Control of Blowing Snow using SNOWMAN:  
Developers Manual  

NYSDOTIRC Subcontract 28311-5823  
Project C-01-67  

M. F. Lamanna, M.S.  
S. S. Chen, Ph.D., P.E.  
Department of Civil, Structural, and Environmental Engineering  
University at Buffalo  
State University of New York  

Revised  
December 2008
ACKNOWLEDGEMENTS

This work was overseen by Project Manager Joseph F. Doherty, MBA, P.E., of the New York State Department of Transportation (NYSDOT). It was sponsored by NYSDOT’s Transportation Infrastructure Research Consortium (TIRC) which is administered by staff at Cornell University. The opinions and conclusions expressed or implied in this report are those of the authors. The advice of Dr. R. D. Tabler of Tabler and Associates and of Mr. D. F. Kaminski, P.E., of NYSDOT, is gratefully acknowledged. The contributions of former graduate students David Schwartz and Xiaotian Wang are also gratefully acknowledged.

DISCLAIMER

The contents of this report reflect the views of the author who is responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the New York State Department of Transportation, the United States Department of Transportation, or the Federal Highway Administration. This report does not constitute a standard, specification, regulation, product endorsement, or an endorsement of manufacturers.
### Control of Blowing Snow using SnowMan: Developer's Manual

Properly engineered passive snow control measures can significantly reduce the safety hazards and winter maintenance costs associated with the problem of blowing and drifting snow. There are two possible mitigation strategies: roadway (cross section) design and snow fencing. This project developed and deployed a software application, named SnowMan (for Snow Management), to run within the NYSDOT’s MicroStation-based CAD environment to assist highway designers and maintenance users in the design of such passive control measures. This effort thus extends the applicability of the earlier PASCON expert system software (Kaminski and Mohan 1991) and incorporates well-established knowledge regarding snow transport and deposition, evaluating roadway cross sections for drift susceptibility, design of passive and living snow fences, and earthwork modification for reducing drifting (Tabler 2003). The SnowMan software brings the science of engineered mitigation of blowing and drifting snow to the desktop. This manual documents information relevant to programmers charged with maintaining and/or extending the software capabilities. Benefits of the use of this software include reducing maintenance costs and closure times and improving crash incidence by improving visibility, preventing drifting on the road, and reducing road icing.
# Table of Contents

## 1 Development and Files
1.1 Source Code Files 1
1.2 Runtime Data Files 2
1.3 Level Standards 3

## 2 Data Structures
2.1 SnowmanGlobals 4
2.2 snowmanDefaults 11
2.3 snowmanLevelStandard 13
2.4 NYSDOTCADDLevelStandard 14
2.5 SnowmanLevelStandards 14
2.6 SnowmanClimateData 17
2.7 SnowmanDataPoint 18
2.8 SnowmanProfile 18
2.9 SnowmanCrossSection 19
2.10 SnowmanTopoData 21
2.11 SnowmanFence 21
2.12 SnowmanDitch 22
2.13 SnowmanResult 23
2.14 SnowmanCaseResult 26
2.15 SnowmanResultsData 27
2.16 SnowmanFenceConstants 28
2.17 SnowmanElements 30

## 3 Module Definitions
3.1 SnowmanEvaluation 32
  3.1.1 External Function Definitions 33
    case1 33
    case2 33
    case3 34
    case4 35
    case5 36
    case6 37
    case7 37
    case8 38
  3.1.2 Internal Function Definitions 39
    createNewGround 39
    closestSetback 39
    effectiveHeight 40
    isBuried 40
    nextSetback 41
    mitigatedProfile 41
    nextUnmitigatedDriftElevation 42
    computeStorageAndDepthAndCheckForProblem 42
setFenceConstants 44
shortestFence 44
nextTallerFence 45
actualHeight 46
numberOfPossibleFences 46
findElevation 47

3.2 FileHandlers 48
3.2.1 External Function Definitions 49
loadDefaultsFile 49
loadLevelStandardsFile 49
loadClimateDataFile 50
saveClimateDataFile 51
loadTopoDataFile 51
saveTopoDataFile 52
saveResultsDataFile 53

4 Main Program 55
4.1 Initialization 57
main 57
startSNOWMAN 57
getUserInputInformation 58
4.2 Climate input methods 59
getClimateData 59
displayClimateData 59
recordClimateData 60
4.3 Topographic input methods 62
getTopoData 62
ilkElements 63
findElements 63
recordStartingPoint 64
recordEndingPoint 65
validULOPPoint 65
generateCrossSections 66
addDataPoint 67
getNextDataPoint 67
getSectionInformation 68
recordSectionInfo 69
manualInputDone 70
displayTopoData 70
displayNextSection 70
displayPrevSection 71
startProcessing 71
4.4 Case selection methods 73
getCaseSelections 73
checkCaseSelections 73
getCaseSpecificData 73
4.5 Output
   getOutputOptions  82
   recordOutputPoint  82
   plotOutputToScreen  82
   outputFenceAndCaseInformation  84
   outputGeneralInformation  84
   outputLabel  85
   outputAxis  85
   outputSectionInformation  86
   outputProfile  86
   outputEarthworkCaseInformation  87

5 Structural Organization  88
5.1 Overall  89
5.2 Input  90
   5.2.1 General  91
   5.2.2 Climate  92
   5.2.3 Topographic  95
5.3 Processing  99
   5.3.1 Case selection  101
   5.3.2 Case specific data  101
   5.3.3 Generate Results  102
5.4 Output  109
List of Figures

Fig. 1 Top-Level Interactions 89
Fig. 2 Input Modules 90
Fig. 3 User Information Input 91
Fig. 4 Climate Information Input 92
Fig. 5 Site-Specific Climate Data 93
Fig. 6 Climate Data Computed by SnowMan 93
Fig. 7 Climate Data from File 94
Fig. 8 Topographic Data Input 95
Fig. 9 Input from SnowMan Topographic Data File 96
Fig. 10 Manual Input of 2D Sections 97
Fig. 11 Topographic Input from Design File 98
Fig. 12 Processing Cross Sections 99
Fig. 13 Get Case Selections 100
Fig. 14 Get Case-Specific Data 101
Fig. 15 Generate Results 102
Fig. 16 Case 1 Results 103
Fig. 17 Case 2 Results 103
Fig. 18 Case 3 Results 104
Fig. 19 Case 4 Results 105
Fig. 20 Case 5 Results 106
Fig. 21 Case 6 Results 107
Fig. 22 Case 7 Results 107
Fig. 23 Case 8 Results 108
Fig. 24 Output Processing 109
Fig. 25 Get Output Options 110
Fig. 26 Results Output to Data File or Screen 110
Fig. 27 Results Output to Screen 111
1 Development and Files

The MDL application SNOWMAN was developed under MicroStation version 08.01.00.07 Windows x86. Compilation was done through the bmake utility, Bentley Systems Make Utility version 08.99.99.99. Resources were compiled with the MicroStation Resource Compiler version 08.01.00. The type resource file was created with the MicroStation Type Resource File Generator version 08.01.00. The modules were compiled with the MicroStation Development Language Compiler version 08.01.00. The resources and modules were liked with the MicroStation Development Language Linker version 08.01.00. The MDL application was assembled with the MicroStation Resource Librarian version 08.01.00.

The file snowman.mke contains the instructions for compiling, linking and assembling the MDL application executable file SNOWMAN.MA when using the Bentley Systems Make Utility. The files snowman.h, snowman.mc, snowmanEvaluation.h, snowmanEvaluation.mc, snowmanFileHandlers.h, snowmanFileHandlers.mc, snowmanDataStructures.h, snowman.r, and snowmancmd.r are required to build the application using the make file and the Bentley Systems Make Utility. These source code files must be in the same directory as the snowman.mke makefile in order for the SnowMan application to be re-compiled and re-built using the MicroStation bmake utility.

The files nym_snowman_defaults.data, nyu_snowman_defaults.data, and ny_snowman_levels.data are also required to run the snowman application. The .data files must be in the path to the directory referenced by the MicroStation Environment Variable “SNOWMAN_HOME”.

The New York State - specific climatological data files originally developed for the SnowMan software are not used directly. An interpolation algorithm developed by Dr. Ron Tabler is used instead.

1.1 Source Code Files

snowman.h
The file snowman.h contains constants for the main program as well as the definition of the snowmanglobals data structures used to transfer information back and forth between the program and the dialog boxes.

snowmanDataStructures.h
This file contains all the definitions for the various data structures used to store information in SnowMan. These data structures are used in both the main program as well as the evaluation module and the file handling module.
snowmanEvaluation.h
This file declares the function names of the routines that are defined in snowmanEvaluation.mc. These functions are called by the main program.

snowmanFileHandlers.h
This file declares the function names of the routines that are defined in snowmanFileHandlers.mc, as well as the constants that define success or failure for files being loaded or saved. These functions are called by the main program.

snowmancmd.h
This file defines the command numbers used by the dialog boxes in SnowMan.

snowman.mc
This file contains all the routines for gathering, storing, and transferring data. This includes communication with the user via dialog boxes, calling evaluation routines to process data, calling file handling routines to read and store data to and from files, and those used to modify the design file to display the results of the processing.

snowmanEvaluation.mc
This file contains the evaluation routines declared in snowmanEvaluation.h. These modules define the functions necessary to process the data and develop results.

snowmanFileHandlers.mc
This file contains the file handling routines declared in snowmanFileHandlers.h. These modules handle the storing of data to file and gathering data from files.

1.2 Runtime Data Files

ny_snowman_levels.data
This file is loaded by SnowMan and contains the relevant level definitions.

nyu_snowman_defaults.data
This file contains the default parameters in US Customary units.

nym_snowman_defaults.data
This file contains the default parameters in metric units.

1.3 Level Standards

SnowMan must be able to find certain information contained in the design file related to the location of any ditches, limits of protection, and topography. For ditches, limits of protection, and topography there is corresponding definition of level, weight and color for the lines of
interest defined in the file ny_snowman_level.data. The current version of SnowMan uses the latest definitions that were available for those elements from the New York State Department of Transportation's CADD level standards.

Ditch information is defined by EXISTING DITCH in ny_snowman_levels.data. The upwind and downwind limits of protection are defined by UPWIND EDGE OF PROTECTION and DOWNWIND EDGED OF PROTECTION respectively in ny_snowman_levels.data. The topography needs to be in the form of triangles defined by EXISTING TOPO in ny_snowman_levels.data.

Currently these values are:
for ditches, level=DD, color=7, weight=0, style=6
for upwind limit of protection, level=LLOPSU_P, color = 5, weight=2, style=0
for downwind limit of protection, level=LLOPSD_P, color = 5, weight=2, style=0
for topography, level=RDEFAULT_P, color=1, weight=0, style=0
2 Data Structures

2.1 snowmanGlobals

Struct: snowmanGlobals
Fields: 62 (described in the following)
Purpose: Communication of data between dialog boxes and the SnowMan program

Field: userName
Type: char[256]
Purpose: Stores the name of the current user of the program.

Field: userLevel
Type: int
Purpose: Stores the user’s level [expert, novice]

Field: PIN (project ID number)
Type int
Purpose: Stores the project ID number for the area being investigated

Field: siteID
Type: char[256]
Purpose: Stores the description of the site in the area under investigation

Field: climateInputOption
Type: int
Purpose: Stores the method for climate data input
[CLIMATE_INPUT_BY_SITE_SPECIFIC_DATA, CLIMATE_INPUT_BY_LAT_LONG_ELEV_COMPUTATION, CLIMATE_INPUT_BY_SNOWMAN_CLIMATE_DATA_FILE]

Field: topoInputOption
Type: int
Purpose: Stores the method for topographic data input
[TOPO_INPUT_BY_EXISTING_DESIGN_FILE, TOPO_INPUT_BY_PROPOSED_DESIGN_FILE, TOPO_INPUT_BY_MANUAL_INPUT, TOPO_INPUT_BY_SNOWMAN_TOPO_DATA_FILE]

Field: latitude
Type: double
Purpose: Stores the North Longitude of the site under investigation.

Field: longitude
Type: double
Purpose: Stores the West Longitude of the site under investigation.
Field: elevation
Type: double
Purpose: Multiple uses. Whenever information is gathered from the user or displayed to the user about the elevation in meters this field conveys that information. It is used for the characteristic elevation of the site for climate computations as well as the elevation of a particular data point for offset elevation pairs in topographic profiles.

Field: St
Type: double
Purpose: Stores the total annual snowfall for a site in millimeters.

Field: Cr
type: double
Purpose: Stores the relocation coefficient [0.0,1.0]
Default value: 0.17 (NYS)

Field: direction
Type: double
Purpose: Stores the true North Azimuth of the direction of the wind. This follows the normal climatological convention whereby the direction of the wind is given by the direction the wind comes from.

Field: k
Type: double
Purpose: Exceedence factor to use for the site under investigation.
Default value: 1.5 (represents 95% confidence level)

Field: ambientSnow
Type: double
Purpose: Stores the ambient snow cover in millimeters for the site under investigation.

Field: saveClimateData
Type: int
Purpose: Indicates if the climate data should be saved to a file. [TRUE, FALSE]

Field: sectionNumber
Type: int
Purpose: Used to display the current section number to the user during the collection and display of topographic data. It is also used to display the current section number to the user for gathering case specific data.

Field: dataPointNumber
Type: int
Purpose: Used to display the current data point during manual input of topographic data.
Field: offset
Type: double
Purpose: Stores the current offset of a data point being entered.

Field: deopOffset
Type: double
Purpose: Stores the downwind (positive) offset measured from the upwind limit of protection of
the downwind limit of protection during the gathering of section information in manual input of
topographic data.

Field: hasDitch:
Type: int
Purpose: Indicates if the current section has relevant ditch information. [TRUE, FALSE]

Field: ditchOffset
Type: double
Purpose: Stores the upwind offset (negative) in meters measured from the upwind limit of
protection to the lowest point in the ditch if the section has a ditch.

Field: ditchElevation
Type: double
Purpose: Stores the elevation in meters of the lowest point in the ditch.

Field: angleToRoad
Type: double
Purpose: Stores the angle between the wind and the road. This information is collected from the
user during the manual input of topographic data and displayed to the user during display of
topographic data. The value is used in computation of offsets along the direction of the wind.

Field: fetch
Type: double
Purpose: Stores the distance from the upwind limit of protection to the limit of fetch in meters
along the direction of the wind.

Field: manualInputDone
Type: int
Purpose: Indicates if the current section is the last section to be input during manual input of
topographic data. [TRUE, FALSE]

Field: DITCHS
Type: char[32]
Purpose: Stores the string of characters used to display the ditch offset and elevation. Will indicate
either the offset elevation point of the lowest point in the ditch or simply that there is no ditch.

Field: downwindEOPS
Type: char[32]
Purpose: Stores the string of characters used to display the distance to the downwind limit of protection.

Field: fetchS
Type: char[32];
Purpose: Stores the string of characters used to display the distance from the upwind limit of protection to the limit of fetch.

Field: angleToRoad
Type: char[32]
Purpose: Used to display the angle between the wind and the road.

Field: saveTopoData
Type: int
Purpose: Used to indicate if the topographic data should be save to a file. [TRUE, FALSE]

Field: caseNumber1
Type: int
Purpose: Indicates if case 1 should be run during the evaluation phase.

Field: caseNumber2
Type: int
Purpose: Indicates if case 2 should be run during the evaluation phase.

Field: caseNumber3
Type: int
Purpose: Indicates if case 3 should be run during the evaluation phase.

Field: caseNumber4
Type: int
Purpose: Indicates if case 4 should be run during the evaluation phase.

Field: caseNumber5
Type: int
Purpose: Indicates if case 5 should be run during the evaluation phase.

Field: caseNumber6
Type: int
Purpose: Indicates if case 6 should be run during the evaluation phase.
Field: caseNumber7
Type: int
Purpose: Indicates if case 7 should be run during the evaluation phase.

Field: caseNumber8
Type: int
Purpose: Indicates if case 8 should be run during the evaluation phase.

Field: height
Type: int
Purpose: Used to collect the height of the fence for cases where the user selects a height.
[HEIGHT_1_4_METERS, HEIGHT_1_8_METERS, HEIGHT_2_0_METERS, HEIGHT_2_1 METERS, HEIGHT_2_4_METERS, HEIGHT_2_7_METERS, HEIGHT_3_0_METERS, HEIGHT_3_6_METERS]

Field: actualHeight
Type: double
Purpose: For the case where the user specifies the actual height instead of selecting from the list of available fence heights, this field is used to store the actual height of the fence.

Field: porosity
Type: int
Purpose: Used to collect the porosity of the fence where the user can specify porosity.
[POROSITY_50_PERCENT, POROSITY_37_5_PERCENT, POROSITY_25_PERCENT, POROSITY_0_PERCENT]

Field: setback
Type: double
Purpose: Stores the setback in meters measured from the upwind limit of protection along the direction of the wind to the item of interest. This could be the maximum allowable setback of a fence or limit of earthwork. It could also be the actual setback of the fence to be used.

Field: fenceType
Type: int
Purpose: Indicates the type of fence/fences to use. [PERMANENT_ONLY, PORTABLE_ONLY, TEMPORARY_ONLY, PERMANENT_OR_PORTABLE]

Field: allowBuriedFence
Type: int
Purpose: Indicates if the fence for the current section is allowed to be buried more than the default allowable buried fence ratio or not. [TRUE, FALSE]
Field: allowableBuriedFenceRatio
Type: double
Purpose: Indicates the percentage of the fence that is allowed to be buried by snow and still considered “unburied”. [0.0, 1.0]
Default value: [0.2] (NYSDOT)

Field: useAmbientSnow
Type: int
Purpose: Indicates if the ambient snow cover should be used during computation of the effective fence height (i.e. the part that is not buried). [TRUE, FALSE]
Default value: FALSE (NYSDOT)

Field: useSameForAll
Type: int
Purpose: Indicates if the case specific data for the current section should be use for all remaining sections in this case. [TRUE, FALSE]
Default value: TRUE

Field: backSlope
Type: double
Purpose: Indicates the back slope of the ditch to be used when generating new ground profiles.
Default value: 1 on 2 slope, 0.5 (NYSDOT)

Field: bottomSlope
Type: double
Purpose: Indicates the bottom slope of the ditch to be used when generating new ground profiles.
Default value: 2% slope, 0.02 (NYSDOT)

Field: toFile
Type: int
Purpose: Indicates if the results data should be sent to a file or not. [TRUE, FALSE]
Default value: FALSE

Field: toScreen
Type: int
Purpose: Indicates if the results data should be displayed to the design file. [TRUE, FALSE]
Default value: TRUE

Field: scaleFactor
Type: int
Purpose: Indicates if the results data should be exaggerated in the Y-axis when outputting to the
design file. [SCALE_FACTOR1, SCALE_FACTOR5]

Field: startPointX
Type: double
Purpose: Stores X-axis value of the starting point along the upwind limit of protection.

Field: startPointY
Type: double
Purpose: Stores Y-axis value of the starting point along the upwind limit of protection.

Field: startPointZ
Type: double
Purpose: Stores Z-axis value of the starting point along the upwind limit of protection.

Field: endPointX
Type: double
Purpose: Stores X-axis value of the ending point along the upwind limit of protection.

Field: endPointY
Type: double
Purpose: Stores Y-axis value of the ending point along the upwind limit of protection.

Field: endPointZ
Type: double
Purpose: Stores Z-axis value of the ending point along the upwind limit of protection.

Field: outputPointX
Type: double
Purpose: Stores the X-axis value of the upper left corner where output is to start.

Field: outputPointY
Type: double
Purpose: Stores the Y-axis value of the upper left corner where output is to start.

Field: outputPointZ
Type: double
Purpose: Stores the Z-axis value of the upper left corner where output is to start.

Field: haveOutputPoint
Type: int
Purpose: indicates if the user has entered a data point for the output.
2.2 snowmanDefaults

Struct: snowmanDefaults
Fields: 16 (described in the following)
Purpose: Storage of default values to use. This data structure stores the contents of the file defaults.data

Field: levelStandardsFile
Type: char[256]
Purpose: unused legacy field from previous version of SNOWMAN

Field: sectionSpacing
Type: double
Purpose: Stores distance in meters to be used between sections for input by design file. The distance is measured along an imaginary line between the starting point to the ending point.

Field: Cr
Type: double
Purpose: Stores the relocation coefficient to be used as default.

Field: rowGap
Type: double
Purpose: unused legacy field from previous version of SNOWMAN

Field: maxFenceOffset
Type: double
Purpose: Stores the farthest distance upwind in meters measured from the upwind limit of protection to a fence. That is, no fence can be placed further than this from the upwind limit of protection.

Field: minFenceOffset
Type: double
Purpose: unused. Minimum is now computed as 15*height

Field: allowableBuriedFenceRatio
Type: double
Purpose: Stores the default allowable buried depth ratio.

Field: maxNaturalAccumOffset
Type: double
Purpose: unused. Storage is not computed from the limit of terrain or limit of fetch (whichever is smaller).
Field: LOCACCMOffset
Type: double
Purpose: Stores the lower bound of the offset location where computation of storage for design transport is computed. That is, storage is measured from the most upwind point or the limit of fetch to this point.

Field: allowableSoR
Type: double
Purpose: Stores the allowable depth in millimeters of snow at either limit of protection. That is, depths greater than this value will result in flagging a section as having a drifting snow problem.

Field: minUpwindTerrain
Type: double
Purpose: Stores the minimum upwind terrain in meters measured from the upwind limit of protection that is necessary for SNOWMAN to perform computations.

Field: minDownwindTerrain
Type: double
Purpose: Stores the minimum downwind terrain in meters measured from the downwind limit of protection that is required for SNOWMAN to perform computations.

Field: minFetch
Type: double
Purpose: Stores the minimum value of fetch in meters measured from the upwind limit of protection along the direction of the wind. That is, values of fetch less than this will not give meaningful results.

Field: maxFetch
Type: double
Purpose: Stores the maximum value of fetch in meters measured from the upwind limit of protection along the direction of the wind. Values greater than this do not change the resulting transports.

Field: bottomSlope
Type: double
Purpose: Stores the value of the bottom slope to be used when SNOWMAN generates new ground profiles for earthwork solution cases.

Field: backSlope
Type: double
Purpose: Stores the value of the back slope for the new ditch when SNOWMAN generates new ground profiles for earthwork solution cases.
2.3 snowmanLevelStandard

Struct: snowmanLevelStandard
Fields: 7 (described in the following)
Purpose: Defines the level standard. Level standards have a level name and/or a level number, a color, a weight, and either a style name or a style number.

Field: useLevelName
Type: int
Purpose: TRUE, indicates that the level name should be used. FALSE, indicates that the level number should be used.

Field: level
Type: char[80]
Purpose: Stores the level name.

Field: levelNumber
Type: int
Purpose: Stores the level number.

Field: color
Type: int
Purpose: Stores the color number.

Field: weight
Type: int
Purpose: Stores the weight number.

Field: useStyleName
Type: int
Purpose: TRUE, indicates that the style name should be used. FALSE, indicates that the style number should be used.

Field: styleName
Type: char[80]
Purpose: Stores the style name.

Field: styleNumber
Type: int
Purpose: Stores the style number.
2.4 NYSDOTCADDLevelStandard

Structure: NYSDOTCADDLevelStandard
Fields: 3 (described in the following)
Purpose: Defines a level standard for 3 specific types of elements. There can be existing and proposed level standards for the ditch, topo, and fence elements. However, due to changes in how the level standards are used there is no need to keep track of existing and proposed. This (the distinction between existing and proposed) is now legacy from the older version. SNOWMAN uses just the relevant standards and no longer has options for using existing or proposed. These element level standards could be removed and placed in SnowmanLevelStandards.

Field: DITCH
Type: SnowmanLevelStandard
Purpose: Stores the SnowmanLevelStand for ditch elements.

Field: TOPO
Type: SnowmanLevelStandard
Purpose: Stores the SnowmanLevelStandard for topographic elements (i.e. triangles)

Field: FENCE
Type: SnowmanLevelStandard
Purpose: Stores the SnowmanLevelStandard for snow fence elements.

2.5 snowmanLevelStandards

Structure: snowmanLevelStandards
Fields: 16 (described in the following)
Purpose: Defines the level standards for all the levels relevant to SNOWMAN. This includes both levels that are used to create topographic data for topographic input by design file as well as the levels used to output results generated by SNOWMAN.

Field: existing
Type: NYSDOTCADDLevelStandard
Purpose: Defines the existing level standards for snow fence, topographic elements (triangles), and ditch lines. This could be removed and ditch, fence, and topo could be imported as fields from NYSDOTCADDLevelStandard in this data structure.

Field: proposed
Type: NYSDOTCADDLevelStandard
Purpose: Defines the proposed level standards for snow fence, topographic elements (triangles), and ditch lines.
Field: ULOP
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the upwind limit of protection. Upwind limit of protection is used to determine the zero offset point in the profile when topographic data is input by design file. All distances for profile computations are based on the location of this point in the profile.

Field: DLOP
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the downwind limit of protection. Downwind limit of protection is used to determine where along the profile on the downwind side that the protection area stops.

Field: LOF
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the limit of fetch. Limit of Fetch is used in the input phase when topographic data is input by design file to determine the fetch distance. That is the distance from the upwind limit of protection to the limit of fetch in meters measured along the direction of the wind.

Field: axis
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the axis lines in the output. Used in the output to set the color, level, weight and style for outputting the axis lines for result plots.

Field: axisText
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the axisText. Used in the output to set the color, level, weight and style for outputting axis labels and numbers on the result plots.

Field: axisTick
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the tick marks on the axis. Used in the output to set the color, level, weight, and style for outputting the tick marks on the axis lines for results.

Field: axisHorizontalGrid
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the horizontal grid lines in the output. Used in the output to set the color, level, weight, and style for outputting horizontal grid lines on the result plots.

Field: axisVerticalGrid
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the vertical grid lines in the output. Used in the output to set the color, level, weight, and style for outputting horizontal grid lines on the result plots.
Field: text
Type: SnowmanLevelStandard
Purpose: Stores the level standard for the textual output. Used in the output to set the color, level, weight, and style for outputting any textual data in the result plots.

Field: ground
Type: SnowmanLevelStandard
Purpose: Stores the level standard for drawing the original unmodified ground profile (OG). Used in the output to set the level, color, weight and style for the lines representing the original ground profile in the result plots for both earthwork and structural snow fence solutions.

Field: newGround
Type: SnowmanLevelStandard
Purpose: Stores the level standard for drawing the modified ground profile. Used in the output to set the level, color, weight, and style for lines representing the new modified ground in the result plots for earthwork solutions.

Field: originalSnow
Type: SnowmanLevelStandard
Purpose: Stores the level standard for drawing the predicted unmitigated snow profile. Used in the output to set the level, color, weight, and style for the lines representing the unmitigated snow profile in the result plots for both earthwork and structural snow fence solutions.

Field: newSnow
Type: SnowmanLevelStandard
Purpose: Stores the level standard for drawing the predicted mitigated snow profile. Used in the output to set the level, color, weight and style for the lines representing the mitigated snow profile in the result plots for both earthwork and structural snow fence solutions.

Field: crossSection
Type: SnowmanLevelStandard
Purpose: Stores the level standard for drawing the sectioning lines. Used in the input phase to draw the lines that represent where the ground profiles were taken when topographic input is by design file.
2.6 snowmanClimateData

**Struct:** snowmanClimateData  
**Fields:** (described in the following)  
**Purpose:** Stores the climate data used by SNOWMAN.

Field: latitude  
Type: double  
Purpose: Stores the characteristic North Latitude of the area under investigation. It is used in computing climate information related to snow accumulation season.

Field: longitude  
Type: double  
Purpose: Stores the characteristic West Longitude of the area under investigation. It is used in computing the climate information related to the snow accumulation season.

Field: elevation  
Type: double  
Purpose: Stores the characteristic elevation in meters above sea level of the area under investigation. It is used in computing the climate information related to the snow accumulation season.

Field: St  
Type: double  
Purpose: Stores the total annual snowfall in millimeters of snow of the area under investigation. It is used in the computing the water equivalent snowfall over the accumulation season.

Field: Cr  
Type: double  
Purpose: Stores the relocation coefficient to be used for the area under investigation. This affects the transport value by affecting the percentage of snow that can be picked up and transported in the fetch area.

Field: k  
Type: double  
Purpose: Stores the exceedence factor used for the area under investigation. This affects the computation of the exceedence transport and provides the level of confidence in determining if drifting is likely to occur.

Field: direction  
Type: double  
Purpose: Stores the prevailing wind direction for the area under investigation. This is converted from the normal climate convention of describing the direction the wind comes from to describing the direction the wind blows to.
Field: ambientSnow
Type: double
Purpose: Stores the ambient snow cover in millimeters for the area under investigation. This is used in computing the effective height by subtracting the amount of ambient snow cover from the structural height of the fence when requested by the user (i.e. useAmbientSnow is TRUE).

Field: useAmbientSnow
Type: int
Purpose: Indicates if the ambient snow cover should be used in the effective snow fence height computation or not. [TRUE, FALSE]
Default value: FALSE (NYSDOT)

2.7 snowmanDataPoint
Struct: snowmanDataPoint
Fields: 2 (described in the following)
Purpose: Stores the offset and elevation of a data point for SNOWMAN.

Field: offset;
Type: double
Purpose: Stores the offset of a data point in a profile. The offset is the distance in meters from the upwind limit of protection measured along the direction of the wind. Negative values denote upwind and positive values denote downwind.

Field: elevation
Type: double
Purpose: Stores the elevation of the data point in a profile. The elevation is the height in meters above sea level.

2.8 snowmanProfile
Struct: snowmanProfile
Fields: 2 (described in the following)
Purpose: Stores a set of data points that constitute a profile. The profile can be either a ground (original, or modified), or a snow (either mitigated or unmitigated) profile.

Field: nDataPoints
Type: int
Purpose: Stores the number of data points in the array of SnowmanDataPoints pointed to by dPoint.
Field: dPoint
Type: array of SnowmanDataPoint
Purpose: Used to store the data points that comprise the profile. Each data point is defined by an offset and elevation pair.

2.9 snowmanCrossSection
Struct: snowmanCrossSection
Fields: 17 (described in the following)
Purpose: Stores all the cross section information pertaining to a single cross section.

Field: profile
Type: SnowmanProfile
Purpose: Stores the original (unmitigated) ground profile (O.G.)

Field: hasULOP
Type: int
Purpose: Indicates if an intersection of the sectioning line and the upwind limit of protection was found for topographic input by design file. TRUE indicates it was found correctly and FALSE indicates it was either not found or multiple ones were found.

Field: hasUDOP
Type: int
Purpose: Indicates if an intersection of the sectioning line and the downwind limit of protection was found for topographic input by design file. TRUE indicates it was found correctly and FALSE indicates it was either not found or multiple ones were found.

Field: hasDitch
Type: int
Purpose: Indicates if an intersection of the sectioning line and a ditch line was found. FALSE indicates that no intersection was found. TRUE indicates that the lowest one was found.

Field: LOCACCM
Type: SnowmanDataPoint
Purpose: Stores the offset in meters measured along the direction of the wind of the 7 meter perpendicular to the road setback required for storage of plow cast. This is computed based on the angle of the wind to the road.

Field: ULOP
Type: SnowmanDataPoint
Purpose: unused. Legacy from older version of SNOWMAN where the reference was the center line and the upwind and downwind limits were the pavement edges.
Field: DLOP
Type: SnowmanDataPoint
Purpose: Stores the location of the downwind limit of protection. This is a data point along the profile that indicates the limit on the downwind side of the profile where the protection stops.

Field: ditch
Type: SnowmanDataPoint
Purpose: Stores the location of the ditch point. This is a point on the profile that indicates where the ditch to be modified is located. SNOWMAN uses the lowest ditch point as the one to modify when multiple ditch lines are present.

Field: fetch
Type: double
Purpose: This is the distance in meters along the profile from the upwind limit of protection to the limit of fetch for the particular cross section.

Field: angleToRoad
Type: double
Purpose: This stores the angle in degrees between the road and wind. It is the minimum angle in the range [0.0, 90.0].

Field: validForSnowfence
Type: int
Purpose: Indicates that all the required information is present for SNOWMAN to evaluate and design structural snow fence mitigation measures.

Field: validForEarthwork
Type: int
Purpose: Indicates that all the required information is present for SNOWMAN to evaluate and design earthwork solutions.

Field: x
Type: double
Purpose: Stores the x-coordinate of the intersection of the cross sectioning line and the upwind limit of protection line when topographic input comes from design file (DGN).

Field: y
Type: double
Purpose: Stores the y-coordinate of the intersection of the cross sectioning line and the upwind limit of protection line when topographic input comes from design file (DGN).

Field: z
Type: double
Purpose: Stores the z-coordinate of the intersection of the cross sectioning line and the upwind limit of protection line when topographic input comes from design file (DGN).
Field: Q50
Type: double
Purpose: Stores the value of the average transport for this section based on this section’s fetch and the climate data for the area under investigation.
Field: Q20
Type: double
Purpose: Stores the value of the exceedence transport for this section based on this section’s fetch and the climate data for the area under investigation (including the k factor).

2.10 snowmanTopoData
Struct: snowmanTopoData
Fields: 2 (described in the following)
Purpose: Stores all the individual cross section information for the area under investigation.

Field: nSections
Type: int
Purpose: Stores the number of cross sections in the field section. This is the total set of cross sections for the entire area under investigation.

Field: section
Type: array of SnowmanCrossSection
Purpose: Stores the individual cross sections for the entire area under investigation.

2.11 snowmanFence
Struct: snowmanFence
Fields: 8 (described in the following)
Purpose: Stores the information about a single fence.

Field: height
Type: int
Purpose: Indicates which fence height the fence is referring to. [HEIGHT_1_4_METERS, HEIGHT_1_8_METERS, HEIGHT_2_0_METERS, HEIGHT_2_1_METERS, HEIGHT_2_4_METERS, HEIGHT_2_7_METERS, HEIGHT_3_0_METERS, HEIGHT_3_6_METERS]

Field: actualHeight
Type: double
Purpose: Stores the actual height of the fence in meters.
Field: effectiveHeight  
Type: double  
Purpose: Stores the effective height of the fence. This is computed by subtracting the portion of the fence that is considered buried from the actual height of the fence.

Field: setback  
Type: double  
Purpose: Stores the offset in meters from the upwind limit of protection to the location of the fence along the direction of the wind.

Field: porosity  
Type: int  
Purpose: Indicates the porosity of the fence selected. [POROSITY_50_PERCENT, POROSITY_37.5_PERCENT, POROSITY_25_PERCENT, POROSITY_0_PERCENT]

Field: bottomGap  
Type: double  
Purpose: unused. Legacy from an older version of snowman

Field: isBuried  
Type: int  
Purpose: Indicates if the fence is considered buried or not. TRUE indicates that the fence is buried by more than the allowable. This may be acceptable in certain cases but needs to be tracked to warn users.

Field: location  
Type: SnowmanDataPoint  
Purpose: The offset and elevation of the location of the fence along the profile. The elevation of this point is the elevation of the base of the structural snow fence.

2.12 snowmanDitch

Struct: snowmanDitch
Fields: 6 (described in the following)
Purpose: Stores the information about the ditch if present.

Field: backSlope  
Type: double  
Purpose: Stores the slope of the back slope. The back slope is the slope measured from the back of the bottom of the ditch to the top of the cut on the upwind side of the ditch.

Field: bottomSlope  
Type: double
Purpose: Stores the slope of the bottom of the ditch. The bottom slope is the slope measured from the most upwind point at the bottom of the ditch to the most downwind point at the bottom of the ditch.

Field: topOfBackSlope
Type: SnowmanDataPoint
Purpose: Stores the location of the top of the back slope. That is the point where the new ditch intersects the remaining ground profile.
Field: bottomOfBackSlope
Type: SnowmanDataPoint
Purpose: Stores the location of the point where the ditch bottom ends and the upwind cut section of the ditch begins.

Field: volumeOfEarthwork
Type: double
Purpose: Stores the volume of earthwork to be removed as a result of the new ditch. This is measured in cubic meters.

Field: additionalDitchWidth
Type: double
Purpose: Stores the horizontal distance that the width of the bottom of the ditch has been expanded by. This is measured in meters.

2.13 snowmanResult
Struct: snowmanResult
Fields: 27 (described in the following)
Purpose: Stores the result of a single case for a single section.

Field: storesQaverage
Type: int
Purpose: Indicates if the average design transport was stored upwind of the limit of storage (LOCACCM). TRUE indicates it was stored upwind of the limit of storage. FALSE indicates it was not stored upwind of the limit of storage.

Field: storesQexceedence
Type: int
Purpose: Indicates if the exceedence transport was stored upwind of the upwind limit of protection. TRUE indicates it was stored upwind of the upwind limit of storage. FALSE indicates it was not stored upwind of the upwind limit of storage.

Field: locQavg
Type: SnowmanDataPoint
Purpose: Stores the location where the average transport is stored.

Field: locQexc
Type: SnowmanDataPoint
Purpose: Stores the location where the exceedence transport is stored.

Field: hasProfile
Type: int
Purpose: Indicates profile contains a valid snow profile. TRUE indicates it does have a valid snow profile. FALSE indicates it does not have a valid snow profile.

Field: profile
Type: SnowmanProfile
Purpose: Stores the snow profile for single section and a single case. This can be an unmitigated or mitigated snow profile.

Field: depth
Type: SnowmanProfile
Purpose: Stores the snow depth profile for a single section and a single case. This can be an unmitigated or mitigated snow depth profile.

Field: hasNewGround
Type: int
Purpose: Indicates if newGround contains a valid ground profile. TRUE indicates it does have a valid ground profile. FALSE indicates it does not have a valid ground profile.

Field: newGround
Type: SnowmanCrossSection
Purpose: Stores the new ground profile for a single section and a single case.

Field: hasFence
Type: int
Purpose: Indicates if fence contains a valid fence. TRUE indicates fence is valid. FALSE indicates fence is not valid.

Field: fence
Type: SnowmanFence
Purpose: Stores the fence information for a single case and section.

Field: ditch
Type: SnowmanDitch
Purpose: Stores the ditch information for a single case and section.
Field: fenceType
Type: int
Purpose: Indicates the type of fence(s) that can be used. [PERMANENT_ONLY, PORTABLE_ONLY, TEMPORARY_ONLY, PERMANENT_OR_PORTABLE]

Field: maxFenceOffset
Type: double
Purpose: Indicates the maximum setback in meters along the direction of the wind from the upwind limit of protection where a fence can be placed.

Field: maxEarthworkOffset
Type: double
Purpose: Indicates the maximum setback in meters along the direction of the wind from the upwind limit of protection where the ground can be modified.

Field: allowBuriedFence
Type: int
Purpose: Indicates if a fence is allowed to be buried by more than the buriedFenceRatio.

Field: buriedFenceRatio
Type: double
Purpose: Stores the ratio of the fence that can be buried by snow and still considered unburied.

Field: useAmbientSnow
Type: int
Purpose: Indicates if the ambient snow depth should be used in the computation of the effective fence height and subsequent check of its buried depth ratio.

Field: storageAtLOCACCMM
Type: double
Purpose: Stores the total storage from the lesser of the limit of fetch or the upwind limit of terrain to the limit of storage in kilograms per meter.

Field: storageAtULOP
Type: double
Purpose: Stores the total storage from the lesser of the limit of fetch or the upwind limit of terrain to the upwind limit of protection in kilograms per meter.

Field: actualSnowDepthAtULOP
Type: double
Purpose: Stores the predicted snow depth at the upwind limit of protection. If the exceedence transport is stored upwind of the upwind limit of protection then this will be zero. Otherwise, this is the snow depth of the snow drift profile at the upwind limit of protection.
Field: actualSnowDepthAtDLOP
Type: double
Purpose: Stores the predicted snow depth at the downwind limit of protection. If the exceedence transport is stored upwind of the upwind limit of protection then this will be zero. Otherwise, this is the snow depth of the snow drift profile at the downwind limit of protection.

Field: unlimitedSnowDepthAtULOP
Type: double
Purpose: Stores the depth of snow at the upwind limit of protection based on the depth of the snow profile at the upwind limit of protection.

Field: unlimitedSnowDepthAtDLOP
Type: double
Purpose: Stores the depth of snow at the downwind limit of protection based on the depth of the snow profile at the downwind limit of protection.

Field: hasDriftingSnowProblem
Type: int
Purpose: Indicates if this section for this case has a drifting snow problem. A drifting snow problem is defined as one where Qexceedence is not stored upwind of the upwind limit of protection and the drift depth at either the upwind limit of protection or the downwind limit of protection is greater than the allowableSoR. TRUE indicates that a drifting snow problem exists. FALSE indicates drifting snow problem does not exist.

Field: hasBlowingSnowProblem
Type: int
Purpose: Indicates if this section for this case has a blowing snow problem. A blowing snow problem is defined as one where the average transport is not stored upwind of the upwind limit of storage. TRUE indicates that a blowing snow problem exists. FALSE indicates that a blowing snow problem does not exist.

Field: needsMitigation
Type: int
Purpose: Indicates if this section for this case has either a blowing or drifting snow problem. TRUE if either hasDriftingSnowProblem or hasBlowingSnowProblem is TRUE. FALSE otherwise.

2.14 snowmanCaseResult
Struct: snowmanCaseResult
Fields: 2 (described in the following)
Purpose: Stores all the results of a single case for a single section.
Field: section  
Type: pointer to SnowmanResult  
Purpose: Stores the results of all the sections for a single case. This is used by every case. For cases 2 and 4 this stores the shortest fence solution.

Field: section2  
Type: pointer to SnowmanResult  
Purpose: Stores the results of all sections for the closest fence solutions in cases 2 and 4.

2.15 snowmanResultsData

Struct: snowmanResultsData
Fields: 2 (described in the following)
Purpose: Stores the whole set of solutions for each case and cross section.

Field: maxOffset  
Type: double  
Purpose: Used to store the largest downwind offset from all sections so that all sections can be plotted on the same size graph.

Field: minOffset  
Type: double  
Purpose: Used to store the largest upwind offset from all sections so that all sections can be plotted on the same size graph.

Field: maxElevation  
Type: double  
Purpose: Used to store the largest elevation from all sections so that all sections can be plotted on the same size graph.

Field: minElevation  
Type: double  
Purpose: Used to store the smallest elevation from all sections so that all sections can be plotted on the same size graph.

Field: nCases  
Type: int  
Purpose: Stores the number of cases in this result set.

Field: caseList  
Type: array of int  
Purpose: Stores the case number in this result set.
Field: nSections
Type: int
Purpose: Stores the number of sections in this result set.

Field: caseNumber
Type: SnowmanCaseResult[12]
Purpose: Stores the results for all cases.

**2.16 snowmanFenceConstants**

**Struct:** snowmanFenceConstants

**Fields:** 15 (described in the following)

**Purpose:** Stores the fence constants used in the Tabler polynomials for profile generations

Field: Au
Type: double
Purpose: Stores the zero order constant in polynomial for predicting profiles upwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Bu
Type: double
Purpose: Stores the first order constant in polynomial for predicting profiles upwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Cu
Type: double
Purpose: Stores the second order constant in polynomial for predicting profiles upwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Du
Type: double
Purpose: Stores the third order constant in polynomial for predicting profiles upwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Eu
Type: double
Purpose: Stores the fourth order constant in polynomial for predicting profiles upwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Fu
Type: double
Purpose: Stores the fifth order constant in polynomial for predicting profiles upwind of the fence.
Value is based on porosity of the fence. See Tabler (2003) for details.

Field: dsLimitu
Type: double
Purposes: Stores the limit in meters upwind from the fence along the approach slope where the upwind equation is valid. The polynomial is valid from this point up to the fence. The drift profile upwind of this point is the same as the unmitigated drift profile. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Al
Type: double
Purpose: Stores the zero order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Bl
Type: double
Purpose: Stores the first order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Cl
Type: double
Purpose: Stores the second order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Dl
Type: double
Purpose: Stores the third order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: El
Type: double
Purpose: Stores the fourth order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: Fl
Type: double
Purpose: Stores the fifth order constant in polynomial for predicting profiles downwind of the fence. Value is based on porosity of the fence. See Tabler (2003) for details.

Field: dsLimitl
Type: double
Purpose: Stores the limit in meters downwind from the fence where the downwind polynomial for predicting drift profiles is valid. The polynomial works to predict drift from the fence to this point.
and uses the unmitigated drift prediction from then on.

### 2.17 snowmanElements

**Struct:** SnowmanElements  
**Fields:** 10 (described in the following)  
**Purpose:** Stores the handles to the design elements of interest to SNOWMAN

- **Field:** nFETCH  
  **Type:** int  
  **Purpose:** Stores the number of elements in the array FETCH. These are the elements that define the limit of fetch for the area under investigation.

- **Field:** nULOP  
  **Type:** int  
  **Purpose:** Stores the number of elements in the array ULOP. These are the elements that define the upwind limit of protection for the area under investigation.

- **Field:** nDLOP  
  **Type:** int  
  **Purpose:** Stores the number of elements in the array DLOP. These are the elements that define the downwind limit of protection for the area under investigation.

- **Field:** nDD  
  **Type:** int  
  **Purpose:** Stores the number of elements in the array DD. These are the elements that define the locations of the drainage ditches for the area under investigation.

- **Field:** nTOPO  
  **Type:** int  
  **Purpose:** Stores the number of elements in the array TOPO. These are the elements that define the surface of the area under investigation (i.e. the triangles).

- **Field:** FETCH  
  **Type:** array of ULong  
  **Purpose:** Stores the handles to the elements that constitute the limit of fetch.

- **Field:** ULOP  
  **Type:** array of ULong  
  **Purpose:** Stores the handles to the elements that constitute the upwind limit of protection.

- **Field:** DLOP  
  **Type:** array of ULong
Purpose: Stores the handles to the elements that constitute the downwind limit of protection.

Field: DD
Type: array of ULong
Purpose: Stores the handles to the elements that constitute the drainage ditches.

Field: TOPO
Type: array of ULong
Purpose: Stores the handles to the elements that constitute the topographic surface.
3 Module Definitions

3.1 SnowmanEvaluation
Module: SnowmanEvaluation
Purpose: Contains the set of functions that generates snow profiles and computes relevant
information for each section. The profiles and accompanying data are used to determine when/if
an acceptable solution is found. An acceptable solution is one that stores the average transport
upwind of the limit of storage and either has drifting less than the acceptable at the limits of
protection or stores the exceedence transport upwind of the upwind limit of protection. Each case
uses the same criteria for determining an acceptable solution. The module has functions to
evaluate unmitigated terrain and terrain that contains a single structural snow fence. The module
also includes functions to determine single structural snow fences that result in acceptable design.
Those functions include user constraints on types of fences, fence heights, porosity, and setbacks.
This module also has functions that generate earthwork solutions. Earthwork solutions are ones
where the profile has an upwind ditch that can be modified to store the required snow. In those
cases the existing ditch is expanded while keeping the low point of the ditch fixed so no changes
are made to existing drainage patterns.
3.1.1 External Function Definitions

External Functions for SnowmanEvaluation
This is the set of functions that other modules can call to perform evaluation or design of structural snow fences and earthwork modifications.

Function: case1
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
data inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: findElevation, malloc, nextUnmitigatedDriftElevation,
                  computeStorageAndDepthAndCheckForProblem
Description: Generates the snow profile and the drift depth profile for the unmitigated terrain.
             Determines the location of the limit of storage
             Determines the snow depths at the upwind and downwind limits of protection.
             Determines if the site had a blowing and/or drifting snow problem (accounts for allowable transport across road)
Called by: generateResults in Snowman module
Pseudo code:
  initialize current section
  has no profile or fence
  determine distance and memory needed for section
  set first data point to be the same as the first ground point
  set current Offset to be just downwind of the first ground point
  while the current offset is less than 45 m upwind from the most downwind point
     compute and record new data point for the snow profile and drift depth
     increment the current offset by inc
     compute location of the limit of storage
     compute the storages
     determine if problem(s) exist

Function: case2
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
data inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: findElevation, malloc, nextUnmitigatedDriftElevation,
                  computeStorageAndDepthAndCheckForProblem, shortestFence,
                  nextTallerFence, actualHeight, closestSetback, effectiveHeight, isBuried,
nextSetback, mitigatedProfile

Description: Generates the two snow profiles and the drift depth profiles for snow fences (closest fence and shortest fence).

Uses the restriction based on the fence type the user picked.

Determines the shortest fence solution then the closest fence solution.

Takes into account the further restrictions: use ambient snow cover and allow buried fences

Called by: generateResults in Snowman module

Pseudo code:

for each section
    if section needs mitigation
        determine limit for fence offset (min of terrain or max allowable fence offset)
        determine number of fences possible for fence types allowed
        start with the shortest fence then select each additionally taller fence based on fence types allowed.
        foreach fence that is allowed
            determine height, actual height, effective height, if buried
            set porosity to 50% and set fence location
            check if starting location is valid
            if not then
                fence's setback is increased until it runs out of room or a suitable location is found
            if we have a suitable location then
                get the profile for that fence
                check for problems (blowing and/or drifting)
                while an acceptable solution is not found and there is room to move the fence back
                    increase the setback and check again
                find shortest and closest solutions
            else section doesn't need mitigation
                do nothing

Function: case3

Version: 1.0

Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
       double inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values

Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case

Return: void

Functions called: findElevation, malloc, nextUnmitigatedDriftElevation, computeStorageAndDepthAndCheckForProblem, shortestFence, nextTallerFence, actualHeight, closestSetback, effectiveHeight, isBuried, nextSetback, mitigatedProfile

Description: generates a snow profile and drift depth profile for the fence height and porosity that was selected by the user (determines the setback for the fence).

determines the setback location for the fence the user specified.

Called by: generateResults in Snowman module
Pseudo code:

```plaintext
def function_name:
    foreach section
        if section needs mitigation then
            set limit for maximum allowable fence offset (specified by user).
            determine number of fences possible for fence types allowed
            start with the shortest fence then select each additionally taller fence based
            on fence types allowed.
```

Function: case4
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
       double inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: findElevation, malloc, nextUnmitigatedDriftElevation,
                  computeStorageAndDepthAndCheckForProblem, shortestFence,
                  nextTallerFence, actualHeight, closestSetback, effectiveHeight, isBuried,
                  nextSetback, mitigatedProfile
Description: generates the two snow profiles and the drift depth profiles for snow fences (closest
             fence and shortest fence). Uses the restriction based on the fence type and
             maximum allowable setback that the user specified. Determines the shortest fence
             solution and the closest fence solution. Takes into account the further restrictions,
             use ambient snow cover and allow buried fences
Called by: generateResults in Snowman module
foreach fence that is allowed
determine height, actual height, effective height, if buried
set porosity to 50% and set fence location
check if starting location is valid, if not then the fence's setback is
increased until it runs out of room or a suitable location is found
if we have a suitable location then
  get the profile for that fence
  check for problems (blowing and/or drifting)
  while an acceptable solution is not found and we have room to and can
  move the fence back
  increase the setback and check again
find shortest and closest solutions
else section doesn't need mitigation
  do nothing

Function: case5
Version: 1.0
Input: 
  SnowmanTopoData *stdP, pointer to the topographic data
  SnowmanClimiteData *scdP, pointer to the climate data
  double inc, increment to step by when generating profiles
  SnowmanDefaults *sdP, pointer to the defaults values
Output: 
  SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: findElevation, malloc, nextUnmitigatedDriftElevation,
  computeStorageAndDepthAndCheckForProblem, shortestFence,
  nextTallerFence, actualHeight, closestSetback, effectiveHeight, isBuried,
  nextSetback, mitigatedProfile
Description: Generates a solution for the snow profile and drift profile for a specified fence with
  a given maximum allowable setback. Uses the restriction based on the fence height,
  porosity and maximum allowable setback that the user specified. Takes into account
  the further restrictions, use ambient snow cover and allow buried fences
Called by: generateResults in Snowman module
Pseudo code:
  foreach section
    if section needs mitigation then
      set limit for maximum allowable fence offset (specified by user).
      determine height, actual height, effective height, if buried
      set porosity to 50% and set fence location
      check if starting location is valid, if not then the fence's setback is
      increased until it runs out of room or a suitable location is found
      if we have a suitable location then
        get the profile for that fence
        check for problems (blowing and/or drifting)
        while an acceptable solution is not found and we have room to and can move
        the fence back
        increase the setback and check again
      else section doesn't need mitigation so
        do nothing
Function: case6
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
       double inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: findElevation, malloc, nextUnmitigatedDriftElevation,
                   computeStorageAndDepthAndCheckForProblem, shortestFence,
                   nextTallerFence, actualHeight, closestSetback, effectiveHeight, isBuried,
                   nextSetback, mitigatedProfile
Description: Generates a snow profile and a drift profile for the fence specified by the user
             (height, setback, and porosity).
Called by: generateResults in Snowman module
Pseudo code:
    foreach section
        determine effective height and if buried
        set location
        generate profile
        compute storage and check for problem

Function: case7
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
       double inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: malloc, createNewGround, nextUnmitigatedDriftElevation,
                   computeStorageAndDepthAndCheckForProblem
Description: Generates a new ground profile that doesn't have any blowing or drifting snow
             problems as well as the snow and drift profiles for that new ground
Called By: generateResults in Snowman module
Pseudo code:
    foreach section
        initialize memory for solutions
        if section needs mitigation then
            if section has ditch
                start with a 5m increment in the width of the bottom of the ditch measured
                upwind from the ditch point (lowest upwind ditch point)
                while we can increase the ditch width and we don't have a solution
                    create a new ground profile with the additional ditch width
                if we created new ground
                    generate profile and check for problems
            if problem(s) still exist then
increase ditch width
else we found a solution
nothing more to do
else
increase ditch width
else there is no ditch to modify so
do nothing
else section doesn't need mitigation so
do nothing

Function: case8
Version: 1.0
Input: SnowmanTopoData *stdP, pointer to the topographic data
       SnowmanClimiteData *scdP, pointer to the climate data
double inc, increment to step by when generating profiles
       SnowmanDefaults *sdP, pointer to the defaults values
Output: SnowmanResultData *srdP, pointer to the results data to put the results from this case
Return: void
Functions called: malloc, createNewGround, nextUnmitigatedDriftElevation,
                 computeStorageAndDepthAndCheckForProblem
Description: generates a new ground profile that doesn't have any blowing or drifting snow
problems as well as the snow and drift profiles for that new ground but only allows
for earthwork up to the maximum allowable offset for earthwork specified by the
user. That is, the new ground is modified only up to the offset the user specified
and no further upwind.
Called By: generateResults in Snowman module
Pseudo code:
foreach section
  initialize memory for solutions
  if section needs mitigation then
    if section has ditch
      start with a 5m increment in the width of the bottom of the ditch measured
      upwind from the ditch point (lowest upwind ditch point)
      while we can increase the ditch width and we don't have a solution
        (constrained by maximum allowable offset for earthwork)
        create a new ground profile with the additional ditch width
        if we created new ground
          generate profile and check for problems
          if problem(s) still exist then
            increase ditch width
          else we found a solution
            nothing more to do
        else
          increase ditch width
      else there is no ditch to modify so
      do nothing
    else section doesn't need mitigation so
    do nothing
  else
    increase ditch width
  else we found a solution
nothing more to do

3.1.2 Internal Function Definitions

Internal Functions for SnowmanEvaluation module

These are the set of internal functions that do the work. These are functions that serve a very simple purpose and are used repeatedly by the external functions. That is, these functions are common to many of the external functions and have been defined explicitly to avoid duplicating code in the external functions. It also allows for the easy modification to procedures detailed by Tabler on evaluating sites and prescribing mitigation measures.

Function: createNewGround
Version: 1.0
Input: SnowmanCrossSection *scsP, pointer to the original ground profile cross section
double additionalDitchWidth, amount to increase the ditch width to
double limit, offset limit for earthwork modification. That is the point where the profile cannot be modified any further upwind.
Output: SnowmanResultData *srdP, pointer to the results data with the new ground profile
Return: void
Functions called: findElevation, malloc
Description: Creates a new ground profile with a ditch that has a bottom width specified by ditchWidth. The bottom and back slopes that are used come from those specified in srdP
Called by: case7 and case8
Pseudo code:

determine the limit of earthwork modification
set the bottom of back slope location (bottomOfCut)
make the top of back slope location the limit and check elevation
if the elevation at the limit is less than the ground then
we can't possibly create a cut that intersects the original ground so no new ground profile is possible
find the most upwind intersection of the back slope to the original ground and make that the topOfCut
if we found the intersection before the location of the bottomOfCut then
create the new ground profile
copy the original ground from the most upwind point to the point of the topOfCut
add points for the topOfCut and bottomOfCut
copy the original ground from the bottomOfCut to the most downwind point
else we can't create a new ground profile (using the current back and bottom slope no new ground is possible)

Function: closestSetback
Version: 1.0
Input: double height, actual fence height
double minOffset, minimum allowable fence setback
Output: double, height or minOffset, which ever is smaller
Return: double, the smaller of 15*height and minFenceOffset
Functions called: none
Description: returns the minimum setback for a given fence of height height
Called by: case2, case3, case4, case5
Pseudo code:

```
return the smaller of minOffset and 15*height
as per Ron Tabler fence cannot be placed closer than 15H (15*height of fence)
as per Darrell Kaminski fences are not to be placed closer than minFenceOffset.
That is the NYSDOT may want to keep fences further than 15H from ULOP
```

Function: effectiveHeight
Version: 1.0
Input: double height, actual fence height
double setback, offset of the fence
SnowmanProfile *originalSnow, unmitigated snow profile
int useAmbientSnow, TRUE or FALSE, indicates if ambientSnow is to be used in the computation
double ambientSnow, >0, value in meters of the ambient snow cover.
Output: none
Return: double, > 0.0, the effective height of the fence (that portion not buried by snow)
Functions: findElevation
Description: returns the effective height of the fence with consideration of the ambient snow and depth of snow from the unmitigated snow profile at the fence location
Called by: case2, case3, case4, case5, case6
Pseudo code:

```
find the elevation of snow at the fence location
if useAmbientSnow then
    compute effective height as actual height - snowdepth at fence location - ambient snow
else don't use ambient snow in computation so
    compute effective height as actual height - snowdepth at fence location
if the effective height should be less than 0.0
    return 0.0
else
    return effectiveHeight
```

Function: isBuried
Version: 1.0
Input: double effectiveHeight, the effective height of the fence
double actualHeight, the structural height of the fence
double buriedRatio, the ratio at which the fence is considered buried
Output: none
Return: int, TRUE or FALSE, indicates if the fence is considered buried or not
Functions called: none
Description: returns TRUE if the fence is buried and returns FALSE if not
Called by: case2, case3, case4, case5, case6
Pseudo code:

```plaintext
if effectiveHeight is < actualHeight - actualHeight*buriedRation then the fence is buried
    return TRUE
else fence is not buried
    return FALSE
```

Function: nextSetback
Version: 1.0
Input: SnowmanFence *fence, pointer to the fence structure
double maxFenceOffset, the maximum setback for the fence
int allowBuriedFence, TRUE or FALSE, indicates if the fence is allowed to be buried more than buriedFenceRatio
double buriedFenceRatio, [0.0, 1.0], ratio of fence that is allowed to be buried and fence is considered not ot be buried.
int useAmbientSnow, TRUE or FALSE, indicates if ambient snow is to be used in the effective height computation
double ambientSnow, > 0.0 meters, ambient snow cover in meters
SnowmanProfile *originalSnowDepth, pointer to the unmitigated snow profile
SnowmanProfile *originalGround, pointer to the original ground profile
Output: SnowmanFence *fence, pointer to the fence structure
Return:int, TRUE or FALSE, indicates if a new setback was found that meets the requirements specified by allow buried fence or not and use ambient snow or not.
Functions called: effectiveHeight, isBuried, findElevation
Description: returns TRUE and updates *fence if a new setback was found, otherwise return FALSE
Called By: case2, case3, case4, case5
Pseudo code:

```plaintext
increase the setback by 1 meter
compute new effective height and if the fence is buried or not
while the fence setback is less than the limit (maxFenceOffset) and (the fence is currently buried and buried fences are not allowed)
    increase the setback by 1 meter
    compute new effective height and if the fence is buried or not
if we couldn't find a new suitable setback then
    return FALSE
else we found a new setback that is acceptable
    update the fence structure
    return TRUE
```

Function: mitigatedProfile
Version: 1.0
Input: SnowmanProfile *topo, pointer to the original ground profile
SnowmanProfile *originalSnow, pointer to the unmitigated snow profile
SnowmanProfile *originalDepth, pointer to the unmitigated snow depth profile
double inc, > 0.0, increment between data points to be used in generating profiles
double *maxElev, pointer to the maxElevation seen so far
SnowmanFence *fence, pointer to the fence to use in generating the snow profile
SnowmanDefaults *sdP, pointer to the defaults data that is used
SnowmanClimateData *scdP, pointer to the climate data to use for this profile generation
(currently unused)
Output: SnowmanProfile *newSnow, pointer to the mitigated snow profile for the given fence
        SnowmanProfile *newDepth, pointer to the mitigated snow depth profile for the given fence
double *maxElev, pointer to the maximum elevation seen so far, updated to include possible changes from this profile
Return: void
Functions called: setFenceConstants, findElevation, malloc, cos
Description: generates a snow profile and drift depth profile for the specified fence and ground.
See tabler report for full description of procedure for generating these profiles.
Called by: case2, case3, case4, case5, case6
Pseudo code:
    see Tabler

Function: nextUnmitigatedDriftElevation
Version: 1.0
Input: SnowmanProfile *depth, pointer to the depth profile upto offset.
        SnowmanProfile *snow, pointer to the unmitigated snow profile upto offset.
        SnowmanProfile *topo, pointer to the original ground profile.
        double offset, current point that we are computing the snow profile (snow) and drift depth for (depth).
        double inc, the distance from the last point in the profiles to the current (offset).
        double *maxElev, pointer to the maxElevation seen so far
Output: SnowmanProfile *snow, pointer to the unmitigated snow profile with the addition of the snow elevation at offset.
        SnowmanProfile *depth, pointer to the unmitigated snow depth profile with the addition of the new snow depth at offset.
        double *maxElev, pointer to the maximum elevation seen so far, updated to include possible changes from this profile
Return: void
Functions called: findElevation
Description: computes the snow depth and elevation at the current point (offset).
Called by: case1, case7, case8, mitigatedProfile
Pseudo code:
    see Tabler (2003)

Function: computeStorageAndDepthAndCheckForProblem
Version: 1.0
Input: SnowmanCrossSection *scsP, pointer to the snow profile
       SnowmanResult *srP, pointer to the result data for a section.
       SnowmanDefaults *sdP, pointer to the defaults data.
Output: SnowmanResult *srP, pointer to the result data for a section.
Return: void
Functions called: findElevation
Description: computes the storages contained in srP, storage at limit of storage
              (storageAtLOCACCM), and storage at upwind limit of protection
              (storageAtULOP).
              computes the location (locQavg) at which the average transport (Q50)
              is stored and the location (locQexc) at which the exceedence
              transport (Q20) is stored.
              computes the depth of drift (predicted and unlimited) at the upwind
              and downwind limits of protection.
Called by: case1, case2, case3, case4, case5, case6, case7, case8
Pseudo code:

foreach data point in the depth profile
  if the current data point is in the storage range (i.e. upwind of the ULOP and
downwind of the fetch barrier) then
    if the current data point is in the storage range of just storageAtULOP
      compute the average depth for the current point and the previous point
      compute the average density over this increment
      compute the density of the snow stored upto the current point
      if the density is > 0.0
        update storage for storageAtULOP
        check if this is where the average and/or exceedence transports are
        stored and update accordingly
    else the current data point is in the storage range of both storageAtULOP and
      StorageAtLOCACCM then
      compute the average depth for the current point and the previous point
      compute the average density over this increment
      compute the density of the snow stored upto the current point
      if the density is > 0.0
        update storage for storageAtULOP and storageAtLOCACCM
        check if this is where the average and/or exceedence transports are
        stored and update accordingly
    else we are past the point where storage matters (i.e. downwind of the ULOP)
      break out of loop
  determine the depths at the limits of protection and if the site has problems
  get unlimited drift depths at ULOP and DLOP
  if storage at the upwind limit of protection (storageAtULOP) is less than the
  exceedence transport then
    drift at the ULOP and DLOP will be the same as the unlimited drift depths
  else there is no drifting to compute so
    set predicted drift at ULOP and DLOP equal to 0.0
  if storage at the upwind limit of storage (storageAtLOCACCM) is > the average
  transport (Q50) then
    indicate this section does not have a blowing snow problem
  else the average transport is not stored upwind of the limit of storage so
    indicate the section has a blowing snow problem.
  if storage at ULOP (storageAtULOP) is greater than the exceedence transport (Q20)
    indicate the section does not have a drifting problem
  else the site may have a drifting problem
    if the actual drift depths at either the ULOP or DLOP exceed the allowed drift at
that location then
    indicate the section has a drifting snow problem
else the actual drift depths are less than the allowable so
    indicate the section does not have a drifting problem
if the site has either a drifting or blowing snow problem then
    indicate the section needs further mitigation
else there are no problems detected
    indicate the section does not need further mitigation

Function: setFenceConstants
Version: 1.0
Input: SnowmanFence *sfP, pointer to the fence struct
Output: SnowmanFenceConstants *sfc, pointer to the snowman fence constants struct
Return: void
Functions called: none
Description: fills in the fence constants structure with the relevant values for the input porosity
             from sfP
Called by: mitigatedProfile
Pseudo code:
if sfP->porosity is POROSITY_50_PERCENT
  set Au, Bu, Cu, Du, Eu, Fu, dsLimitu, Al, Bl, Cl, Dl, El, Fl, dsLimitl, and eta
else if sfP->porosity is POROSITY_37_5_PERCENT
  set Au, Bu, Cu, Du, Eu, Fu, dsLimitu, Al, Bl, Cl, Dl, El, Fl, dsLimitl, and eta
else if sfP->porosity is POROSITY_25_PERCENT
  set Au, Bu, Cu, Du, Eu, Fu, dsLimitu, Al, Bl, Cl, Dl, El, Fl, dsLimitl, and eta
else if sfP->porosity is POROSITY_0_PERCENT
  set Au, Bu, Cu, Du, Eu, Fu, dsLimitu, Al, Bl, Cl, Dl, El, Fl, dsLimitl, and eta
else
  return error

Function: shortestFence
Version: 1.0
Input: int fenceType, [PERMANENT_ONLY, PORTABLE_ONLY, TEMPORARY_ONLY,
            PERMANENT_OR_PORTABLE], indicates the type of fences that are allowed
Output: none
Return: int, [HEIGHT_1_8_METERS, HEIGHT_2_0_METERS, HEIGHT_1_4_METERS,
              FENCE_HEIGHT_ERROR] the shortest fence indicator
Functions called: none
Description: returns the indicator to the shortest fence based on the type(s) of fences allowed.
Called by: case2, case4
Pseudo code:
if fenceType is PERMANENT_ONLY
  return HEIGHT_1_8_METERS
else if fenceType is PORTABLE_ONLY
  return HEIGHT_2_0_METERS
else if fenceType is TEMPORARY_ONLY
  return HEIGHT_1_4_METERS
else if fenceType is PERMANENT_OR_PORTABLE
  return HEIGHT_1_8_METERS
else
    return FENCE_HEIGHT_ERROR

Function: nextTallerFence
Version: 1.0
Input: int currentFenceHeight, indicator of the current fence height
       int fenceType, [PERMANENT_ONLY, PORTABLE_ONLY, TEMPORARY_ONLY, PERMANENT_OR_PORTABLE], indicates the type of fences that are allowed
Output: none
Return:int, [HEIGHT_1_8_METERS, HEIGHT_2_0_METERS, HEIGHT_1_4_METERS, FENCE_HEIGHT_ERROR] the shortest fence indicator
void
Functions called: none
Description: returns the indicator to the next taller fence based on the type(s) of fences allowed and what the current height is.
            if there is no taller fence or the fence type is unknown then FENCE_HEIGHT_ERROR is returned
Called by: case2, case4
Pseudo code:
if fenceType is PERMANENT_ONLY
    if currentFenceHeight is HEIGHT_1_8_METERS
        return HEIGHT_2_1_METERS;
    else if currentFenceHeight is HEIGHT_2_1_METERS
        return HEIGHT_2_4_METERS
    else if currentFenceHeight is HEIGHT_2_4_METERS
        return HEIGHT_2_7_METERS
    else if currentFenceHeight is HEIGHT_2_7_METERS
        return HEIGHT_3_0_METERS
    else if currentFenceHeight is HEIGHT_3_0_METERS
        return HEIGHT_3_6_METERS
    else
        return FENCE_HEIGHT_ERROR
else if fenceType is PORTABLE_ONLY
    if currentFenceHeight is HEIGHT_2_0_METERS
        return HEIGHT_2_4_METERS;
    else
        return FENCE_HEIGHT_ERROR
else if fenceType is TEMPORARY_ONLY
    return FENCE_HEIGHT_ERROR
else if fenceType is PERMANENT_OR_PORTABLE
    if currentFenceHeight is HEIGHT_1_8_METERS
        return HEIGHT_2_0_METERS
    else if currentFenceHeight is HEIGHT_2_0_METERS
        return HEIGHT_2_1_METERS
    else if currentFenceHeight is HEIGHT_2_1_METERS
        return HEIGHT_2_4_METERS
    else if currentFenceHeight is HEIGHT_2_4_METERS
        return HEIGHT_2_7_METERS
    else if currentFenceHeight is HEIGHT_2_7_METERS
        return HEIGHT_3_0_METERS
    else if currentFenceHeight is HEIGHT_3_0_METERS
        return HEIGHT_3_6_METERS
    else if currentFenceHeight is HEIGHT_3_6_METERS
        return FENCE_HEIGHT_ERROR
else
    return FENCE_HEIGHT_ERROR
else
    return FENCE_HEIGHT_ERROR
else
    return FENCE_HEIGHT_ERROR

Function: actualHeight
Version: 1.0
Input: int fenceHeight, indicator of the current fence height
Output: none
Return: double, the actual structural height of the current fence indicator
Functions called: none
Description: returns the structural height of the input fence indicator
Called by: case2, case3, case4, case5, case6
Pseudo code:

    if fenceHeight is HEIGHT_1_4_METERS
        return 1.4
    else if fenceHeight is HEIGHT_1_8_METERS
        return 1.8
    else if fenceHeight is HEIGHT_2_0_METERS
        return 2.0
    else if fenceHeight is HEIGHT_2_1_METERS
        return 2.1
    else if fenceHeight is HEIGHT_2_4_METERS
        return 2.4
    else if fenceHeight is HEIGHT_2_7_METERS
        return 2.7
    else if fenceHeight is HEIGHT_3_0_METERS
        return 3.0
    else if fenceHeight is HEIGHT_3_6_METERS
        return 3.6
    else
        return 0.0

Function: numberOfPossibleFences
Version: 1.0
Input: int fenceType, [PERMANENT_ONLY, PORTABLE_ONLY, TEMPORARY_ONLY, PERMANENT_OR_PORTABLE], indicates the type of fences that are allowed
Output: none
Return: int, [0,1,2,6,7], the number of possible fence based on the input type
Functions called: none
Description: returns the number of fences based on fence type
Called by: case2, case4
Pseudo code:

    there are 6 PERMANENT_ONLY type fences
    there are 2 PORTABLE_ONLY type fences
    there is 1 TEMPORARY_ONLY type fence
    there are 7 PERMANENT_OR_PORTABLE type fences
Function: findElevation

Version: 1.0

Input: SnowmanProfile *spP, pointer to a profile (can be any kind, ground, snow, depth, ...)
   double offset, an offset at which the elevation is to be found in the input profile

Output: double *elevation, the elevation that corresponds to the input offset in the input profile

Return: int, [FOUND_ELEVATION, OFFSET_OUT_OF_BOUNDS] indicates if the offset was
   found and the elevation set or if the offset is not within the *spP

Functions called: none

Description: finds the elevation of a data point corresponding to the input offset in the array of
   data points in *spP

Called by: case1, case2, case3, case4, case5, case6, case7, case8, createNewGround,
   effectiveHeight, nextSetback, mitigatedProfile, nextUnmitigatedDriftElevation,
   computeStorageAndDepthAndCheckForProblem

Pseudo code:
   search spP to find the data points just upwind and downwind of offset
   interpolate to the elevations
   return result
3.2 FileHandlers Module

Module: FileHandlers

Purpose: Contains the set of functions that handle the saving and reading of various types of data to and from files for SNOWMAN. This includes the saving and loading of climate, topographic and results data. In addition, the file loaders for defaults are in this module.
3.2.1 External Function Definitions

Function: loadDefaultsFile
Version: 1.0
Input: char *fName, pointer to a string of characters containing the name of the file containing the defaults data to be opened and loaded.
Output: SnowmanDefaults *sdP, pointer to the defaults data structure to load data from the file into
Return: int, [SNOWMAN_FILE_LOAD_FAILURE, SNOWMAN_FILE_LOAD_SUCCESS],
    SNOWMAN_FILE_LOAD_SUCCESS, if the file is open and loaded successfully,
    SNOWMAN_FILE_LOAD_FAILURE otherwise
Functions called: strlen, fopen, fscanf, fclose
Description: Opens the file pointed to by fName and reads in the contents to the SnowmanDefaults data structure pointed to by sdP. If the file is successfully opened and loaded and the defaults data successfully read in then SNOWMAN_FILE_LOAD_SUCCESS is returned and sdP contains valid defaults values. If the file is not successfully opened or contains bad data then SNOWMAN_FILE_LOAD_FAILURE is returned and sdP may not contain valid data.
Called by: main in Snowman
Pseudo code:
    if inputs are not valid
        return SNOWMAN_FILE_LOAD_FAILURE
    else if we cannot successfully open the file for reading
        return SNOWMAN_FILE_LOAD_FAILURE
    else we have valid inputs and the file is open for reading
        scan in the data for all fields in the defaults data structure (SnowmanDefaults) in the following order: levelStandardsFile, sectionSpacing, Cr, maxFenceOffset, minFenceOffset, allowableBuriedFenceRatio, maxNaturalAccumOffset, LOCACMOffset, allowableSoR, minUpwindTerrain, minDownwindTerrain, minFetch, maxFetch, bottomSlope, backSlope
        if any value cannot be read in then
            return SNOWMAN_FILE_LOAD_FAILURE
        if all data is read in correctly
            close the open file
            return SNOWMAN_FILE_LOAD_SUCCESS

Function: loadLevelStandardsFile
Version: 1.0
Input: char *fName, pointer to a string of characters containing the name of the file containing the level standards data to be opened and loaded.
Output: SnowmanLevelStandards *slsP, pointer to the level standards structure to hold the data
Return: int, [SNOWMAN_FILE_LOAD_FAILURE, SNOWMAN_FILE_LOAD_SUCCESS],
    SNOWMAN_FILE_LOAD_SUCCESS, if the file is open and loaded successfully,
    SNOWMAN_FILE_LOAD_FAILURE otherwise
Functions called: strlen, fopen, fscanf, atoi, fclose
Description: Opens the file pointed to by fName and reads in the contents into the
SnowmanLevelStandards data structure pointed to by slsP. If the file is successfully opened and loaded and the level standards data successfully read in then SNOWMAN_FILE_LOAD_SUCCESS is returned and slsP contains valid level standards data. If the file is not successfully opened or contains bad data then SNOWMAN_FILE_LOAD_FAILURE is returned and slsP may not contain valid data.

Called by: main in Snowman

Pseudo code:

```c
if inputs are not valid
    return SNOWMAN_FILE_LOAD_FAILURE
else if we cannot successfully open the file for reading
    return SNOWMAN_FILE_LOAD_FAILURE
else we have valid inputs and the file is open for reading
    read in the data for the fields in the SnowmanLevelStandards data structure in the following order: axis, axisText, axisTick, axisHorizontalGrid, axisVerticalGrid, text, ground, newGround, originalSnow, newSnow, existing.FENCE, proposed.FENCE, crossSection, ULOP, DLOP, proposed.DITCH, existing.DITCH, proposed.TOPO, existing.TOPO, LOF, note that each entry must the following in this order, useLevelName, level, color, weight, useStyleName, styleName
    if any value is not read in properly
        return SNOWMAN_FILE_LOAD_FAILURE
    if all data is read in correctly
        close the open file
        for each entry check if it uses level name or number. if it uses level number
            then convert level to an integer
        for each entry check if it uses style name or number. if it uses style number
            then convert styleName to an integer
```

Function: loadClimateDataFile

Version: 1.0

Input: char *fName, pointer to a string of characters containing the name of the file containing the climate data to be opened and loaded.

Output: SnowmanClimteData *scdP, pointer to the climate data structure to hold the data

Return: int, [SNOWMAN_FILE_LOAD_FAILURE, SNOWMAN_FILE_LOAD_SUCCESS], SNOWMAN_FILE_LOAD_SUCCESS, if the file is open and loaded successfully, SNOWMAN_FILE_LOAD_FAILURE otherwise

Functions called: strlen, fopen, fscanf, fclose

Description: opens the file pointed to by fName and reads the climate data from that file. If the file is a valid climate data file then SNOWMAN_FILE_LOAD_SUCCESS is returned and scdP contains valid climate data. If the file is not valid or doesn’t contain valid climate data then SNOWMAN_FILE_LOAD_FAILURE is returned and scdP may not contain valid data.

Called by: getClimateData in Snowman

Pseudo code:

```c
if inputs are not valid
    return SNOWMAN_FILE_LOAD_FAILURE
else if we cannot successfully open the file for reading
```
return SNOWMAN_FILE_LOAD_FAILURE
else we have valid inputs and the file is open for reading
read in the data for the fields in the SnowmanClimateData data structure in the
following order: latitude, longitude, elevation, St, Cr, k, direction,
ambientSnow
if any value is not read in properly then
return SNOWMAN_FILE_LOAD_FAILURE
else all values are read in properly so
close the open file
SNOWMAN_FILE_LOAD_SUCCESS

Function: saveClimateDataFile
Version: 1.0
Input: char *fName, pointer to a string of characters containing the name of the file the climate
data is to be written to.
SnowmanClimateData *scdP, pointer to the climate data structure that holds the climate
data to be written to the file.
Output: none
Return: int, [SNOWMAN_FILE_SAVE_FAILURE, SNOWMAN_FILE_SAVE_SUCCESS],
SNOWMAN_FILE_SAVE_SUCCESS, if the file is open/created and saved
successfully, SNOWMAN_FILE_SAVE_FAILURE otherwise
Functions called: strlen, fopen, fprintf, fclose
Description: opens/creates a file specified by fName and outputs the data contained in scdP to
that file. The output format is such that the function loadClimateData can open and
re-load the data contained in this file.
 Called by: recordClimateData in Snowma
Pseudo code:
  if inputs are not valid
    return SNOWMAN_FILE_SAVE_FAILURE
  else if we cannot successfully open/create the file for writing
    return SNOWMAN_FILE_SAVE_FAILURE
  else we have valid inputs and the file is open for writing
    write the following fields to the open file: latitude, longitude, elevation, St,
    Cr, k, direction, ambientSnow
    close the open file
    return SNOWMAN_FILE_SAVE_SUCCESS

Function: loadTopoDataFile
Version: 1.0
Input: char *fName, pointer to a string of charcters containing the name of the file containing the
topographic data to be opened and loaded.
SnowmanDefaults *sdP, pointer to the defaults data used to check certain topographic data
input bounds.
Output: SnowmanTopoData *stdP, pointer to the topographic data structure to hold the data
Return: int, [SNOWMAN_FILE_LOAD_FAILURE, SNOWMAN_FILE_LOAD_SUCCESS],
SNOWMAN_FILE_LOAD_SUCCESS, if the file is opened and loaded successfully,
SNOWMAN_FILE_LOAD_FAILURE otherwise
Functions called: strlen, fopen, fscanf, malloc, exit
Description: opens the file pointed to by fName and reads the topographic data from that file. If the file is a valid topographic data file then SNOWMAN_FILE_LOAD_SUCCESS is returned and stdP contains valid topographic data. If the file is not valid or doesn’t contain valid topographic data then SNOWMAN_FILE_LOAD_FAILURE is returned and stdP may not contain valid data.

Called by: getTopoData in Snowman

Pseudo code:
if inputs are not valid
    return SNOWMAN_FILE_LOAD_FAILURE
else if we cannot successfully open the file for reading
    return SNOWMAN_FILE_LOAD_FAILURE
else we have valid inputs and the file is open for reading
    read the number of cross sections in the file (nSections)
    if there are an invalid (<1) number of sections then
        return SNOWMAN_FILE_LOAD_FAILURE
    allocate the memory to hold the cross sectional data for nSections in section
    for each section
        read in the number of data points for the section (nDataPoints)
        size the profile.dPoint array to hold the nDataPoints
        for each data point
            read in the offset and elevation of that data point
            check that it is within the right range of elevations
            if not then return SNOWMAN_FILE_LOAD_FAILURE
        read in the location of the upwind limit of protection
        read in the location of the downwind limit of protection
        read in the ditch present indicator
        if there is a ditch then
            read in the location of the ditch low point and check that it’s valid
            set hasDitch to TRUE
            set validForEarthwork to TRUE
        else if there is no ditch then
            set hasDitch to FALSE
            set validForEarthwork to FALSE
        else something bad was read in so
            return SNOWMAN_FILE_LOAD_FAILURE
    read in and check the fetch
    read in and check the angle to the road
    if all the data for all the sections was read in fine then return
    SNOWMAN_FILE_LOAD_SUCCESS

Function: saveTopoDataFile
Version: 1.0
Input: char *fName, pointer to a string of characters containing the name of the file the topographic data is to be written to.
        SnowmanTopoData *stdP, pointer to the topographic data structure that holds the topographic data to be written to the file.
Output: none
Return: int, [SNOWMAN_FILE_SAVE_FAILURE, SNOWMAN_FILE_SAVE_SUCCESS], SNOWMAN_FILE_SAVE_SUCCESS, if the file is open/created and saved successfully, SNOWMAN_FILE_SAVE_FAILURE otherwise
Functions called: strlen, fopen, fprintf, fclose
Description: opens/creates a file specified by fName and outputs the data contained in stdP to that file. The output format is such that the function loadTopoData can open and reload the data contained in this file.

Called by: startProcessing in Snowman

Pseudo code:
```c
if inputs are not valid
    return SNOWMAN_FILE_SAVE_FAILURE
else if we cannot successfully open/create the file for writing
    return SNOWMAN_FILE_SAVE_FAILURE
else we have valid inputs and the file is open for writing
    write the number of cross sections
    foreach cross section
        write the number data points in the section
        foreach data point in the current section
            write the location of the data point (offset, and elevation)
            write the location of the upwind limit of protection
            write the location of the downwind limit of protection
            write the fetch
            write the angle to the road
    close the file
    return SNOWMAN_FILE_SAVE_SUCCESS
```

Function: saveResultsDataFile

Version: 1.0

Input: char *fName, pointer to a string of characters containing the name of the file the results data is to be written to.
SnowmanResultsData *srdP, pointer to the results data structure that holds the results data to be written to the file.
SnowmanClimateData *scdP, pointer to the climate data structure that holds the climate data to be written to the file.
SnowmanTopoData *stdP, pointer to the topographic data structure that holds the topographic data to be written to the file.

Output: none

Return: int, [SNOWMAN_FILE_SAVE_FAILURE, SNOWMAN_FILE_SAVE_SUCCESS],
SNOWMAN_FILE_SAVE_SUCCESS, if the file is open/created and saved successfully, SNOWMAN_FILE_SAVE_FAILURE otherwise

Functions called: strlen, fopen, fprintf, fclose

Description: output the topo, climate and results data to the file fName

Called by: getOutputOptions in Snowman

Pseudo code:
```c
if inputs are not valid
    return SNOWMAN_FILE_SAVE_FAILURE
else if we cannot successfully open/create the file for writing
    return SNOWMAN_FILE_SAVE_FAILURE
else we have valid inputs and the file is open for writing
    output the topo data:
        write number of sections
        foreach section
            write number of data points
            write offset and elevation foreach data point
```
write location of limits of protection
write ditch indicator and ditch location if applicable
write fetch and angle to road
write transports (Qaverage, Qexceedence)
write location of limit of storage (LOCACC offset)
write indicators if the section was considered valid for snow fences and or earthwork
output the climate data:
write latitude, longitude, elevation, St, Cr, direction
output the results data:
for each case to be output
output the case number
for each section
output the section number
output fence information (height, effectiveHeight, setback, and porosity)
if there is a fence
output the profile information (number of data points and all the data points offset and location pairs, and storages) if there is a result for this section and case
4 Main Program

The main program is responsible for communications with the user. The functions here interact with the user to gather data from the user or to display data to the user. In addition, some minor computations are carried out regarding climate data. These computations mostly involve the determination of climate factors needed by the evaluation module based on inputs from the user.

The program starts by loading some defaults files which define various constants and variables used during computation. The defaults.data file contains information regarding default values to use and bounds on certain parameters. The levels.data file contains the level standards definitions of MicroStation levels containing features that are of interest to SNOWMAN.

Once the defaults are loaded, the program gathers information about the user and the methods the user would like to use to input climate and topographic data. The user information is used to determine what level of access users can have and what they are allowed to modify during the program as well as for identification purposes in the output.

There are 3 ways to input climate data. The first is to provide SNOWMAN with the latitude, longitude and elevation of the site. Based on this information SNOWMAN can roughly estimate the climate data. The second method of input is to provide SNOWMAN with site specific data. That is, instead of having SNOWMAN compute the climate information, the user supplies the actual climate data for the site. The final method is to load it from a file. This is equivalent to providing SNOWMAN with site-specific data.

There are 3 ways to provide topographic data to SNOWMAN. A user can manually input the offset and elevation pairs for the profiles along with some specific section information. This manual input can be done for any number of cross sections. Alternatively, the information in the design (DGN) file for the site (if available) is used, and SNOWMAN automatically generates the cross sections. This approach requires level standard compliance in the DGN file. The final method is to load topographic data from a data file that was created from a previous run of SNOWMAN or from site survey (or rod and level) data.

After both the climate and topographic data have been collected, the program generates snow profiles for unmitigated terrain for each cross section and identifies problems with those sections if any problems exist. The user is provided some choice about how (scaling) and where (design file, or SNOWMAN data file) to send the results. The user can then review those results before making any case selections for possible mitigation if needed.

The mitigation choices come in two types. Mitigation can be accomplished through the placement of a structural snow fence or by a change in the upwind topography. The user can constrain the fence solutions by specifying or limiting certain attributes of the fences. These constraints include the types of fences allowed (permanent, temporary, or portable), the fence height and porosity, and the location or maximum setback of the fence. The earthwork solutions can be constrained by
limiting the maximum upwind offset where the topography can be modified.

The output is sent to either a snowman data file (ascii text) or to the design file in MicroStation. The output to design file allows the user to view the plots of the original ground and unmitigated profile along with the mitigated profiles and mitigation measures (fences, or modified terrain). After each run the user can review the results generated so far and continue to refine the solutions with additional constraints or can exit the program at any time.
4.1 Initialization

This section covers the functions that load defaults and initialize SNOWMAN.

Function: main
Version: 1.0
Input: int, argc, the number of arguments in argv
char[], argv, the array of arguments to main
Output: none
Return: void
Functions called: mdlResource_openFile, mdl_loadCommandTable,
mdlCExpression_initializeSet, mdlDialog_publishComplexVariable,
mdl_dialog_hookPublish, mdlSystem_setFunction, mdlSystem_getCfgVar,
loadDefaultsFile, loadLevelStandardsFile
Description: Performs the initialization and loading of resource files. Publishes the
snowmanGlobals variable so the fields can be accessed by the dialog boxes. Reads
the system variable SNOWMAN_HOME to get the location of the defaults and
levels files. Loads the defaults file and the file with the level standards definitions.
Called by: system
Pseudo code:
load resource files
publish global structure for communications with dialog boxes
publish hood functions
read environment variable SNOWMAN_HOME to get home location of program
load defaults data file
load levels data file
start SNOWMAN

Function: startSNOWMAN
Version: 1.0
Command Number: none
Input: void
Output: void
Return: void
Functions called: mdlDialog_open
Description: Starting function that opens the welcome to SNOWMAN dialog box.
Called by: main
Pseudo code:
open welcome screen dialog box (DIALOGID_Snowman)
Function: getUserInputInformation
Version: 1.0
Command number: CMD_GET_USER_INPUT_INFORMATION
Input  void
Output: void
Return: void
Functions called: mdlDialog_open
Description: Initializes SNOWMAN and opens the user input information dialog box.
Called by: DIALOGID_Snowman via command number
Pseudo code:
initialize userName and siteID to be blank
set user level to novice (default)
set climate input method to input by snowman data file
set topographic input method to input by snowman data file.
set default save for climate and topographic data to FALSE
set manualInputDone to FALSE
set firstTime to TRUE
set output to screen
set scale factor to 1:1
open the get user input information dialog box (DIALOGID_GetUserInputInformation)
4.2 Climate input methods

Function: getClimateData
Version: 1.0
Command number: CMD_GET_CLIMATE_DATA
Input: void
Output: void
Return: void
Functions called: mdlDialog_open, sprintf, mdlDialog_fileOpen, loadClimateDataFile,
displayClimateData, mdlOutput Printf, printf, exitSnowman
Description: Either opens the corresponding dialog box for input by lat, long elev, or user
specified input, or loads a SNOWMAN climate data file.
Called by: DIALOGID_GetUserInputInformation via command number
Pseudo code:
check the user information
if input by site specific
    open get climate by site specific data dialog box
else if input by latitude, longitude, and elevation
    open the get climate by lat long elev computation dialog box
else if input by snowman climate data file
    while data file not loaded
        prompt user for name of data file
        if name given
            attempt to load file
            if successful
                call displayClimateData
            else
                display error message and reprompt
        else if user canceled input
            go back to the user input information dialog box to get new input method
        else
            go back to the user input information dialog box to get new input method

Function: displayClimateData
Version: 1.0
Command number: CMD_DISPLAY_CLIMATE_DATA
Input: void
Output: void
Return: void
Functions called: sprintf, mdlOutput_printf, mdlDialog_open,
Description: Checks the inputs (latitude, longitude, elevation, St, Cr, and direction) if
CLIMATE_INPUT_BY_SITE_SPECIFIC_DATA. Checks the inputs (latitude,
longitude, and elevation) and computes St, and direction, and assigns Cr if
CLIMATE_INPUT_BY_LAT_LONG_ELEV_COMPUTATION. Then updates
the corresponding variables in snowmanGobals and opens the display climate data
dialog box if the values are OK. If the values contain errors (i.e. not representative of NYS) then the input dialog box is opened again and the user is reprompted for valid input.

Called by: getClimateData, DIALOGID_GetClimateBySiteSpecificData via command number, DIALOGID_GetClimateByLatLongElevComputation via command number.

Pseudo code:

if input by site specific data
    check each input for proper bounds
    if any input is out of range
        display message with proper range
        reopen DIALOGID_GetClimateBySiteSpecificData
    else
        use default value for k (1.5, corresponds to 5% exceedence)
        open display climate information dialog box (DIALIOGID_DisplayClimateData)
        if novice user
            disable the fields for ambient snow cover usage and allowing buried fences
        else if input by latitude, longitude and elevation computation
            check each input for proper bounds
            if any input is out of range
                display message with proper range
                reopen DIALOGID_GetClimateByLatLongElevComputation
            else
                use default value for k (1.5, corresponds to 5% exceedence)
                use Cr from defaults data
                Compute:
                St = 25.4*(-686.6 + 0.083*elevation +17.251*latitude);
                direction = 987.0 - 9.42*longitude;
                ambientSnow=((0.7775-(0.00914*longitude))*(-686.6+(0.083*elevation)+(17.251*latitude)))*25.4;
                open display climate information dialog box (DIALIOGID_DisplayClimateData)
                if novice user
                    disable the fields for ambient snow cover usage and allowing buried fences
                else if input by snowman data file
                    open display climate information dialog box (DIALIOGID_DisplayClimateData)
                    if novice user
                        disable the fields for ambient snow cover usage and allowing buried fences

Function: recordClimateData

Version: 1.0
Command number: CMD_RECORD_CLIMATE_DATA
Input: void
Output: void
Return: void

Functions called: sprintf, mdlOutput_printf, mdlDialog_open, mdlDialog_fileCreate, saveClimateDataFile, getTopoData, printf, exitSnowman

Description: Once the user sees the climate data they can make changes to the values so those values need to be rechecked (latitude, longitude, elevation, St, Cr, and direction). If the values are OK they are recorded and the data is either saved and not saved to a file (depending on user selection) before moving on to gather topographic data.

Called by: DIALOGID_DisplayClimateData via command number
Pseudo code:

if invalid latitude
    prompt user with bounds for latitude and re-open DisplayClimateData dialog box
else if invalid longitude
    prompt user with bounds for longitude and re-open DisplayClimateData dialog box
else if invalid elevation
    prompt user with bounds for elevation and re-open DisplayClimateData dialog box
else if total annual snowfall is less than 0
    prompt user for value greater than 0 and re-open DisplayClimateData dialog box
else if relocation coefficient is less than 0 or greater than 1
    prompt user for new value and re-open DisplayClimateData dialog box
else if exceedence factor is less than 1
    prompt user for new value and re-open DisplayClimateData dialog box
else if wind direction is less than 0 or greater than 360
    prompt user for proper wind direction and re-open DisplayClimateData dialog box
else if ambient snow cover is less than 0
    prompt user for value greater than 0 and re-open DisplayClimateData dialog box
else everything is OK,
    record all values in the climate data structure
    if user wants to save data to file
        prompt for data file and write data to file
    call get topographic data
4.3 Topographic input methods

Function: getTopoData
Version: 1.0
Command number: none
Input: void
Output: void
Return: void
Functions called: ilkElements, mdlDialog_open, mdlState_startPrimitive, mdlDialog_fileOpen,
loadTopoDataFile, displayTopoData, printf, exitSnowman

Description: Based on user selection for topo input information either opens a dialog box to
prompt for the starting location in the design file along the upwind limit of
protection, or opens a dialog box to load a snowman topographic data file, or opens
a dialog box to gather topo data manually from user. If topographic input method is
by SNOWMAN data file then after the file is loaded (loading functions checks
inputs) the information is displayed to the user. In all other cases different
functions still need to be called to complete the gathering of topographic data. This
function is just the decision maker as to which branch to follow for the various topo
input methods.

Called by: recordClimateData

Pseudo code:
if input by existing or proposed design file
    ilkElements
    prompt user for starting point
    note: since this is done through the start primitive function control passes out
of this function and into recordStartingPoint
else if input by manual input
    initialize section number and data point number
    initialize memory for cross sections
    initialize cross sections
    open dialog box to get data points (DIALOGID_GetDataPoint)
else if input by snowman data file
    while data file not loaded
        prompt user for name of data file
        if name given
            attempt to load file
            if successful
                call displayTopoData
            else
                display error message and re prompt
        else if user canceled input
            go back to the user input information dialog box to get new input method
        else
            go back to the user input information dialog box to get new input method

Function: ilkElements
Version: 1.0
Command number: none
Input: int type: indicates which type to search for existing or proposed level standards
Output: none
Return:int (0,1)  0 - indicates that there was an error (i.e. definite missing information)
        1 - indicates that there was no errors (i.e. possible missing information)
Functions called: findElements, printf, exitSnowman
Description: finds elements of interest to SNOWMAN: upwind limit of protection line(s),
downwind limit of protection line(s), topo triangles, ditch line(s), and fetch barrier
line(s). If we don't find a minimum number of these elements then we know we
can't construct the topographic information to do anything with SNOWMAN.
Based on the type (existing or proposed) this function calls findElements to get a set
of elements matching that types settings (level, color, weight, style) then the
number of elements found for each type is checked to insure we have sufficient
information in the design file.
Called by: getTopoData
Pseudo code:
// if type is SNOWMAN_EXISTING
//   find existing ULOP, DLOP, TOPO, DD, and FETCH elements
//   check for proper/minimum/maximum number of each element type
//   if insufficient information
//     return 0
//   else
//     return 1
// if type is SNOWMAN_PROPOSED
//   find proposed ULOP, DLOP, TOPO, DD, and FETCH elements
//   check for proper/minimum/maximum number of each element type
//   if insufficient information
//     return 0
//   else
//     return 1

Function: findElements
Version: 1.0
Input: SnowmanLevelStandard ls: st ucture that defines what elements are sought (level, color,
        weight, and style)
Output: int nElements: number of elements of type specified by ls that were found.
Return:ULong*: pointer to an array of elements that match the description in ls.
Functions called: mdlBitMask_create, mdlLevel_getIdFromName, mdlOutput_printf,
                mdlBitMask_setBit, mdlScanCriteria_create, mdlScanCriteria_setReturnType,
                mdlScanCriteria_setModel, mdlScanCriteria_setDrawnElements,
                mdlScanCriteria_setLevelTest, malloc, exitSnowman, sizeof,
                mdlScanCriteria_scan, mdlElement_read, mdlElement_getSymbology,
                mdlElement_getLineStyle, realloc
Description: Searches the design file (MASTERFILE) for elements that match level, color,
weight, and style specified by input ls. Collects all the matching elements and returns a pointer to the array of those elements as well are outputing the number of elements in the array being returned. This function was updated to match the MicroStation version 8 routines that replace the older searching functions. It provided vast improvement in computation time since it allowed limiting the search to specified levels instead of the whole design file.

Called by: ilkElements

Pseudo code:

use a bitmask to eliminate the elements not on a level of interest
define the search criteria
return file position from masterfile of drawn elements that are on the level of interest
begin scan
foreach element returned by the scan check color, weight, and style
if it’s a match
store the address of the element in foundElements
else
skip element
return all the matching elements found

Function: recordStartingPoint
Version: 1.0
Command number: none
Input: Dpoint3d *pntP: MicroStation data point indicating the data point the user entered.
int view: The number of the view that the data point was entered in (unused).
Output: none
Return: void
Functions called: validULOPPoint, mdlOutput_printf, mdlState_startPrimitive,
mdlDialog_closeCommandQueue, mdlDialog_find, mdlDialog_open
Description: Determines if the input data point is along the upwind limit of protection (ULOP). If it is along the ULOP, then the data point is recorded in snowmanGlobals.startPoint and a new state is started to get the ending point along the ULOP. If it is not along the ULOP, then the user is informed of that and prompted to enter a data point along the ULOP.
Called by: via mdlState_startPrimitive command in getTopoData, and recordStartingPoint
Pseudo code:

check that pntP is valid
if valid (i.e. along the ULOP) then
record the data point that was entered and use that as the starting point
get the ending data point (again this is done through the mdlState_startPrimitive)
else
re-prompt user for a new data point along the ULOP
Function: recordEndingPoint
Version: 1.0
Command number: none
Input:  DPoint3d *pntP: MicroStation data point indicating the data point the user entered.
         int view: The number of the view that the data point was entered in (unused).
Output:  none
Return: void
Functions called:  validULOPPoint, mdlOutput_printf, mdlState_startPrimitive,
                            mdlDialog_closeCommandQueue, mdlDialog_find, mdlDialog_open
Description:  Determines if the input data point is along the upwind limit of protection (ULOP).
If it is along the ULOP, then the data point is recorded in snowmanGlobals.endPoint If it is not along the ULOP, then the user is informed of that and prompted to enter a data point along the ULOP. Once the data point is found and is valid then generateCrossSections is called.
Called by: via mdlState_mdlPrimitive command in recordStartingPoint, recordEndingPoint
Pseudo code:
    check that pntP is valid
    if valid (i.e. along the ULOP) then
        record the data point that was entered and use that as the ending point
        call generateCrossSections
    else
        re-prompt user for a new data point along the ULOP

Function: validULOPPoint
Version: 1.0
Command number: none
Input:  DPoint3d *pntP: MicroStation data point to be checked to see if it is along the ULOP.
Output:  DPoint3d *pntP: The exact point along the ULOP.
Return: int (0,1): 0 - indicates it was not along the ULOP.
          1 - indicates it was along the ULOP.
Functions called:  sin, cos, malloc, exitSnowman, mdlLineString_create, mdlElmdscr_new,
                             mdlElmdscr_read, mdlIntersect_allBetweenElms, mdlElmdscr_freeAll, sprintf,
                             printf
Description:  This is needed since Darrell Kaminski requested that the user be able to just click near the start location and not have to enter the data point exactly along the ULOP. This function tries to find the intersection of an imaginary line in the direction of the wind that passes through both the ULOP line and the data point entered by the user. If it does, then the intersection of that imaginary line and the ULOP is recorded in pntP, and 1 is returned. If not then 0 is returned.
Called by: recordStartingPoint, recordEndingPoint
Pseudo code:
    check that pntP is not null
    create a small element parallel to the direction of the wind at pntP
    find intersection of that small element and the ULOP line(s)
    if no intersections then
        errorMessage to indicate no intersections found

65
return FALSE
if just 1 intersection
then return TRUE
if more than 1 intersection
set errorMessage to indicate multiple intersections found
return FALSE

Function: generateCrossSections
Version: 1.0
Command number: none
Input: void
Output: void
Return: void
Functions called: sqrt, atan, malloc, exitSnowman, fabs, mdlLineString_create, mdlElmdscr_new,
mdlElmdscr_read, mdlIntersect_allBetweenElms, printf, mdlElement_display,
mdlElement_add, pow, mdlElmdscr_freeAll, free, displayTopoData,
mdlDialog_open
Description: Based on the starting and ending points in snowmanGlobals cross sections are
created at intervals between the specified points along the direction of the wind.
The intersections with these cross sections lines and the ULOP, DLOP, possibly the
ditch lines, the TRIANGLES, and the FETCH lines are used to determine the cross
sections’ profile (based on TRIANGLES) and section information (ULOP offset,
DLOP offset, FETCH offset, DITCH offset (if present)). If any violate the
minimum specified requirement for those parameters and error is flagged and the
user is told about the missing or invalid information. Presently, SNOWMAN
requires all sections to pass the requirements, but this could be changed to allow it
to skip the invalid input sections.
Called by: recordEndingPoint
Pseudo code:

determine direction along road from start to end point
determine number of cross sections from length from start to end point
allocate memory for cross sections and initialize them
check angle between wind and road (must be > 10 degrees)
foreach cross section
find intersection point between cross section line and ULOP
create the cross sectioning line
find the DLOP
if no intersections found
  display error message
  open get user information dialog box (DIALOGID_GetUserInputInformation)
else if multiple intersections found
  display error message
  open get user information dialog box (DIALOGID_GetUserInputInformation)
create the ground profile from intersection of cross section line with triangles
find lowest upwind ditch location if any
find the intersection with fetch
if no intersections found
  make fetch maximum fetch distance
if one intersection found
if computed fetch is greater than max
    make fetch max fetch distance
else if computed fetch is less than min
    make fetch min fetch distance
else
    make fetch computed distance

foreach cross section
    verify correct ULOP,DLOP,...
    set valid for earthwork and snow fences accordingly
    draw the sectioning line from the most upwind point the most downwind point
    if all sections are valid for at least snow fences
        continue on to display topographic data
    else
        return to the get user input information dialog box to get new input method

Function: addDataPoint
Version: 1.0
Command number: none
Input: int n: section number to add the offset and elevation to.
        double offset: offset of the new data point to be added.
        double elevation: elevation of the new data point to be added.
Output: none
Return:int:  
        ADDED_DATA_POINT - indicates data point was successfully added.
        DUPLICATE_OFFSET - indicates that a data point with that offset already exists and new data point will not be added.
Functions called: malloc, sizeof, realloc, printf, exitSnowman
Description: Adds a new data point specified by the pair (offset,elevation) to section n of the sTopoData structure. If there exists an entry in that structure with the same offset then the new data point is not added and DUPLICATE_OFFSET is returned. In addition the new data point is added in ascending offset sorted order.
Called by: generateCrossSections, getNextDataPoint, getSectionInformation
Pseudo code:
if at the start of a new cross section
    allocate memory for profile and initialize profile
    add data point
    return ADDED_DATA_POINT
else
    if duplicate offset
        return DUPLICATE_OFFSET
    else
        reallocate memory
        add/insert data point
        return ADDED_DATA_POINT

Function: getNextDataPoint
Version: 1.0
Command number: CMD_GET_NEXT_DATA POINT
Input: void
Output: void
Return:void
Functions: sprintf, mdlOuptut_printf, mdlDialog_open, addDataPoint
Description: Checks the input of the new data point from the user through Data Point Entry
Dialog Box (offset and elevation) to insure that it’s within bounds for NYS. Then
calls addDataPoint to added it to the current section. If ADDED_DATA_POINT is
returned then the next data point is gathered. If DUPLICATE_OFFSET is returned
then the user is prompted again to enter a valid data point.
Called by: getNextDataPoint, DIALOGID_GetDataPoint via command number
Pseudo code:
if elevation out of range
    prompt user with error message
    reopen dialog box DIALOGID_GetDataPoint
else
    add data point
    if data point added
        increment data point number
        reopen dialog box DIALOGID_GetDataPoint
    else if duplicate point
        prompt user with error message
        reopen dialog box DIALOGID_GetDataPoint

Function: getSectionInformation
Version: 1.0
Command number: CMD_GET_NEXT_CROSS_SECTION
Input: void
Output: void
Return:void
Functions: sprintf, mdlOutput_printf, mdlDialog_open, addDataPoint, printf, exitSnowman
Description: Checks the input of the new data point from the user through Data Point Entry
Dialog Box (offset and elevation) to insure that its within bounds for NYS. Then
calls addDataPoint to added it to the current section. If ADDED_DATA_POINT is
returned then the section is checked for sufficient upwind and downwind terrain.
Then the dialog box to get the seicon information is opened. If
DUPLICATE_OFFSET is returned then the user is prompted again to enter a valid
data point.
Called by: manualInputDone, DIALOGID_GetDataPoint via command number
Pseudo code:
if elevation out of range
    prompt user with error message
    reopen dialog box DIALOGID_GetDataPoint
else
    add data point
    if data point added
        increment data point number
    else if duplicate point
        just ignore it
check for min upwind and downwind terrain and number of data points
if any is out of range
    prompt user of mistake
    re-open the DIALOGID_GetDataPoint
else
    open dialog box to get section information (DIALOGID_GetSectionInfo)

Function: recordSectionInfo
Version: 1.0
Command number: CMD_RECORD_SECTION_GET_NEXT
Input: void
Output: void
Return: void
Functions called: sprintf, mdlOutput_printf, mdlDialog_open, displayTopoData, malloc, realloc, printf, exitSnowman
Description: Checks the values for the section information (ditch offset and elevation (if present), downwind limit of protection offset). The DLOP must be downwind (i.e. positive offset). The ditch, if present, must be upwind (i.e. negative offset). Also checks that we have minDownwindTerrain past the DLOP. Fetch is checked against the default min and max fetch values. Angle to road is checked to insure that the angle between the wind and road is between 10 and 90 degrees. If all checks pass the data is recorded in the sTopoData structure for the current section. The flag snowmanGlobals.manualInputDone is checked. If the flag is set then displayTopoData is called. If the flag is not set then the next section's memory is allocated and the getDataPoint dialog box is opened.

Called by: DIALOGID_GetSectionInfo

Pseudo code:
check offset and elevations for the ditch (if present), DLOP
check for min upwind and downwind terrain and number of data points
check fetch and angle to road
if any is out of range
    prompt user about error
    re-open the DIALOGID_GetSectionInfo
else
    record data (ULOP, DLOP, FETCH, angleToRoad, ...)
    mark as valid for snowfence
    if ditch is present
        mark as valid for earthwork
    else
        mark as not valid for earthwork
if manual input is done
    call displayTopoData
else
    create next cross section
    allocate memory
    initialize new section
    open dialog box to get data points (DIALOGID_GetDataPoint)
Function: manualInputDone
Version: 1.0
Command number: CMD_MANUAL_INPUT_DONE
Input: void
Output: void
Return: void
Functions called: getSectionInformation
Description: sets the flag in snowmanGlobals to indicate that manual input is done before going to recordSectionInfo (i.e. no more sections to be input)
Called by: DIALOGID_GetSectionInfo
Pseudo code:
   set flag for manual input done to TRUE
   call getSectionInformation

Function: displayTopoData
Version: 1.0
Command number: CMD_DISPLAY_TOPO_DATA
Input: void
Output: void
Return: void
Functions called: sprintf, mdlDialog_open
Description: Copies the value from the sTopoData structure into the corresponding variables in snowmanGlobals then opens the DisplayTopoData dialog box to display the data to the user.
Called by: getTopoData, generateCrossSections, recordSectionInfo, displayNextSection, displayPrevSection, startProcessing
Pseudo code:
   copy DLOP offset, angle to road, and ditch information (if any)
   open the display topo data dialog box (DIALOGID_DisplayTopoData)

Function: displayNextSection
Version: 1.0
Command number: CMD_NEXT_SECTION
Input: void
Output: void
Functions called: displayTopoData, mdlOutput_printf
Description: Since the user could change the value of fetch (i.e. it’s a modifiable textbox) the value is rechecked and updated if within bounds. If the fetch is OK then the next crossSection in sTopoData is set up for display. If the current section is the last section then the first section will be displayed.
Called by: DIALOGID_DisplayTopoData via command number
Pseudo code:
check fetch
if in range
    increment section number
    if section number now greater than number of sections
        reset section number to 1
    copy fetch
    call displayTopoData
else
    prompt user for correct fetch range
    call displayTopoData

Function: displayPrevSection
Version: 1.0
Command number: CMD_PREV_SECTION
Input: void
Output: void
Functions called: displayTopoData, mdlOutput_printf
Description: Since the user could change the value of fetch (i.e. it’s a modifiable textbox) the value is rechecked and updated if within bounds. If the fetch is OK then the previous crossSection in sTopoData is set up for display. If the current section is the first section then the last section will be displayed.
Called by: DIALOGID_DisplayTopoData via command number
Pseudo code:
check fetch
if in range
    decrement section number
    if section number now less than 1
        reset section number to number of sections
    copy fetch
    call displayTopoData
else
    prompt user for correct fetch range
    call displayTopoData

Function: startProcessing
Version: 1.0
Command number: CMD_SNOWMAN_EVALUATION
Input: void
Output: void
Return: void
Functions called: sprintf, mdlDialog_fileCreate, saveTopoDataFile, mdlOutput_printf, exitSnowman, getCaseSelections, displayTopoData
Description: Checks that the fetch is still valid (could be changed in dialog box). If it’s not valid then the display topo data dialog box is reopened. Then saves the topographic data to file if user has chosen to do so. Computes the transport values and calls getCaseSelection
Called by: DIALOGID_DisplayTopoData
Pseudo code:

check fetch input from user
if within range
    record fetch
    if user selected to save data to file
        prompt user for name of file
        if user gave name
            open/create file
            write topographic data to file
        else
            re-open DIALOGID_DisplayTopoData
    else
        prompt user for proper fetch range
        re-open DIALOGID_DisplayTopoData
4.4 Case selection methods

Function: getCaseSelections
Version: 1.0
Command number: CMD_GET_CASE_SELECTIONS
Input: void
Output: void
Return: void
Functions called: generateResults, mdlDialog_open, malloc
Description: checks the global flag firstTime to see if this is the first time (run). If so then a
preselected number of cases is run. If not then the dialog box that collects the case
selections is opened to determine what cases to run.
Called by: startProcessing, getCaseSelections, plotOutputToScreen
Pseudo code:
if first time
  run case 1 only (unmitigated snow)
else
  free old memory from previous runs (excluding case 1)
  reset results case list
  reset case selections
  open case selection dialog box (DIALOGID_CaseSelection)

Function: checkCaseSelections
Version: 1.0
Command number: CMD_CHECK_CASE_SELECTIONS
Input: void
Output: void
Return: void
Functions called: sprintf, mdlOutput_printf, mdlDialog_open, getCaseSpecificData
Description: Checks that the user has entered a valid list of cases to run; i.e., at least 1 case has
been selected
Called by: DIALOGID_CaseSelection via command number
Pseudo code:
if no cases were selected
  prompt user to select at least 1 case
  re-open DIALOGID_CaseSelection
else
  call getCaseSpecificData

Function: getCaseSpecificData
Version: 1.0
Command number: none
Input: void  
Output: void  
Return: void  
Functions called: recordCase2Data  
Description: For each case the user selected to run that requires additional case specific input (fence selection, fence setback, ...) The user is prompted for that information. The information is checked and recorded for that case.

Called by: checkCaseSelections

Pseudo code:

```c
set number of sections per case in results to number of sections
initialize current section number to 0
```

Function: recordCase2Data
Version: 1.0
Command number: CMD_RECORD_CASE2_DATA
Input: void
Output: void
Return: void
Functions called: recordCase3Data, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf
Description: This function checks to see if case 2 was selected. If not then flow continues on to recordCase3Data. However, if case 2 was selected this function initializes the data structures to hold the results for case 2 and records the user’s options for case 2 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, and the type(s) of fences to possibly use)

Called by: getCaseSpecificData, DIALOGID_Case2Data via command number

Pseudo code:

```c
if case 2 was selected
    if section number is 0 this is the first time
        initialize sResultsdata to collect the user constraints
        open dialog box to collect case 2 data
        if novice user
            disable access to buried fence and ambient snow
        else not the first time
            if buried fence ratio is out of range
                prompt user for proper range
                re-open dialog box for collecting case 2 data
            else
                if last section
                    record data for last section
                    reset section number to 0
                    call recordCase3Data
                else
                    if use same for all was selected
                        record current set of inputs for all sections
                        reset section number to 0
                        call recordCase3Data
                    else
                        record current set of inputs for current section
                        increment section number
```
Function: recordCase3Data
Version: 1.0
Command number: CMD_RECORD_CASE3_DATA
Input: void
Output: void
Return: void
Functions called: recordCase4Data, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf

Description: This function checks to see if case 3 was selected. If not then flow continues on to recordCase4Data. However, if case 3 was selected this function initializes the data structures to hold the results for case 3 and records the user’s options for case 3 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, fence height, and fence porosity)

Called by: recordCase2Data, DIALOGID_Case3Data via command number

Pseudo code:
if case number 3 was selected
    if first time through
        initialize sResultsData to hold case 3 constraints
        set default selections
        open dialog box to collect case 3 data
        if novice user
            disable access to buried fence and ambient snow
    else
        if buried fence ratio is out of range
            prompt user for proper range
            re-open dialog box to collect case 3 data
        if novice user
            disable access to buried fence and ambient snow
    else
        if last section
            record current constraints
            reset section number to 0
            call recordCase4Data
        else
            if user selected use same for all
                record current constraints for all remaining sections
                reset section number to 0
                call recordCase4Data
            else
                record current constraints for current section
                increment section number
                re-open dialog box to collect case 3 data
            if novice user
                disable access to buried fence and ambient snow
            else
                call recordCase4Data
Function: recordCase4Data
Version: 1.0
Input: void
Output: void
Return: void
Functions called: recordCase4Data, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf
Description: This function checks to see if case 4 was selected. If not then flow continues on to recordCase5Data. However, if case 4 was selected this function initializes the data structures to hold the results for case 4 and records the user’s options for case 4 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, fence type and max fence setback)
Called by: recordCase3Data, DIALOGID_Case4Data via command number
Pseudo code:
if case 4 was selected
  if first time through
    initialize sResults data to hold case4 constraints
    set default constraints
    open dialog box to collect case 4 data (DIALOGID_Case4Data)
    if novice user
      disable access to buried fence and ambient snow
  else
    if buried fence ratio is out of range
      prompt user with proper range
      re-open dialog box to collect case 4 data
      if novice user
        disable access to buried fence and ambient snow
    else if setback constraint is out of range
      prompt user with proper range
      re-open dialog box to collect case 4 data
      if novice user
        disable access to buried fence and ambient snow
  else
    if last section
      record constraints for this section
      reset section number to 0
      call recordCase5Data
    else
      if use same for all was selected
        foreach remaining section
          check if current constraint is within range
          if not valid for all
            prompt user for valid setback constraint
            re-open dialog box to collect case 4 data
            if novice user
              disable access to buried fence and ambient snow
          else
            foreach remaining section
              record constraints
              reset section number to 0
              call recordCase5Data

else not using same for all remaining sections
    record current constraints for current section
    increment section number
    re-open dialog box to collect case 4 data
    if novice user
        disable access to buried fence and ambient snow
else case 4 was not selected
    call recordCase5Data

Function: recordCase5Data
Version: 1.0
Input: void
Output: void
Return: void
Functions called: recordCase6Data, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf
Description: This function checks to see if case 5 was selected. If not then flow continues on to recordCase6Data. However, if case 5 was selected this function initializes the data structures to hold the results for case 5 and records the user’s options for case 5 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, fence height, fence porosity, and max fence setback)
Called by:
Pseudo code:
    if case 5 was selected
        if first time through
            initialize sResultsData to hold case 5 results and constraints
            set default constraints
            open dialog box to get case 5 data (DIALOGID_Case5Data)
            if novice user
                disable access to buried fence and ambient snow
            else not first time
                if buried fence ratio is out of range
                    prompt user with proper range
                    re-open dialog box to get case 5 data
                    if novice user
                        disable access to buried fence and ambient snow
                else if setback is out of range
                    prompt user with proper range
                    re-open dialog box to get case 5 data
                    if novice user
                        disable access to buried fence and ambient snow
        else
            if last section
                record current constraints for current section
                reset section number to 0
                call recordCase6Data
            else
                if user selected use same for remaining
                    check if setback is valid for all remaining sections
                    if not valid for all
                        prompt user for new setback constraint
                        re-open dialog box to get case 5 data
if novice user
disable access to buried fence and ambient snow
else it is valid for all

foreach remaining section
record current constraints
reset section number to 0
call recordCase6Data
else use for remaining was not selected
record current constraints for current section
re-open dialog box to get case 5 data
if novice user
disable access to buried fence and ambient snow
else case 5 was not selected
call recordCase6Data

Function: recordCase6Data
Version: 1.0
Input: void
Output: void
Return: void
Functions called: recordCase7Data, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf
Description: This function checks to see if case 6 was selected. If not then flow continues on to recordCase7Data. However, if case 6 was selected this function initializes the data structures to hold the results for case 6 and records the user’s options for case 6 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, fence height, fence porosity, and fence setback)
Called by: recordCase5Data, DIALOGID_Case6Data via command number
Pseudo code:
if case 6 was selected
if first time through
initialize sResultsData to hold case 6 and constraints
set default constraints
open dialog box to get case 6 data (DIALOGID_Case6Data)
if novice user
disable access to buried fence and ambient snow
else not first time
if height constraint is out of range
prompt user with proper range
open dialog box to get case 6 data
if novice user
disable access to buried fence and ambient snow
else if use same for all was selected
check if setback is valid for all sections
if not
prompt user with proper range
open dialog box to get case 6 data
if novice user
disable access to buried fence and ambient snow
else
for each remaining section
record current constraints for each section
reset section number to 0
call recordCase7Data
else
    record current constraints for current section
    increment section number
    open dialog box to get case 6 data
    if novice user
        disable access to buried fence and ambient snow
    else case 6 was not selected
    call recordCase7Data

Function: recordCase7Data
Version: 1.0
Command number: CMD_RECORD_CASE7_DATA
Input: void
Output: void
Return: void
Functions called: generateResults, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf
Description: This function checks to see if case 7 was selected. If not then flow continues on to recordCase8Data. However, if case 7 was selected this function initializes the data structures to hold the results for case 7 and records the user’s options for case 7 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation)
Called by: recordCase6Data, DIALOGID_Case7Data via command number
Pseudo code:
if case number 7 was selected
    if first time through
        check if valid for earthwork solutions
        if not
            prompt user that no sections are valid for earthwork
            open case selection dialog box (DIALOGID_CaseSelection)
        else there are sections valid for earthwork
            initialize sResultsData to hold case 7 and constraints
            set default constraints
            open dialog box to collect case 7 data
    else not first time through
        if last section
            record constraints
            reset section number to 0
            call recordCase8Data
        else not last section
            if user selected use same for remaining
                record current constraints for all remaining sections
                reset section number to 0
                call recordCase8Data
            else user didn’t select use same for remaining
                record current constraints for this section
                increment section number
                re-open dialog box to collect case 7 data
        else case number 7 was not selected
        call recordCase8Data

79
Function: recordCase8Data  
Version: 1.0  
Command number: CMD_RECORD_CASE8_DATA  
Input: void  
Output: void  
Return: void  
Functions: generateResults, mdlDialog_open, mdlDialog_itemSetEnableState, sprintf, mdlOutput_printf  
Description: This function checks to see if case 8 was selected. If not then flow continues on to generateResults. However, if case 8 was selected this function initializes the data structures to hold the results for case 8 and records the user’s options for case 8 (allow buried fences or not, allowable buried depth ratio, and if ambient snow should be used in computation, limit of earthwork)  
Called by: recordCase7Data, DIALOGID_Case8Data via command number  
Pseudo code:  
if case 8 was selected  
if first time through  
check if valid for earthwork solutions  
if not  
prompt user that no sections are valid for earthwork  
open case selection dialog box (DIALOGID_CaseSelection)  
else there are sections valid for earthwork  
initialize sResultsData to hold case 8 and constraints  
set default constraints  
open dialog box to collect case 8 data (DIALOGID_Case8Data)  
else not first time  
if last section  
record constraints  
call generateResults  
else not last section  
if user selected use same for all  
foreach section  
record constraints  
call generate results  
else user didn't select use same for all  
record constraints for current section  
increment section number  
re-open dialog box to collect case 8 data  
else case 8 was no selected  
call generateResults

Function: generateResults  
Version: 1.0  
Command number: none  
Input: void  
Output: void  
Functions called: malloc, case1, case2, case3, case4, case5, case6, mdlDialog_open
Description: initializes the data structures to hold the results and calls the corresponding function to generate the results in the snowmanEvaluation module.

Called by:

Pseudo code:

```
initialize the min and max offset and elevations
foreach case selected
    put that case number in the case list
call that case’s evaluation function to generate the results
open output options dialog box
```
4.5 Output

Function: getOutputOptions
Version: 1.0
Command number: CMD_GET_OUTPUT_OPTIONS
Input: void
Output: void
Return: void
Functions called: sprintf, mdlOutput_printf, mdlDialog_open, mdlDialog_fileCreate,
                   saveResultsDataFile, exitSnowman, mdlState_startPrimitive
Description: This function gathers the output options from the user (to screen, to file, and/or
             vertical scaling factor (1:1 or 1:5)). If the user selects to send to a file the name of
             the file is input and results written to that file. If the user selects to output to screen
             then they are prompted for the location of the output.
Called by: DIALOGID_OutputOptions via command number
Pseudo code:
  check that the user picked either screen (design file) or file (snowman data file)
  if not
    prompt user to pick one
    re-open the dialog box to get output options (DIALOGID_OutputOptions)
  else user picked at least 1
    if user picked to file (snowman data file)
      prompt user for name of file
      if user gave name
        open/create file
        write results data to file
      else
        re-open DIALOGID_OutputOptions
    if user didn't pick to screen
      call getCaseSelections
    if user did pick to screen
      use mdlState_startPrimitive to get output data point

Function: recordOutputPoint
Version: 1.0
Command number: none
Input: Dpoint3d *pntP pointer to the data point that defines the point where the user wants the
       output to start in the design file
       int view the view number that the user entered the data point in.
Output: none
Return: void
Functions called: plotOutputToScreen
Description: records the data point the user entered to start the output to the design file.
Called by: system via mdlState_startPrimitive in getOutputOptions
Pseudo code:
  check that pntP is not null
  if null
else
    record the x, y, z coordinates of the data point entered
    call plotOutputToScreen

Function: plotOutputToScreen
Version: 1.0
Command number: none
Input: void
Output: void
Return: void
Functions called: sprintf, outputProfile, outputGeneralInformation, outputSectionInformation,
                   outputFenceAndCaseInformation, mdlDialog_open, outputLabel, outputAxis,
                   outputEarthworkCaseInformation
Description: This function calls the appropriate routines to output the various SNOWMAN
results information to the user.
Called by: recordOutputPoint
Pseudo code:
    adjust min and max offset and elevations to the nearest 10 meters
    output the general information about the site with outputGeneralInformation
    for each section
        for each case in the list of selected cases
            output that case’s name and section number
            draw the axis and labels
            draw the original ground profile
            draw the unmitigated snow profile
            if case 1
                output the section information for this case
            if case 2, 3, 4, 5, or 6
                output the section information from case 1
                if section has a mitigated snow profile
                    draw the mitigated snow profile
                    output the fence and mitigated information
                else
                    output that no solution was found
                if case 2 or 4
                    if has shortest fence solution
                        draw the mitigated snow profile for the shortest fence solution
                        output the fence and mitigated information for the shortest fence
data solution
                    else
                        output that no shortest fence solution was found
                else if case 7, or 8
                    output the section information from case 1
                    if section has a mitigated snow profile
                        output the new ground profile
                        output the mitigated snow profile
                        output the earthwork and mitigated information
                    else
                        output that no solution was found
        call getCaseSelections
Function: outputFenceAndCaseInformation
Version: 1.0
Command number: none
Input: DPoint3d *location, the upper left corner of the bounding box where this case and sections output goes
      SnowmnResult *result, pointer to this case and sections results
      int secondFence, unused
Output: void
Return: void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle,
                  mdlCnv_masterUnitsToUors, mdlLineString_create, mdlLevel_setActive,
                  mdlElement_display, mdlElement_add, mdlElmdscr_new, strlen,
                  mdlLevel_setActiveByName, sprintf, mdlText_create
Description: Draws the fence (if present) and storage markers (if stored upwind of ULOP).
             Outputs all the associated text with it regarding the mitigated profile and the snow fence (if present).
Called by: plotOutputToScreen
Pseudo code:
create text template
create line template
plot the fence in the proper location on the profile
if the transports are stored upwind of ULOP
   draw an arrow at the point where they are stored and label them
output the following fence information:
   location, actual height, effective height, porosity, if buried
output the following mitigated profile information:
   snow depth at limits of protection (predicted and unlimited) and limit, storage
   at LOS, storage at ULOP, location where transports are stored if upwind of ULOP
output what problems if any the section still has

Function: outputGeneralInformation
Version: 1.0
Command number: none
Input: DPoint3d *location, the upper left corner of the bounding box where this case and sections output goes
Output: none
Return: void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle,
                  mdlCnv_masterUnitsToUors, mdlLineString_create, mdlLevel_setActive,
                  mdlElement_display, mdlElement_add, mdlElmdscr_new, strlen,
                  mdlLevel_setActiveByName, sprintf, mdlText_create
Description: Outputs the information that is general to all sections. User info, climate info, and key
Called by: plotOutputToScreen
Pseudo code:
create a text template
output the following user information
   name, user level, PIN, siteID
output the following climate information
   lat, long, elev, annual snowfall, ambient snow, wind direction, relocation
coefficient, exceedence factor,
output key for the following lines:
   original ground, new ground, mitigated profile, unmitigated profile, snow fence

Function: outputLabel
Version: 1.0
Command number: none
Input: char *label, text string to output
       Dpoint3d *location, location where the text starts (i.e. left side)
       SnowmanLevelStandard *ls, pointer the level standard definition for the label
Output: none
Return:void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle,
                 mdlLevel_setActive, mdlLevel_setActiveByName, mdlLevel_setActive,
                 mdlElement_display, mdlElement_add, mdlElmdscr_new
Description: Outputs the text in label following level standard ls.
Called by: plotOutputToScreen
Pseudo code:
   create a text template
   output the label using the level standard ls

Function: outputAxis
Version: 1.0
Command number: none
Input: Dpoint3d *location, the upper left corner of the bounding box where this case and sections
       output goes
Output: none
Return:void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle,
                 mdlLevel_setActive, mdlLevel_setActiveByName, mdlLevel_setActive,
                 mdlElement_display, mdlElement_add, mdlElmdscr_new
Description: Draws the axis, tick marks, grid lines and labels for the axis
Called by: plotOutputToScreen
Pseudo code:
   create a line template
   create a text template
   set line template for axis level
   plot the axis
   set line template for tick marks
   plot the tick marks
   set the line template for axis labels
   label the tick marks, label the axis
   set line template for grid lines
   draw horizontal and vertical grid lines
Function: outputSectionInformation
Version: 1.0
Command number: none
Input: Dpoint3d *location, the upper left corner of the bounding box where this case and sections output goes
int n, the section number
int plotStorageLocations, TRUE - indicates to plot storage location information FALSE - indicates to skip storage location information
Output: none
Return:void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle,
mdlCnv_masterUnitsToUors, sprintf, strlen, mdlLevel_setActiveByName,
mdlLevel_setActive, mdlElement_display, mdlElement_add, mdlElmdscr_new,
mdlLineString_create
Description: Marks the limit of storage on the profile. Can draw the markers for the locations were transports are stored. Labels the ULOP and DLOP. Outputs the information about the unmitigated snow profile for section n. Note the unmitigated is section dependent not case dependent. Hence you may or may not get the storage locations drawn. To plot them for the mitigated and unmitigated would be confusing.
Called by: plotOutputToScreen
Pseudo code:
create a line template
create a text template
draw lines for the ULOP and DLOP
if plotStorageLocations is TRUE
  if average transport is stored upwind of limit of storage
    plot an arrow at the location where this occurs and label it
  if exceedence transport is stored upwind of the upwind limit of protection
    plot an arrow at the location where this occurs and label it
plot the location of the upwind limit of protection
plot the location of the downwind limit of protection
output the following section information for unmitigated snow profile
  angle between road and wind, limit of storage offset, upwind limit of protection offset, downwind limit of protection offset, fetch, average transport, exceedence transport, depth of snow at ULOP, depth of snow at DLOP, Storage at LOS, storage at ULOP, location where average transport is stored, location where exceedence transport is stored, types of problems if any

Function: outputProfile
Version: 1.0
Command number: none
Input: SnowmanProfile *spP, pointer to the profile to be plotted
SnowmanLevelStandard *slP, pointer to the level standard definition to use to plot the profile
Function: outputEarthworkCaseInformation
Version: 1.0
Command number
Input: Dpoint3d *location, the upper left corner of the bounding box where this case and sections output goes
SnowmanResult *result, pointer to the snowman result data that is to be output for this earthwork case
Output: none
Return:void
Functions called: malloc, mdlText_create, mdlElement_setSymbology, mdlElement_setLineStyle, strlen, mdlLevel_setActiveByName, mdlLevel_setActive, mdlLineString_create, mdLocale_setLineStyle, mdlCnv_masterUnitsToUors, mdlElement_display, mdlElement_add, mdlElmdscr_new, sprintf, mdlText_create
Description: Outputs the section information for this earthwork case contained in result.
Called by: plotOutputToScreen
Pseudo code:
create a text template
output the lines for the storage if stored
output the following text information about the section:
  snow depth (unlimited and predicted) at limits of protection, storage at LOS,
  storage at ULOP, location where average transport stored, location where exceedence transport stored, ditch back and bottom slope, location of the top and bottom of back slope, additional ditch width, volume of earthwork to remove, any problems that may still exist
5 Structural Organization

This section provides visual information about which functions interact with which other functions. The purpose is to show where control and data flow from one module to the next. This section includes descriptions of what each module shown does.
5.1 Overall

This section shows top level interactions among the main parts of the program.

User, climate and topographic data are all collected by the input portion of the program. This information is then passed along to the processing portion which handles the evaluations and designs. The processing portion evaluates unmitigated terrain and existing snow fences. The processing portion also recommends structural snow fences that satisfy the design criteria as well as changes to topography to meet the same design criteria. The output sends those results and evaluations generated by the processing portion to the design file or to a snowman data file.
5.2 Input

Fig. 2 details the different input modules. There are 4 basic types of inputs: general user information, climate data, topographic data and case specific data. Case specific data is detailed in section 5.3 on case selection and processing. The initial inputs consist of user information, followed by the climate information, then the topographic information in that order.

User information consists of information about the user and the site that is not relevant to the evaluation or design of snow mitigation measures. Climate data is that which affects the computations for snow over the accumulation season and associated transports. Topographic data consists of the ground profiles and corresponding section information.
5.2.1 User Information Input

The user information is gathered through the use of getUserInputInformation, DIALOGID_GetUserInputInformation, and getClimateData. getUserInputInformation initializes the fields for the get user input information dialog box. The OK button in that dialog box takes us from the gathering of user information to gathering climate information.

The data obtained in this portion is the userName, userLevel, PIN, siteID, climateInputOption, topoInputOption which correspond to the user’s name, the user’s level, the project ID number, the site identification, the method to use for climate input, the method to use for topographic input respectively.

Figure 3 User Information Input
5.2.2 Climate input

Fig. 4 shows the climate input methods. Climate data input can be by SNOWMAN data file, site specific data, and computation from latitude, longitude and elevation.

![Climate Input Methods Diagram](image)

Figure 4  Climate Input Methods

Input from of site-specific data prompts the user for all inputs. Input of just latitude, longitude, and elevation results in SNOWMAN computing the remainder of the climate data. Input by data file loads all the climate data from a previous run of the program.

The site-specific data input method shown in Fig. 5 prompts the user for the site’s latitude, longitude, and elevation, total annual snowfall, relocation coefficient, prevailing wind direction, and ambient snow cover. The value of the exceedance factor is set to the default value of 1.5. Then the dialog box for displaying the climate data is opened. At that point the user can modify data and has the option to save the data to a data file before continuing on to topographic data input.

The method of inputting just the latitude, longitude and elevation, shown in Fig. 6, results in computation by SNOWMAN for the total annual snowfall, prevailing wind direction, and ambient snow cover. The relocation coefficient and the exceedance factor are both set to the default values. In this method the user is asked to provide only the information about the longitude, latitude and elevation. Once the information is input the remaining values are computed or set by default and displayed to the user. The user will have the option changing values and saving the new data to a snowman climate data file before continuing on to topographic data input.
Figure 5  Site-Specific Climate Data

Figure 6  Climate Data Computed by SNOWMAN
Input from a SNOWMAN data file, shown in Fig. 7, loads the values from a previous run of SNOWMAN. The previous run may have generated the results from either input by site specific data, or by computation from latitude, longitude and elevation. In this case the user is prompted for the name of a SNOWMAN climate data file and once provided the file is opened and read in. If the file is corrupted or not a valid snowman climate data file the user is prompted again for the name of a valid SNOWMAN climate data file. Once the file is successfully loaded in the information is displayed to the user for review and possible modification. The user has the option to save the data to a new file or overwrite the file loaded before continuing on.

![Diagram of Snowman Climate data file process]

Figure 7 Climate Data from File
5.2.3 Topographic Input

The topographic input methods, as shown in Fig. 8, consist of input by snowman topographic data file, manual input of 2 cross sections and section information, and automated generation from level standard compliant MicroStation design (DGN) file. This topographic information is needed so snow profiles can be generated.

![Figure 8  Topographic Data Input](Image)

Input from snowman topographic data file, shown in Fig. 9, loads all the necessary topographic information from a file that was generated from a previous run of snowman. The previous run could have generated the information by either design file or manual input. The manual input results in the user being prompted for section profiles and section information for each section to be input through a series of dialog boxes. The input by design file prompts the user for a starting and ending location along the upwind limit of protection line. The ground profiles and the cross section information are then extracted from the elements in the design file.

Input from snowman data file is similar to input from climate data file. That is, all the relevant topographic information is read back in from a previous run and save of topographic data. This includes all the offset / elevation pairs for the ground profile as well as the locations of the upwind and downwind limits of protection, indication if there is a ditch and location of the lowest point of the ditch if present, the value of fetch and the angle between the road and the direction of the prevailing wind. If all the data in the file provided by the user is valid then the data is displayed back to the user where they can review each section. If the file is not valid then the user is prompted again for the name of a valid snowman topographic data file. After a valid file is loaded and the user has reviewed the sections they have the option of saving the data to a file before continuing to processing.
Input by 2-D sections, shown in Fig. 10, involves repeated gathering of offset elevation pairs with the use of the DIALOGID_GetDataPoint dialog box. Each data point is checked to insure there are no duplicate offsets. Once the data points are gathered, the section information is gathered. This consists of the ditch information (if ditch is present), location of the upwind and downwind limit of protection, fetch, and angle to road. That information is gathered through the use of the DIALOGID_GetSectionInfo dialog box and is checked by the function recordSectionInfo. This process is repeated as long as the user clicks the next section button in the get data point dialog box. The loop ends when the user selects done. At that point the last of the section information is gathered and checked. If at any point the user inputs an invalid value for a field they are prompted for a correct value. Once all data is gathered and checked the information is displayed to the user through the DIALOGID_DisplayTopoData where the user can review each section and has the option to save the data to a snowman topographic data file before continuing on to processing.
Input from design file, shown in Fig. 11, starts by searching the design file for elements of interest (triangles, ditch, ULOP, DLOP, and fetch) and storing the access to those elements in the sElements variable.

After the elements of interest are found, the user is prompted to enter a data point along the upwind limit of protection. This is done through the use of the mdlState_startPrimitive command and the recordStartingPoint and recordEndingPoint functions. Both of those functions use the validULOPPoint to insure the user entered a point along the upwind limit of protection. If they did not then they are prompted until they do.

After the starting and ending locations are called the generateCrossSections function determines the number of cross sections needed to cover the area the user identified. Each cross section is drawn and the intersections with the triangle elements determine the ground profile. The intersection with the other lines of interest allow the function to determine the fetch, location of the upwind and downwind limits of protection and the location of the lowest ditch point if there is a
After all the sections are generated the displayTopoData function is used to display the data to the user for review. The user can select to record the data to a file before continuing on to processing.

Figure 11  Topographic Input from Design File
5.3 Processing

This section details the different modules used in the processing of cross sections. That is, these modules determine what cases the user wants and gathers the case-specific data for those cases before calling the appropriate functions to produce the requested results or analysis. There are 3 main sections: determining what cases to run, collecting the relevant data for those cases, then generating the results (fences, snow profiles, storage computations, depth predictions, etc.)

Get case selections determines which of the available cases the user would like to run for the current set of cross-sections and climate data. The available cases are as follows:

1. evaluate existing terrain,
2. user selects fence type (permanent, portable, temporary) and SNOWMAN determines the fence height, porosity, and location,
3. user selects fence height and porosity and SNOWMAN determines location,
4. user specifies fence type and maximum setback and SNOWMAN determines fence height, porosity and actual location,
5. user specifies fence height, porosity and maximum setback and SNOWMAN determines actual location,
6. user specifies actual fence height, porosity, and actual location and SNOWMAN determines the profiles and computes the storage and depths.

The first time through the processing portion of the program, the set of cases is pre-determined. Only the evaluate existing terrain is run. In all subsequent runs the user selects the cases to run.
5.3.1 Get Case Selection

Get Case Selections, shown in Fig. 13, uses the dialog box DIALOGID_CaseSelection to get the selection set of cases to run. The set is checked to ensure that it is not empty. If the user did not select a set of cases to run they are prompted for a set of cases. Once they choose which cases they would like to run the program moves on to collect the specific data related to the cases that were chosen.

Figure 13  Get Case Selections
5.3.2 Get Case Specific Data

Getting the case specific data, shown in Fig. 14, is done through a sequence of functions. It starts with a call to the function recordCase2Data. If case 2 was selected, the DIALOGID_Case2Data dialog box is used to collect the specific information to case 2. Once all the data is collected for each section, or case 2 was not in the set of selected cases then recordCase2Data calls recordCase3Data. If case 3 was selected, the DIALOGID_Case2Data dialog box is used to collect the specific information for case 3. Once all the data is collected for each section or case 3 was not in the set of selected cases then recordCase3Data calls recordCase4Data. The sequence continues until recordCase8Data is called. Instead of calling the next recordCaseXData generateResults is called instead, since Case 8 is the last case SNOWMAN handles.

![Diagram](image-url)

Figure 14 Get Case-Specific Data
5.3.3 Generate Results

This section details both the functions that are called to generate the results: Case1, Case2, Case3, etc., and the support functions that each of those case functions uses. These functions are all contained in the evaluation module of the program.

The function generateResults, shown in Fig. 15, calls only the cases that need to be called, unlike the way the case specific data is acquired. That is, just the cases that are in the case selection set get called.

Figures 16 – 23 show the generation of results for cases 1 through 8 in turn.
Figure 16 Case 1 Results

Figure 17 Case 2 Results
Figure 18 Case 3 Results
Figure 19  Case 4 Results
Figure 20  Case 5 Results
Figure 21  Case 6 Results

Figure 22  Case 7 Results
Figure 23  Case 8 Results
5.4 Output

This section shows the functions that are responsible for output. The output options are gathered; then the results for the cases from the last run are output. Once that is complete the control returns to gathering the list of cases for the next run.

Get Output Options, shown in Fig. 25, consists of prompting the user for where the output should go. There are two options for where to send the results. The results can go either to a SNOWMAN results data file, or they can go to the design file (screen). The only other output option is to select a scale factor when output goes to the screen. For that, there are only two choices, either 1:1 or 5:1 scale factor for the vertical scale. When output goes to the screen the user is prompted for a data point for the upper left corner where the output is to start.

As shown in Fig. 26, outputting the results to a data file consists of opening/creating a data file and using the saveResultsDataFile function to output the data to a file. Outputting the results to the screen involves plotting the information to the current MicroStation design file.
Figure 25  Get Output Options

Get Output Options

- DIALOGID_OutputOptions
- DIALOGID_GetOutputPoint
- recordOutputPoint

Output Results

Figure 26  Results Output to Data File or Screen

Output Results

- saveResultsDataFile
- plotOutputToScreen

Get Case Selections
Saving the results to a data file only involves the call to one function. Plotting the output to the screen involves several functions since there are several things to plot and different ways the data is plotted, as shown in Fig. 27.

Figure 27  Results Output to Screen