save\_Files() saves all files that the ReadFileValue.dll currently has in memory to disk.

#### **RF\_CheckVersion**

RF\_CheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit) returns True if the version number of the PIE is greater than or equal to the version number passed in the arguments. The version number of the PIE may be checked by right clicking on the dll, selecting "Properties" and going to the version information tab.

## **Conversion functions**

Kipp (1987) provides conversion formulas for converting from metric units to US customary units. These formulas were programmed into functions listed in Table 3. No explanation of these functions is needed beyond that in the table so they are not listed individually here. These conversions supplement those in the Unit Conversions PIE on the Argus Interware web site (<http://www.argusint.com/>).

## **Hidden Functions**

Argus ONE gives PIE developers the ability to "hide" functions. If a function is hidden, it will still work properly but it will not show up in the list of PIE functions. This helps keep inexperienced users from using functions improperly that have a high potential for misuse. The functions in this section can only be used in export templates. For most of them, this is because they either allocate memory that must be released using a different function or use memory allocated by a call to a previous function. Others display dialog boxes that would be inappropriate for use with expressions for layer parameters.

To make the hidden functions visible create a text file in the same directory as the PIE whose functions you want to reveal. Name the file the same as the dll except change the extension from dll to ini. Make the first line of the file "Show". The next time you start Argus ONE, the functions in that PIE will be visible in the expression editor. To hide the functions again, delete the "ini" text file or change its first line to something other than "Show." "Show" is case insensitive.

## **OKCancel Functions**

The OkCancel PIE contains functions that can be used to ask the user questions and get responses in an export template. The OkCancel PIE is used in the export template for SUTRA if the bandwidth of the mesh appears high. In such cases, the user is asked if they want to continue exporting or stop. The second template in Appendix 5 also uses the OkCancel PIE.

### **IsOK**

IsOK(Message, [HideCancel]) displays "Message" in a dialog box with "Yes", "No", and "Cancel" buttons. If the optional "HideCancel" parameter is set to true, the "Cancel" button, is not shown. It returns True if the user clicks on "Yes". Otherwise it returns False. The following is an example export template using the IsOK function

```
#
# Ask the user a question and do different things depending on the result.
If: IsOK("Do you want to click the Yes button?")
    Alert: "You clicked Yes."
Else
    Alert: "You clicked No or Cancel."
End if
```

### **Ok\_Add\_Radio\_Choice**

Ok\_Add\_Radio\_Choice, ok\_Get\_Radio\_Choice, and ok\_Radio\_Free are functions related to displaying a dialog box with a series of radio buttons and getting a response from

the user depending on the radio button selected. `Ok_Add_Radio_Choice(Message, [Message] , [Message] , [Message] , [Message])` adds up to five radio buttons to the list of radio buttons that are displayed. The text of each radio button will be "Message".

`Ok_Add_Radio_Choice` creates, but does not display, the dialog box if the dialog box does not already exist. If the dialog box already exists, `ok_Add_Radio_Choice` adds Message to the list of choices displayed in the dialog box.

`Ok_Add_Radio_Choice` returns True if it is successful. It should succeed unless the computer's memory has been exhausted.

### **Ok\_Get\_Radio\_Choice**

`Ok_Get_Radio_Choice(Message, [Choices_Height], [Width], [Question_Height])` displays a dialog box with a question set in the first parameter. There are three optional parameters. The first sets the height in pixels of the box containing the choices. The second sets the width of the dialog box in pixels, and the third sets the height in pixels of the question to be displayed. The choices that the user may select as responses must have previously been set with a call to `ok_Add_Radio_Choice`. The dialog box is destroyed after the response is obtained in `ok_Get_Radio_Choice`. The function returns the index of the radio button that the user selected as his or her response. The radio buttons are numbered consecutively starting at zero.

### **Ok\_Radio\_Free**

`Ok_Radio_Free()` destroys the dialog box without displaying it. This function is only needed if `ok_Add_Radio_Choice` is used without a subsequent `ok_Get_Radio_Choice`.

The following is an export template using `IsOK`, `ok_Add_Radio_Choice`, `ok_Get_Radio_Choice`, and `ok_Radio_Free`.

```
#
Evaluate expression: ok_Add_Radio_Choice("Choice 1", "Choice 2")
Evaluate expression: ok_Add_Radio_Choice("Choice 3")
If: IsOK("Do you want to see Choices 1, 2, and 3?", 1)
    Alert: "You chose choice number "+(1+ok_Get_Radio_Choice("What is your choice?",80,300,45))
Else
    Evaluate expression: ok_Radio_Free()
End if
```

### **Ok\_UserFloat and Ok\_UserInteger**

`Ok_UserFloat(Message, Response, [Minimum], [Maximum])` and `Ok_UserInteger(Message, Response, [Minimum], [Maximum])` displays a dialog box with a question set by Message and an edit box containing Response. If Minimum and/or Maximum are set, they are the minimum and maximum allowable value of the response. The function returns the value shown in the edit box as a real number or integer when the user closes the dialog box depending on which function was called.

### **Ok\_CheckVersion**

`Ok_CheckVersion(First_Digit, Second_Digit, Third_Digit, Fourth_Digit)` returns True if the version number of the dynamic link library is greater than or equal to the version number passed in the arguments. The version number of the OkCancel dynamic link library

may be checked by right clicking on OkCancel.dll, selecting "Properties" and going to the version information tab.

## ***ProgressBar Functions***

The functions in the ProgressBar PIE have the function names shown below except that they may have a prefix such as "MODFLOW\_" or "SUTRA\_".

The ProgressBar PIE allows you to display a progress bar during the export of a model that advances as the export progresses. It also has a label and a memo box with which messages to the user can be displayed without halting the export process. The contents of the memo can be saved to a text file. The progress bar shows the elapsed time and gives an estimate of the time remaining to complete the export process.

The ProgressBar PIE is used extensively in the export template for MODFLOW-96 (Winston, 2000).

Appendix 2 has an example export template that uses the ProgressBar PIE.

### **ProgressBarInitialize**

ProgressBarInitialize(Number, [Show\_Cancel]) creates and shows the progress bar and sets the maximum position of the progress bar to Number. Initially the progress bar position is at 0. If Show\_Cancel is true, an "Abort" button will be visible. This function returns True if it succeeds. Any of the remaining ProgressBar functions will fail if ProgressBarInitialize is not called first.

### **ProgressBarFree**

ProgressBarFree() frees the progress bar and frees all memory associated with the progress bar. This function returns True if it succeeds. If ProgressBarFree(), is not called at the end of the export process, the progress bar will remain visible after the export process is complete.

### **ProgressBarMax**

ProgressBarMax(Number) sets the maximum value of the progress bar. This is also set in ProgressBarInitialize(Number). This function returns True if the Abort button has not been pressed and the function succeeds.

### **ProgressBarAdvance**

ProgressBarAdvance() increases the position of the progress bar by one. This function returns True if the Abort button has not been pressed and the function succeeds.

### **ProgressBarSetMessage**

ProgressBarSetMessage(Message) sets the message in the label displayed in the progress bar to Message. This function returns True if the Abort button has not been pressed and the function succeeds.

### **ProgressBarAddLine**

ProgressBarAddLine(Message) adds Message to the list of messages in the memo. This function returns True if the Abort button has not been pressed and the function succeeds.

## **ProgressBarSaveToFile**

ProgressBarSaveToFile(File\_Name) saves the messages in the memo to a text file named File\_Name and returns the number of lines in the file it saves.

## **ProgressBarCheckVersion**

ProgressBarCheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit) returns True if the version number of the dynamic link library is greater than or equal to the version number passed in the arguments. To check the version number of the ProgressBar dynamic link library right-click on Progressbar.dll, select "Properties", and go to the version information tab.

## **JoinFiles Functions**

The JoinFiles PIE contains functions for performing operations on files.

The functions in the JoinFiles PIE have the function names shown below except that they may have a prefix such as "MODFLOW\_" or "SUTRA\_". The JoinFiles PIE is used extensively in the export template for MODFLOW-96 because it allows the template to be more efficient (Winston, 2000). Certain items at the beginnings of the input files are counts of items that occur later in the file. The JoinFiles PIE makes it possible to count the items as they are exported, write the count to a separate file, and then join the files to create a properly formatted, MODFLOW, input file.

### **Join\_Files**

Join\_Files(First\_File, Second\_File, Result\_File) appends Second\_File to First\_File and stores the result in Result\_File.

### **Delete\_File**

Delete\_File(File\_Name) deletes the file indicated by File\_Name.

### **Rename\_File**

Rename\_File(Old\_File\_Name, New\_File\_Name) changes the name of a file from Old\_File\_Name to New\_File\_Name.

### **Split\_File**

Split\_File('Input\_File, First\_File, Search\_String, Second\_File [, Search\_String, Third\_File] [, Search\_String, Fourth\_File]...[, Search\_String, Thirtieth\_File]) Splits a file into up to 30 sections. The input file is read one line at a time and saved to **First\_File**. If **Search\_String** is found in a line, **First\_File** is closed and subsequent lines are saved in **Second\_File** and the last line of **First\_File** will be **Search\_String**. If there are additional parameters (**Search\_String**, **Third\_File**, etc.) the remainder of **Input\_File** will continue to be searched by occurrences of the new **Search\_String** and split into separate files as before.

### **Int2Str**

Int2Str(Number) converts a number to its character representation in Base 36. (0, 1, ... 9, A, B, ... Z, 10, 11, ... 19, 1A, ....). This can be useful if a file extension must be less than 3 characters long and the file extension must have a number. For example, MODFLOW

(Harbaugh, Banta, Hill, and McDonald, 2000) can be compiled in such a way that only file names less than or equal to eight characters in length with extensions less than or equal to three characters in length will be recognized. If the IBOUND array for each of 100 layers is stored in separate files with names of the form MyModel.i1 to MyModel.i100, there will be a problem. The last file, MyModel.i100 won't be recognized because the extension has more than 3 characters. In an export template that used Int2Str, the file name could be MyModel.i2s which would be acceptable.

### **JF\_CopyLines**

JF\_CopyLines(Old\_File\_Name, New\_File\_Name, Line\_Count, [Is\_Local\_File]) creates a new text file the current directory named New\_File\_Name and opens an existing file named Old\_File\_Name. It then reads Line\_Count lines from Old\_File\_Name and writes them to New\_File\_Name. If it can do all this, it returns True. Otherwise it returns False. If the optional Is\_Local\_File parameter is present and is set to False, Old\_File\_Name represents the full path of the existing file. Otherwise it represents the name of a file in the current directory.

### **JF\_CheckVersion**

JF\_CheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit) returns True if the version number of the PIE is greater than or equal to the version number passed in the arguments. To check the version number of the PIE right-click on JoinFiles.dll, select "Properties", and go to the version information tab.

### **List functions**

The functions in the List PIE have the function names shown below except that they may have a prefix such as "MODFLOW\_" or "SUTRA\_". The List PIE maintains variable-size, zero-based lists (arrays) of real numbers. The number of dimensions in the lists can be either one or three.

It is possible to call the functions in the List PIE with incorrect arguments. In such cases, the List PIE will increment an internal variable that contains a count of the number of errors that have occurred and then return a result. Where possible, the result will indicate that an error occurred but it is not always possible for the function to do this. Users can call the L\_GetErrorCount function.get determine the number of errors that have occurred and then take appropriate action.

The List PIE is used extensively in the export template for MODFLOW 96 (Winston, 2000). It isn't needed for MODFLOW-2000.

An example export template using the List PIE is in Appendix 3.

### **L\_Initialize**

L\_Initialize() (obsolete) was formerly used to initialize the List PIE. It is no longer needed but is still present for backwards compatibility.

### **L\_CreateNewList**

L\_CreateNewList() create a new list and returns the "ListIndex" of the new list. ListIndex is used to refer to a particular list when it is used. If the function is unsuccessful, it increments the error count and returns -1. L\_CreateNewList should only be unsuccessful if

the computer's memory has been exhausted. The values of ListIndex returned by L\_CreateNewList are consecutive and start at 0.

### **L\_SetListSize**

L\_SetListSize(ListIndex, Size) adds additional items with values of 0 to the list indicated by ListIndex until the number of items in the list is equal to Size. You can still add additional items to a list after using SetListSize. The function returns True if it is successful, and False if unsuccessful. If it is unsuccessful, it will increment the error count. If ListIndex indicates a list that does not exist, SetListSize will be unsuccessful. If the list is already larger than Size, L\_SetListSize does nothing.

### **L\_GetListSize**

L\_GetListSize(ListIndex) returns the number of items in the list indicated by ListIndex. The function returns -1 if it is unsuccessful. If ListIndex indicates a list that does not exist, L\_GetListSize will be unsuccessful and will increment the error count.

### **L\_FreeAList**

L\_FreeAList(ListIndex) deletes all items in the list indicated by ListIndex. It does not delete the list itself so the ListIndices of lists created after the list that has been emptied do not change. The function returns True if successful, and False if unsuccessful. If ListIndex indicates a list that does not exist, L\_FreeAList will be unsuccessful and will increment the error count.

### **L\_AddToList**

L\_AddToList(ListIndex, Value) adds an item to the end of the list indicated by ListIndex. The numeric value of the item is Value. Value is stored as a double-precision real number. The function returns the "Index" of the item if successful, and -1 if unsuccessful. "Index" is the position of the item in the list. If ListIndex indicates a list that does not exist or if memory has been exhausted, L\_AddToList will be unsuccessful and will increment the error count. The values of "Index" are consecutive and start at zero.

### **L\_GetFromList**

L\_GetFromList(ListIndex, Index) returns the item at the position indicated by Index from the list indicated by ListIndex. The function returns 0 if unsuccessful. If ListIndex and Index indicate a list or an item that does not exist, L\_GetFromList will be unsuccessful and will increment the error count.

### **L\_SetListItem**

L\_SetListItem(ListIndex, Index, Value) sets the value of the item at the position indicated by Index in the list indicated by ListIndex to Value. The function returns True is successful and False if unsuccessful. If ListIndex and Index indicate a list or an item that does not exist, L\_SetListItem will be unsuccessful and will increment the error count. For example, if L\_GetListSize(ListIndex) returns 5, L\_SetListItem(ListIndex, 5, Value) would be unsuccessful but L\_SetListItem(ListIndex, 4, Value) would succeed.

## **L\_DeleteListItem**

L\_DeleteListItem(ListIndex, [Index]) deletes the item at the position indicated by Index from the list indicated by ListIndex. This reduces the number of items in the list by one. The function returns True if successful and False if unsuccessful. If ListIndex and Index indicate a list or an item that does not exist, L\_DeleteListItem will be unsuccessful and will increment the error count. "Index" is optional. If Index is not included, the last item in the list indicated by ListIndex will be deleted.

## **L\_SortList**

L\_SortList(ListIndex) sorts the items in the list indicated by ListIndex from lowest at the beginning of the List to highest at the end of the list. The function returns True if it succeeds and False if it fails. If the list indicated by ListIndex does not exist, L\_SortList will fail and will increment the error count.

## **L\_EliminateDuplicates**

If two adjacent items in a list are identical, L\_EliminateDuplicates(ListIndex) will delete one of them from the list. Usually, you should call L\_SortList or otherwise ensure that the list is sorted before calling L\_EliminateDuplicates to eliminate all duplicate entries in the list. The function returns True if it succeeds and False if it fails. If the list indicated by ListIndex does not exist, L\_EliminateDuplicates will fail and will increment the error count.

## **L\_IndexOf**

L\_IndexOf(ListIndex, Value) returns the position within a sorted list indicated by ListIndex of the first occurrence of Value. Normally, you should call L\_SortList or otherwise ensure that the list is sorted before calling L\_IndexOf.

If Value is not in the list, it will return the index of the first occurrence of the largest item smaller than Value.

For example if the list contains 0, 1, 1, 3: L\_IndexOf(ListIndex, 0) returns 0, L\_IndexOf(ListIndex, 1) returns 1, L\_IndexOf(ListIndex, 2) returns 1, and L\_IndexOf(ListIndex, 3) returns 3.

If the list indicated by ListIndex does not exist, L\_IndexOf will fail. It will then return -1 and will increment the error count.

## **L\_UnsortedIndexOf**

L\_UnsortedIndexOf(ListIndex, Value) returns the position of Value in the list indicated by ListIndex. If it is not found, the function returns -1. If the list indicated by ListIndex does not exist, L\_UnsortedIndexOf will fail. It will then return -1 and will increment the error count.

## **L\_CreateNew3DList**

L\_CreateNew3DList(Maximum\_X, Maximum\_Y, Maximum\_Z) creates a new 3D List with dimensions [0..Maximum\_X-1, 0..Maximum\_Y-1, 0..Maximum\_Z-1]. If it succeeds, L\_CreateNew3DList returns the ListIndex of the new 3D List. The function returns -1 if unsuccessful. L\_CreateNew3DList will be unsuccessful if memory has been exhausted. If the function is unsuccessful, it will increment the error count by 1. The values of ListIndex returned by L\_CreateNew3DList are consecutive and start at 0. The numbering



of 3D lists is independent of the numbering of lists created with `L_CreateNewList`. Thus, the initial calls to `L_CreateNew3DList` and `L_CreateNewList` would both return 0. Initially all items in a 3D list have a value of 0.

### **L\_FreeA3DList**

`L_FreeA3DList(ListIndex)` deletes all items from the 3D List indicated by `ListIndex` and sets `Maximum_X`, `Maximum_Y` and `Maximum_Z` to 0. The function returns `True` if successful and `False` if unsuccessful. If `ListIndex` indicates a 3D list that does not exist, `L_FreeA3DList` will be unsuccessful. If the function is unsuccessful, it will increment the error count.

### **L\_GetFrom3Dlist and L\_GetFromOneBased3DList**

`L_GetFrom3DList(ListIndex, X_Index, Y_Index, Z_Index)` returns the value of the item within the 3D list indicated by `ListIndex` at the position indicated by `X_Index`, `Y_Index`, and `Z_Index`. The function returns 0 if it fails. If `ListIndex`, `X_Index`, `Y_Index`, `Z_Index` indicate a 3D list or an item in a 3D list that does not exist, `L_GetFrom3DList` will be unsuccessful and will increment the error count.

`L_GetFromOneBased3DList` is identical to `L_GetFrom3DList` except that the 3D list is treated as having the following limits [`1..Maximum_X`, `1..Maximum_Y`, `1..Maximum_Z`] instead of [`0..Maximum_X-1`, `0..Maximum_Y-1`, `0..Maximum_Z-1`].

### **L\_Set3DlistItem and L\_SetOneBased3DlistItem**

`L_Set3DlistItem(ListIndex, X_Index, Y_Index, Z_Index, Value)` sets the value of the item within the 3D list indicated by `ListIndex` at the position indicated by `X_Index`, `Y_Index`, and `Z_Index` to `Value`. The function returns `True` if it succeeds and `False` if it fails. It will fail if the 3D list indicated by `ListIndex` does not exist or if the position indicated by `X_Index`, `Y_Index`, and `Z_Index` is outside the extents of the 3D List. If it fails, it will increment the error count.

`L_SetOneBased3DlistItem` is identical to `L_Set3DlistItem` except that the 3D list is treated as having the following limits [`1..Maximum_X`, `1..Maximum_Y`, `1..Maximum_Z`] instead of [`0..Maximum_X-1`, `0..Maximum_Y-1`, `0..Maximum_Z-1`].

### **L\_ResetA3DList**

`L_ResetA3DList(ListIndex)` sets all items in the 3D list indicated by `ListIndex` to 0. The function returns `True` if it succeeds and `False` if it fails. It will fail if the 3D list indicated by `ListIndex` does not exist. If it fails, it will increment the error count.

### **L\_Add3DLists**

`L_Add3DLists(FirstListIndex, SecondListIndex, ResultListIndex)` adds each member of the 3D list indicated by `FirstListIndex` to the corresponding member of the 3D list indicated by `SecondListIndex` and places the result in the corresponding member of the 3D list indicated by `ResultListIndex`. The function will fail and return `False` if `FirstListIndex`, `SecondListIndex`, or `ResultListIndex` indicate a 3D list that does not exist. It will also fail if all of the 3D lists do not have the same limits. If it fails, it will increment the error count.

### **L\_Subtract3DLists**

L\_Subtract3DLists(FirstListIndex, SecondListIndex, ResultListIndex) subtracts each member of the 3D list indicated by SecondListIndex to the corresponding member of the 3D list indicated by FirstListIndex and places the result in the corresponding member of the 3D list indicated by ResultListIndex. The function will fail and return False if FirstListIndex, SecondListIndex, or ResultListIndex indicate a 3D list that does not exist. It will also fail if all of the 3D lists do not have the same limits. If it fails, it will increment the error count.

### **L\_Multiply3DLists**

L\_Multiply3DLists(FirstListIndex, SecondListIndex, ResultListIndex) multiplies each member of the 3D list indicated by FirstListIndex by the corresponding member of the 3D list indicated by SecondListIndex and places the result in the corresponding member of the 3D list indicated by ResultListIndex. The function will fail and return False if FirstListIndex, SecondListIndex, or ResultListIndex indicate a 3D list that does not exist. It will also fail if all of the 3D lists do not have the same limits. If it fails, it will increment the error count.

### **L\_Divide3DLists**

L\_Divide3DLists(FirstListIndex, SecondListIndex, ResultListIndex) divides each member of the 3D list indicated by FirstListIndex by the corresponding member of the 3D list indicated by SecondListIndex and places the result in the corresponding member of the 3D list indicated by ResultListIndex. The function will fail and return False if FirstListIndex, SecondListIndex, or ResultListIndex indicate a 3D list that does not exist. It will also fail if all of the 3D lists do not have the same limits. If it fails, it will increment the error count.

### **L\_Multiply3DByConstant**

L\_Multiply3DByConstant(ListIndex, ResultListIndex, Value) multiplies each member of the 3D list indicated by ListIndex by Value and places the result in the corresponding member of the 3D list indicated by ResultListIndex. The function will fail and return False if ListIndex or ResultListIndex indicate a 3D list that does not exist. It will also fail if the 3D lists indicated by ListIndex and ResultListIndex do not have the same limits. If it fails, it will increment the error count.

### **L\_Invert3DListMembers**

L\_Invert3DListMembers(ListIndex, ResultListIndex) takes the reciprocal of each member of the 3D list indicated by ListIndex and places the result in the corresponding member of the 3D list indicated by ResultListIndex. The function will fail and return False if ListIndex or ResultListIndex indicate a 3D list that does not exist. It will also fail if the 3D lists indicated by ListIndex and ResultListIndex do not have the same limits. If it fails, it will increment the error count.

### **L\_IsSingPrecUniform**

L\_IsSingPrecUniform(ListIndex) returns True if all members of the list indicated by ListIndex have the same value after being converted to single precision. The function will fail and return False if ListIndex indicates a 3D list that does not exist. If it fails, it will increment the error count.

This function is used in the export template for MOC3D to check that the cells in the export subgrid are of uniform size.

### **L\_GetErrorCount**

L\_GetErrorCount() returns the number of errors that have occurred in the List PIE since the PIE was loaded or the last call to L\_FreeAllLists. Normally an error count greater than 0 indicates that the export process did not go as planned. This function is used in the export template for MODFLOW to check for programming errors in the export template.

### **L\_FreeAllLists**

L\_FreeAllLists() Deletes all items in all lists and all 3D Lists, deletes the lists and sets the error count to 0. The function returns True if it is successful and False if unsuccessful. L\_FreeAllLists should never be unsuccessful. L\_FreeAllLists() should be called at the end of any export template that uses the List PIE because it will release memory allocated by the List PIE. If it is not called, the memory will be released when Argus ONE is shut down.

### **L\_CheckVersion**

L\_CheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit) returns True if the version number of the dynamic link library is greater than or equal to the version number passed in the arguments. The version number of the dynamic link library may be checked by right clicking on List.dll, selecting "Properties" and going to the version information tab.

## ***BlockList functions***

The functions in the BlockList PIE have the function names shown below except that they may have a prefix such as "MODFLOW\_" or "SUTRA\_". The BlockList PIE contains GIS functions deal with the relationships between contours and grid cells. The functions indicate (1) which cells are intersected by a contour, (2) which cells are inside a contour, (3) the order in which cells are intersected by a contour (4) the length of the section of the contour intersected by a cell, (5) the cells next to the ones intersected by the contour but on the other side of the contour, and (6) many other data relating the contours and cells. The BlockList PIE is, by far, the most complicated PIE discussed in this paper. It can provide information that would be difficult to calculate with just an export template. In the MODFLOW GUI (Winston, 2000) the BlockList PIE is used extensively in preparing the input for the Horizontal Flow Barrier (Hsieh, and Freckleton, 1993) and Stream (Prudic, 1989) packages for MODFLOW-96. For MODFLOW-2000, the blocklist functions are called directly by the MODFLOW GUI rather than through an export template and the functions are used for all boundary conditions rather than just the Horizontal-Flow Barrier and Stream packages.

In these functions, ListIndex is used to designate a list of cells intersected by a contour. CellIndex indicates a particular cell in the list. VertexIndex indicates a vertex in a cell. SegmentIndex indicates a segment of a contour within a cell. In any of the functions, invalid values of Listindex, CellIndex, VertexIndex, or SegmentIndex are used, an error will occur.

It is possible to call the functions in the BlockList PIE with incorrect arguments. In such cases, the BlockList PIE will increment an internal variable that contains a count of the

number of errors that have occurred and then return a result. Where possible, the result will indicate that an error occurred but it is not always possible for the function to do this. Users can call the `BL_GetErrorCount` function get determine the number of errors that have occurred and take appropriate action.

Appendix 4 contains an example export template that uses the BlockList PIE.

### **BL\_InitializeGridInformation**

`BL_InitializeGridInformation(Grid_Layer_Name_as_String, [GridType])` reads the positions of row and columns from the grid layer named `Grid_Layer_Name_as_String`. Because `Grid_Layer_Name_as_String` is a string, it must either be surrounded by quotes or be turned into a string by some other method. `GridType` is an optional parameter that determines how the grid will be treated in subsequent processing. `GridType = 0` indicates a block-centered grid. `GridType = 1` indicates a node-centered grid. The default is a block-centered grid. `GridType` need not match the actual type of the grid. For example a block-centered grid could be used with a `GridType` of 1 in order to evaluate all information as if it were a node-centered grid in which all the nodal points were at the intersections of the row and column boundaries in the block-centered grid.

### **BL\_AddVertexLayer**

`BL_AddVertexLayer(Information_Layer_Name_as_String)` reads the contour information from the information layer named `Information_Layer_Name_as_String` and determines which cells intersect the contours on the information layer. Because `Information_Layer_Name_as_String` is a string, it must either be surrounded by quotes or be turned into a string by some other method. `BL_AddVertexLayer` does not erase the information stored in previous calls to `BL_AddVertexLayer` but instead adds the information in `Information_Layer_Name_as_String` to the information that was read previously. To erase the contour information, call `BL_ReInitializeVertexList()`.

### **BL\_ReInitializeVertexList**

`BL_ReInitializeVertexList()` deletes all contour information that has already been entered.

### **BL\_GetCountOfCellLists**

`BL_GetCountOfCellLists()` returns the number of lists of cells. This is normally the sum of all the objects on all the layers that have been added using `BL_AddVertexLayer` since the last call to `BL_ReInitializeVertexList`. Each list of cells contains the cells intersected by the contour in the order in which they were intersected along the length of the contour.

### **BL\_GetCountOfACellList**

`BL_GetCountOfACellList(ListIndex)` returns the number of cells in a list of cells intersected by a contour. The same cell may be intersected by a contour more than once so the cells in this list may not all be unique.

### **BL\_GetCellRow and BL\_GetCellColumn**

`BL_GetCellRow(ListIndex, CellIndex)` and `BL_GetCellColumn(ListIndex, CellIndex)` return a row or column number in a list of cells. The first cell is the one at the

beginning of the contour and the cells are listed in order from the beginning to end of the contour.

### **BL\_GetVertexCount**

BL\_GetVertexCount(ListIndex, CellIndex) returns the number of vertices associated with a cell. Vertices will be present wherever a contour has a vertex or a contour intersects the cell boundary.

### **BL\_GetVertexXPos and BL\_GetVertexYPos**

BL\_GetVertexXPos(ListIndex, CellIndex, VertexIndex) and BL\_GetVertexYPos(ListIndex, CellIndex, VertexIndex) return the X and Y coordinates of a vertex.

### **BL\_SegmentCount**

BL\_SegmentCount(ListIndex, CellIndex) returns the number of line segments within a cell. A segment is the section of a contour within a cell that connects two adjacent vertices.

### **BL\_SegmentFirstX and BL\_SegmentFirstY**

BL\_SegmentFirstX(ListIndex, CellIndex, SegmentIndex) and BL\_SegmentFirstY(ListIndex, CellIndex, SegmentIndex) returns the X and Y coordinates of the first vertex defining a segment.

### **BL\_SegmentSecondX and BL\_SegmentSecondY**

BL\_SegmentSecondX(ListIndex, CellIndex, SegmentIndex) and BL\_SegmentSecondY(ListIndex, CellIndex, SegmentIndex) return the X and Y coordinates of the second vertex defining a segment.

### **BL\_SegmentLengthX and BL\_SegmentLengthY**

BL\_SegmentLengthX(ListIndex, CellIndex, SegmentIndex) and BL\_SegmentLengthY(ListIndex, CellIndex, SegmentIndex) return the X or Y coordinate of the second vertex defining a segment minus the X or Y coordinate of the first vertex defining a segment.

### **BL\_SegmentLength**

BL\_SegmentLength(ListIndex, CellIndex, SegmentIndex) returns the length of a segment.

### **BL\_SumSegmentsX and BL\_SumSegmentsY**

BL\_SumSegmentsX(ListIndex, CellIndex) and BL\_SumSegmentsY(ListIndex, CellIndex) return the sum of all the differences between the X or Y coordinate of the second vertex defining each segment minus the X or Y coordinate of the first vertex defining each segment.

### **BL\_SumSegmentLengths**

BL\_SumSegmentLengths(ListIndex, CellIndex) returns the sum of the lengths of all the segments in a cell.

### **BL\_GetCountOfCombinedCellList**

BL\_GetCountOfCombinedCellList() returns the number of all the cells intersected by any contour. Each cell in this list has a unique row and column number. The cells are listed in no particular order but no two cells are identical.

### **BL\_GetCellRowFromCombinedList and BL\_GetCellColumnFromCombinedList**

BL\_GetCellRowFromCombinedList(CellIndex) and BL\_GetCellColumnFromCombinedList(CellIndex) return a row or column in the list of all cells intersected by contours.

### **BL\_GetCountOfCrossRowLists and BL\_GetCountOfCrossColumnLists**

BL\_GetCountOfCrossRowLists() returns the number of lists of cells in which a contour crosses the Y-coordinate of the cell node. For block-centered grids, the nodal position is at the center of the grid but in grid-centered grids, the nodal position may be off-center. For the purposes of this function, a contour is considered not to have crossed the center if its first and last vertices in segments within the cell lie on the same side of the Y-coordinate of the cell node. BL\_GetCountOfCrossColumnLists() similar to BL\_GetCountOfCrossRowLists() except that it deals with cells in which a contour crosses the X-coordinate of a cell node.

### **BL\_GetCountOfACrossRowList and BL\_GetCountOfACrossColumnList**

BL\_GetCountOfACrossRowList(ListIndex) returns the number of cells in which a contour crosses the Y-coordinate of the cell node. BL\_GetCountOfACrossColumnList(ListIndex) is similar to BL\_GetCountOfACrossRowList except that it deals with cells in which a contour crosses the X-coordinate of a cell node.

### **BL\_GetCrossRowRow and BL\_GetCrossColumnRow**

BL\_GetCrossRowRow(ListIndex, CellIndex) returns the row number of a cell in which the contour crosses the Y-coordinate of the cell node. BL\_GetCrossColumnRow(ListIndex, CellIndex) is similar to BL\_GetCrossRowRow except that it deals with cells in which a contour crosses the X-coordinate of a cell node. CellIndex must be greater than or equal to zero and less than or equal to the result of BL\_GetCountOfACrossRowList or BL\_GetCountOfACrossColumnList

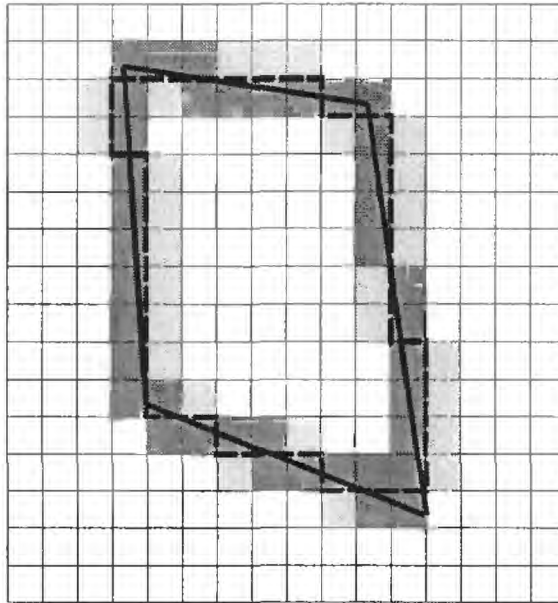
### **BL\_GetCrossRowColumn and BL\_GetCrossColumnColumn**

BL\_GetCrossRowColumn(ListIndex, CellIndex) returns the column number of a cell in which the contour crosses the Y-coordinate of the cell node. BL\_GetCrossColumnColumn(ListIndex, CellIndex) is similar to BL\_GetCrossRowColumn except that it deals with cells in which a contour crosses the X-coordinate of a cell node. CellIndex must be greater than or equal to zero and less than or equal to the result of BL\_GetCountOfACrossRowList or BL\_GetCountOfACrossColumnList

### **BL\_GetCrossRowNeighborColumn and BL\_GetCrossColumnNeighborRow**

BL\_GetCrossRowNeighborColumn(ListIndex, CellIndex) returns the column number of the neighboring cell to the one in which the contour crosses the Y-coordinate of the cell node (Figure 4). BL\_GetCrossColumnNeighborRow(ListIndex, CellIndex) is similar to

**BL\_GetCrossRowNeighborColumn** except that it deals with cells in which a contour crosses the X-coordinate of a cell node and it returns a row number rather than a column number. **CellIndex** must be greater than or equal to zero and less than or equal to the result of **BL\_GetCountOfACrossRowList** or **BL\_GetCountOfACrossColumnList**

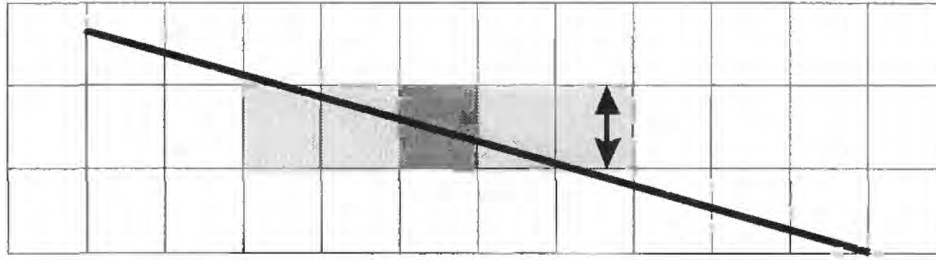


**Figure 4.** The cells intercepted by the contour (heavy black line) are shown in dark gray. Those cells which are their neighbors, as determined by **BL\_GetCrossRowNeighborColumn** and **BL\_GetCrossColumnNeighborRow**, are shown in light gray except for those which are also intercepted by the contour. The heavy dashed line separates each cell intercepted by the contour from its neighbor or neighbors.

### **BL\_GetCrossRowCompositeY and BL\_GetCrossColumnCompositeX**

**BL\_GetCrossRowCompositeY(ListIndex, CellIndex)** returns a total difference in Y-coordinate of segments in one or more cells. The cells over which the sum is returned must be in the same row as the cell that has a contour that crosses the Y-coordinate of the cell node. If there are two such cells in the same row with no other such cells between them on different rows, a maximum or minimum point of the contour between the two cells will be the limit of segments that will be summed for the purpose of this function (Figure 5).

**BL\_GetCrossColumnCompositeX(ListIndex, CellIndex)** is similar to **BL\_GetCrossRowCompositeY** except that it deals with cells in which a contour crosses the X-coordinate of a cell node and it returns a difference in the X-coordinate. **CellIndex** must be greater than or equal to zero and less than or equal to the result of **BL\_GetCountOfACrossRowList** or **BL\_GetCountOfACrossColumnList**



**Figure 5.** For the cell indicated by the dark gray square, the length returned by **BL\_GetCrossRowCompositeY** is indicated by the double-headed arrow. To determine this, the function retrieved data not only from the dark gray cell but also from the light gray cells.

### **BL\_GetSumCrossRowCompositeY and BL\_GetSumCrossColumnCompositeX**

**BL\_GetSumCrossRowCompositeY(ListIndex)** returns the sum of the **BL\_GetCrossRowCompositeY**'s for all the cells in the list that cross the Y-coordinate of the cell node. **BL\_GetSumCrossColumnCompositeX(ListIndex)** is similar to **BL\_GetSumCrossRowCompositeY** except that it deals with cells in which a contour crosses the X-coordinate of a cell node. **CellIndex** must be greater than or equal to zero and less than or equal to the result of **BL\_GetCountOfACrossRowList** or **BL\_GetCountOfACrossColumnList**

### **BL\_GetCrossRowCompositeLength and BL\_GetCrossColumnCompositeLength**

**BL\_GetCrossRowCompositeLength(ListIndex, CellIndex)** is similar to **BL\_GetCrossRowCompositeY** except that it returns the total length rather than the total difference in Y-coordinates. **BL\_GetCrossColumnCompositeLength(ListIndex, CellIndex)** is similar to **BL\_GetCrossRowCompositeLength** except that it deals with cells in which a contour crosses the X-coordinate of a cell node. **CellIndex** must be greater than or equal to zero and less than or equal to the result of **BL\_GetCountOfACrossRowList** or **BL\_GetCountOfACrossColumnList**

### **BL\_GetRowBoundary and BL\_GetColumnBoundary**

**BL\_GetRowBoundary(Row)** and **BL\_GetColumnBoundary(Column)** return the position of the Row or Column boundary indicated by Row or Column. These are equivalent to the **Row()** and **Column()** functions of Argus ONE.

### **BL\_PointInsideContour**

**BL\_PointInsideContour(ListIndex,X,Y)** returns True if (X,Y) is inside the contour indicated by ListIndex but is not inside any contours inside the one indicated by ListIndex. (This function may not work if "Allow Intersection" is on and contours cross each other.)

### **BL\_GetRowNodePosition and BL\_GetColumnNodePosition**

**BL\_GetRowNodePosition(Row)** and **BL\_GetColumnNodePosition(Column)** return the Y position of the node of the Row indicated by "Row" and the X position of the node of the Column indicated by "Column". These are equivalent to the Argus ONE functions **NthRowPos(Row)** and **NthColumnPos(Column)**.



### **BL\_GetRowBoundaryCount and BL\_GetColumnBoundaryCount**

BL\_GetRowBoundaryCount() and BL\_GetColumnBoundaryCount() return the number of row and column boundaries in the grid.

### **BL\_GetRowNodeCount and BL\_GetColumnNodeCount**

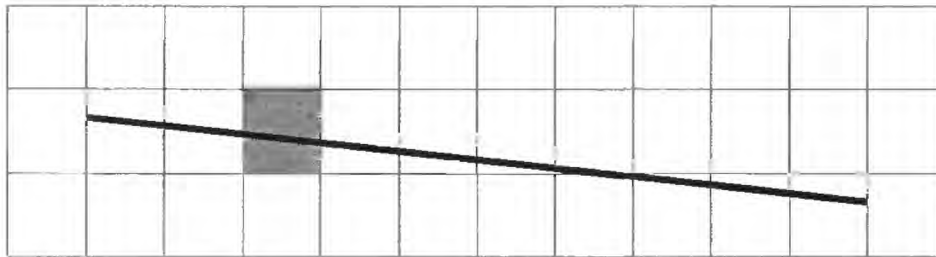
BL\_GetRowNodeCount() and BL\_GetColumnNodeCount() return the number of row and column nodes in the grid.

### **BL\_GetCellArea**

BL\_GetCellArea(Column, Row) indicates the area of the cell indicated by Column, Row.

### **BL\_FractionOfLine**

BL\_FractionOfLine(ListIndex, CellIndex) gives the fraction of the total length of the line inside the cell indicated by CellIndex (Figure 6).



**Figure 6.** The result of BL\_FractionOfLine for the cell indicated by the dark square would be 0.10 because 10 percent of the contour lies within the cell.

### **BL\_FreeVertexList**

BL\_FreeVertexList() Frees all memory associated with a list of vertices.

### **BL\_FreeAllBlockLists**

BL\_FreeAllBlockLists() Frees all memory associated with the BlockList PIE.

### **BL\_GetErrorCount**

BL\_GetErrorCount() returns the number of errors that have occurred since the last time the PIE was initialized in BL\_InitializeGridInformation. This function is used in the export template for MODFLOW-96 as a check for programming errors in the Blocklist PIE.

### **BL\_CheckVersion**

BL\_CheckVersion(First\_Digit, Second\_Digit, Third\_Digit, Fourth\_Digit) returns True if the version number of the dynamic link library is greater than or equal to the version number passed in the arguments. The version number of the dynamic link library may be checked by right clicking on BlockList.dll, selecting "Properties" and going to the version information tab.

## Conclusions

One motivation for creating the PIE commands, functions, and interpolation methods described in this report was to provide a mechanism for performing actions that could not be done easily using Argus ONE. For example, when editing a grid in Argus ONE, it was time consuming to create a grid that did not have a uniform cell size and that also did not have abrupt changes in cell sizes. The **Edit Grid** command makes creating such grids easy. Another motivation was to speed up procedures that could already be done in Argus ONE. A good example of this is the **QT\_Nearest** interpolation method. This gives exactly the same results as the **NN2D** interpolation method, but it is faster for large data sets because it uses a more efficient algorithm. This report documents the PIEs so that users will have a reference document to which they can turn when attempting to use them in their own work.

## References

- Anderson, M.P., and Woessner, W.W., 1992, Applied groundwater modeling, simulation of flow and advective transport: San Diego, California, Academic Press, 381 p.
- Argus Interware, Inc., 1997, User's guide Argus ONE™, Argus Open Numerical Environments – A GIS modeling system, version 4.0: Jericho, NY, Argus Holdings, Limited, 506 p.
- Harbaugh, A.W., Banta, E.R., Hill, M.C., and McDonald, M.G., 2000, MODFLOW-2000, the U.S. Geological Survey modular ground-water model – User's guide to modularization concepts and the ground-water flow process: U.S. Geological Survey Open-File Report 00-92, 191 p.
- Hsieh, P.A., and Freckleton, J.R., 1993, Documentation of a computer program to simulate horizontal-flow barriers using the U.S. Geological Survey modular three-dimensional finite-difference ground-water flow model: U.S. Geological Survey Open-File Report 92-477, 32 p.
- Kipp, K.L., Jr., 1987, HST3D: A computer code for simulation of heat and solute transport in three-dimensional ground-water flow systems: U.S. Geological Survey, Water-Resources Investigations Report 86-4095, 597 p.
- \_\_\_\_\_, 1997, Guide to the revised heat and solute transport simulator, HST3D – version 2: U.S. Geological Survey Water-Resources Investigations Report 97-4157, 149 p.
- Konopka, Ray, 1997, Developing custom Delphi 3 components: Albany, New York, Coriolis Group Books, 725 p.
- Poeter, E.P., and Hill, M.C., 1998, Documentation of UCODE, a computer code for universal inverse modeling: U.S. Geological Survey Water-Resources Investigations Report 98-4080, 116 p.
- Prudic, D.E., 1989, Documentation of a computer program to simulate stream-aquifer relations using a modular, finite-difference, ground-water flow model: U.S. Geological Survey Open-File Report 88-729, 113 p.
- Stephens, Rod, 1998, Ready-to-run Delphi 3.0 algorithms: Wiley, New York, 398 p.
- U.S. Department of the Interior, U.S. Geological Survey, 1992, Standards for digital elevation models: Reston, Va., 102 p.
- Voss, C.I., Boldt, David, and Shapiro, A.M., 1997, A graphical-user interface for the U.S. Geological Survey SUTRA code using Argus ONE: U.S. Geological Survey, Open-File Report 97-421, 106 p.

Winston, R.B., 1999, Upgrade to MODFLOW-GUI: Addition of MODPATH, ZONEBDGT, and additional MODFLOW packages to the U.S. Geological Survey MODFLOW-96 graphical-user interface: U.S. Geological Survey Open-File Report 99-184, 63 p.  
\_\_\_\_\_ 2000, Graphical user interface for MODFLOW: Version 4: U.S. Geological Survey Open-File Report 00-315, 27 p.

## Appendix 1: Custom Components Used in PIEs

Several custom components were developed and used as part of the PIEs described here. These components can also be used in any program or dll created using Borland Delphi. The names of the components follow the naming conventions discussed by Konopka (1997) by including the author's initials or a company name in the name of the component. Detailed descriptions of the properties, methods, and events in the components are contained in Windows help files that come with the components.

### **TRbwZoomBox**

**TRbwZoomBox** provides methods for converting real-number coordinates to screen coordinates so that points at those coordinates may be easily displayed.

By default, the positive directions for the real-number coordinate system are to the right (X-axis) and upward (Y-axis). However, the direction of either axis may be reversed. By default, the vertical exaggeration is one but the vertical exaggeration may be set to any positive real number. However, if the vertical exaggeration is set to too high or too low a value, **EInvalidOp** may be raised when calculating the screen coordinates.

**TRbwZoomBox** has two embedded components, a **TPaintBox** and a **TShape**; the **TPaintBox** provides the drawing surface. The **TShape** is only visible during zooming operations. Some of the more important methods of **TRbwZoomBox** are listed below:

Zooming methods; methods for changing the magnification at which the data are displayed:

- AbortZoom**
- BeginZoom**
- ContinueZoom**
- FinishZoom**
- ZoomBy**
- ZoomByAt**
- ZoomOut**
- SetZoom**

Panning methods; methods for changing what area is visible:

- BeginPan**
- EndPan**

Coordinate Conversion methods; methods for converting from real-number to screen coordinates and back:

- MouseToCoordinates**
- X**
- XCoord**
- Y**
- YCoord**

**TRbwZoomPoint** is a helper-class that store the real-number coordinates. Many of the zooming operations require that **TRbwZoomPoint** be used to store the coordinates.

## ***TRbwDynamicCursor***

**TRbwDynamicCursor** provides a convenient way to draw a cursor at runtime. Assign the cursor hot spot with the **HotPointX** and **HotPointY** properties. Draw the cursor on the bitmaps provided in the **OnDrawCursor** event. Call the **RedrawCursor** method whenever the cursor needs to be changed.

## ***TArgusDataEntry***

**TArgusDataEntry** provides data-checking for user-entered data. All data entered in a **TArgusDataEntry** are first checked to be sure it can be converted to another type of data as specified by the **DataType** property. In the event that the data can not be converted, characters will be stripped from the end of **Text** until the conversion is possible. If the **DataType** is **dtInteger** or **dtReal**, it is also possible to check that the numeric representation of **Text** does not lie outside the range **Min..Max**. Use **CheckMin** and **CheckMax** to turn on or off range checking. **OnExceedingBounds** and **OnExceededBounds** can be used to provide custom handling for cases where the data do lie outside the range specified by **Min..Max** when range checking is on.

**TArgusDataEntry** was originally developed while the author worked for Argus Interware. It was subsequently adopted by the U.S. Geological Survey.

## ***TRbwQuadTree***

**TRbwQuadTree** provides methods for storing data associated with X, Y coordinates in a way that allows the data close to a specific point can be retrieved quickly. Data are added using the **AddPoint** procedure and removed with the **RemovePoint** procedure. All the data in the **TRbwQuadTree** can be removed using the **Clear** procedure. Data can be retrieved using the **FindClosestPointsData**, **FindNearestPoints**, **FindPointsInBlock**, **FindPointsInCircle**, and **NearestPointsFirstData** procedures. Before adding points to the **TRbwQuadTree** it is best to set **Xmax**, **Xmin**, **Ymax**, and **Ymin** to the limits of the data to be added or something close to the limits. Performance may suffer if the limits are greatly different from the actual limits of the data. **MaxPoints** specifies the maximum number of data points in any one leaf of a **TRbwQuadTree**. Performance may be affected by **MaxPoints** in a way that depends on the nature of the task to be performed.

## ***TRbwOctTree***

**TRbwOctTree** is the three-dimensional equivalent of **TRbwQuadTree**.

## ***Installation***

To install **TRbwZoomBox**, **TRbwDynamicCursor**, **TArgusDataEntry**, **TRbwQuadTree**, and **TRbwOctTree** in Delphi 5, start Delphi 5 and select **Component|Install Component**. Select the file **RbwZoomBox.pas**, **RbwDynamicCursor.pas**, **ArgusDataEntry.pas**, **QuadtreeClass**, or **OctTreeClass** as the "Unit file name" and click the OK button. To install context-sensitive help for the components in Delphi 5 select **Help|Customize**. On the Contents tab select **Edit|Add Files** and select **TRBWZoomBox.cnt**, **TDynamicCursor.cnt**, or **TArgusDataEntry.cnt**. Next, on the Index tab select **Edit|Add Files** and select **TRbwZoomBox.hlp**, **TDynamicCursor.hlp**, or **TArgusDataEntry.hlp**. Then on the Link tab select **Edit|Add Files** and select

TRbwZoomBox.hlp, TDynamicCursor.hlp, or TArgusDataEntry.hlp. Finally, select **File|Save Project** and close the dialog box.

## Appendix 2: Example Export Template for Progress Bar PIE

```

#
Define Variable: OK [Boolean]
Set Variable: OK:= ProgressBarCheckVersion(1,3,0,0)
#
# Check if the PIE is installed.
If: IsNAN(OK)
    Set Variable: OK:= 0
    Alert: "Aborting: You either have an outdated version of the Progress Bar PIE or else you don't
have the Progress Bar PIE installed."
Else
    # Check if the PIE is up-to-date.
    If: OK=0
        Alert: "Aborting: You have an outdated version of the Progress Bar PIE."
    End if
End if
If: OK
    # Initialize the progress bar. Set the maximum for the
    # progress bar to the number of blocks.
    Evaluate expression: ProgressBarInitialize(NumBlocks(), 1)
    #
    # Set the message shown in the progress bar
    Set Variable: OK:= ProgressBarSetMessage("Exporting Blocks")
    #
    # Check if the user has pressed the Cancel button.
    If: OK
        Redirect output to: $BaseName$
        Loop for: Blocks
            #
            # Advance the progress bar by one. At the end of the
            # loop over blocks this function will have been called
            # NumBlocks() times.
            Set Variable: OK:= ProgressBarAdvance()
            #
            # Check if the user has pressed the Cancel button.
            If: OK
                # Make a test such as a test for an error condition.
                If: Column()=Row()
                    #
                    # If the test succeeds, add a message to the progress
bar memo.
                    Evaluate expression: ProgressBarAddLine("Column: "
+ Column() + " = Row: " + Row())
                End if
            End if
        End loop
    End file
    #
    # Save the messages in the progress bar memo to a file.
    Evaluate expression: ProgressBarSaveToFile("AFile")
End if
#
# Get rid of the progress bar.
Evaluate expression: ProgressBarFree()
End if

```

## Appendix 3: Example Export Template for List PIE

```
#
# These will be assigned values of ListIndex.
Define Variable: FirstList [Integer]
Define Variable: SecondList [Integer]
# These will be used to determine how many items are in the list.
Define Variable: LoopIndex [Integer]
Redirect output to: $BaseName$
  Start a new line
    # FirstList is assigned a value of 0.
    Set Variable: FirstList:= L_CreateNewList()
    Export expression: FirstList; [G0]
    # SecondList is assigned a value of 1.
    Set Variable: SecondList:= L_CreateNewList()
    Export expression: SecondList; [G0]
    # We now add variables to the first and second lists.
    Evaluate expression: L_AddToList(FirstList, 5)
    Evaluate expression: L_AddToList(FirstList, 7)
    Evaluate expression: L_AddToList(SecondList, 9)
    # The number of items in FirstList is 2 and
    # the number of items in SecondList is 1.
    Export expression: L_GetListSize(FirstList); [G0]
    Export expression: L_GetListSize(SecondList); [G0]
  End line
# This exports all values in the first list.
Start a new line
  Export expression: "FirstList" [G0]
End line
Start a new line
  Loop for: Variable LoopIndex from: 0 to: L_GetListSize(FirstList)-1 step: 1
    Export expression: L_GetFromList(FirstList, LoopIndex); [G0]
  End loop
End line
# This exports all values in the second list.
Start a new line
  Export expression: "SecondList" [G0]
End line
Start a new line
  Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
    Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
  End loop
End line
# This does nothing because there are already more than one item if FirstList.
Evaluate expression: SetListSize(FirstList, 1)
# This adds two more items with a values of 0 to the end of SecondList.
Evaluate expression: L_SetListSize(SecondList, 3)
# This exports all values in the first list.
Start a new line
  Export expression: "FirstList" [G0]
End line
Start a new line
  Loop for: Variable LoopIndex from: 0 to: L_GetListSize(FirstList)-1 step: 1
    Export expression: L_GetFromList(FirstList, LoopIndex); [G0]
  End loop
End line
# This exports all values in the second list.
Start a new line
  Export expression: "SecondList" [G0]
End line
Start a new line
```



```

        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
            Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
        End loop
    End line
    # This deletes all items in FirstList but does not change SecondList.
    Evaluate expression: L_FreeAList(FirstList)
    # This exports all values in the first list.
    Start a new line
        Export expression: "FirstList is Empty"; [G0]
        # This returns 0
        Export expression: L_GetListSize(FirstList) [G0]
    End line
    Start a new line
        # These will cause an error because all loops are executed at least once.
        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(FirstList)-1 step: 1
            # The error occurs here because FirstList contains nothing.
            Export expression: L_GetFromList(FirstList, LoopIndex); [G0]
        End loop
    End line
    # This exports all values in the second list.
    Start a new line
        Export expression: "SecondList" [G0]
    End line
    Start a new line
        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
            Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
        End loop
    End line
    # This sets the value of the second item in SecondList to 20.
    Evaluate expression: L_SetListItem(SecondList, 1, 20)
    # This exports all values in the second list.
    Start a new line
        Export expression: "SecondList" [G0]
    End line
    Start a new line
        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
            Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
        End loop
    End line
    # This deletes the first item of SecondList.
    Evaluate expression: L_DeleteListItem(SecondList, 0)
    # This exports all values in the second list.
    Start a new line
        Export expression: "SecondList" [G0]
    End line
    Start a new line
        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
            Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
        End loop
    End line
    # This deletes the last item of SecondList.
    Evaluate expression: L_DeleteListItem(SecondList)
    # This exports all values in the second list.
    Start a new line
        Export expression: "SecondList" [G0]
    End line
    Start a new line
        Loop for: Variable LoopIndex from: 0 to: L_GetListSize(SecondList)-1 step: 1
            Export expression: L_GetFromList(SecondList, LoopIndex); [G0]
        End loop
    End line
    # Test if error has occurred. This export template was designed to have one error in it.

```

```

    If: L_GetErrorCount()
        Start a new line
        Export expression: "Whoops. The number of errors was " + L_GetErrorCount()
[G0]
        End line
    End if
    # This frees up memory. It returns True if successful.
    Start a new line
        Export expression: L_FreeAllLists() [G0]
    End line
End file

```

The contents of the file exported by this template are as follows:

```

0 1 2 1
FirstList
5. 7.
SecondList
9.
FirstList
5. 7.
SecondList
9. 0. 0.
FirstList is Empty 0
0.
SecondList
9. 0. 0.
SecondList
9. 20. 0.
SecondList
20. 0.
SecondList
20.
Whoops. The number of errors was 1
1

```

## Appendix 4: Example Export Template for BlockList PIE

```
#
Define Variable: CellListIndex [Integer]
Define Variable: CellIndex [Integer]
Redirect output to: $BaseName$
  Start a new line
    Export expression: BL_InitializeGridInformation("Grid"); [G0]
    Export expression: BL_AddVertexLayer("New Layer1"); [G0]
    Export expression: BL_AddVertexLayer("New Layer2"); [G0]
  End line
  Start a new line
    Export expression: "Number of Cell Lists: " [G0]
    Export expression: BL_GetCountOfCellLists() [G0]
  End line
  Loop for: Variable CellListIndex from: 0 to: BL_GetCountOfCellLists()-1 step: 1
    Start a new line
      Export expression: "Cells for Object " [G0]
      Export expression: BL_CellListIndex [G0]
    End line
    Loop for: Variable CellIndex from: 0 to: BL_GetCountOfACellList(CellListIndex)-1 step: 1
      Start a new line
        Export expression: "Column"; [G0]
        Export expression: BL_GetCellColumn(CellListIndex, CellIndex); [G0]
        Export expression: "Row"; [G0]
        Export expression: BL_GetCellRow(CellListIndex, CellIndex); [G0]
      End line
    End loop
  End loop
  Start a new line
    Export expression: "Combined List" [G0]
  End line
  Loop for: Variable CellIndex from: 0 to: BL_GetCountOfCombinedCellList()-1 step: 1
    Start a new line
      Export expression: "Column"; [G0]
      Export expression: BL_GetCellColumnFromCombinedList(CellIndex); [G0]
      Export expression: "Row"; [G0]
      Export expression: BL_GetCellRowFromCombinedList(CellIndex) [G0]
    End line
  End loop
  Start a new line
    Export expression: "Reinitialize: "; [G0]
    Export expression: BL_ReinitializeVertexList(); [G0]
    Export expression: BL_AddVertexLayer("New Layer3") [G0]
  End line
  Start a new line
    Export expression: "Number of Cell Lists: " [G0]
    Export expression: BL_GetCountOfCellLists() [G0]
  End line
  Loop for: Variable CellListIndex from: 0 to: BL_GetCountOfCellLists()-1 step: 1
    Start a new line
      Export expression: "Cells for Object " [G0]
      Export expression: CellListIndex [G0]
    End line
    Loop for: Variable CellIndex from: 0 to: BL_GetCountOfACellList(CellListIndex)-1 step: 1
      Start a new line
        Export expression: "Column"; [G0]
        Export expression: BL_GetCellColumn(CellListIndex, CellIndex); [G0]
        Export expression: "Row"; [G0]
        Export expression: BL_GetCellRow(CellListIndex, CellIndex); [G0]
      End line
```

```

        End loop
    End loop
    Start a new line
        Export expression: "Combined List" [G0]
    End line
    Loop for: Variable CellIndex from: 0 to: BL_GetCountOfCombinedCellList()-1 step: 1
        Start a new line
            Export expression: "Column"; [G0]
            Export expression: BL_GetCellColumnFromCombinedList(CellIndex); [G0]
            Export expression: "Row"; [G0]
            Export expression: BL_GetCellRowFromCombinedList(CellIndex) [G0]
        End line
    End loop
    Start a new line
        Export expression: BL_FreeAllBlockLists() [G0]
    End line
End file

```

Executing this export template can yield a file with the following contents.

```

1 1 1
Number of Cell Lists: 2
Cells for Object 0
Column 1 Row 3
Column 2 Row 3
Column 3 Row 3
Column 4 Row 3
Column 5 Row 3
Column 6 Row 3
Column 6 Row 4
Cells for Object 1
Column 2 Row 5
Column 2 Row 4
Column 3 Row 4
Column 3 Row 3
Column 4 Row 3
Column 4 Row 2
Column 5 Row 2
Column 6 Row 2
Combined List
Column 1 Row 3
Column 2 Row 3
Column 3 Row 3
Column 4 Row 3
Column 5 Row 3
Column 6 Row 3
Column 6 Row 4
Column 2 Row 5
Column 2 Row 4
Column 3 Row 4
Column 4 Row 2
Column 5 Row 2
Column 6 Row 2
Reinitialize: 1 1
Number of Cell Lists: 1
Cells for Object 0
Column 1 Row 7
Column 2 Row 7
Column 3 Row 7
Column 4 Row 7
Column 5 Row 7

```

Column 6 Row 7  
Combined List  
Column 1 Row 7  
Column 2 Row 7  
Column 3 Row 7  
Column 4 Row 7  
Column 5 Row 7  
Column 6 Row 7  
1

## Appendix 5: Export Templates Using Rotated X and Rotated Y

### Template 1

```
#
# Export template to export the row and column numbers
# and the locations of the four corners of each block.
#
Define Variable: X [Real]
Define Variable: Y [Real]
#
Redirect output to: $BaseName$
  Start a new line
    Export expression: "Row Number"; [G0]
    Export expression: "Column Number"; [G0]
    Export expression: "X-coord 1"; [G0]
    Export expression: "Y-coord 1"; [G0]
    Export expression: "X-coord 2"; [G0]
    Export expression: "Y-coord 2"; [G0]
    Export expression: "X-coord 3"; [G0]
    Export expression: "Y-coord 3"; [G0]
    Export expression: "X-coord 4"; [G0]
    Export expression: "Y-coord 4"; [G0]
  End line
  Loop for: Blocks
    Start a new line
      Export expression: Row(); [G0]
      Export expression: Column(); [G0]
      Set Variable: X:= NthColumnPos(Column()-1)
      Set Variable: Y:= NthRowPos(Row()-1)
      Export expression: Rotated X(X, Y, GridAngle()); [G0]
      Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      Set Variable: X:= NthColumnPos(Column())
      Export expression: Rotated X(X, Y, GridAngle()); [G0]
      Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      Set Variable: Y:= NthRowPos(Row())
      Export expression: Rotated X(X, Y, GridAngle()); [G0]
      Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      Set Variable: X:= NthColumnPos(Column()-1)
      Export expression: Rotated X(X, Y, GridAngle()); [G0]
      Export expression: Rotated Y(X, Y, GridAngle()); [G0]
    End line
  End loop
End file
```

## Template 2

```
#
# This template uses the OkCancel and Utility PIEs
# to export a contour that surrounds a single cell.
# The contour can be used as a domain outline for a
# model that defines a subgrid within that cell.
#
Define Variable: RowToUse [Integer]
Define Variable: ColumnToUse [Integer]
Define Variable: X [Real]
Define Variable: Y [Real]
Redirect output to: $BaseName$
  Set Variable: RowToUse:= OK_UserInteger("What row do you want to use?", 1, 1, NumRows())
  Set Variable: ColumnToUse:= OK_UserInteger("What column do you want to use?", 1, 1,
NumColumns())
  Start a new line
    Export expression: "## Name:" [G0]
  End line
  Start a new line
    Export expression: "## Icon:"; [G0]
    Export expression: 0 [G0]
  End line
  Start a new line
    Export expression: "# Points Count"; [G0]
    Export expression: "Value" [G0]
  End line
  Start a new line
    Export expression: 5 [G0]
  End line
  Start a new line
    Export expression: "# X pos"; [G0]
    Export expression: "Y pos" [G0]
  End line
  Loop for: Blocks
    If: (Row()=RowToUse)&(Column()=ColumnToUse)
      Start a new line
        Set Variable: X:= NthColumnPos(Column()-1)
        Set Variable: Y:= NthRowPos(Row()-1)
        Export expression: Rotated X(X, Y, GridAngle()); [G0]
        Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      End line
      Start a new line
        Set Variable: X:= NthColumnPos(Column())
        Export expression: Rotated X(X, Y, GridAngle()); [G0]
        Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      End line
      Start a new line
        Set Variable: Y:= NthRowPos(Row())
        Export expression: Rotated X(X, Y, GridAngle()); [G0]
        Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      End line
      Start a new line
        Set Variable: X:= NthColumnPos(Column()-1)
        Export expression: Rotated X(X, Y, GridAngle()); [G0]
        Export expression: Rotated Y(X, Y, GridAngle()); [G0]
      End line
      Start a new line
        Set Variable: Y:= NthRowPos(Row()-1)
        Export expression: Rotated X(X, Y, GridAngle()); [G0]
        Export expression: Rotated Y(X, Y, GridAngle()); [G0]
```

End file  
End loop  
End if  
End line



## **Appendix 6: DEM2Image.exe**

**DEM2Image.exe** reads Digital Elevation Models (DEMs) in the format described in the DEM data users guide (U.S. Department of the Interior, U.S. Geological Survey, 1992) (<http://rockyweb.cr.usgs.gov/nmpstds/demstds.html>) and displays them as bitmaps. Once the program has been started by double-clicking on its icon, its operation is much the same as the Sample DEM Data command described in the main body of this report. However, you do not need to specify a grid nor does DEM2Image extract gridded data.