Some tools for three-dimensional modelling in structural geology and tectonics

Report

Author(s):
Maxelon, Michael

Publication date:
2004

Permanent link:
https://doi.org/10.3929/ethz-a-004719066

SOME TOOLS FOR THREE-DIMENSIONAL MODELLING IN STRUCTURAL GEOLOGY AND TECTONICS

Michael Maxelon

Geologisches Institut, ETH Zentrum, 8092 Zürich, Switzerland

Contents

1 Introduction

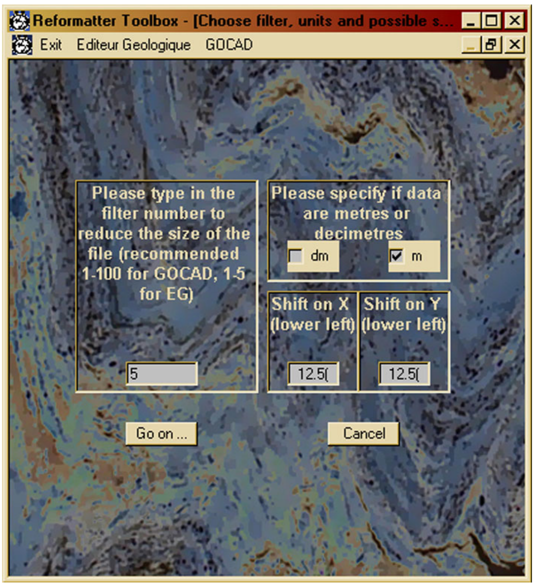
Two small collections of computer routines for PCs running on a Microsoft Windows© system are described in this text. One is written in VisualBasic (VB) as a stand-alone application, the other is written in VisualBasic for Applications© (VBA) in the ArcMap© environment. The VB-application is called *Reformatter Toolbox* and can be used especially for changing formats of digital elevation models (DEM) available as ASCII-files so that they comply with the requirements of some geoscientific modelling tools (GOCAD© or Editeur Géologique[1]). Useful preprocessing routines in ArcInfo© GIS are also described. The VBA-macros provide tools for data export and structural data assessment in ArcView© GIS (mainly ArcMap©). They are organised in a toolbar called *Export Toolbox* within the ArcMap© environment, but are also available independently. The source codes of all tools are also listed in the appendix (page 28 onwards).

2 Dealing with Digital Elevation Models

The programme *Reformatter Toolbox* (Maxelon, 2004j) contains three main menus called *Exit, Editeur Geologique* and *GOCAD* (see Fig. 1). Their meaning is obvious: *Exit* terminates the programme, *Editeur Geologique* provides a routine to export a DEM to the 3D modelling tool Editeur Géologique and *GOCAD* provides three routines that produce files readable by the 3D modelling tool GOCAD©. The input ASCII-file of a DEM must have the following format (z-values are in decimetres in this example):

```
ncols (number)          [number of columns]
nrows (number)          [number of rows]
xllcorner (coordinate)  [x-coordinate of lower left corner]
yllcorner (coordinate)  [y-coordinate of lower left corner]
cellsize (number)       [distance between two rows/columns]
 19389 19512 19618 ...  [elevation values]
```

[1] developed by BRGM – 3, avenue Claude-Guillemin – BP 6009 – 45060 Orléans Cedex 2 – France; http://3dweg.brgm.fr/



Reformatter Toolbox - [Choose filter, units and possible s...
Exit
Editeur Geologique
GOCAD
Go on ...
Cancel

Please type in the filter number to reduce the size of the file (recommended 1-100 for GOCAD, 1-5 for EG): 5

Please specify if data are metres or decimetres: □ dm ☑ m

Shift on X (lower left): 12.5(

Shift on Y (lower left): 12.5(

Figure 1: Screenshot of the programme *Reformatter Toolbox*. The DEM export routine is active, allowing specification of a filter density, of the measured unit and of a possible shift.

The input file must correspond to this format. It is the standard format obtained if an ESRI grid is exported from ArcInfo© with the command line tool *GRIDASCII*. Thus this format should be commonly available. The syntax of the *GRIDASCII* tool is:

```
GRIDASCII <full_name_of_the_grid> <full_name_of_the_output_file>
```

2.1 Export of a DEM to Editeur Géologique

The routine *DHM 2 EG* in the menu *Editeur Geologique* opens the subform shown in Fig. 1. The filter number specifies the number of lines and rows of the GRID to be ignored. So if for instance ”5” is specified, only the 5^{th} line and row, the 10^{th} line and row, the 15^{th} line and row ... etc will be exported. Specifying that data are in decimetres simply multiplies the input data with a factor ”0.1”, specifying metres leaves them unchanged. Other measures for conversion are not supported. In this case data is exported *as* *is*. The values given in the *shift boxes* will be added to the cellsize in the x- or y-direction (default value is zero; usually identical shifts are used for x- and y-direction). The specifications are confirmed by pressing the button *Go on ...*, which also opens the next interactive form. This form shows a standard file selection environment, in which the drive can be chosen from a drop-down list. The respective folder and the input file are selected by doubleclicking. If the export was successful, this is confirmed in a message box.

The exported file has the identifier *.semi*. Its format for usage in the Editeur Géologique looks like this (z-values are metres in this example):

```
W XMIN= (number) XMAX= (number) YMIN= (number) YMAX= (number) ...
 ...NUMBERX= (number) NUMBERY= (number)
 641000  143950  2698
 641100  143950  2697
 641200  143950  2691
 .
    X       Y      Z
 ..
 ... more XYZ-data-sets
```

As can be inferred from the first line of this example Editeur Géologique needs a constant distance between rows and lines, although all points of the DEM are fully specified (with X-, Y- and Z-coordinate).

2.2 Export to GOCAD©

2.2.1 DHM 2 GOCAD

The routine *DHM 2 GOCAD* in the menu *GOCAD* of the *Reformatter Toolbox* opens the same subform already discussed in chapter 2.1 (see also Fig. 1). However, in this context the filter number specifies how many single values are to be ignored. So if ”3” is specified, only the 3rd, 6th, 9th ... etc elevation number of the input file are exported. The other parameters of this form have the same meaning as already discussed in chapter 2.1, and the subsequent procedures are also identical.

The exported file is a standard GOCAD© point set with the identifier *.vs*. The format looks like this (units are metres in this example):

```
GOCAD VSet 0.01
HEADER{
name:(your_filename_here)
}

VRTX 0  654987  157987  3044
VRTX 1  661237  157987  2087
VRTX 2  667487  157987  1909
  .
VRTX n    X       Y      Z
  ..
  ...
```

2.2.2 Ungenerated 2 GOCAD

As can be inferred from the file format example in chapter 2.2.1, every point in a GOCAD© point set is also fully defined by X-, Y- and Z-coordinates. Moreover these data can be inhomogeneously distributed in space. Because of this the ArcInfo© command line tool **V**ery**I**mportant**P**oints can be used to reduce the amount of data significantly, while still maintaining their regional relevance. In other words, in areas where there is little change in z-values (e.g. a wide plane) more data will be left out, whereas on a crest or in a narrow gorge with rapid changes in z-values most data will be kept. The input data format is an ESRI grid. The usage of the tool is as follows:

1. `type` *vip* `name_in name_out [Percentage of reduction]`
   `e.g. vip D:\WorkSpace\Grids\1251_gr1 D:\WorkSpace\Grids\1251_red 15`

2. `change to 'Arcedit' simply by typing` *arcedit*

3. `type` *mape* `name_out`
   `e.g. mape D:\WorkSpace\Grids\1251_red`

4. `type` *ec* `name_out (i.e. edit coverage)`
   `e.g. ec D:\WorkSpace\Grids\1251_red`

5. `type` *ef Points* `(i.e. edit feature)`

6. `type` *select all*
7. `type` *calculate* `name_out-`*ID = SPOT* `(this takes a while)`
   `e.g. calculate 1251_RED-ID = SPOT`
8. `type` *save*
9. `leave Arcedit (type` *q*`)`
10. `in ARC type:` *ungenerate POINT* `name_out your_filename`*.txt # FIXED*

The resulting file is of the following format:

```
  19389      672512.500000000000000      158012.500000000000000
  19512      672537.500000000000000      158012.500000000000000
  19618      672562.500000000000000      158012.500000000000000
  19686      672587.500000000000000      158012.500000000000000
 .
 ..
 ...
```

The routine *Ungenerated 2 Gocad* in the *GOCAD* menu of the *Reformatter Toolbox* transforms the exported file into GOCAD© point set format (see page 6). The file selection works as described in chapter 2.1.

2.2.3 Polyline Decomposer

Lines and polygons typically stored in shape files can be exported from ArcMap© to GOCAD© using the tools described in chapter 3.2. However, a shape file usually contains more than one single geometry, so the resulting GOCAD© polyline file (identifier *.pl*) contains numerous lines and polygons. Often it is convenient to have these available as single lines in independent files. This job is done by the routine *Polyline Decomposer* in the *GOCAD* menu of the *Reformatter Toolbox*. The file selection works as described in chapter 2.1. The file format of a polyline file looks like this:

```
GOCAD PLine 0.01
HEADER {
name: your_filename_here
}

ILINE
VRTX 1  697026  112081  1131
VRTX 2  697306  112394  307
VRTX 3  697536  112653 -610
...
SEG 1  2
SEG 2  3
SEG 3  4
...

ILINE
VRTX 1  697153  112223 -7698
VRTX 2  697101  112165 -7053
VRTX 3  696886  111923 -5704
...
SEG 1  2
SEG 2  3
...

END
```

The *Polyline Decomposer* stores every single *ILINE* in a completely independent *.pl*-file.

3 Data Export and Assessment in ArcMap©

ESRI ArcView© GIS (especially the ArcMap© module) is a common tool for map production and data evaluation in structural geology. The macro routines described in the following subchapters provide methods to export data managed in ArcMap© to three-dimensional geoscientific modelling tools (Editeur Géologique, GOCAD©; chapters 3.1, 3.2). Two data assessment and examination tools are presented in chapter 3.3. The tools can be implemented

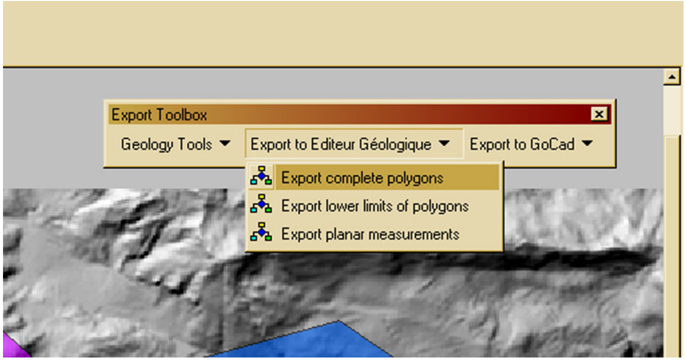

Export Toolbox
Geology Tools
Export to Editeur Géologique
Export to GoCad
Export complete polygons
Export lower limits of polygons
Export planar measurements

Figure 2: Export Toolbox with opened Editeur Géologique menu in ArcMap©. Complete polygons, lower limits of geological units (with respect to a (tectono)stratigraphy defined by sequence of ID numbers) or planar measurements can be exported with the tools of this submenu.

into ArcMap© by opening the file *Moex_tools.mxt* that comes with this publication (Maxelon, 2004g). The map documents (indentifier *.mxd*) based on this template will then have the tools available. Forms and codes included in *Moex_tools.mxt* can also be copied to the standard template in ArcMap© using the internal *Macro Editor* for VBA routines (opens with ALT+F11 within ArcMap©). However, the *Export Toolbox* toolbar (see Fig. 2) contained in *Moex_tools.mxt* will then have to be set up again (Menu *Tools* → *Customize* in ArcMap©), in order to rearrange the references to the standard template. If desired *Moex_tools.mxt* can also completely replace the ArcMap© standard template. To establish this it has to be renamed to *Normal.mxt* and copied into the local ArcMap© template folder[2]. The *Export Toolbox* toolbar would then remain available. Nonetheless you should backup your old *Normal.mxt* for safety reasons. Finally the tools are also available separately as module- and form-files (*.bas* and *.frm/.frx*) for VBA (Maxelon, 2004a,b,c,d,e,f,g,h,i) and can be imported using the *Macro Editor* of ArcMap© (menu *File* → *Import file ...*).

The tools usually require ArcMap© to be in *MapView*-mode. The func-

[2] Usually located at C:\Documents and Settings*your_user_name_here*\Application... ... Data\ESRI\ArcMap\Templates

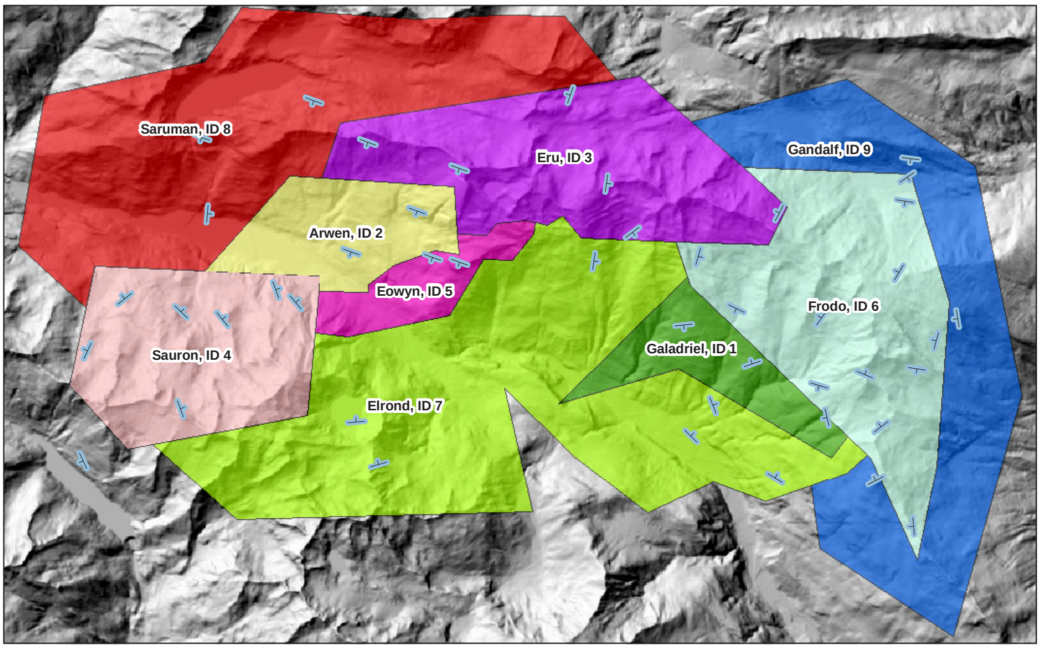


Figure 3: Showcase map with several units identified by a name and an ID number. Representative examples of orientation measurements are also shown. The background shows the hillshade representation of topography as calculated from a DEM.

tionality of the tools will be explained below considering the showcase map illustrated in Fig. 3 as an example. Note: *All geometries involved* ***must*** *refer to identical* ***projected*** *coordinate systems (e.g. UTM, SwissGrid).*

3.1 Export to Editeur Géologique

Two tools to export lines or polygons and one tool to export planar measurements from ArcMap© to Editeur Géologique are included in the *Export Toolbox* (Fig. 2).

3.1.1 Export complete polygons

The *Export complete polygons* routine (Fig. 2; Maxelon, 2004a) exports all lines/polygons of the feature layer selected in the *Table of Contents* of the current map in ArcMap© and stores them in a *.data*-file. File name and path can be specified in the user interface. The exported file format looks like this:

```
9 INTERFACES
INTERFACE XX 4
13 POINTS
695408  151341
697823  151238
...
INTERFACE XX 2
13 POINTS
697853  153293
700647  153106
...
```

The interface IDs ("4" and "2" in the example above) correspond to formation names in the Editeur Géologique (compare Fig. 3). They are read from the ID column of the shape file selected in the Table of Contents in ArcMap©. This field is created by default for every shape file (*.shp*) and is therefore generally available. As a consequence the *ID numbers should be carefully chosen* during creation or modification of a shape file, so as to reflect the (tectono)stratigraphic sequence (lowest number = lowest unit; compare chapter 3.1.2).

This tool can also be used to export polygons or lines from digitised cross-sections to Editeur Géologique.

3.1.2 Export lower limits of polygons

The export routine *Export complete polygons* (chapter 3.1.1; Maxelon, 2004a) works comparatively fast. However, since it exports the full polygons, boundaries between two polygons will be duplicated in the exported file. This shortcoming is avoided in the slower export routine *Export lower limits of polygons* (Fig. 2; Maxelon, 2004d). The selection of the geometries to be exported works the same way as described for the *Export complete polygons* routine (chapter 3.1.1). However, in addition this routine checks the boundaries of the polygons with respect to their (tectono)stratigraphic position in a geologic pile. It is assumed that highest ID values correspond to those units situated in the highest levels of the pile. Because of this the exported *.data*-file for the showcase lithologies in Fig. 3 looks like this:

```
9 INTERFACES
INTERFACE XX 4
3POINTS
695408 151341
697823 151238
697792 150909

INTERFACE XX 3
3POINTS
700710 152112
700647 153106
697853 153293

INTERFACE XX 6
7POINTS
708898 147875
705600 151035
...
```

No points are exported for *Galadriel* and *Arwen* (compare Fig. 3) because their IDs (”1” and ”2”) are the lowest IDs of all and their lower limits do not crop out in the map area. For both *Eru* and *Sauron* (IDs ”3” and ”4”) only three points are exported, namely their boundaries with *Arwen* that has a lower ID. Thus these points are interpreted as markers for the lower limit of *Eru* and *Sauron*.

This tool can also be used to export polygons *or lines* from digitised cross-sections to Editeur Géologique.

3.1.3 Export planar measurements

Editeur Géologique requires planar measurements (e.g. foliations, bedding) to be defined by X- and Y-coordinate, by dip direction and dip angle (in degrees) and by their polarity (or younging direction; +1 or -1). Moreover they should be uniquely assigned to one geologic unit. The Z-value is automatically taken from the separately imported DEM within Editeur Géologique (compare chapter 2.1). The required file also has the identifier *.data* and the following format:

```
45 FOLIATIONS
695816 150340 51 31 1 Sauron
697357 150696 50 30 -1 Sauron
700173 151555 200 80 1 Eowyn
699847 152533 200 80 -1 Arwen
703908 153156 98 78 1 Eru
700736 153422 200 80 1 Eru
707435 152563 298 54 1 Eru
709955 151318 300 65 1 Frodo
710370 149213 189 78 -1 Frodo
710251 145953 265 12 1 Frodo
692793 147316 245 20 -1
693682 150784 321 19 1 Sauron
695312 154075 200 80 -1 Saruman
...
```

The information stored herein is:

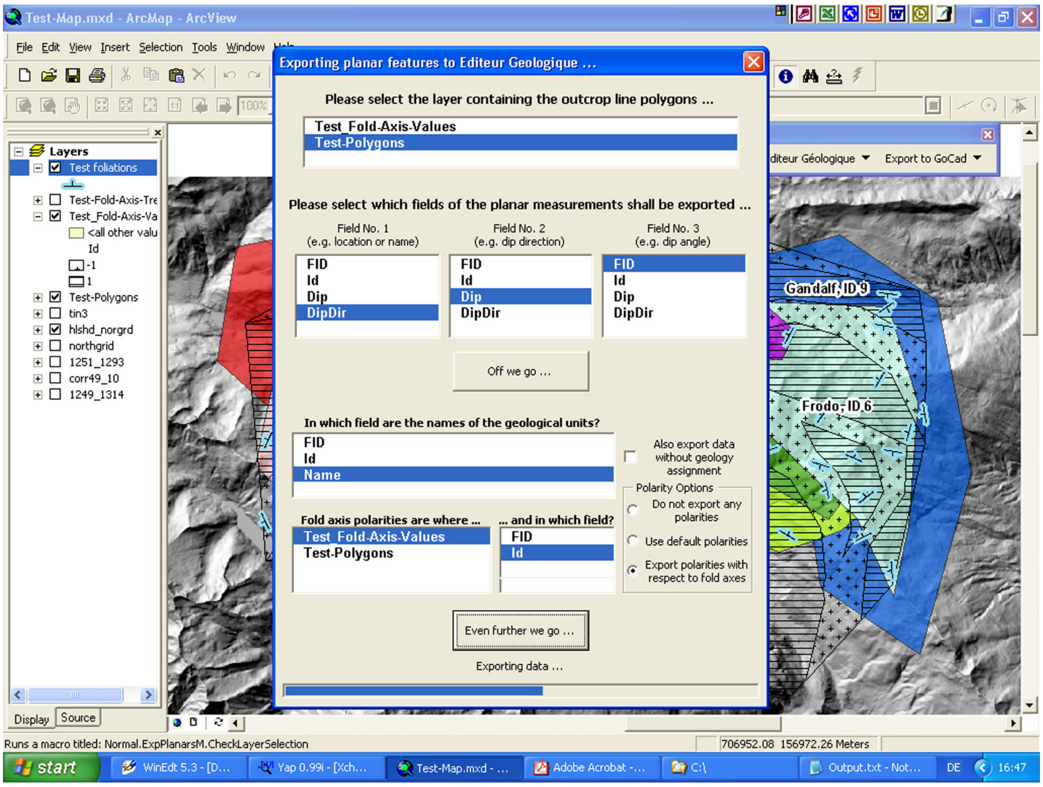
```
X-coordinate  Y-coordinate  dip direction  dip angle  polarity  name
```

The above *.data*-file should be compared to the data shown in Fig. 3. One value (in the third line counted from bottom) is not assigned to a specific unit. The respective measurement is situated in the SE of Fig. 3. The polarity values (±1) have been assigned taking into account an additional feature layer, which specifies the polarities (visible in the background of Fig. 4).

The following paragraphs provide a stepwise instruction manual for the respective export routine:

1. The layer containing the planar measurements to be exported must be selected before the routine *Export planar measurements* (Fig. 2; Maxelon, 2004e) is started (e.g. *Test foliations* in the upper left of the *Table of Contents* in Fig. 4).

2. The routine (user interface shown in Fig. 4) first asks for selection of the feature layer containing the geologic units. The selection requires a confirmation and the routine then opens the menus for further selections. Here the names of the fields containing dip direction and dip angle must be selected.



Exporting planar features to Editeur Geologique ...

Please select the layer containing the outcrop line polygons ...

Test_Fold-Axis-Values
Test-Polygons (selected)

Please select which fields of the planar measurements shall be exported ...

Field No. 1 (e.g. location or name)	Field No. 2 (e.g. dip direction)	Field No. 3 (e.g. dip angle)
FID	FID	FID (selected)
Id	Id	Id
Dip	Dip (selected)	Dip
DipDir (selected)	DipDir	DipDir

In which field are the names of the geological units?

FID
Id
Name (selected)

☐ Also export data without geology assignment

Fold axis polarities are where ...

Test_Fold-Axis-Values (selected)
Test-Polygons

... and in which field?

FID
Id (selected)

Polarity Options
○ Do not export any polarities
○ Use default polarities
● Export polarities with respect to fold axes

Exporting data ...

Runs a macro titled: Normal.ExpPlanarsM.CheckLayerSelection

Figure 4: Screenshot showing the user interface of the *Export planar measurements* routine during data export. The additional transparent layer in the map (compare to Fig. 3) contains the polarity values (horizontal lines: +1; crosses: -1). Usage of the interface is explained in the text.

3. If polarity values have already been stored with the planar measurements, the field they are stored in should be selected in the third box (according to the sequence in which measurements are specified in the *.data*-file described on page 13).

4. If polarity data are contained in another polygon layer (as shown in Fig. 4, fro example) the third box should be left empty.

5. The selection is confirmed by pressing the button *Off we go ...* .

6. If the third box was left empty it has to be confirmed that this happened intentionally. A dummy item will then be selected in the respective list box, but it will not be exported.

7. The field containing the names of the geologic units (e.g. *ID* or *Name*, as chosen in Fig. 4) has to be selected next.

8. With a tick box to the right it can be specified afterwards, if data outside the mapped area are to be exported also.

9. Below the above mentioned tick box, a selection must be made of the way in which polarities will be treated.

   (a) If they are already exported together with the planar measurements (i.e. the name of their field in the respective feature layer is selected in the right list box above, as described before) *Do not export any polarities* should be selected (compare Fig. 4).

   (b) If all data are to be exported with the same polarity value (+1 in this case) *Use default polarities* should be selected.

   (c) If the polarity data are to be assigned based on the position of the data with respect to traces of fold axial planes, *Export polarities with respect to fold axes* should be selected. This subroutine can be helpful if dealing with flat-lying recumbent folds like nappes. In such cases polarity information is not always available immediately. Then another polygon feature layer should be created roughly following the fold axial traces as outlines, because they mark the positions where polarity values change their sign. If this polarity option is chosen, two more boxes open (the two lowermost in Fig. 4), where the name of the feature layer containing the polarity values and the respective data field in that layer must be specified.

10. If all these selections are done, the export is started by pressing the button *Even further we go ...*.

In principal this routine can also be used for other data export purposes dealing with topological relationships (i.e. inside ↔ outside a polygon) of data (e.g. assigning the names of other polygons to various point data).

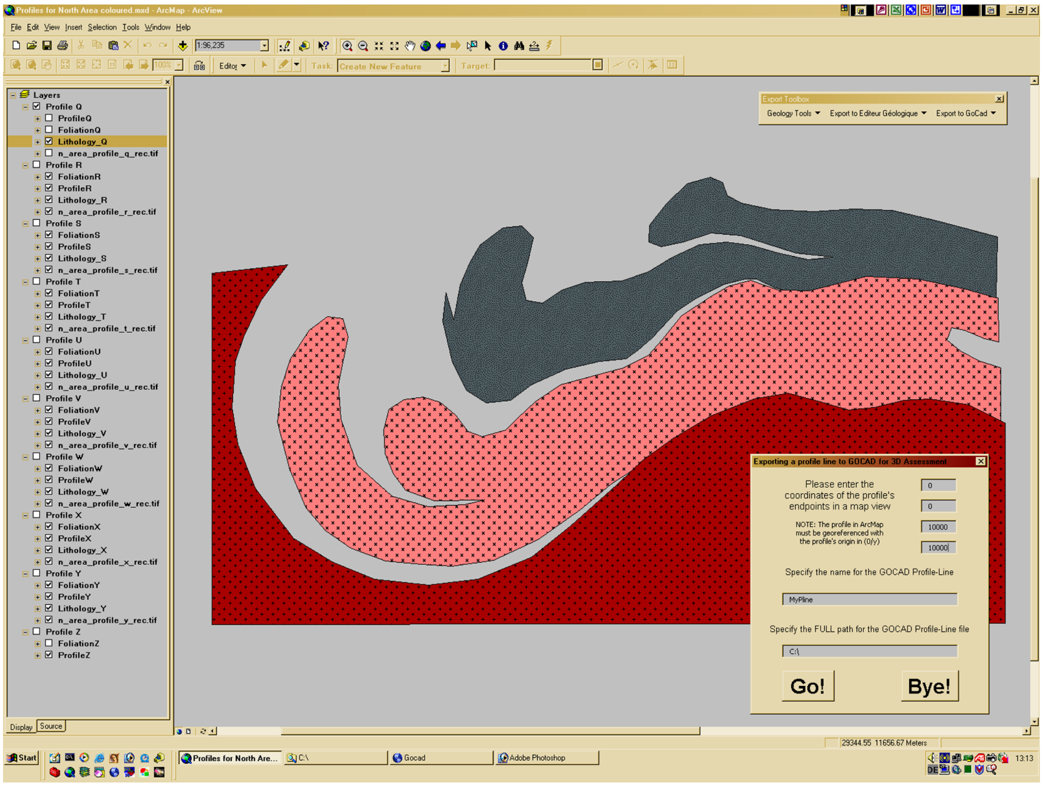
3.2 Export to GOCAD©

Three tools exporting geometries from ArcMap© to GOCAD© are presented. Two of them export lines and polygons in the correct GOCAD©-format (i.e. polylines (*.pl*-files); compare format example on page 8). The third routine exports digitised cross-sections as *.pl*-files, taking georeferenced start- and end-points into account.

3.2.1 Exporting lines and polygons to GOCAD©

Lines and polygons can be exported to GOCAD with two fast and easy-to-handle routines.

- **Lines** (Maxelon, 2004c)
  The feature layer to be exported must be selected in the *Table of Contents* in ArcMap© (like *Test foliations* in the upper left of Fig. 4). The routine for line export is started by clicking the button *Export lines to GOCAD* in the *Export to GOCAD* menu of the *Export Toolbox* (compare Fig. 2). The user interface allows both a path and file name to be specified and is started with the button *GO!*. A message box informs the user when the export is finished and shows where the exported file has been stored.



Exporting a profile line to GOCAD for 3D Assessment

Please enter the coordinates of the profile's endpoints in a map view: 0, 0, 10000, 10000

NOTE: The profile in ArcMap must be georeferenced with the profile's origin in (0/y)

Specify the name for the GOCAD Profile-Line: MyPline

Specify the FULL path for the GOCAD Profile-Line file: C:\

Figure 5: Digitised cross-section in ArcMap©. The user interface to export the cross-section to GOCAD© is active (lower right). The feature layer containing the respective geometries is selected (upper left).

- **Polygons** (Maxelon, 2004f)
  The routine for polygon export works exactly the same way as the routine for line export, but is started with the *Export polygons to GOCAD* button.

3.2.2 Export digitised profiles to GOCAD (Maxelon, 2004b)

The button *Export digitised profiles to GOCAD* in the *Export to GOCAD* menu of the *Export Toolbox* opens the user interface shown in the lower right of Fig. 5. The feature layer containing the geometries to be exported must be selected in the *Table of Contents* in ArcMap© (as shown in Fig. 5). The geometries (polygons or polylines in ArcMap©) are exported as GOCAD© *.pl*-files (an example of the file-format is illustrated on page 8). The export

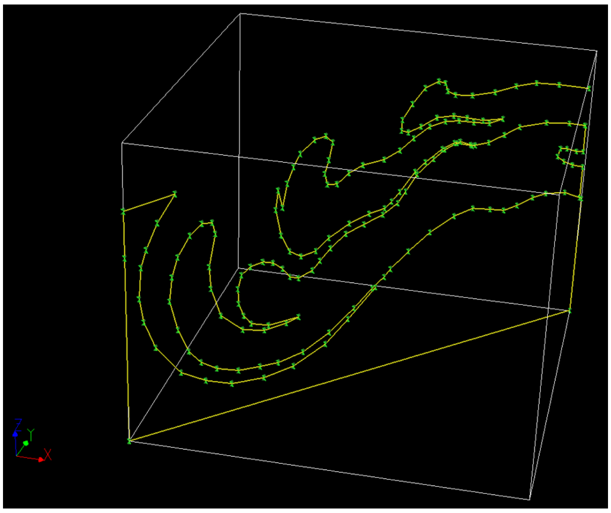


Figure 6: Three-dimensional view of a cross-section exported from ArcMap© to GOCAD© according to the specifications illustrated in Fig. 5. The principal directions of the coordinate system correspond to the limits of the bounding box and are also shown in the lower left of the figure.

routine requires the point(s) defining the left boundary of the digitised cross-section to be defined by the coordinates (0,y). In other words, it is assumed that the section is referenced to its own independent (u,v)-coordinate system with the point of origin at (0,0). The u-coordinate relates to the X-coordinate in the 3D space and the v-coordinate represents the Z-coordinate. Additionally the start- and end-point of the cross-section have to be specified in the user interface. The coordinates entered here must be referenced to the projected coordinate system used for the three-dimensional modelling (e.g. X- and Y-coordinate in SwissGrid notation). From these declarations the correct X- and Y-coordinates of each constituent point in three dimensions are calculated. The showcase numbers entered in Fig. 5 (namely (0,0/10000,10000)) result in an export as visualised in Fig. 6.

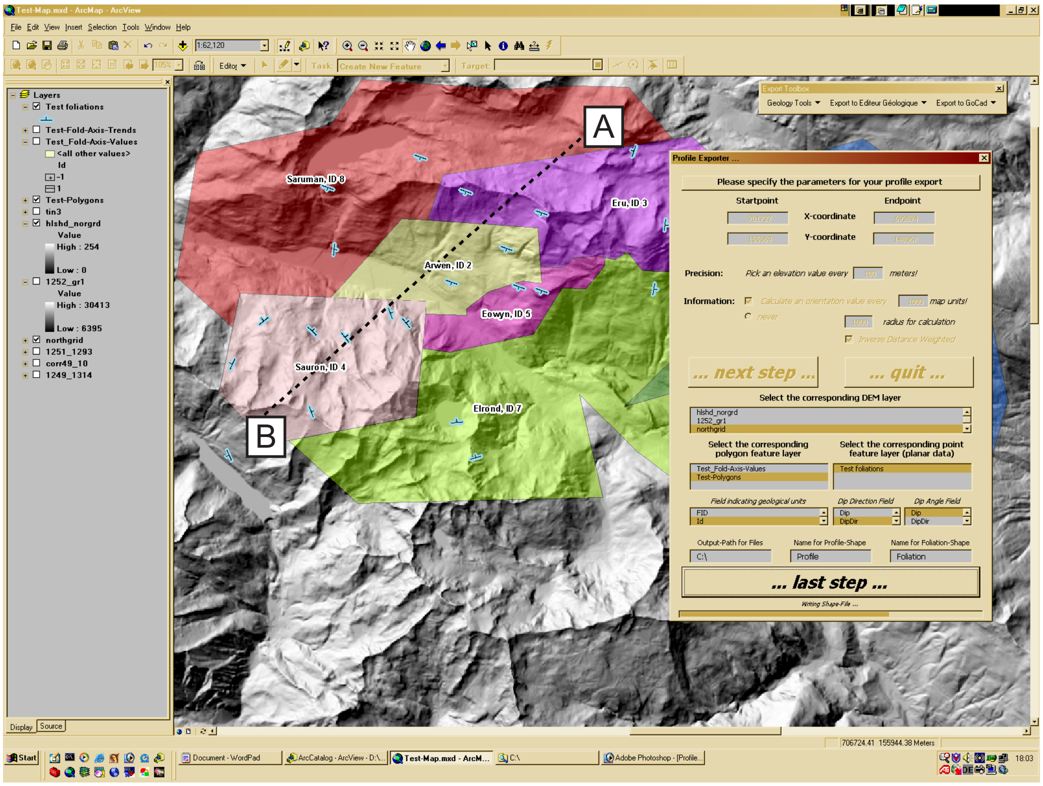
3.3 Data Assessment

The menu *Geology Tools* in the *Export Toolbox* contains routines for two common tasks in geology. The routine *Profile Calculation* automatically calculates a cross-section from a given DEM, also recording information about different geologic units and planar measurements along the profile. The routine *Orientation Averaging* creates a regularly meshed grid of spatially averaged orientations (e.g. foliations) from a given set of typically heterogeneously distributed measurements.

3.3.1 Creating Cross-Sections

This routine (Maxelon, 2004i) opens with the button *Profile Calculation* in the menu *Geology Tools* of the *Export Toolbox*. The user interface is shown in Fig. 7. The start- and endpoints must be specified using the appropriate coordinate system in the respective text boxes (in the upper part of the user interface). The precision of the cross-section is determined by specifying at which interval elevation points are to be spaced – the bigger the interval, the faster (but less precise) the calculation. Furthermore, the interval along the cross-section at which orientation values are to be calculated can be prescribed. These orientation values are calculated from all measurements available within a circular buffer region around the point of interest by component adding (possibly inverse distance weighted, if the respective tick is set → *inverse distance weighted*). The radius of the buffer can also be specified in the user interface by setting the *radius for calculation*.

Once these values have been entered and options set, the button *...next step...* is pressed and the second part of the user interface opens (the upper part of the user interface will then be locked, like it is in Fig. 7). In this step, the appropriate DEM has to be selected first, then the layer containing the polygons corresponding to geologic units and – as the case may be – that containing the planar measurements. The respective boxes in the lower part of the dialogue box then list the data fields available in the chosen feature layers. Although several fields may be listed, *it is highly recommended to **choose the ID field** as the field indicating geological units*, in order to avoid data format incompatibilities. Finally the path and the file name for the output files have to be specified. The export routine is started with the button *...last step...*.



Profile Exporter ...

Please specify the parameters for your profile export

	Startpoint		Endpoint
X-coordinate	701727		693674
Y-coordinate	155369		148367

Precision: Pick an elevation value every 100 meters!

Information: ☑ Calculate an orientation value every 1000 map units!
○ never
1000 radius for calculation
☑ Inverse Distance Weighted

... next step ... ... quit ...

Select the corresponding DEM layer: hlshd_norgrd, 1252_gr1, **northgrid**

Select the corresponding polygon feature layer: Test_Fold-Axis-Values, **Test-Polygons**

Select the corresponding point feature layer (planar data): **Test foliations**

Field indicating geological units: FID, **Id**

Dip Direction Field: Dip, **DipDir**

Dip Angle Field: **Dip**, DipDir

Output-Path for Files: C:\

Name for Profile-Shape: Profile

Name for Foliation-Shape: Foliation

... last step ...

Writing Shape-File ...

Figure 7: Screenshot of ArcMap© showing the user interface of the *Profile Exporter*. The dashed line in the northwestern part of the showcase map delineates the trace of the cross-section (labels A and B indicate start- and endpoints; compare Fig. 8).

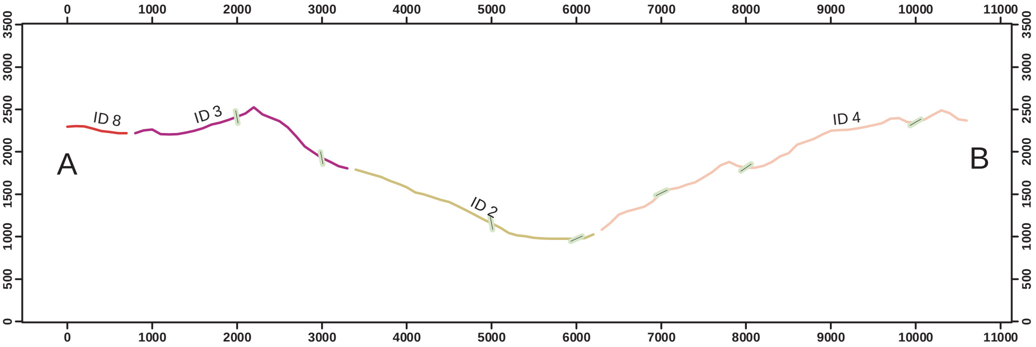


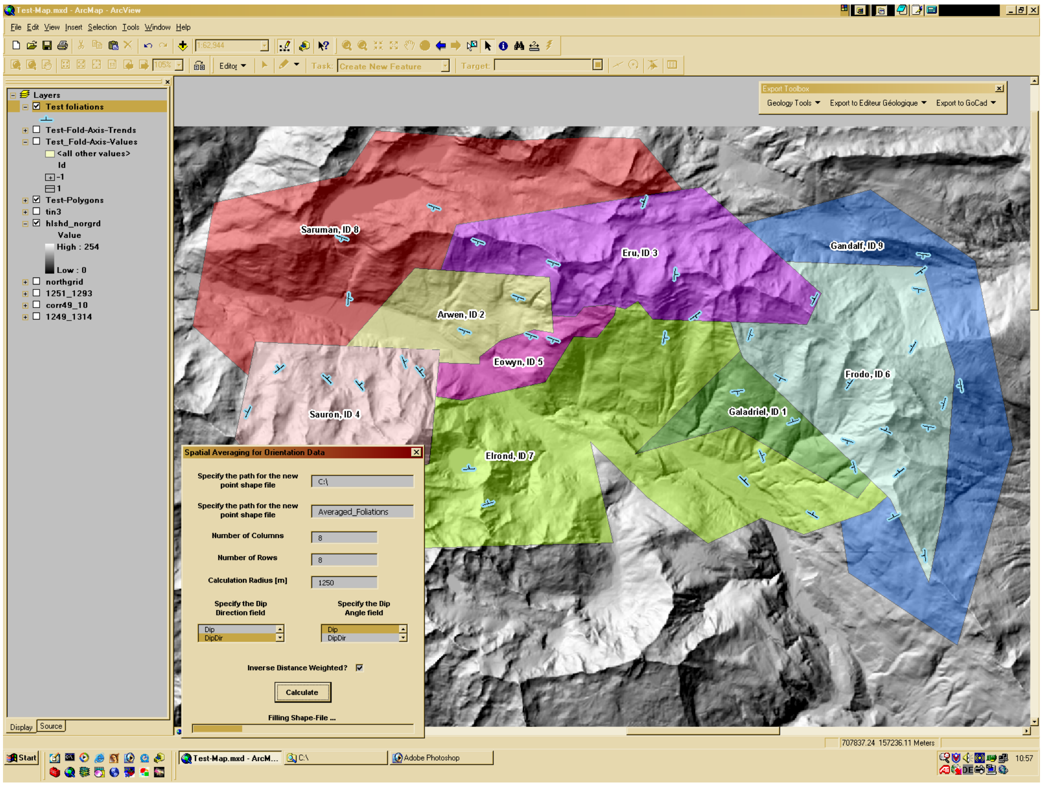
Figure 8: Cross-section calculated with the *Profile Exporter* (compare Fig. 7). The IDs of geologic units are labelled. The apparent dips of orientations projected onto the profile trace are indicated by line markers with a light green background.

The output is stored in two shape files: a line shape, containing the profile line, and a point shape, containing the averaged orientation measurements. Make sure that these files do not already exist, because the routine does not overwrite pre-existing files and will produce an error if you try to do so. The two shape files can then be added to a new map document in ArcMap©. The line shape consists of several single consecutive lines, corresponding to the outcrop zones of geologic units crossed by the profile trace. Choosing an appropriate pattern according to their ID-assignments gives the different parts of the profile line the same colour as the geologic units have in the map (compare Figs. 7, 8). In the point-shape file, three data fields for dip direction, dip angle and apparent dip are created. If the latter is used as a rotation field for a linear symbol representing the averaged point measurements along the cross-section, the changes in orientation along the cross-section can be plotted (as in Fig. 8).

3.3.2 Spatial Averaging of Orientation Data

The routine *Orientation Averaging* (Maxelon, 2004h) is started with the respective button in the menu *Geology Tools* of the *Export Toolbox*. The layer containing the measurements to be averaged must be selected in the *Table of Contents* in ArcMap© (like *Test foliations* in Fig. 9). In the user interface the path and name of the shape file that will store the averaged data must be specified. Make sure that the shape file you want to store does not already exist, because the routine does not overwrite existing files and produces an error when you try to do so.

The numbers of columns and rows specify how closely meshed the grid of averaged data points must be - these numbers should be chosen with respect to the distribution of the input data. Of course the calculation time increases when more points have to be created and calculated. The *Calculation Radius* specifies the size of the buffer (area of averaging) around the newly created points. All measurements within this buffer are taken into account for averaging. If the *Inverse Distance Weighted?* (IDW) tick box is selected, the averaging algorithm also uses IDW to produce the set of averaged data.



Spatial Averaging for Orientation Data

Specify the path for the new point shape file: C:\

Specify the path for the new point shape file: Averaged_Foliations

Number of Columns: 8

Number of Rows: 8

Calculation Radius [m]: 1250

Specify the Dip Direction field: Dip DipDir

Specify the Dip Angle field: Dip DipDir

Inverse Distance Weighted? ☑

Calculate

Filling Shape-File ...

Figure 9: Screenshot of ArcMap©. The routine *Orientation Averaging* is active (lower left) and the layer containing the data to be averaged is selected in the *Table of Contents* (upper left).

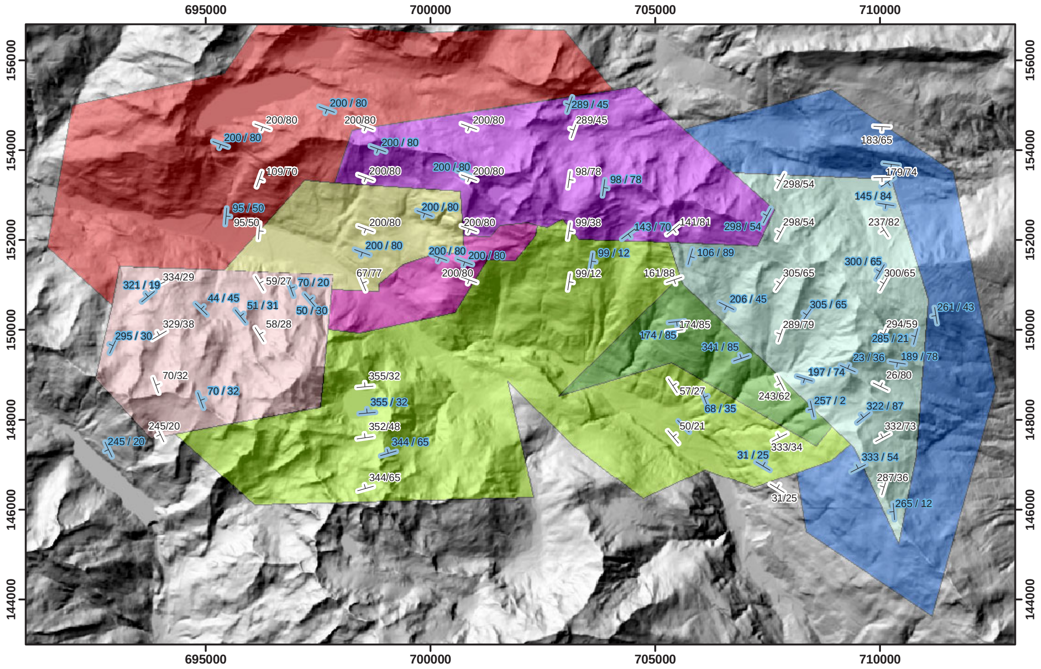


Figure 10: Showcase map illustrating input data (blue background) and results (white background) of spatial averaging (compare Fig. 9). Labels indicate the exact values.

The routine is started by pressing the button *Calculate*. The result of the showcase calculation of Fig. 9 is illustrated in Fig. 10.

The resulting shape-file (*.shp*) contains two important data fields: one is called *DipDir* and provides the dip direction, the other is called *DipAn* and stores the dip angle. If no data were found inside the buffer around the point of calculation, a dummy value of -9999 is stored in both fields. Points with no data can then be identified and easily excluded from plots.

Acknowledgements

I want to thank Martin Brändli, who introduced me patiently and with expert knowledge into programming with VBA© in ArcMap©. Reviews of Diane Seward and Neil S. Mancktelow are also gratefully acknowledged.

Warranty and Liability

DISCLAIMER OF WARRANTY

Since the Software is provided free of charge, it is provided on an *as is* basis, without warranty of any kind, including, without limitation, the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the Software is borne by the user. Should the Software prove defective, the user will assume the entire cost of any service and repair.

LIMITATION OF LIABILITY

Under no circumstances and under no legal theory, tort, contract, or otherwise, shall the author be liable to the user or any other person for any indirect, special, incidental, or consequential damages of any character including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses.

References

Maxelon, M., 2004a. Export complete polygons. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script3.zip.

Maxelon, M., 2004b. Export digitised profiles to GOCAD. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script7.zip.

Maxelon, M., 2004c. Export lines to GOCAD. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script6a.zip.

Maxelon, M., 2004d. Export lower limits of polygons. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script4.zip.

Maxelon, M., 2004e. Export planar measurements. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script5.zip.

Maxelon, M., 2004f. Export polygons to GOCAD. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script6b.zip.

Maxelon, M., 2004g. Moex_tools.mxt. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script2.zip.

Maxelon, M., 2004h. Orientation Averaging. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script9.zip.

Maxelon, M., 2004i. Profile Calculation. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script8.zip.

Maxelon, M., 2004j. Reformatter.exe. In: E-collection. ETH Zürich, http://e-collection.ethbib.ethz.ch/ecol-pool/bericht/bericht_377_script1.zip.

The following chapters list the source codes
of the VisualBasic© programmes
and the Visual Basic for Applications© macros.

A Reformatter Toolbox (VB© codes)

The source codes in this chapter were written in the VisualStudio© 6.0 environment.

A.1 Reformatter Core

The name of this form in the VisualBasic© programming environment is *Reformatter_Core*!

```
Option Explicit
Public MnuSlct As String

Private Sub MDIForm_Resize()
  Select Case Reformatter_Core.WindowState
    Case 0
      Reformatter_Core.Width = 6300
      Reformatter_Core.Height = 6885
    Case 2
      Reformatter_Core.WindowState = 0
      Reformatter_Core.Width = 6300
      Reformatter_Core.Height = 6885
  End Select
End Sub

Private Sub mnuEGdhm2eg_Click()
  Reformatter_Core.Tag = "dhm2eg"
  Filter_Query_dhm_exp.Show
End Sub

Private Sub mnuGCDdhm2gcd_Click()
  Reformatter_Core.Tag = "dhm2goc"
  Filter_Query_dhm_exp.Show
End Sub

Private Sub mnuGCDpllndcmp_Click()
  File_Dialog.Caption = "Polyline Decomposer"
  File_Dialog.Show
End Sub

Private Sub mnuGCDung2gcd_Click()
  File_Dialog.Caption = "Ungenerated 2 Gocad"
  File_Dialog.Show
End Sub

Private Sub mnuODLTpo_Click()
  Unload Me
End Sub
```

A.2 Filter Query

The name of this form in the VisualBasic© programming environment is *Filter_Query_dhm_exp*!

```
Option Explicit

Private Sub Check_dm_Click()
  If Check_m.Value = 1 Then Check_m.Value = 0
End Sub

Private Sub Check_m_Click()
  If Check_dm.Value = 1 Then Check_dm.Value = 0
End Sub

Private Sub Reformat_Click()
  Select Case Reformatter_Core.Tag
    Case "dhm2eg"
      File_Dialog.Caption = "DHM 2 EG"
    Case "dhm2goc"
      File_Dialog.Caption = "DHM 2 GOCAD"
  End Select
  File_Dialog.Show
End Sub

Private Sub Buzz_Off_Click()
  Unload Me
End Sub

Private Sub Xll_shift_Change()
  Yll_shift.Text = Xll_shift.Text
End Sub
```

A.3 File Dialog

The name of this form in the VisualBasic© programming environment is *File_Dialog*!

```
 Option Explicit
 Dim OldFile As String               ' file name of the input file
 Dim NewFile As String               ' file name of the output file
 Dim PurFilNam As String             ' input file name without path
 Dim i As Double                     ' used for for-next loops within the line constituents
 Dim c As Double                     ' used for item counting
 Dim k As Double                     ' used for item counting
 Dim LineReader As String            ' reads whole lines from input file
 Dim LnPrts() As String              ' stores the constituents of a line in an array

 Private Sub Drive1_Change()         ' updates directory view when drive is changed
  Dir1.Path = Drive1.Drive
End Sub

 Private Sub Dir1_Change()           ' updates file view when directory is changed
  File1.Path = Dir1.Path
End Sub

Private Sub File1_DblClick()        ' file chosen to be reformatted

  Dim PointPos As Integer           ' stores the position of the point in the filename

  PointPos = InStr(1, File1.FileName, ".") - 1
  PurFilNam = Left$(File1.FileName, PointPos)
  OldFile = File1.Path & "\" & File1.FileName
  ProgressBar.Visible = True

  Select Case File_Dialog.Caption   ' selects formatting routine based on the choice from Reformatter_Core
    Case "Ungenerated 2 Gocad"
      Call Formatter1
    Case "DHM 2 EG"
      Call Formatter2
    Case "DHM 2 GOCAD"
      Call Formatter2
    Case "Polyline Decomposer"
      Call Formatter3
  End Select
  Unload Filter_Query_dhm_exp
End Sub

 Private Sub Formatter1()              ' Reformats ungenerated files from ArcInfo to GOCAD-
                                       ' vs-Files

  Dim AnswrPrts(3) As Double          ' stores the resorted line constituents

  k = 1
  c = 0
  NewFile = File1.Path & "\" & PurFilNam & ".vs"
  Open OldFile For Input As #1          ' input file opened
  Open NewFile For Output As #2         ' file opened for output
  Print #2, "GOCAD VSet 0.01"
  Print #2, "HEADER{"
  Print #2, "name:"; PurFilNam
  Print #2, "}"
  Print #2, ""
    While Not EOF(1)
      Line Input #1, LineReader         ' reads input file line by line
      LnPrts = Split(LineReader)        ' splits line content into array of strings
      For i = 1 To UBound(LnPrts)       ' checks all strings of the given array
```

```
        If LnPrts(i) <> "" Then         ' chooses strings that are not empty
          AnswrPrts(c) = Round(Val(Trim$(LnPrts(i))))   ' non-empty strings are stored in a separate array
          c = c + 1
        End If
      Next i
      Print #2, "VRTX"; k; AnswrPrts(1); AnswrPrts(2); AnswrPrts(0)     ' strings are plotted in correct sequence
      k = k + 1
      c = 0
    Wend
  Close #1
  Close #2
  MsgBox "Your files have been stored in: " & Chr(13) & NewFile, 64, "Hey Babe!"
  Unload Me
End Sub

 Private Sub Formatter2()              ' Reformats DHM-Files either into Editeur Geologique format
                                      ' or into GOCAD-vs-Files

  Dim CS As Single                    ' stores the cellsize of the DHM
  Dim NCol As Single                  ' stores the number of columns of the DHM
  Dim NRow As Single                  ' stores the number of rows of the DHM
  Dim NDV As Single                   ' stores nodata_value
  Dim Xcnt As Single                  ' stores the increments in x-direction
  Dim Ycnt As Single                  ' stores the increments in y-direction
  Dim Xll As Single                   ' stores x (lower left) of the DHM
  Dim Yll As Single                   ' stores y (lower left) of  the DHM
  Dim Yact As Single                  ' stores the actual value of y for Gocad or the EG
  Dim Fltr As Single                  ' stores the factor by which the DHM will be filtered
  Dim Unt As Single                   ' divides values for metres or decimetres
  Dim PrmtrPos As Single              ' stores UBound(LnPrts) in order to access NCol etc.
  Dim Xmax As Single                  ' stores maximum X according to filtering effects (drop-out of lines)
  Dim Ymin As Single                  ' stores minimum Y according to filtering effects (drop-out of lines)
  Dim NmbX As Single                  ' stores number of columns according to filtering effects
  Dim NmbY As Single                  ' stores number of rows according to filtering effects

  c = 0
  k = 1
  Unt = 1
  Xcnt = 0
  Ycnt = 0
  Fltr = Filter_Query_dhm_exp.Filter_Number.Text    ' reads the filter number from the filter_query...-form
  If Filter_Query_dhm_exp.Check_dm.Value = 1 Then Unt = 10

  Select Case Reformatter_Core.Tag    ' selects file-ending based on choice in the reformatter core
    Case "dhm2eg"
      NewFile = File1.Path & "\" & PurFilNam & ".semi"
    Case "dhm2goc"
      NewFile = File1.Path & "\" & PurFilNam & ".vs"
  End Select
  Open OldFile For Input As #1
  Open NewFile For Output As #2


    While Not EOF(1)                  ' reads input file line by line
      Line Input #1, LineReader
      If LineReader <> "" Then        ' checks that line ist not empty
        LnPrts = Split(LineReader, " ")

        If k < 7 Then
          PrmtrPos = UBound(LnPrts)
          Select Case UCase(LnPrts(0))
            Case "NCOLS"
              NCol = LnPrts(PrmtrPos)
            Case "NROWS"
              NRow = LnPrts(PrmtrPos)
            Case "XLLCORNER"
              Xll = LnPrts(PrmtrPos)
```

```
   Case "YLLCORNER"
    Yll = LnPrts(PrmtrPos)
   Case "CELLSIZE"
    CS = LnPrts(PrmtrPos)
   Case "NODATA_VALUE"
    NDV = LnPrts(PrmtrPos)
    ProgressBar.Min = c + 1
    ProgressBar.Max = NCol * NRow
  End Select
  If k = 6 Then

  Xll = Xll + Filter_Query_dhm_exp.Xll_shift.Text
  Yll = Yll + Filter_Query_dhm_exp.Yll_shift.Text
  Yact = Yll + (NRow - 1) * CS

   Select Case Reformatter_Core.Tag
    Case "dhm2goc"
     Print #2, "GOCAD VSet 0.01"
     Print #2, "HEADER{"
     Print #2, "name:"; PurFilNam
     Print #2, "}"
     Print #2, ""

    Case "dhm2eg"                        ’math explained below
     Xmax = Int((NCol - 1) / Fltr) * CS * Fltr + Xll
     Ymin = Yll + CS * (NRow - Int((NRow - 1) / Fltr) * Fltr - 1)
     NmbX = Int((Xmax - Xll) / (CS * Fltr)) + 1
     NmbY = Int((Yact - Ymin) / (CS * Fltr)) + 1
     Print #2, "W XMIN="; Xll; "XMAX="; Xmax; "YMIN="; Ymin; "YMAX="; Yact; "NUMBERX="; NmbX; "NUMBERY="; NmbY
   End Select

  End If
 End If

 If k > 6 Then
  For i = 0 To UBound(LnPrts)
   If LnPrts(i) <> "" Then

    Select Case Reformatter_Core.Tag
     Case "dhm2eg"                        ’math explained below

      If (Int((Xcnt) / Fltr) = (Xcnt) / Fltr) And (Int(Ycnt / Fltr) = Ycnt / Fltr) Then
       Print #2, Int(Xll + Xcnt * CS); Int(Yact); Int(LnPrts(i) / Unt)
      End If

     Case "dhm2goc"
      If Int(c / Fltr) = c / Fltr Then
       Print #2, "VRTX"; c / Fltr; Int(Xll + Xcnt * CS); Int(Yact); Int(LnPrts(i) / Unt)
      End If
    End Select

    ProgressBar.Value = c + 1

    c = c + 1
    Xcnt = Xcnt + 1

    If Int(Xcnt / NCol) = Xcnt / NCol Then
     Ycnt = Ycnt + 1
     Xcnt = 0
     Yact = Yact - CS
    End If

   End If
  Next i
 End If
k = k + 1
```

```
      End If

    Wend
  Close #1
  Close #2
  ProgressBar.Visible = False
  ProgressBar.Value = ProgressBar.Min
  MsgBox "File(s) stored in: " & Chr(13) & NewFile, 64, "Hey Babe!"
  Unload Me
End Sub
 '
 'calculation of Xmax after filtering:
 '   delta x = -(xmin - xmaxorig)
 '   stepsize = cellsize * filter-factor
 '   => xmax = Int [ delta x / stepsize ] * stepsize + xmin
 '           = Int [ ( xmax - xmin ) / ( cellsize * filter ) ] * ( cellsize * filter ) + xmin
 '      xmax = xll + (Ncol - 1) * cellsize
 '   => xmax


 Private Sub Formatter3()            ' decomposes a polyline into its constituents
                                    ' saving them as single files

  k = 1
  NewFile = File1.Path & "\" & PurFilNam & k & ".pl"
  Open OldFile For Input As #1
  Open NewFile For Output As #2


  While Not EOF(1)
    Line Input #1, LineReader
    If LineReader <> "" Then

      LnPrts = Split(LineReader, " ")

      If LnPrts(0) = "ILINE" And k > 1 Then
        Print #2, "END"
        Close #2
        NewFile = File1.Path & "\" & PurFilNam & k & ".pl"
        Open NewFile For Output As #2
        Print #2, "GOCAD Pline 0.01"
        Print #2, "HEADER{"
        Print #2, "name:"; PurFilNam & k
        Print #2, "}"
        Print #2, ""
        k = k + 1
      End If

      If LnPrts(0) = "ILINE" And k = 1 Then k = 2

      Print #2, LineReader
    End If
  Wend
  Close #1
  Close #2
  MsgBox "Your files have been stored in: " & Chr(13) & File1.Path, 64, "Hey Babe!"
  Unload Me
End Sub
```

B Export Toolbox (VBA© codes)

The source codes in this chapter were written in the VBA© *Macro Editor* included in the ArcView© 8.2 environment.

B.1 Export to Editeur Géologique

B.1.1 Export complete polygons

The name of this form in the VBA© programming environment in ArcMap© is *Frm_Shp2EG_Full_Polygon*!

```
Private Sub ExitButton_Click()
  Unload Me
End Sub

Private Sub StartShp2Goc_Click()
  Call GetGeometryFullPolygon_2EG
  Unload Me
End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *Mod_Shp2EG_Full_Polygon*!

```
Dim PolygNr As Long

 ' Export of complete polygons (not single points
 ' according to relationship criteria) from ArcMap
 ' into Mif/Mid-format

Sub Export_Shp2EG_Full_Polygon()
  Frm_Shp2EG_Full_Polygon.Show
End Sub

Public Sub GetGeometryFullPolygon_2EG()
  Dim aoiFeatureLayer As IFeatureLayer

  ' Get the selected feature layer

  Set aoiFeatureLayer = GetSelectedFeatureLayer()

  If Not aoiFeatureLayer Is Nothing Then
    LookForGeometries aoiFeatureLayer
  End If
End Sub

Public Function GetSelectedFeatureLayer() As IFeatureLayer
  Dim aoiFeatureLayer As IFeatureLayer

  'Access the actual document
  Dim aoiDoc As IMxDocument
  Set aoiDoc = ThisDocument
  'Verify that the active view is a data frame.
  'Then access the selected layer

  If TypeOf aoiDoc.ActiveView Is IMap Then
    Dim aoiLayer As ILayer
    Set aoiLayer = aoiDoc.SelectedLayer
    If Not aoiLayer Is Nothing Then
      If TypeOf aoiLayer Is IFeatureLayer Then
        Set aoiFeatureLayer = aoiLayer
      Else
        MsgBox "Selected layer is not a FeatureLayer!"
      End If
    Else
      MsgBox "Exactly one Layer must be active!"
    End If
  ElseIf TypeOf aoiDoc.ActiveView Is IPageLayout Then
    MsgBox "Current View is a Page Layout"
  End If
  Set GetSelectedFeatureLayer = aoiFeatureLayer
End Function

 ' adressing the geometries in the selected feature layer
 ' and looping through all features

Public Sub LookForGeometries(aoiFeatureLayer As IFeatureLayer)
  Dim aoiFeatureClass As IFeatureClass
  Dim geometryType As esriGeometryType
  Dim aoiCursor As IFeatureCursor
  Dim aoiFeature As IFeature
  Dim aoiPolygon As IGeometryCollection
  Dim aoiIDField As IField
  Dim aoiIDFieldIndex As Long
  Dim FileParts(2) As String
  Dim saveFileData As String

  Set aoiFeatureClass = aoiFeatureLayer.FeatureClass
  aoiIDFieldIndex = aoiFeatureClass.FindField("ID")

  ' Check the geometry
    geometryType = aoiFeatureClass.ShapeType

    If geometryType = esriGeometryPolygon Or esriGeometryPolyline Then
      FileParts(0) = Frm_Shp2EG_Full_Polygon.MifPath.Value & "\"
      FileParts(1) = Frm_Shp2EG_Full_Polygon.MifName.Value
      FileParts(2) = FileParts(0) & FileParts(1)
      saveFileData = FileParts(2) & ".data"
      Open saveFileData For Output As #1
      Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
      Set aoiFeature = aoiCursor.NextFeature
      Print #1, aoiFeatureClass.FeatureCount(Nothing) & " INTERFACES"
      Do While Not aoiFeature Is Nothing
        PolygNr = aoiFeature.Value(aoiIDFieldIndex)
        Print #1, "INTERFACE XX " & PolygNr
        Set aoiPolygon = aoiFeature.Shape
        DecomposePolygon aoiPolygon
        Set aoiFeature = aoiCursor.NextFeature
      Loop
      Close #1
    MsgBox "Your file has been stored as: " & Chr(13) & saveFileData, 64
    Else: MsgBox "You selected a wrong geometry type!", vbExclamation, "Unable to process!"
      Unload Shp2MifF_Polygon
    End If
End Sub

 ' Function gets a polygon as GeomteryCollection (possibly comprising
 ' more than one part) and decomposes it to find the coordinates
```

```
Public Sub DecomposePolygon(aoiPolygon As IGeometryCollection)

  Dim aoiRing As ISegmentCollection
  Dim aoiCurve As ICurve
  Dim fromPoint As IPoint
  Dim toPoint As IPoint
  Dim segCounter As Long
  Dim lngCounter As Long

  ' Get the Rings
  For lngCounter = 0 To aoiPolygon.GeometryCount - 1
    Set aoiRing = aoiPolygon.Geometry(lngCounter)
    Print #1, aoiRing.SegmentCount + 1 & " POINTS"
    For segCounter = 0 To aoiRing.SegmentCount - 1
      Set aoiCurve = aoiRing.Segment(segCounter)
      Set fromPoint = aoiCurve.fromPoint
      Set toPoint = aoiCurve.toPoint
      Print #1, Round(fromPoint.x, 0); Round(fromPoint.y, 0)
    Next segCounter

    ' close the polygon ...
    Print #1, Round(toPoint.x, 0); Round(toPoint.y, 0)
  Next lngCounter

End Sub
```

B.1.2 Export lower limits of polygons

The name of this form in the VBA© programming environment in ArcMap© is *Frm_Shp2EG_Limits_Polygon*!

```
Private Sub ExitButton_Click()
  Unload Me
End Sub

Private Sub StartShp2Goc_Click()
  prgBr.Visible = True
  Call GetGeometryPolygonLimits
  Unload Me
End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *Mod_Shp2EG_Limits_Polygon*!

```
 Dim PolygNr As Long
 Dim aoiFeatureClass As IFeatureClass
 Dim aoiFeatureClass2 As IFeatureClass
 Dim geometryType As esriGeometryType
 Dim aoiCursor As IFeatureCursor
 Dim aoiCursor2 As IFeatureCursor
 Dim aoiFeature As IFeature
 Dim aoiFeature2 As IFeature
 Dim aoiFeatureLayer As IFeatureLayer
 Dim aoiPolygon As IGeometryCollection
 Dim aoiPolygon2 As IGeometryCollection
 Dim aoiIDField As IField
 Dim aoiRing As ISegmentCollection
 Dim aoiCurve As ICurve
 Dim fromPoint As IPoint
 Dim aoiRing2 As ISegmentCollection
 Dim aoiCurve2 As ICurve
 Dim fromPoint2 As IPoint
 Dim intMidCntr As Integer
 Dim lgGeomCntr As Long
 Dim lgGeomCntr2 As Long
 Dim segCounter As Long
 Dim segCounter2 As Long
 Dim aoiIDFieldIndex As Long
 Dim lgMinID As Long
 Dim dbl As Double
 Dim strCoords() As String
 Dim FileParts(2) As String
 Dim saveFileMif As String

Sub Export_Shp2EG_Limits_Polygon()

  'Shows the respective GUI

  Frm_Shp2EG_Limits_Polygon.Show

End Sub

Public Sub GetGeometryPolygonLimits()
  Dim aoiFeatureLayer As IFeatureLayer
  ' Get the selected feature layer

  Set aoiFeatureLayer = GetSelectedFeatureLayer()
  If Not aoiFeatureLayer Is Nothing Then

    ' Calls the function that indentifies and extracts the geometries
    ' in the selected feature layer

    LookForGeometries aoiFeatureLayer
  Else: Exit Sub
  End If
End Sub

Public Function GetSelectedFeatureLayer() As IFeatureLayer

  'Access the actual document

  Dim aoiDoc As IMxDocument
  Set aoiDoc = ThisDocument

  'Verify that the active view is a data frame.
  'Then access the selected layer

  If TypeOf aoiDoc.ActiveView Is IMap Then
    Dim aoiLayer As ILayer
    Set aoiLayer = aoiDoc.SelectedLayer

    If Not aoiLayer Is Nothing Then
      If TypeOf aoiLayer Is IFeatureLayer Then
        Set aoiFeatureLayer = aoiLayer
      Else
        MsgBox "Selected layer is not a FeatureLayer!"
      End If
    Else
      MsgBox "Exactly one Layer must be active!"
    End If

  ElseIf TypeOf aoiDoc.ActiveView Is IPageLayout Then
    MsgBox "Current View is a Page Layout"

  End If

  Set GetSelectedFeatureLayer = aoiFeatureLayer

End Function

 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub LookForGeometries(aoiFeatureLayer As IFeatureLayer) Dim k
As Integer

  Set aoiFeatureClass = aoiFeatureLayer.FeatureClass
  Set aoiFeatureClass2 = aoiFeatureLayer.FeatureClass
  k = 1
  ReDim strOIDStrg(0)


  aoiIDFieldIndex = aoiFeatureClass.FindField("ID")

    geometryType = aoiFeatureClass.ShapeType

    ' make sure that correct geometry type is selected

    If geometryType = esriGeometryPolygon Then

      ' compose the names for the export files
      FileParts(0) = Frm_Shp2EG_Limits_Polygon.MifPath.Text
      FileParts(1) = Frm_Shp2EG_Limits_Polygon.MifName.Text
      FileParts(2) = FileParts(0) & FileParts(1)

      ' decide if "upper limits" (i.e. only one ID)
      ' or "contacts" (i.e. both adjacent IDs) are exported

      saveFileMif = FileParts(2) & ".data"

      Open saveFileMif For Output As #1

      Call FindMinID

      Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
      Set aoiFeature = aoiCursor.NextFeature

      ' Initialises the progress bar for the export

      Frm_Shp2EG_Limits_Polygon.prgBr.Max = aoiFeatureClass.FeatureCount(Nothing)
      Frm_Shp2EG_Limits_Polygon.prgBrName.Visible = True
      Frm_Shp2EG_Limits_Polygon.Repaint

      ' loops through all features with an ID higher than the lowest ID

      Print #1, aoiFeatureClass.FeatureCount(Nothing) & " INTERFACES"
      Do While Not aoiFeature Is Nothing
      Frm_Shp2EG_Limits_Polygon.prgBr.Value = k
        If aoiFeature.Value(aoiIDFieldIndex) > lgMinID Then
          PolygNr = aoiFeature.Value(aoiIDFieldIndex)
          Set aoiPolygon = aoiFeature.Shape
          DecomposePolygon aoiPolygon
        End If
      Set aoiFeature = aoiCursor.NextFeature
      k = k + 1
      Loop

      Close #1

    MsgBox "Your file has been stored as: " & Chr(13) & saveFileMif, 64
    Else: MsgBox "Guess you selected the wrong file!", vbExclamation, "Impossible to process!"
      Unload Frm_Shp2EG_Limits_Polygon
    End If
End Sub

 ' finds the lowest ID of all the features in the
 ' selected polygon feature

Private Sub FindMinID()

  Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
  Set aoiFeature = aoiCursor.NextFeature

  lgMinID = aoiFeature.Value(aoiIDFieldIndex)

  Do While Not aoiFeature Is Nothing

    If lgMinID > aoiFeature.Value(aoiIDFieldIndex) Then
      lgMinID = aoiFeature.Value(aoiIDFieldIndex)
    End If

    Set aoiFeature = aoiCursor.NextFeature

  Loop

End Sub
```

```
 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub DecomposePolygon(aoiPolygon As IGeometryCollection)

  dbl = 1

  ' loops through all PARTS of the feature

  For lgGeomCntr = 0 To aoiPolygon.GeometryCount - 1
  Set aoiRing = aoiPolygon.Geometry(lgGeomCntr)
    For segCounter = 0 To aoiRing.SegmentCount - 1
      Set aoiCurve = aoiRing.Segment(segCounter)
      Set fromPoint = aoiCurve.fromPoint

      Call BoundaryChecker

    Next segCounter
  Next lgGeomCntr

  If strCoords(0) <> "gibbdnix" Then
    Print #1, "INTERFACE XX " & PolygNr
    Print #1, UBound(strCoords) & "POINTS"

    ' the array containing 'touch-points' is written into the MIF-file

    For segCounter = 0 To UBound(strCoords)
      Print #1, strCoords(segCounter)
    Next

    strCoords(0) = "gibbdnix"
  End If

End Sub

 ' the current point is compared with all other points
 ' in features with a lower ID to look for a "touch-point" and
 ' store it in an array with pre-existing "touch-points"

Private Sub BoundaryChecker()

Dim i As Integer Dim aoiRelOp As esriCore.IRelationalOperator

  Set aoiCursor2 = aoiFeatureClass2.Search(Nothing, False)
  Set aoiFeature2 = aoiCursor2.NextFeature

  ' for every single point of "aoiFeature" all points in all
  ' other features are checked for their relationship to it

  Do While Not aoiFeature2 Is Nothing
    If aoiFeature2.Value(aoiIDFieldIndex) < aoiFeature.Value(aoiIDFieldIndex) Then
      Set aoiPolygon2 = aoiFeature2.Shape
      For lgGeomCntr2 = 0 To aoiPolygon2.GeometryCount - 1
        Set aoiRing2 = aoiPolygon2.Geometry(lgGeomCntr2)
        For segCounter2 = 0 To aoiRing2.SegmentCount - 1
          Set aoiCurve2 = aoiRing2.Segment(segCounter2)
          Set fromPoint2 = aoiCurve2.fromPoint
          If Round(fromPoint2.x, 2) = Round(fromPoint.x, 2) And Round(fromPoint2.y, 2) = Round(fromPoint.y, 2) Then

            ' the array containing the coordinates of all
            ' "touch-points" is filled

            ReDim Preserve strCoords(dbl)
            strCoords(dbl - 1) = Round(fromPoint.x, 0) & " " & Round(fromPoint.y, 0)
            dbl = dbl + 1

            Exit Do
          End If
        Next segCounter2
      Next lgGeomCntr2
    End If
  Set aoiFeature2 = aoiCursor2.NextFeature
  Loop

End Sub
```

B.1.3 Export planar measurements

The name of this form in the VBA© programming environment in ArcMap© is *FrmSlctGeoLayer*!

```
Option Explicit

 Public aoiCrntDoc As esriCore.IMxDocument            ' Active ArcMap application
 Public aoiCrntLyr As esriCore.IFeatureLayer          ' Active Layer in active document
 Public aoiCrntLyr2 As esriCore.IFeatureLayer         ' Layer containing the geology for assignment
 Public aoiCrntLyr3 As esriCore.IFeatureLayer         ' Layer containing the polarity for assignment
 Public aoiEnumLyrs As esriCore.IEnumLayer            ' List of all layers from which the geology is chosen
 Public aoiCrntMp As esriCore.IMap                    ' Active Map
 Public aoiCrntFtCls As esriCore.IFeatureClass        ' Feature Class for points feature (foliations)
 Public aoiCrntFtCls2 As esriCore.IFeatureClass       ' Feature Class for polygon feature (geology)
 Public aoiCrntFtCls3 As esriCore.IFeatureClass       ' Feature Class for polygon feature (polarity)
 Public aoiCrntFt As esriCore.IFeature                ' Current Feature in point feature class
 Public aoiCrntFt2 As esriCore.IFeature               ' Current Feature in polygon feature class
 Public aoiCrntFt3 As esriCore.IFeature               ' Current Feature in polarity feature class
 Public aoiFields As esriCore.IFields                 ' Field Interface to access fields in feature classes
 Public aoiCrntField As esriCore.IField               ' Field at specified index in fields collection
 Public aoiFtCrs As esriCore.IFeatureCursor           ' moving through the features in aoiCrntFtCls
 Public aoiFtCrs2 As esriCore.IFeatureCursor          ' moving through the features in aoiCrntFtCls2
 Public aoiFtCrs3 As esriCore.IFeatureCursor          ' moving through the features in aoiCrntFtCls3
 Public aoiGeomPlg As esriCore.IGeometryCollection    ' group of parts of (polygon) features
 Public aoiRelOpPlg As esriCore.IRelationalOperator   ' checks if points are in/outside a polygon
 Public pUID As IUID                                  ' unique identifier for certain (here polygon) layers
 Public intFltnsCntr As Integer                       ' counts the number of relevant points (foliations)
 Public intFtCntr As Integer                          ' counts the number of features in the point feature
 Public intFtCntr2 As Integer                         ' counts the number of features in the polygon feature
 Public intFtCntr3 As Integer                         ' counts the number of features in the polarity feature
 Public lgFldCntr As Long                             ' loops through fields containing export-data
 Public strLctnFld As String                          ' stores selected field in 'location field'
 Public strDipDirFld As String                        ' stores selected field in 'DipDir field'
 Public strDipAngFld As String                        ' stores selected field in 'DipAn field'
 Public strGeoUntFld As String                        ' stores selected field in 'geology unit field'
 Public strPlrtUntFld As String                       ' stores selected field in 'polarity unit field'
 Public strFldSlctnIndx As String                     ' stores selected field in polygon (geology) layer list
 Public strPrntStrng As String                        ' continuously appended string assembling the final result
 Public varLctnVal As Variant                         ' stores the value of 'location field'
 Public varDipDirVal As Variant                       ' stores the value of 'DipDir field'
 Public varDipAngVal As Variant                       ' stores the value of 'DipAn field'
 Public varPlrtUntVal As Variant                      ' stores the value of 'polarity field'
 Public varGeoUntVal As Variant                       ' stores the value of 'geology unit field'

Public Sub LstBxPlgnLyrs_Change()

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntLyr = aoiCrntDoc.SelectedLayer
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFields = aoiCrntFtCls.Fields

  Select Case MsgBox("So the layer containing the correct polygons is " _
    & FrmSlctGeoLayer.LstBxPlgnLyrs.Value & "?", vbQuestion + vbYesNo, _
    "Correct layer chosen?")

  Case 6

' store the polygon layer in aoiCrntLyr2

  Set pUID = New UID
  pUID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
  Set aoiEnumLyrs = aoiCrntMp.Layers(pUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr2 = aoiEnumLyrs.Next

  Do While Not aoiCrntLyr2 Is Nothing
    If aoiCrntLyr2.Name = FrmSlctGeoLayer.LstBxPlgnLyrs.Value Then
      Exit Do
    End If
    Set aoiCrntLyr2 = aoiEnumLyrs.Next
  Loop

' fill list boxes for selection of value fields

    For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
      Set aoiCrntField = aoiFields.Field(lgFldCntr)

      If aoiCrntField.Name <> "Shape" Then
        With FrmSlctGeoLayer
          .LstBxFldDipDir.AddItem aoiCrntField.Name
          .LstBxFldDip.AddItem aoiCrntField.Name
          .LstBxFldLctn.AddItem aoiCrntField.Name
        End With
      End If
    Next

 ' making list boxes and further menu visible (for choosing labels
 ' of Location, Dip Dir and Dip columns)

    With FrmSlctGeoLayer
      .cmdBtStrt.Visible = True
      .lblDipAngle.Visible = True
      .lblDipDir.Visible = True
      .lblLctn.Visible = True
      .LstBxFldDip.Visible = True
      .LstBxFldDipDir.Visible = True
      .LstBxFldLctn.Visible = True
      .lblPlnrFlds.Visible = True
    End With

  Case 7
    Unload FrmSlctGeoLayer
    Call ExpPlanarsM.ChooseGeologyLayer

  End Select

End Sub

Private Sub cmdBtStrt_Click()

 ' get identification for Location, Dip Direction, Dip columns

  If (LstBxFldLctn.ListIndex = -1) _
    Or (LstBxFldDipDir.ListIndex = -1) _
    Or (LstBxFldDip.ListIndex = -1) Then

      strFldSlctnIndx = "111"

      Select Case MsgBox("Is it ok that there's nothing selected in at least one box?" _
        , vbQuestion + vbYesNo, "Are you sure?")

      Case 6
        If LstBxFldLctn.ListIndex = -1 Then
          LstBxFldLctn.ListIndex = 0
          Mid(strFldSlctnIndx, 1, 1) = "0"
        End If

        If LstBxFldDipDir.ListIndex = -1 Then
```

```
          LstBxFldDipDir.ListIndex = 0
          Mid(strFldSlctnIndx, 2, 1) = "0"
        End If

        If LstBxFldDip.ListIndex = -1 Then
          LstBxFldDip.ListIndex = 0
          Mid(strFldSlctnIndx, 3, 1) = "0"
        End If

      Case 7
        Exit Sub

      End Select

  End If

  strLctnFld = LstBxFldLctn.Value
  strDipDirFld = LstBxFldDipDir.Value
  strDipAngFld = LstBxFldDip.Value

 ' fill list box for selection of field containing the geology polygons

  Set aoiCrntFtCls2 = aoiCrntLyr2.FeatureClass
  Set aoiFields = aoiCrntFtCls2.Fields
  For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
    Set aoiCrntField = aoiFields.Field(lgFldCntr)
    If aoiCrntField.Name <> "Shape" Then
      FrmSlctGeoLayer.LstBxGeoUnits.AddItem aoiCrntField.Name
    End If
  Next

 ' making list boxes and further menu visible (for choosing label
 ' of geology column)

  With FrmSlctGeoLayer
    .cmdBtFrthr.Visible = True
    .LstBxGeoUnits.Visible = True
    .lblGeoUnits.Visible = True
    .chkBxExprtWthtGeol.Visible = True
    .framPlrtOpt.Visible = True
    .LstBxGeoUnits.Height = 120
    .Repaint
  End With

End Sub

Private Sub cmdBtFrthr_Click()

 ' if-clauses preventing errors that result from missing selections

        If LstBxGeoUnits.ListIndex = -1 Then
          MsgBox "Please select the field containing the names of the geological units!", vbCritical + vbOKOnly, "Unit name ambiguities!"
          Exit Sub
        End If

        If optBtPlrtRgrdFlds.Value = False _
          And optBtDfltPlrtAssgn.Value = False _
          And optBtNoPlrt.Value = False Then
            MsgBox "Please specify your polarity parameters!", vbCritical + vbOKOnly, "Polarity ambiguities!"
            Exit Sub
        End If

        If optBtPlrtRgrdFlds.Value = True Then
          If lstBxPlrtUnts.ListIndex = -1 Or lstBxPlrtFld.ListIndex = -1 Then
            MsgBox "Please specify your polarity parameters!", vbCritical + vbOKOnly, "Polarity ambiguities!"
            Exit Sub
          End If
          MsgBox "Data outside the area of well-defined polarities will be assigned a default polarity!", _
            vbInformation + vbOKOnly, "Polarity assignment rules ..."
          strPlrtUntFld = lstBxPlrtFld.Value
        End If

 ' end of the aforementioned if-clauses

  strGeoUntFld = LstBxGeoUnits.Value

  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)

  Open "C:\Output_temp.txt" For Output As #1

  With FrmSlctGeoLayer
    .lblPrgBr.Visible = True
    .prgBr.Visible = True
    .prgBr.Max = aoiCrntFtCls.FeatureCount(Nothing) - 1
    .Repaint
  End With

  intFltnsCntr = 0

  For intFtCntr = 0 To aoiCrntFtCls.FeatureCount(Nothing) - 1

  With prgBr
    .Value = intFtCntr
    .Refresh
  End With

    Set aoiCrntFt = aoiFtCrs.NextFeature

    Call Print_String_Assembler

    ' selecting the polarity assignments for the exported foliations

    Select Case FrmSlctGeoLayer.framPlrtOpt.ActiveControl.Name

    ' default polarities (i.e. 1) will be assigned
    Case "optBtDfltPlrtAssgn"
      strPrntStrng = strPrntStrng & " 1"

    ' polarities will be assigned regarding the fold axis trends
    Case "optBtPlrtRgrdFlds"
      Call PolarityFinder

    End Select

    Call GeologyFinder

  Next intFtCntr

  Close #1

  With FrmSlctGeoLayer
    .lblPrgBr.Caption = "Writing output file ..."
    .prgBr.Max = intFltnsCntr
    .Repaint
  End With

  lgFldCntr = 0

  Open "C:\Output.txt" For Output As #1

    Print #1, intFltnsCntr & " FOLIATIONS"

    Open "C:\Output_temp.txt" For Input As #2
      While Not EOF(2)
```

```
        lgFldCntr = lgFldCntr + 1
        With prgBr
          .Value = lgFldCntr
          .Refresh
        End With
        Line Input #2, strFldSlctnIndx
        Print #1, strFldSlctnIndx
      Wend
    Close #2

  Close #1

  Kill "C:\Output_temp.txt"
  lblPrgBr.Caption = "Finished ..."

  MsgBox ("Your file has been stored in C:\Output.txt! Please rename it immediately" _
    & " to prevent overwriting during next use of this export-macro!")
  Unload Me

End Sub

' refreshes the ListBox showing the fields of the polarity table

Private Sub lstBxPlrtUnts_Change()

  ' store polarity layer in aoiCrntLyr3

  FrmSlctGeoLayer.lstBxPlrtFld.Clear

  Set pUID = New UID
  pUID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"

  Set aoiEnumLyrs = aoiCrntMp.Layers(pUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr3 = aoiEnumLyrs.Next

  Do While Not aoiCrntLyr3 Is Nothing
    If aoiCrntLyr3.Name = FrmSlctGeoLayer.lstBxPlrtUnts.Value Then
      Exit Do
    End If
    Set aoiCrntLyr3 = aoiEnumLyrs.Next
  Loop

  ' fill ListBox with field names of polarity layer

  Set aoiCrntFtCls3 = aoiCrntLyr3.FeatureClass
  Set aoiFields = aoiCrntFtCls3.Fields
  For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
    Set aoiCrntField = aoiFields.Field(lgFldCntr)
    If aoiCrntField.Name <> "Shape" Then
      FrmSlctGeoLayer.lstBxPlrtFld.AddItem aoiCrntField.Name
    End If
  Next

  FrmSlctGeoLayer.lstBxPlrtFld.Height = 50

End Sub

Private Sub Print_String_Assembler()

  strPrntStrng = Round(aoiCrntFt.Extent.xmax, 0) & " " & Round(aoiCrntFt.Extent.ymax, 0) & " "

  Select Case strFldSlctnIndx

  Case "111"
    varLctnVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strLctnFld))
    varDipDirVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirFld))
    varDipAngVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAngFld))
    strPrntStrng = strPrntStrng & varLctnVal & " " & varDipDirVal & " " & varDipAngVal

  Case "110"
    varLctnVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strLctnFld))
    varDipDirVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirFld))
    strPrntStrng = strPrntStrng & varLctnVal & " " & varDipDirVal

  Case "101"
    varLctnVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strLctnFld))
    varDipAngVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAngFld))
    strPrntStrng = strPrntStrng & varLctnVal & " " & varDipAngVal

  Case "100"
    varLctnVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strLctnFld))
    strPrntStrng = strPrntStrng & varLctnVal

  Case "011"
    varDipDirVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirFld))
    varDipAngVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAngFld))
    strPrntStrng = strPrntStrng & varDipDirVal & " " & varDipAngVal

  Case "010"
    varDipDirVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirFld))
    strPrntStrng = strPrntStrng & varDipDirVal

  Case "001"
    varDipAngVal = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAngFld))
    strPrntStrng = strPrntStrng & varDipAngVal

  Case "000"
    strPrntStrng = strPrntStrng & ""

  End Select
End Sub

 ' checks polarity value for respective foliation point

Private Sub PolarityFinder()

  Set aoiFtCrs3 = aoiCrntFtCls3.Search(Nothing, False)

  For intFtCntr3 = 0 To aoiCrntFtCls3.FeatureCount(Nothing) - 1
    Set aoiCrntFt3 = aoiFtCrs3.NextFeature

    If Not aoiCrntFt3 Is Nothing Then
      Set aoiGeomPlg = aoiCrntFt3.Shape
      Set aoiRelOpPlg = aoiGeomPlg
      varPlrtUntVal = aoiCrntFt3.Value(aoiCrntFt3.Fields.FindField(strPlrtUntFld))
      If aoiRelOpPlg.Contains(aoiCrntFt.Shape) Then
        strPrntStrng = strPrntStrng & " " & varPlrtUntVal
        Exit Sub
      End If
    End If

  Next intFtCntr3
  strPrntStrng = strPrntStrng & " 1"

End Sub

 ' checks within which geology polygon the respective
 ' foliation point is situated

Private Sub GeologyFinder()

  Set aoiFtCrs2 = aoiCrntFtCls2.Search(Nothing, False)
  For intFtCntr2 = 0 To aoiCrntFtCls2.FeatureCount(Nothing) - 1
    Set aoiCrntFt2 = aoiFtCrs2.NextFeature

    If Not aoiCrntFt2 Is Nothing Then
      Set aoiGeomPlg = aoiCrntFt2.Shape
      Set aoiRelOpPlg = aoiGeomPlg
      varGeoUntVal = aoiCrntFt2.Value(aoiCrntFt2.Fields.FindField(strGeoUntFld))
      If aoiRelOpPlg.Contains(aoiCrntFt.Shape) Then
        Print #1, strPrntStrng & " " & varGeoUntVal
        intFltnsCntr = intFltnsCntr + 1
        Exit For
      End If
    End If

    If FrmSlctGeoLayer.chkBxExprtWthtGeol.Value = True Then
      If intFtCntr2 = aoiCrntFtCls2.FeatureCount(Nothing) - 1 Then
        Print #1, strPrntStrng
        intFltnsCntr = intFltnsCntr + 1
      End If
    End If

  Next intFtCntr2

End Sub

 ' adjusts polarity list boxes after re-choosing geology name field

Private Sub LstBxGeoUnits_AfterUpdate()
  FrmSlctGeoLayer.LstBxGeoUnits.Height = 120
  With FrmSlctGeoLayer
    .lstBxPlrtFld.Visible = False
    .lstBxPlrtUnts.Visible = False
    .optBtPlrtRgrdFlds.Value = False
  End With
End Sub

 ' adjusts height of polarity field

Private Sub lstBxPlrtFld_AfterUpdate()
  FrmSlctGeoLayer.lstBxPlrtFld.Height = 50
End Sub

 '  visualises fields necessary for polarity assignments
 ' if fold axis trends are to be regarded

Private Sub optBtPlrtRgrdFlds_Click()
  FrmSlctGeoLayer.lstBxPlrtFld.Clear
  With FrmSlctGeoLayer
    .lstBxPlrtFld.Visible = True
    .lstBxPlrtUnts.Visible = True
    .lblPlrtFld.Visible = True
    .lblPlrtUnt.Visible = True
    .LstBxGeoUnits.Height = 50
    .lstBxPlrtFld.Height = 50
    .lstBxPlrtUnts.Height = 50
  End With

End Sub

 ' hides fields necessary for polarity assignments
 ' if default polarities (i.e. 1) are to be assigned
Private Sub optBtDfltPlrtAssgn_Click()
  With FrmSlctGeoLayer
    .lstBxPlrtFld.Visible = False
    .lstBxPlrtUnts.Visible = False
    .lblPlrtFld.Visible = False
    .lblPlrtUnt.Visible = False
    .LstBxGeoUnits.Height = 120
  End With
End Sub

 ' hides fields necessary for polarity assignments
 ' if no polarities are to be assigned
Private Sub optBtNoPlrt_Click()
  With FrmSlctGeoLayer
    .lstBxPlrtFld.Visible = False
    .lstBxPlrtUnts.Visible = False
    .lblPlrtFld.Visible = False
    .lblPlrtUnt.Visible = False
    .LstBxGeoUnits.Height = 120
  End With
End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *ExpPlanarsM*!

```
Sub CheckLayerSelection()

  Dim aoiCrntDoc As esriCore.IMxDocument
  Set aoiCrntDoc = ThisDocument

  If Not aoiCrntDoc.SelectedLayer Is Nothing Then

    If TypeOf aoiCrntDoc.SelectedLayer Is IFeatureLayer Then
      Dim aoiCrntLayer As esriCore.IFeatureLayer
      Dim aoiCrntFtCls As esriCore.IFeatureClass
      Set aoiCrntLayer = aoiCrntDoc.SelectedLayer
      Set aoiCrntFtCls = aoiCrntLayer.FeatureClass

      If aoiCrntFtCls.ShapeType = esriGeometryPoint Then
        Call ChooseGeologyLayer
      Else
        MsgBox ("The selected layer doesn't contain appropriate data!")
      End If

    Else
      MsgBox ("The selected layer is not a feature layer!")
    End If

  Else
    MsgBox ("Please select the layer containing your planar measuremets!")
  End If

End Sub

Public Sub ChooseGeologyLayer()

  Dim aoiLayers As esriCore.IEnumLayer
  Dim aoiCrntLayer As esriCore.ILayer
  Dim aoiCrntFtLayer As esriCore.IFeatureLayer
  Dim aoiUID As New UID
  Dim aoiCrntMap As esriCore.IMap
  Dim aoiCrntDoc As esriCore.IMxDocument
  Dim aoiCrntFtCls As esriCore.IFeatureClass

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMap = aoiCrntDoc.FocusMap
  aoiUID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
  Set aoiLayers = aoiCrntMap.Layers(aoiUID, True)
  aoiLayers.Reset
  Set aoiCrntLayer = aoiLayers.Next
  Do While Not aoiCrntLayer Is Nothing

    If TypeOf aoiCrntLayer Is IFeatureLayer Then
      Set aoiCrntFtLayer = aoiCrntLayer
      Set aoiCrntFtCls = aoiCrntFtLayer.FeatureClass

      If aoiCrntFtCls.ShapeType = esriGeometryPolygon Then
        FrmSlctGeoLayer.LstBxPlgnLyrs.AddItem aoiCrntLayer.Name
        FrmSlctGeoLayer.lstBxPlrtUnts.AddItem aoiCrntLayer.Name
      End If

    End If

    Set aoiCrntLayer = aoiLayers.Next
  Loop

  FrmSlctGeoLayer.LstBxGeoUnits.Height = 120
  FrmSlctGeoLayer.Show

End Sub
```

B.2 Export to GOCAD©

B.2.1 Export lines

The name of this form in the VBA© programming environment in ArcMap© is *Shp2GocadF_Line*!

```
Private Sub ExitButton_Click()
  Unload Me
End Sub

Private Sub StartShp2Goc_Click()
  Call GetGeometryLine
End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *Shp2GocadM_Lines*!

```
Public Sub GetGeometryLine()
  Dim aoiFeatureLayer As IFeatureLayer
  Set aoiFeatureLayer = GetSelectedFeatureLayer()
  If Not aoiFeatureLayer Is Nothing Then
    LookForGeometries aoiFeatureLayer
  End If
End Sub

Public Function GetSelectedFeatureLayer() As IFeatureLayer
  Dim aoiFeatureLayer As IFeatureLayer

  'Access the actual document

  Dim aoiDoc As IMxDocument
  Set aoiDoc = ThisDocument

  'Verify that the active view is a data frame.
  'Then access the selected layer

  If TypeOf aoiDoc.ActiveView Is IMap Then
    Dim aoiLayer As ILayer
    Set aoiLayer = aoiDoc.SelectedLayer
    If Not aoiLayer Is Nothing Then
      If TypeOf aoiLayer Is IFeatureLayer Then
        Set aoiFeatureLayer = aoiLayer
      Else
        MsgBox "Selected Layer is not a FeatureLayer"
      End If
    Else
      MsgBox "Exactly one layer has to be selected!"
    End If
    If TypeOf aoiDoc.ActiveView Is IPageLayout Then
      MsgBox "Current View is a Page Layout"
    End If
  End If
  Set GetSelectedFeatureLayer = aoiFeatureLayer
End Function

 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates
```

```
Public Sub LookForGeometries(aoiFeatureLayer As IFeatureLayer)
  Dim aoiFeatureClass As IFeatureClass
  Dim aoiGeometryType As esriGeometryType
  Dim aoiCursor As IFeatureCursor
  Dim aoiFeature As IFeature
  Dim aoiPolyLine As IGeometryCollection
  Dim aoiIDField As IField
  Dim lgIDFldIndx As Long
  Dim strFlPrts(2) As String
  Dim strSvFl As String

  Set aoiFeatureClass = aoiFeatureLayer.FeatureClass
  lgIDFldIndx = aoiFeatureClass.FindField("ID")

  aoiGeometryType = aoiFeatureClass.ShapeType

    If aoiGeometryType = esriGeometryPolyline Then
      strFlPrts(0) = Shp2GocadF_Line.PlinePath.Text
      strFlPrts(1) = Shp2GocadF_Line.PlineName.Text
      strFlPrts(2) = ".pl"
      strSvFl = Join(strFlPrts, "")
      Open strSvFl For Output As #1
      Print #1, "GOCAD PLine 0.01"
      Print #1, "HEADER {"
      Print #1, "name:"; strFlPrts(1)
      Print #1, "}"
      Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
      Set aoiFeature = aoiCursor.NextFeature
      Do While Not aoiFeature Is Nothing
        Print #1, ""
        Print #1, "ILINE"
        lgPlgnNr = aoiFeature.Value(lgIDFldIndx)
        Set aoiPolyLine = aoiFeature.Shape
        DecomposeCurve aoiPolyLine
        Set aoiFeature = aoiCursor.NextFeature
      Loop
      Print #1, "END"
      Close #1
      MsgBox "File(s) stored in " & strSvFl
    Else: MsgBox "Probably you selected a layer with a wrong geometry type! The Macro will be reset!", _
          vbExclamation, "Wrong geometry type?"
      Unload Shp2GocadF_Line
    End If
End Sub


 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub DecomposeCurve(aoiPolyLine As IGeometryCollection)

  Dim aoiLines As ISegmentCollection
  Dim aoiCurve As ICurve
  Dim aoiFrmPnt As IPoint
  Dim aoiTPnt As IPoint
  Dim intSgmntCntr As Integer
  Dim intStrtCnt As Integer
  Dim intGeomCntr As Integer
  intStrtCnt = 0
  intGeomCntr = 0

    Do
      Set aoiLines = aoiPolyLine.Geometry(intGeomCntr)
      For intSgmntCntr = intStrtCnt To aoiLines.SegmentCount + intStrtCnt - 1
        Set aoiCurve = aoiLines.Segment(intSgmntCntr - intStrtCnt)
        Set aoiFrmPnt = aoiCurve.fromPoint
        Set aoiTPnt = aoiCurve.toPoint
        Print #1, "VRTX"; intSgmntCntr + 1; Round(aoiFrmPnt.x, 0); Round(aoiFrmPnt.y, 0); 0; lgPlgnNr
      Next intSgmntCntr
      Print #1, "VRTX"; intSgmntCntr + 1; Round(aoiTPnt.x, 0); Round(aoiTPnt.y, 0); 0; lgPlgnNr
      intGeomCntr = intGeomCntr + 1
      intStrtCnt = intSgmntCntr + 1
    Loop While intGeomCntr < aoiPolyLine.GeometryCount

    intSgmntCntr = 0
    Do While intSgmntCntr < intStrtCnt - 1
      intSgmntCntr = intSgmntCntr + 1
      Print #1, "SEG"; intSgmntCntr; intSgmntCntr + 1
    Loop

End Sub
```

B.2.2 Export polygons

The name of this form in the VBA© programming environment in ArcMap© is *Shp2GocadF_Polygon*!

```
Private Sub ExitButton_Click()
  Unload Me
End Sub

Private Sub StartShp2Goc_Click()
  Call GetGeometryPolygon
End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *Shp2GocadM_Polygon*!

```
Sub GocadExporter()
  Shp2GocadF_Polygon.Show
End Sub

Public Function GetSelectedFeatureLayer() As IFeatureLayer
  Dim aoiFeatureLayer As IFeatureLayer

  'Access the actual document
  Dim aoiDoc As IMxDocument
  Set aoiDoc = ThisDocument

  'Verify that the active view is a data frame.
  'Then access the selected layer
  If TypeOf aoiDoc.ActiveView Is IMap Then
    Dim aoiLayer As ILayer
    Set aoiLayer = aoiDoc.SelectedLayer

    If Not aoiLayer Is Nothing Then
      If TypeOf aoiLayer Is IFeatureLayer Then
        Set aoiFeatureLayer = aoiLayer
      Else
        MsgBox "Selected layer is not a feature layer!"
      End If
    Else
      MsgBox "Exactly one layer must be active!"
    End If
  ElseIf TypeOf aoiDoc.ActiveView Is IPageLayout Then
    MsgBox "Current view is a Page Layout"
  End If
  Set GetSelectedFeatureLayer = aoiFeatureLayer
End Function


 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub DecomposePolygon(aoiPolygon As IGeometryCollection)
  ' Get the Rings
  Dim aoiRing As ISegmentCollection
  Dim aoiCurve As ICurve
  Dim fromPoint As IPoint
  Dim toPoint As IPoint
  ' Dim lngCounter As Long
  Dim segCounter As Long

    Set aoiRing = aoiPolygon.Geometry(lngCounter)
    For segCounter = 0 To aoiRing.SegmentCount - 1
      Set aoiCurve = aoiRing.Segment(segCounter)
      Set fromPoint = aoiCurve.fromPoint
      Set toPoint = aoiCurve.toPoint
      Print #1, "VRTX"; segCounter + 1; Round(fromPoint.x, 0); Round(fromPoint.y, 0); 0
    Next segCounter
    segCounter = 1
    Do While segCounter <= aoiRing.SegmentCount
      If segCounter <> aoiRing.SegmentCount Then
        Print #1, "SEG"; segCounter; segCounter + 1
          Else: Print #1, "SEG"; segCounter; 1
      End If
      segCounter = segCounter + 1
    Loop

End Sub

 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub LookForGeometries(aoiFeatureLayer As IFeatureLayer)
  Dim aoiFeatureClass As IFeatureClass
  Dim geometryType As esriGeometryType
  Dim aoiCursor As IFeatureCursor
  Dim aoiFeature As IFeature
  Dim aoiPolygon As IGeometryCollection
  Dim aoiIDField As IField
  Dim aoiIDFieldIndex As Long
  Dim FileParts(2) As String
  Dim saveFile As String

  Set aoiFeatureClass = aoiFeatureLayer.FeatureClass
  aoiIDFieldIndex = aoiFeatureClass.FindField("ID")

  ' Check the geometry
    geometryType = aoiFeatureClass.ShapeType

    If geometryType = esriGeometryPolygon Then
      FileParts(0) = Shp2GocadF_Polygon.PlinePath.Text
      FileParts(1) = Shp2GocadF_Polygon.PlineName.Text
      FileParts(2) = ".pl"
      saveFile = Join(FileParts, "")
      Open saveFile For Output As #1
      Print #1, "GOCAD PLine 0.01"
      Print #1, "HEADER {"
      Print #1, "name:"; FileParts(1)
      Print #1, "}"
      Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
      Set aoiFeature = aoiCursor.NextFeature
      Do While Not aoiFeature Is Nothing
        Print #1, ""
        Print #1, "ILINE"
        PolygNr = aoiFeature.Value(aoiIDFieldIndex)
        Set aoiPolygon = aoiFeature.Shape
        DecomposePolygon aoiPolygon
        Set aoiFeature = aoiCursor.NextFeature
      Loop
      Close #1
      MsgBox "Your file has been stored as: " & Chr(13) & saveFile, 64
    Else: MsgBox "You probably selected a layer with a wrong geometry type! The macro will be reset!", vbExclamation, "Wrong geometry type!"
      Unload Shp2GocadF_Polygon
    End If
End Sub
```

```
Public Sub GetGeometryPolygon()
  Dim aoiFeatureLayer As IFeatureLayer
  ' Get the selected feature layer
  Set aoiFeatureLayer = GetSelectedFeatureLayer()
  If Not aoiFeatureLayer Is Nothing Then
    LookForGeometries aoiFeatureLayer
  End If
End Sub
```

B.2.3 Export cross-sections

The name of this form in the VBA$^{©}$ programming environment in ArcMap$^{©}$ is *Shp2GocadF_ProfLin*!

```
Option Explicit

Private Sub ExitButton_Click()
  Unload Me
End Sub

Private Sub StartShp2Goc_Click()
  Call GetGeometryProfLin
End Sub

Private Sub UserForm_Activate()
  Xleft.SelStart = 0
  Xleft.SelLength = Len(Xleft.Text)
End Sub
```

The name of this module in the VBA$^{©}$ programming environment in ArcMap$^{©}$ is *Shp2GocadM_ProfLin*!

```
Option Explicit

Dim lgPlgnNr As Long

Sub GocadExporter()

  'enable access by a form based user interface

  Shp2GocadF_ProfLin.Show

End Sub

Public Sub GetGeometryProfLin()

  Dim aoiFeatureLayer As IFeatureLayer

  'Function GetSelectedFeatureLayer() is called to fill aoiFeatureLayer

  Set aoiFeatureLayer = GetSelectedFeatureLayer()

  If Not aoiFeatureLayer Is Nothing Then

    'Passes the (active!!) aoiFeatureLayer on to function LookForGeometries

    LookForGeometries aoiFeatureLayer
  End If

  'Closing the user interface

  Unload Shp2GocadF_ProfLin

End Sub

Public Function GetSelectedFeatureLayer() As IFeatureLayer

  'Variable aoiFeatureLayer is defined as IFeatureLayer
  Dim aoiFeatureLayer As IFeatureLayer

  'Access the current document

  Dim aoiDoc As IMxDocument
  Set aoiDoc = ThisDocument

  'Verify that the active view is a data frame.

  If TypeOf aoiDoc.ActiveView Is IMap Then
    Dim aoiLayer As ILayer

    Set aoiLayer = aoiDoc.SelectedLayer

  'Access the selected layer if not empty

    If Not aoiLayer Is Nothing Then
      If TypeOf aoiLayer Is IFeatureLayer Then
        Set aoiFeatureLayer = aoiLayer
      Else
        MsgBox "Selected layer is no FeatureLayer"
      End If
    Else
      MsgBox "Exactly one layer must be active!"
    End If

    'Checks that view is a data view

    If TypeOf aoiDoc.ActiveView Is IPageLayout Then
      MsgBox "Current view is a Page Layout. Change to Data View, please!"
    End If
    Else: MsgBox "You must select at least one layer!"

  End If
  Set GetSelectedFeatureLayer = aoiFeatureLayer
End Function

 ' Function gets a polygon as GeometryCollection and decomposes it
 ' to find the coordinates

 ' Variable aoiFeatureLayer is passed on to LookForGeometries

Public Sub LookForGeometries(aoiFeatureLayer As IFeatureLayer)
  Dim aoiFeatureClass As IFeatureClass
  Dim aoiGeometryType As esriGeometryType
  Dim aoiCursor As IFeatureCursor
  Dim aoiFeature As IFeature
  Dim aoiPolyLine As IGeometryCollection
  Dim aoiIDField As IField
  Dim lgIDFldIndx As Long
  Dim strFlPrts(2) As String
  Dim strSvFl As String

  Set aoiFeatureClass = aoiFeatureLayer.FeatureClass

  'lgIDFldIndx is index number of the item selected

  lgIDFldIndx = aoiFeatureClass.FindField("ID")

  'check if shape are points, polylines, or polygons

  aoiGeometryType = aoiFeatureClass.ShapeType

    If aoiGeometryType = esriGeometryPolyline Or _
      aoiGeometryType = esriGeometryPolygon Then
      'create the file name for the file
      'in which data are going to be saved

      strFlPrts(0) = Shp2GocadF_ProfLin.PlinePath.Text
      strFlPrts(1) = Shp2GocadF_ProfLin.PlineName.Text
      strFlPrts(2) = ".pl"
      strSvFl = Join(strFlPrts, "")

      'writing the GOCAD header

      Open strSvFl For Output As #1
      Print #1, "GOCAD PLine 0.01"
      Print #1, "HEADER {"
      Print #1, "name:"; strFlPrts(1)
      Print #1, "}"

      'Print the start of the next ILINE

      Set aoiCursor = aoiFeatureClass.Search(Nothing, False)
      Set aoiFeature = aoiCursor.NextFeature
      Do While Not aoiFeature Is Nothing
        Print #1, ""
        Print #1, "ILINE"

        'Set up the index for the items and read the item into aoiPolyLine

        lgPlgnNr = aoiFeature.Value(lgIDFldIndx)
        Set aoiPolyLine = aoiFeature.Shape

        'aoiPolyLine is passed on to function DecomposeCurve

        DecomposeCurve aoiPolyLine

        'addressing the next feature (i.e. next line of the shape)

        Set aoiFeature = aoiCursor.NextFeature
      Loop

      'End and close the output file

      Print #1, "END"
      Close #1
      MsgBox "File(s) stored as " & strSvFl

    'Error message if no polyline

    Else: MsgBox "You probably selected the wrong geometry type! The macro will be reset", vbExclamation, "Wrong geometry type?"
      Unload Shp2GocadF_ProfLin
    End If
End Sub


 ' Function gets a polygon as GeomteryCollection and decomposes it
 ' to find the coordinates

Public Sub DecomposeCurve(aoiPolyLine As IGeometryCollection)

  Dim aoiLines As ISegmentCollection
  Dim aoiCurve As ICurve
  Dim aoiFrmPnt As IPoint
  Dim aoiTPnt As IPoint
  Dim intSgmntCntr As Integer
  Dim intStrtCnt As Integer
  Dim intGeomCntr As Integer
  Dim PI As Double
  Dim dbXlmp As Double          ' x left in map
  Dim dbYlmp As Double          ' y left in map
  Dim dbXrmp As Double          ’ x right in map
  Dim dbYrmp As Double          ’ y right in map
  Dim dbAlpha As Double         ’ angle alpha (explained below)
  Dim dbDltxprfl As Double      ’ delta x in profile
  Dim dbTruex As Double         ’ true x-coordinate pf profile-point
  Dim dbTruey As Double         ’ true y-coordinate pf profile-point
  Dim strCsInctr As String      ’ indicates the relationship between the profiles start-/endpoint
                                ’ in GOCAD

  ’Initialisation of counters and reading the coordinates for calculating the profile

  PI = 3.14159265358979
  intStrtCnt = 0
  intGeomCntr = 0
  dbXlmp = Shp2GocadF_ProfLin.Xleft.Text
  dbYlmp = Shp2GocadF_ProfLin.Yleft.Text
  dbXrmp = Shp2GocadF_ProfLin.Xright.Text
  dbYrmp = Shp2GocadF_ProfLin.Yright.Text

  ’Assign the case indicator including cases of equal x- or y-coords

  If dbXlmp < dbXrmp And dbYlmp > dbYrmp Then
    strCsInctr = "-11"
  ElseIf dbXlmp >= dbXrmp And dbYlmp >= dbYrmp Then
    strCsInctr = "11"
  ElseIf dbXlmp > dbXrmp And dbYlmp < dbYrmp Then
    strCsInctr = "1-1"
  ElseIf dbXlmp <= dbXrmp And dbYlmp <= dbYrmp Then
    strCsInctr = "-1-1"
  End If

  ’Prevent division by zero in coordinate’s angle calculation - math explained below

  If Not (dbXrmp = dbXlmp Or dbYrmp = dbYlmp) Then
    dbAlpha = Atn((Abs(dbYrmp - dbYlmp) / Abs(dbXrmp - dbXlmp)) ^ _
              Sgn((dbYrmp - dbYlmp) / (dbXrmp - dbXlmp)))
    Else
      If dbXrmp = dbXlmp Then
        dbAlpha = PI / 2
        Else: dbAlpha = 0
      End If
  End If

    ’Run through profile exporting all segments

    Do

      ’Chooses the next line part of the polyline

      Set aoiLines = aoiPolyLine.Geometry(intGeomCntr)

      ’loop through all segments of the line part

      For intSgmntCntr = intStrtCnt To aoiLines.SegmentCount + intStrtCnt - 1

        Set aoiCurve = aoiLines.Segment(intSgmntCntr - intStrtCnt)
        Set aoiFrmPnt = aoiCurve.fromPoint
        Set aoiTPnt = aoiCurve.toPoint

        ’Calculate and print true coordinates - math explained below
        Select Case strCsInctr
          Case "-11"
            dbTruex = Round(dbXlmp + aoiFrmPnt.x * Sin(dbAlpha), 0)
            dbTruey = Round(dbYlmp - aoiFrmPnt.x * Cos(dbAlpha), 0)
          Case "11"
            dbTruex = Round(dbXlmp - aoiFrmPnt.x * Cos(dbAlpha), 0)
            dbTruey = Round(dbYlmp - aoiFrmPnt.x * Sin(dbAlpha), 0)
          Case "1-1"
            dbTruex = Round(dbXlmp - aoiFrmPnt.x * Sin(dbAlpha), 0)
            dbTruey = Round(dbYlmp + aoiFrmPnt.x * Cos(dbAlpha), 0)
          Case "-1-1"
            dbTruex = Round(dbXlmp + aoiFrmPnt.x * Cos(dbAlpha), 0)
            dbTruey = Round(dbYlmp + aoiFrmPnt.x * Sin(dbAlpha), 0)
        End Select

        Print #1, "VRTX"; intSgmntCntr + 1; dbTruex; dbTruey; Round(aoiFrmPnt.y, 0)

      Next intSgmntCntr

      'Adjust for reading next linepart and add it DIRECTLY
      'to the continuous line in GOCAD in the next step

      intGeomCntr = intGeomCntr + 1
      intStrtCnt = intSgmntCntr + 1

    'loops until all parts AND segments of the polyline are exported
    'as VRTX-points in the GOCAD-file

    Loop While intGeomCntr < aoiPolyLine.GeometryCount

    'loop until all segments are described in the GOCAD file

    intSgmntCntr = 0
    Do While intSgmntCntr < intStrtCnt - 2
      intSgmntCntr = intSgmntCntr + 1
      Print #1, "SEG"; intSgmntCntr; intSgmntCntr + 1
    Loop

 ' Definitions:
 ' alpha = Angle between vertical EW-striking plane (M-system)
 '         and strike of profile line between points a and b (P-system)
 '         in the map view (varies also with the coordinate situation)
 ' alpha = arctan [ (x(b)-x(a))/(y(b)-y(a)) ] in general
 ' delta y(M) = delta x(P) * sin [ alpha ] in general
 ' delta x(M) = delta x(P) * cos [ alpha ] in general

End Sub
```

B.3 Data Assessment

B.3.1 Creating cross-sections

The name of this form in the VBA© programming environment in ArcMap© is *FrmExprtPrfl*!

```
Option Explicit

 Private aoiCrntDoc As esriCore.IMxDocument        ' actual Map Document
 Private aoiCrntMap As esriCore.IMap               ' Map with focus in the Map Document
 Private aoiLayers As esriCore.IEnumLayer          ' collection of layers selected due to UID
 Private aoiCrntLayer As esriCore.ILayer           ' the current layer when looping through many of them
 Private aoiCrntFtLayer As esriCore.IFeatureLayer  ' stores selected/checked feature layers
 Private aoiUID As New UID                         ' Unique Identifier for layer identification
 Private aoiCrntFtCls As esriCore.IFeatureClass    ' feature class for point feature storage
 Dim aoiFields As esriCore.IFields                 ' fields in the feature layers
 Dim lgFldCntr As Long                             ' counter for the aoiFields
 Dim aoiField As esriCore.IField                   ' one field in the aoiFields

' exits the export form

Private Sub cmdBtQuit_Click()
  Unload Me
End Sub

' marks the first TextBox in the TabOrder

Private Sub UserForm_Activate()
  txtBxXstart.SelStart = 0
  txtBxXstart.SelLength = Len(txtBxXstart.Text)
  FrmExprtPrfl.Height = 280
End Sub

 ' unselects the orientation calculation "every ... meters"
 ' and IDW if OptionButton "never" is toggled and locks the
 ' orientation distance TextBox

Private Sub optBtOrientNever_Click()
  With FrmExprtPrfl
    .chkBxCalcOrient.Value = False
    .chkBxIDW = False
    .txtBxOrientDist.Enabled = False
    .txtBxOrientRad.Enabled = False
  End With
End Sub

 ' unselects Option Button "Never" if orientation calculation
 ' every ... meters is selected and (un)locks the orientation
 ' distance TextBox

Private Sub chkBxCalcOrient_Click()
  If chkBxCalcOrient.Value = True Then
    optBtOrientNever.Value = False
    txtBxOrientDist.Enabled = True
    txtBxOrientRad.Enabled = True
  Else
    txtBxOrientDist.Enabled = False
    txtBxOrientRad.Enabled = False
  End If
End Sub

 ' prevents selection of IDW if OptionButton "Never" is selected
```

```
Private Sub chkBxIDW_Click()
  If optBtOrientNever.Value = True Then
    chkBxIDW.Value = False
  End If
End Sub

 ' checks that all values entered are numeric & correct values
 ' and leads on to the 2nd step for entering data

Private Sub cmdBtStart_Click()
  If Not (IsNumeric(txtBxXend.Text) = True _
    And IsNumeric(txtBxYend.Text) = True _
    And IsNumeric(txtBxXstart.Text) = True _
    And IsNumeric(txtBxYstart.Text) = True _
    And IsNumeric(txtBxElevDist.Text) = True _
    And IsNumeric(txtBxOrientDist.Text) = True _
    And IsNumeric(txtBxOrientRad.Text) = True) _
    Or (txtBxXend.Text = txtBxXstart.Text _
    And txtBxYstart.Text = txtBxYend.Text) Then
      MsgBox "That must be an interesting profile ;-)", vbCritical + vbOKOnly, "Take a look at your data again!"
      Exit Sub

  Else:
    With FrmExprtPrfl
      .Height = 530
      .txtBxElevDist.Enabled = False
      .txtBxOrientDist.Enabled = False
      .txtBxXend.Enabled = False
      .txtBxXstart.Enabled = False
      .txtBxYend.Enabled = False
      .txtBxYstart.Enabled = False
      .chkBxCalcOrient.Enabled = False
      .txtBxOrientRad.Enabled = False
      .chkBxIDW.Enabled = False
'      .optBtOrientContact.Enabled = False
      .optBtOrientNever.Enabled = False
      .cmdBtStart.Enabled = False
      .cmdBtQuit.Enabled = False
    End With

    Call ListBoxFiller

 ' locks the list boxes for orientation measurements if
 ' orientation measurements are not exported

    If optBtOrientNever.Value = True Then
      With FrmExprtPrfl
        .lstBxPlanars.Clear
        .lstBxPlanars.ForeColor = &H80000011
        .lstBxDipDir.ForeColor = &H80000011
        .lstBxDipAn.ForeColor = &H80000011
        .lstBxPlanars.Enabled = False
        .lstBxDipDir.Enabled = False
        .lstBxDipAn.Enabled = False
        .lstBxPlanars.AddItem ("not available")
        .lstBxDipDir.AddItem ("not available")
        .lstBxDipAn.AddItem ("not available")
        .txtBxFoliationName.Text = "not available"
        .txtBxFoliationName.Enabled = False
      End With
    End If
  End If
End Sub

 ' checks if necessary layers and fields in
 ' the 2nd part of the form are selected
```

```
Private Sub cmdBtStart2_Click()

  Select Case FrmExprtPrfl.optBtOrientNever.Value
  Case True
    If FrmExprtPrfl.lstBxDEMs.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxGeoID.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxPlgns.ListIndex = -1 Then
      MsgBox "Please specify the necessary selections in the respective list boxes!", vbCritical _
             + vbOKOnly, "Selection ambiguity!"
      Exit Sub
    End If
  Case False
    If FrmExprtPrfl.lstBxDEMs.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxGeoID.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxPlgns.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxDipAn.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxDipDir.ListIndex = -1 _
      Or FrmExprtPrfl.lstBxPlanars.ListIndex = -1 Then
      MsgBox "Please specify the necessary selections in the respective list boxes!", vbCritical + vbOKOnly, "Selection ambiguity!"
      Exit Sub
    End If
  End Select
  FrmExprtPrfl.Height = 545
  FrmExprtPrfl.lblPrgBr.Visible = True

  Call ProfileCalculatorM.Main

End Sub

 ' fills the list boxes for selection of the correct
 ' DEM-, Geology- & planar measurement-layer

Private Sub ListBoxFiller()

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMap = aoiCrntDoc.FocusMap
  aoiUID.Value = "{6CA416B1-E160-11D2-9F4E-00C04F6BC78E}"
  Set aoiLayers = aoiCrntMap.Layers(aoiUID, True)

  aoiLayers.Reset
  Set aoiCrntLayer = aoiLayers.Next

 ' loop through all layers to find all raster layers
 ' and list them in one box and to find all point feature
 ' layers and list them in the other box

  Do While Not aoiCrntLayer Is Nothing

    If TypeOf aoiCrntLayer Is IFeatureLayer Then
      Set aoiCrntFtLayer = aoiCrntLayer
      Set aoiCrntFtCls = aoiCrntFtLayer.FeatureClass

      If aoiCrntFtCls.ShapeType = esriGeometryPoint Then
        FrmExprtPrfl.lstBxPlanars.AddItem aoiCrntLayer.Name
      End If

      If aoiCrntFtCls.ShapeType = esriGeometryPolygon Then
        FrmExprtPrfl.lstBxPlgns.AddItem aoiCrntLayer.Name
      End If

    End If

    If TypeOf aoiCrntLayer Is IRasterLayer Then
      FrmExprtPrfl.lstBxDEMs.AddItem aoiCrntLayer.Name
    End If

    Set aoiCrntLayer = aoiLayers.Next
  Loop

End Sub

 ' updates the Dip Direction and Dip Angle Fields
 ' after a point feature layer is selected

Private Sub lstBxPlanars_Change()

  FrmExprtPrfl.lstBxDipAn.Clear
  FrmExprtPrfl.lstBxDipDir.Clear

  aoiUID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
  Set aoiLayers = aoiCrntMap.Layers(aoiUID, True)
  aoiLayers.Reset
  Set aoiCrntLayer = aoiLayers.Next

 ' loop through all feature layers to find the one
 ' selected feature class in the point layer list box

  Do While Not aoiCrntLayer Is Nothing
    If aoiCrntLayer.Name = FrmExprtPrfl.lstBxPlanars.Value Then
      Set aoiCrntFtLayer = aoiCrntLayer
      Set aoiCrntFtCls = aoiCrntFtLayer.FeatureClass
      Exit Do
    End If
    Set aoiCrntLayer = aoiLayers.Next
  Loop

 ' loop through all fields of the feature class
 ' that has been found above

  Set aoiFields = aoiCrntFtCls.Fields
  For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
    Set aoiField = aoiFields.Field(lgFldCntr)
    If aoiField.Name <> "Shape" Then
      With FrmExprtPrfl
        .lstBxDipDir.AddItem aoiField.Name
        .lstBxDipAn.AddItem aoiField.Name
      End With
    End If
  Next lgFldCntr

End Sub

Private Sub lstBxPlgns_Change()

  FrmExprtPrfl.lstBxGeoID.Clear

  aoiUID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
  Set aoiLayers = aoiCrntMap.Layers(aoiUID, True)
  aoiLayers.Reset
  Set aoiCrntLayer = aoiLayers.Next

 ' loop through all feature layers to find the one
 ' selected feature class in the polygon layer list box

  Do While Not aoiCrntLayer Is Nothing
    If aoiCrntLayer.Name = FrmExprtPrfl.lstBxPlgns.Value Then
      Set aoiCrntFtLayer = aoiCrntLayer
      Set aoiCrntFtCls = aoiCrntFtLayer.FeatureClass
      Exit Do
    End If
    Set aoiCrntLayer = aoiLayers.Next
  Loop

 ' loop through all fields of the feature class
 ' that has been found above

  Set aoiFields = aoiCrntFtCls.Fields
  For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
    Set aoiField = aoiFields.Field(lgFldCntr)
    If aoiField.Name <> "Shape" Then
      With FrmExprtPrfl
        .lstBxGeoID.AddItem aoiField.Name
      End With
    End If
  Next lgFldCntr

End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *ProfileCalculatorM*!

```
Option Explicit

 Private blIDW As Boolean                 ' indicates exported orientations are to be IDW calculated
 Private blOrntEvrMtr As Boolean          ' indicates if orientations are exported at regular intervals
 'Private blContact As Boolean             ' indicates if orientations are exported at contacts additionally
 Private blNoOrient As Boolean            ' indicates if any orientations are exported at all
 Private dbElevDist As Double             ' stores value for the distance elevation points
 Private dbOrientDist As Double           ' stores value for the distance between orientation calculations
 Private lgXstart As Long                 ' X coord of the profile's startpoint
 Private lgYstart As Long                 ' Y coord of the profile's startpoint
 Private lgXend As Long                   ' X coord of the profile's endpoint
 Private lgYend As Long                   ' Y coord of the profile's endpoint
 Private dbPrflCrdsWrt() As Double        ' Array for all coordinates of the profile to be WRITTEN
 Private dbPrflCrdsRd() As Double         ' Array for all coordinates to be READ from the DEMs for the profile
 Private dbPtCrdsWrt() As Double          ' Array for all coordinates of the foliations to be WRITTEN
 Private dbPtCrdsRd() As Double           ' Array for all coordinates to be READ from the DEMs for the foliations
 Private strRstrLyrName As String         ' stores the name of the Raster Layer
 Private strPntLyrName As String          ' stores the name of the Point Feature Layer
 Private strDipDirName As String          ' stores the name of the Dip Direction Field
 Private strDipAnName As String           ' stores the name of the Dip Angle Field
 Private strGeoLayer As String            ' stores the name of the Polygon Feature Layer
 Private strGeoID As String               ' stores the name of the Geology Unit-ID Field
 Private strFolder As String              ' stores the name of the Workspace Path
 Private strPllnName As String            ' stores the name of the Polyline Shape (i.e. profile-line)
 Private strPointName As String           ' stores the name of the Point Shape (i.e. foliations)
 Const PI As Double = 3.14159265358979

 ' module start - shows the user form
Public Sub Start()
  FrmExprtPrfl.Show
End Sub

' calls all subfunctions Public Sub Main()

  Call VariableAssigner                  ' assigns the global variables derived from the user form

  Call ShapeConstruct                    ' creates the necessary shape files (polyline (and points))

  Call CoordinateCalcProfile             ' calculates the coordinates for the profile

  Call ElevationGripProfile              ' retrieves the elevation values from the respective DEM fro the profile

  Call FillPolyline                      ' writes the coordinates in the line shape and
                                         ' internally calls a function to check points' ID assignments

  If blNoOrient = False Then             ' accesses the export routines for foliation export
```

```
    Call CoordinateCalcFoliations            ' calculates the coords for the foliation points

    Call ElevationGripFoliations             ' retrieves the elevation values at the foliation points

    Call FillMultipoint                      ' writes the respective values into the point shape's fields

  End If

 ' notify the user about the files
 ' locations

  MsgBox "Your files are stored in" & " " & strFolder & Chr$(13), vbInformation + vbOKOnly, "That was it ..."
  Unload FrmExprtPrfl

End Sub

 ' loads all variable values from "FrmExprtPrfl"
 ' into the respective (global) variables of this module

Private Sub VariableAssigner()

  With FrmExprtPrfl
    blIDW = .chkBxIDW.Value
    blOrntEvrMtr = .chkBxCalcOrient.Value
    blNoOrient = .optBtOrientNever.Value
'    blContact = .optBtOrientContact.Value
    dbElevDist = Round(.txtBxElevDist.Value, 0)
    dbOrientDist = Round(.txtBxOrientDist.Value, 0)
    lgXstart = CLng(.txtBxXstart.Value)
    lgYstart = CLng(.txtBxYstart.Value)
    lgXend = CLng(.txtBxXend.Value)
    lgYend = CLng(.txtBxYend.Value)
    strRstrLyrName = .lstBxDEMs.Value
    strGeoLayer = .lstBxPlgns.Value
    strGeoID = .lstBxGeoID.Value
    strFolder = .txtBxPath.Value
    strPllnName = .txtBxProfileName.Value
    If blNoOrient = False Then
      strPntLyrName = .lstBxPlanars.Value
      strDipAnName = .lstBxDipAn.Value
      strDipDirName = .lstBxDipDir.Value
      strPointName = .txtBxFoliationName.Value
    End If
  End With

End Sub

 ' calculates the points for the new polyline assemblage

Private Sub CoordinateCalcProfile()

 Dim strCsIndctr As String              ' indicates the relationship between the profile's start-/endpoint
 Dim dbAlpha As Double                  ' angle between a horizontal or vertical line and profile trend
 Dim dbPrflLen As Double                ' length of the profile
 Dim dbNmbPts As Double                 ' total number of points
 Dim dbPtCntr As Double                 ' Counter for the profile Points


 'Assign the case indicator including cases of equal x- or y-coords

  If lgXstart < lgXend And lgYstart > lgYend Then
    strCsIndctr = "-11"
  ElseIf lgXstart >= lgXend And lgYstart >= lgYend Then
    strCsIndctr = "11"
  ElseIf lgXstart > lgXend And lgYstart < lgYend Then
    strCsIndctr = "1-1"
  ElseIf lgXstart <= lgXend And lgYstart <= lgYend Then
    strCsIndctr = "-1-1"
  End If

 ’Prevent division by zero in coordinate’s angle calculation

  If Not (lgXend = lgXstart Or lgYend = lgYstart) Then
    dbAlpha = Atn((Abs(lgYend - lgYstart) / Abs(lgXend - lgXstart)) ^ _
              Sgn((lgYend - lgYstart) / (lgXend - lgXstart)))
    Else
      If lgXend = lgXstart Then
        dbAlpha = PI / 2
        Else: dbAlpha = 0
      End If
  End If

 ’ Calculate the length of the profile

  dbPrflLen = Sqr(((lgYend - lgYstart) ^ 2 + (lgXend - lgXstart) ^ 2))

 ’ Calculate the number of profile points and resize the coordinate array
 ’ (..., 0) = x-coord; (..., 1) = y-coord

  If Not (dbPrflLen / dbElevDist) = Int(dbPrflLen / dbElevDist) Then
    dbNmbPts = Int(dbPrflLen / dbElevDist) + 2
  Else: dbNmbPts = Int(dbPrflLen / dbElevDist) + 1
  End If
  ReDim dbPrflCrdsWrt(dbNmbPts - 1, 1)
  ReDim dbPrflCrdsRd(dbNmbPts - 1, 1)

 ’ looping through all points that have to be constructed in the profile
 ’ assigning their X-values (i.e. along the profile’s length)

  For dbPtCntr = 0 To dbNmbPts - 2
    dbPrflCrdsWrt(dbPtCntr, 0) = dbPtCntr * dbElevDist
  Next dbPtCntr
  dbPrflCrdsWrt(dbNmbPts - 1, 0) = dbPrflLen

 ’ looping through all points that have to be addressed on the
 ’ DEM and in the Geology-Polygon to find their X and Y values

  Select Case strCsIndctr
  Case "-11"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPrflCrdsRd(dbPtCntr, 0) = lgXstart + dbPtCntr * dbElevDist * Sin(dbAlpha)
      dbPrflCrdsRd(dbPtCntr, 1) = lgYstart - dbPtCntr * dbElevDist * Cos(dbAlpha)
    Next dbPtCntr
  Case "11"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPrflCrdsRd(dbPtCntr, 0) = lgXstart - dbPtCntr * dbElevDist * Cos(dbAlpha)
      dbPrflCrdsRd(dbPtCntr, 1) = lgYstart - dbPtCntr * dbElevDist * Sin(dbAlpha)
    Next dbPtCntr
  Case "1-1"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPrflCrdsRd(dbPtCntr, 0) = lgXstart - dbPtCntr * dbElevDist * Sin(dbAlpha)
      dbPrflCrdsRd(dbPtCntr, 1) = lgYstart + dbPtCntr * dbElevDist * Cos(dbAlpha)
    Next dbPtCntr
  Case "-1-1"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPrflCrdsRd(dbPtCntr, 0) = lgXstart + dbPtCntr * dbElevDist * Cos(dbAlpha)
      dbPrflCrdsRd(dbPtCntr, 1) = lgYstart + dbPtCntr * dbElevDist * Sin(dbAlpha)
    Next dbPtCntr
  End Select
  dbPrflCrdsRd(dbPtCntr, 0) = lgXend
  dbPrflCrdsRd(dbPtCntr, 1) = lgYend

End Sub
```

```
 ' identifies the DEM containing the elevation data and
 ' assigns the data to the respective points in the
 ' dbPrflCrdsWrt array

Private Sub ElevationGripProfile()

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntMp As IMap
  Dim aoiUID As UID
  Dim aoiEnumLyrs As IEnumLayer
  Dim aoiCrntLyr As ILayer
  Dim aoiCrntRasLyr As IRasterLayer
  Dim aoiRaster As IRaster
  Dim aoiRstrProp As IRasterProps
  Dim aoiIdentify As IIdentify
  Dim aoiRasIdentObj As IRasterIdentifyObj
  Dim aoiRstrArr As IArray
  Dim aoiChkPoint As IPoint
  Dim lgPtCntr As Long

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiUID = New UID
  aoiUID = "{6CA416B1-E160-11D2-9F4E-00C04F6BC78E}"
  Set aoiEnumLyrs = aoiCrntMp.Layers(aoiUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr = aoiEnumLyrs.Next

  FrmExprtPrfl.prgBr.Max = UBound(dbPrflCrdsWrt, 1)

 ' look for the specified raster layer
  Do While Not aoiCrntLyr Is Nothing
    If aoiCrntLyr.Name = strRstrLyrName Then
      Set aoiCrntRasLyr = aoiCrntLyr
      Set aoiRaster = aoiCrntRasLyr.Raster
      Set aoiIdentify = aoiCrntRasLyr
      Exit Do
    End If
    Set aoiCrntLyr = aoiEnumLyrs.Next
  Loop

  Set aoiChkPoint = New Point
  FrmExprtPrfl.Repaint

 ' loop through all points in the ...Wrt-Array

  For lgPtCntr = 0 To UBound(dbPrflCrdsWrt, 1)
    aoiChkPoint.x = dbPrflCrdsRd(lgPtCntr, 0)
    aoiChkPoint.y = dbPrflCrdsRd(lgPtCntr, 1)

 ' get the raster values at the respective point

    Set aoiRstrArr = aoiIdentify.Identify(aoiChkPoint)
    If Not aoiRstrArr Is Nothing Then
      Set aoiRasIdentObj = aoiRstrArr.Element(0)
      If aoiRasIdentObj.Name <> "NoData" Then

 ' store the value in the y-coord of the array
 ' containing the values for the profile-shape

        dbPrflCrdsWrt(lgPtCntr, 1) = CDbl(aoiRasIdentObj.Name)
      End If
    End If
    With FrmExprtPrfl.prgBr
      .Value = lgPtCntr
      .Refresh
    End With
  Next lgPtCntr

End Sub

 ’ constructs a Polyline shapefile (for the profile line)
 ’ and - if necessary - also a mulitipoint shapefile (for
 ’ the foliation measurements)

Private Sub ShapeConstruct()

  Const strShapeFieldName As String = "Shape"

 ’ Mark for ’goto-jump’ used if foliations are also exported

scndRun:

 ’ Open the folder to contain the shapefile as a workspace

  Dim aoiFWS As IFeatureWorkspace
  Dim aoiWorkspaceFactory As IWorkspaceFactory
  Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
  Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strFolder, 0)

 ’ Set up a simple fields collection

  Dim aoiFields As IFields
  Dim aoiFieldsEdit As IFieldsEdit
  Set aoiFields = New esriCore.Fields
  Set aoiFieldsEdit = aoiFields
  Dim aoiField As IField
  Dim aoiFieldEdit As IFieldEdit

 ’ Make the shape field including its spatial reference

  Set aoiField = New esriCore.Field
  Set aoiFieldEdit = aoiField

 ’ specify that you deal with a shape file

  aoiFieldEdit.Name = strShapeFieldName
  aoiFieldEdit.Type = esriFieldTypeGeometry
  Dim aoiGeomDef As IGeometryDef
  Dim aoiGeomDefEdit As IGeometryDefEdit
  Set aoiGeomDef = New GeometryDef
  Set aoiGeomDefEdit = aoiGeomDef
  With aoiGeomDefEdit

 ’ check if profile or foliation file are created

    If strPllnName <> strPointName Then
      .geometryType = esriGeometryPolyline
    Else
      .geometryType = esriGeometryPoint
    End If
    .GridCount = 1
    .GridSize(0) = 10
    .AvgNumPoints = 2
    .HasM = False
    .HasZ = False

 ’ calls a function to find out the spatial reference of the
 ’ geology layer on which the calculations will be based on

    Set .SpatialReference = GetSpatialReference()
  End With
  Set aoiFieldEdit.GeometryDef = aoiGeomDef
  aoiFieldsEdit.AddField aoiField

 ' Add an ID integer field

  If strPllnName <> strPointName Then
    Set aoiField = New esriCore.Field
    Set aoiFieldEdit = aoiField
    With aoiFieldEdit
        .Length = 10
        .Name = "ID"
        .Type = esriFieldTypeInteger
    End With
    aoiFieldsEdit.AddField aoiField
  End If

 ' if foliation-shape is created the fields for dip
 ' direction, dip angle and apparent dip angle are added

  If strPllnName = strPointName Then
    Set aoiField = New esriCore.Field
    Set aoiFieldEdit = aoiField
    With aoiFieldEdit
        .Length = 5
        .Name = "DipDir"
        .Type = esriFieldTypeInteger
    End With
    aoiFieldsEdit.AddField aoiField

    Set aoiField = New esriCore.Field
    Set aoiFieldEdit = aoiField
    With aoiFieldEdit
        .Length = 5
        .Name = "DipAn"
        .Type = esriFieldTypeInteger
    End With
    aoiFieldsEdit.AddField aoiField

    Set aoiField = New esriCore.Field
    Set aoiFieldEdit = aoiField
    With aoiFieldEdit
        .Length = 5
        .Name = "AppDip"
        .Type = esriFieldTypeInteger
    End With
    aoiFieldsEdit.AddField aoiField
  End If

 ' Create the shapefile - i.e. a new Feature Class

  Dim aoiFeatClass As IFeatureClass
  Set aoiFeatClass = aoiFWS.CreateFeatureClass(strPllnName, aoiFields, Nothing, _
                                          Nothing, esriFTSimple, strShapeFieldName, "")

 ' Jump back to create a second shapefile if necessary
 ' and not already done

  If blNoOrient = False And strPllnName <> strPointName Then
    strPllnName = strPointName
    GoTo scndRun
  End If

  strPllnName = FrmExprtPrfl.txtBxProfileName.Value

End Sub

Private Sub FillPolyline()
```

```
’ Open the folder to contain the shapefile as a workspace

 Dim aoiFWS As IFeatureWorkspace
 Dim aoiWorkspaceFactory As IWorkspaceFactory
 Dim aoiCrntFtCls As IFeatureClass
 Dim aoiFeat As IFeature

 Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
 Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strFolder, 0)
 Set aoiCrntFtCls = aoiFWS.OpenFeatureClass(strPllnName)

 Dim intCaseSlctr As Integer                          ’ specifies what the function "CheckPointID" is used for
 Dim lgSgmntCntr As Long                              ’ counter for ’for...next-loops’
 Dim blPointsSameID As Boolean                        ’ specifies if 2 points are in the same polygon
 Dim aoiFromPt As IPoint                              ’ start point of a segment
 Dim aoiToPt As IPoint                                ’ end point of a segment
 Dim aoiLine As ILine                                 ’ a single 2-point-line
 Dim aoiSegColl As ISegmentCollection                 ’ a collection of aoiLines
 Dim aoiPath As IPath                                 ’ geometry for the SegColl
 Dim aoiGeomColl As IGeometryCollection               ’ all paths
 Dim aoiPlln As IPolyline                             ’ the polyline made of several Paths with unique IDs

 Set aoiSegColl = New Path
 Set aoiGeomColl = New Polyline
 With FrmExprtPrfl
   .lblPrgBr.Caption = "Writing Shape-File ..."
   .Repaint
 End With

’ loop through all points on the polyline

 For lgSgmntCntr = 0 To UBound(dbPrflCrdsWrt, 1) - 1

   intCaseSlctr = 1
   Set aoiToPt = New Point
   Set aoiFromPt = New Point
   Set aoiLine = New Line

   aoiFromPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr, 0), dbPrflCrdsWrt(lgSgmntCntr, 1)
   aoiToPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr + 1, 0), dbPrflCrdsWrt(lgSgmntCntr + 1, 1)
   aoiLine.PutCoords aoiFromPt, aoiToPt

’ check if the two points of a segment are in the same geology polygon

   blPointsSameID = CheckPointID(aoiFromPt, aoiToPt, intCaseSlctr, lgSgmntCntr)

’ at boundary or at end of profile ...

   If blPointsSameID = False _
      Or lgSgmntCntr = UBound(dbPrflCrdsWrt, 1) - 1 Then

’ specifies that CheckPointID-Function is now used for
’ returning ID-values and not booleans for boundary checking

     intCaseSlctr = 2
     aoiGeomColl.AddGeometry aoiSegColl
     Set aoiPlln = aoiGeomColl
     Set aoiFeat = aoiCrntFtCls.CreateFeature
     Set aoiFeat.Shape = aoiPlln

’ assign the ID of this profile part

     aoiFeat.Value(aoiFeat.Fields.FindField(strGeoID)) = CheckPointID(aoiFromPt, aoiToPt, _
                                                                    intCaseSlctr, lgSgmntCntr)
     aoiFeat.Store
     aoiLine.SetEmpty
     Set aoiFeat = New Feature
```

```
      Set aoiSegColl = New Path
      Set aoiGeomColl = New Polyline
    End If

    If Not aoiLine.IsEmpty = True Then aoiSegColl.AddSegment aoiLine

    With FrmExprtPrfl.prgBr
      .Value = lgSgmntCntr
      .Refresh
    End With
  Next lgSgmntCntr

  Set aoiPlln = aoiGeomColl
  Set aoiFeat = aoiCrntFtCls.CreateFeature
  Set aoiFeat.Shape = aoiPlln
  aoiFeat.Store

End Sub

 ' checks if two points of a segment belong two the
 ' same geology unit specified by the ID

Private Function CheckPointID(aoiFromPt As IPoint, aoiToPt As IPoint,
intCaseSlctr As Integer, _
                              lgSgmntCntr As Long)

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntMp As IMap
  Dim aoiUID As UID
  Dim aoiEnumLyrs As IEnumLayer
  Dim aoiCrntLyr As IFeatureLayer
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFtCrs As IFeatureCursor
  Dim aoiCrntFt As IFeature
  Dim aoiRelOpPlg As IRelationalOperator
  Dim aoiPlgn As IGeometryCollection
  Dim intFtCntr As Integer

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiUID = New UID
  aoiUID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"

  Set aoiEnumLyrs = aoiCrntMp.Layers(aoiUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr = aoiEnumLyrs.Next

  Do While Not aoiCrntLyr Is Nothing
    If aoiCrntLyr.Name = strGeoLayer Then
      Exit Do
    End If
    Set aoiCrntLyr = aoiEnumLyrs.Next
  Loop

  aoiFromPt.PutCoords dbPrflCrdsRd(lgSgmntCntr, 0), dbPrflCrdsRd(lgSgmntCntr, 1)
  aoiToPt.PutCoords dbPrflCrdsRd(lgSgmntCntr + 1, 0), dbPrflCrdsRd(lgSgmntCntr + 1, 1)

  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)

 ' loop through all polygons to find those containing
 ' the ToPoint and FromPoint of the respective segment

  For intFtCntr = 0 To aoiCrntFtCls.FeatureCount(Nothing) - 1

    Set aoiCrntFt = aoiFtCrs.NextFeature
    Set aoiPlgn = aoiCrntFt.Shape
    Set aoiRelOpPlg = aoiPlgn

 ' check different possibilities of points' positions
 ' 1) both points in same geology unit and not end of profile

    If (aoiRelOpPlg.Contains(aoiFromPt) _
      And aoiRelOpPlg.Contains(aoiToPt)) _
      And Not lgSgmntCntr = UBound(dbPrflCrdsWrt, 1) - 1 Then
        aoiFromPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr, 0), dbPrflCrdsWrt(lgSgmntCntr, 1)
        aoiToPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr + 1, 0), dbPrflCrdsWrt(lgSgmntCntr + 1, 1)
        CheckPointID = True
        Exit Function

 ' 2)  points are in two different geology units or
 '     at the profile's end

    ElseIf (aoiRelOpPlg.Contains(aoiFromPt) _
      And Not aoiRelOpPlg.Contains(aoiToPt)) _
      Or lgSgmntCntr = UBound(dbPrflCrdsWrt, 1) - 1 Then

 ' if not at the end of the profile's end
 ' there must be a boundary

        Select Case intCaseSlctr
        Case 1
          aoiFromPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr, 0), dbPrflCrdsWrt(lgSgmntCntr, 1)
          aoiToPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr + 1, 0), dbPrflCrdsWrt(lgSgmntCntr + 1, 1)
          CheckPointID = False
          Exit Function

 ' if at the profile's end, the last ID is returned
 ' in order to assign it to the last polygon part

        Case 2
          If aoiRelOpPlg.Contains(aoiFromPt) Then
            aoiFromPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr, 0), dbPrflCrdsWrt(lgSgmntCntr, 1)
            aoiToPt.PutCoords dbPrflCrdsWrt(lgSgmntCntr + 1, 0), dbPrflCrdsWrt(lgSgmntCntr + 1, 1)
            CheckPointID = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strGeoID))
            Exit Function
          End If
        End Select
    End If

  Next intFtCntr

  MsgBox "Your profile trace leads through an area with no geology assignments!" _
  & Chr$(13) & "The macro will be reset!", vbCritical + vbOKOnly, "Fatal error!"

  Unload FrmExprtPrfl
  Kill strFolder & strPllnName & ".*"
  If blNoOrient = False Then Kill strFolder & strPointName & ".*"
  FrmExprtPrfl.Show

  End

End Function

 Private Sub CoordinateCalcFoliations()
 Dim strCsIndctr As String              ' indicates the relationship between the profiles start-/endpoint
 Dim dbAlpha As Double                  ' angle between a horizontal or vertical line and profile trend
 Dim dbPrflLen As Double                ' length of the profile
 Dim dbNmbPts As Double                 ' total number of points
 Dim dbPtCntr As Double                 ' Counter for the Fol-Points


 'Assign the case indicator including cases of equal x- or y-coords
  If lgXstart < lgXend And lgYstart > lgYend Then
    strCsIndctr = "-11"
  ElseIf lgXstart >= lgXend And lgYstart >= lgYend Then
    strCsIndctr = "11"
  ElseIf lgXstart > lgXend And lgYstart < lgYend Then
    strCsIndctr = "1-1"
  ElseIf lgXstart <= lgXend And lgYstart <= lgYend Then
    strCsIndctr = "-1-1"
  End If

 'Prevent division by zero in coordinate's angle calculation

  If Not (lgXend = lgXstart Or lgYend = lgYstart) Then
    dbAlpha = Atn((Abs(lgYend - lgYstart) / Abs(lgXend - lgXstart)) ^ _
              Sgn((lgYend - lgYstart) / (lgXend - lgXstart)))
    Else
      If lgXend = lgXstart Then
        dbAlpha = PI / 2
        Else: dbAlpha = 0
      End If
  End If

 ' Calculate the length of the profile

  dbPrflLen = Sqr(((lgYend - lgYstart) ^ 2 + (lgXend - lgXstart) ^ 2))

 ' Calculate the number of profile points and resize the coordinate array
 ' (..., 0) = x-coord; (..., 1) = y-coord

  If Not (dbPrflLen / dbOrientDist) = Int(dbPrflLen / dbOrientDist) Then
    dbNmbPts = Int(dbPrflLen / dbOrientDist) + 2
  Else: dbNmbPts = Int(dbPrflLen / dbOrientDist) + 1
  End If
  ReDim dbPtCrdsWrt(dbNmbPts - 1, 1)
  ReDim dbPtCrdsRd(dbNmbPts - 1, 1)

 ' looping through all points that have to be constructed in the profile
 ' assigning their X-values

  For dbPtCntr = 0 To dbNmbPts - 2
    dbPtCrdsWrt(dbPtCntr, 0) = dbPtCntr * dbOrientDist
  Next dbPtCntr
  dbPtCrdsWrt(dbNmbPts - 1, 0) = dbPrflLen

 ' looping through all points that have to be addressed on the
 ' DEM to find their x and Y values

  Select Case strCsIndctr
  Case "-11"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPtCrdsRd(dbPtCntr, 0) = lgXstart + dbPtCntr * dbOrientDist * Sin(dbAlpha)
      dbPtCrdsRd(dbPtCntr, 1) = lgYstart - dbPtCntr * dbOrientDist * Cos(dbAlpha)
    Next dbPtCntr
  Case "11"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPtCrdsRd(dbPtCntr, 0) = lgXstart - dbPtCntr * dbOrientDist * Cos(dbAlpha)
      dbPtCrdsRd(dbPtCntr, 1) = lgYstart - dbPtCntr * dbOrientDist * Sin(dbAlpha)
    Next dbPtCntr
  Case "1-1"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPtCrdsRd(dbPtCntr, 0) = lgXstart - dbPtCntr * dbOrientDist * Sin(dbAlpha)
      dbPtCrdsRd(dbPtCntr, 1) = lgYstart + dbPtCntr * dbOrientDist * Cos(dbAlpha)
    Next dbPtCntr
  Case "-1-1"
    For dbPtCntr = 0 To dbNmbPts - 2
      dbPtCrdsRd(dbPtCntr, 0) = lgXstart + dbPtCntr * dbOrientDist * Cos(dbAlpha)
      dbPtCrdsRd(dbPtCntr, 1) = lgYstart + dbPtCntr * dbOrientDist * Sin(dbAlpha)
```

```
    Next dbPtCntr
  End Select
  dbPtCrdsRd(dbPtCntr, 0) = lgXend
  dbPtCrdsRd(dbPtCntr, 1) = lgYend

End Sub

 ' works the same way  as ElevationGripProfile

Private Sub ElevationGripFoliations()

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntMp As IMap
  Dim aoiUID As UID
  Dim aoiEnumLyrs As IEnumLayer
  Dim aoiCrntLyr As ILayer
  Dim aoiCrntRasLyr As IRasterLayer
  Dim aoiRaster As IRaster
  Dim aoiRstrProp As IRasterProps
  Dim aoiIdentify As IIdentify
  Dim aoiRasIdentObj As IRasterIdentifyObj
  Dim aoiRstrArr As IArray
  Dim aoiChkPoint As IPoint
  Dim lgPtCntr As Long

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiUID = New UID
  aoiUID = "{6CA416B1-E160-11D2-9F4E-00C04F6BC78E}"
  Set aoiEnumLyrs = aoiCrntMp.Layers(aoiUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr = aoiEnumLyrs.Next

  With FrmExprtPrfl
    .prgBr.Max = UBound(dbPtCrdsWrt, 1)
    .lblPrgBr.Caption = "Reading DEM for foliations ..."
  End With

  Do While Not aoiCrntLyr Is Nothing
    If aoiCrntLyr.Name = strRstrLyrName Then
      Set aoiCrntRasLyr = aoiCrntLyr
      Set aoiRaster = aoiCrntRasLyr.Raster
      Set aoiIdentify = aoiCrntRasLyr
      Exit Do
    End If
    Set aoiCrntLyr = aoiEnumLyrs.Next
  Loop

  Set aoiChkPoint = New Point

  FrmExprtPrfl.Repaint

  For lgPtCntr = 0 To UBound(dbPtCrdsWrt, 1)
    aoiChkPoint.x = dbPtCrdsRd(lgPtCntr, 0)
    aoiChkPoint.y = dbPtCrdsRd(lgPtCntr, 1)
    Set aoiRstrArr = aoiIdentify.Identify(aoiChkPoint)
    If Not aoiRstrArr Is Nothing Then
      Set aoiRasIdentObj = aoiRstrArr.Element(0)
      If aoiRasIdentObj.Name <> "NoData" Then
        dbPtCrdsWrt(lgPtCntr, 1) = CDbl(aoiRasIdentObj.Name)
      End If
    End If
    With FrmExprtPrfl.prgBr
      .Value = lgPtCntr
      .Refresh
    End With
  Next lgPtCntr
```

```
End Sub

Private Sub FillMultipoint()

 ' Open the folder to contain the shapefile as a workspace

  Dim aoiFWS As IFeatureWorkspace
  Dim aoiWorkspaceFactory As IWorkspaceFactory
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFeat As IFeature

  Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
  Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strFolder, 0)
  Set aoiCrntFtCls = aoiFWS.OpenFeatureClass(strPointName)

  Dim intDipDir As Integer
  Dim intDipAn As Integer
  Dim intAppDip As Integer
  Dim lgSgmntCntr As Long
  Dim strFolFlds As String
  Dim aoiFolPt As IPoint

  With FrmExprtPrfl
    .lblPrgBr.Caption = "Writing Point-Shape-File ..."
    .prgBr.Max = UBound(dbPtCrdsWrt, 1)
    .Repaint
  End With

  For lgSgmntCntr = 0 To UBound(dbPtCrdsWrt, 1)

    Set aoiFolPt = New Point
    aoiFolPt.PutCoords dbPtCrdsWrt(lgSgmntCntr, 0), dbPtCrdsWrt(lgSgmntCntr, 1)

    Set aoiFeat = aoiCrntFtCls.CreateFeature
    Set aoiFeat.Shape = aoiFolPt

 ' retrieve the orientation values in one string
 ' variable and split it into its constituents

    strFolFlds = FillFolFields(aoiFolPt, lgSgmntCntr)
    If strFolFlds <> "" Then
      intDipDir = Split(strFolFlds, " ")(0)
      intDipAn = Split(strFolFlds, " ")(1)

 ' calculating the apparent dip angle in an external function

      intAppDip = ApparentDipCalc(CDbl(intDipDir), CDbl(intDipAn))

      aoiFolPt.PutCoords dbPtCrdsWrt(lgSgmntCntr, 0), dbPtCrdsWrt(lgSgmntCntr, 1)
      aoiFeat.Value(aoiFeat.Fields.FindField("DipDir")) = intDipDir
      aoiFeat.Value(aoiFeat.Fields.FindField("DipAn")) = intDipAn
      aoiFeat.Value(aoiFeat.Fields.FindField("AppDip")) = Round(intAppDip, 0)
      aoiFeat.Store
    End If

    With FrmExprtPrfl.prgBr
      .Value = lgSgmntCntr
      .Refresh
    End With

  Next lgSgmntCntr

End Sub

 ' returns the components' calculation for the orientation measurement
 ' with or without Inverse Distance Weighted averaging
```

```
Function FillFolFields(aoiFolPt As IPoint, lgSgmntCntr As Long)

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntMp As IMap
  Dim aoiUID As UID
  Dim aoiEnumLyrs As IEnumLayer
  Dim aoiCrntLyr As IFeatureLayer
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFtCrs As IFeatureCursor
  Dim aoiCrntFt As IFeature
  Dim aoiRelOpPt As IRelationalOperator
  Dim aoiTopOpPt As ITopologicalOperator
  Dim aoiPtBuffer As IGeometry
  Dim aoiPnt As IPoint
  Dim intFtCntr As Integer
  Dim dbFolXYZ() As Double
  Dim dbX As Double
  Dim dbY As Double
  Dim dbZ As Double
  Dim dbIDWsum As Double
  Dim dbDipAn As Double
  Dim dbDipDir As Double
  Dim dbRadius As Double

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiUID = New UID
  aoiUID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"

  Set aoiEnumLyrs = aoiCrntMp.Layers(aoiUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr = aoiEnumLyrs.Next

 ' look for the respective layer

  Do While Not aoiCrntLyr Is Nothing
    If aoiCrntLyr.Name = strPntLyrName Then
      Exit Do
    End If
    Set aoiCrntLyr = aoiEnumLyrs.Next
  Loop

  aoiFolPt.PutCoords dbPtCrdsRd(lgSgmntCntr, 0), dbPtCrdsRd(lgSgmntCntr, 1)

 ' the array will have to be resized, so the
 ' coord.-values are stored in the first
 ' index [(0,...), (1,...) etc.]

  ReDim dbFolXYZ(3, 0)
  Set aoiTopOpPt = aoiFolPt
  Set aoiPtBuffer = aoiTopOpPt.Buffer(FrmExprtPrfl.txtBxOrientRad.Value)
  Set aoiRelOpPt = aoiPtBuffer
  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)

 ' loop through all foliation features, look for those
 ' inside the buffer region and store their values in a
 ' dynamic array

  For intFtCntr = 0 To aoiCrntFtCls.FeatureCount(Nothing) - 1

    Set aoiCrntFt = aoiFtCrs.NextFeature
    Set aoiPnt = aoiCrntFt.Shape

    If aoiRelOpPt.Contains(aoiPnt) Then
      dbDipDir = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirName)) * PI / 180
      dbDipAn = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAnName)) * PI / 180

' calculate the XYZ-coords from dip angle and dip direction

      dbFolXYZ(0, UBound(dbFolXYZ, 2)) = Sqr((aoiFolPt.x - aoiPnt.x) ^ 2 + (aoiFolPt.y - aoiPnt.y) ^ 2)
      dbFolXYZ(1, UBound(dbFolXYZ, 2)) = Cos(dbDipAn) * Sin(dbDipDir)
      dbFolXYZ(2, UBound(dbFolXYZ, 2)) = Cos(dbDipAn) * Cos(dbDipDir)
      dbFolXYZ(3, UBound(dbFolXYZ, 2)) = -Sin(dbDipAn)
      ReDim Preserve dbFolXYZ(3, UBound(dbFolXYZ, 2) + 1)
    End If
  Next intFtCntr

' storing the components' sum in the "last" record of the array
' if the buffer was not empty and regarding IDW if necessary

  If UBound(dbFolXYZ, 2) <> 0 Then
    Select Case blIDW

  ' without IDW
    Case False

  ' filling the last fields of the array with the sum of X-, Y- and Z-components
      For intFtCntr = 0 To UBound(dbFolXYZ, 2) - 1
        dbFolXYZ(1, UBound(dbFolXYZ, 2)) = dbFolXYZ(1, UBound(dbFolXYZ, 2)) _
                                           + dbFolXYZ(1, intFtCntr)
        dbFolXYZ(2, UBound(dbFolXYZ, 2)) = dbFolXYZ(2, UBound(dbFolXYZ, 2)) _
                                           + dbFolXYZ(2, intFtCntr)
        dbFolXYZ(3, UBound(dbFolXYZ, 2)) = dbFolXYZ(3, UBound(dbFolXYZ, 2)) _
                                           + dbFolXYZ(3, intFtCntr)
      Next intFtCntr

      dbX = dbFolXYZ(1, UBound(dbFolXYZ, 2))
      dbY = dbFolXYZ(2, UBound(dbFolXYZ, 2))
      dbZ = dbFolXYZ(3, UBound(dbFolXYZ, 2))

  ' with IDW
    Case True
  ' filling the last fields of the array with the sum of X-, Y- and Z-components
  ' and weighting it by division through the distance
        For intFtCntr = 0 To UBound(dbFolXYZ, 2) - 1
          dbFolXYZ(0, UBound(dbFolXYZ, 2)) = dbFolXYZ(0, UBound(dbFolXYZ, 2)) _
                                             + dbFolXYZ(0, intFtCntr) ^ -1
          dbFolXYZ(1, UBound(dbFolXYZ, 2)) = dbFolXYZ(1, UBound(dbFolXYZ, 2)) _
                                             + dbFolXYZ(1, intFtCntr) / dbFolXYZ(0, intFtCntr)
          dbFolXYZ(2, UBound(dbFolXYZ, 2)) = dbFolXYZ(2, UBound(dbFolXYZ, 2)) _
                                             + dbFolXYZ(2, intFtCntr) / dbFolXYZ(0, intFtCntr)
          dbFolXYZ(3, UBound(dbFolXYZ, 2)) = dbFolXYZ(3, UBound(dbFolXYZ, 2)) _
                                             + dbFolXYZ(3, intFtCntr) / dbFolXYZ(0, intFtCntr)
        Next intFtCntr

  ' normalising back for IDW
        dbIDWsum = dbFolXYZ(0, UBound(dbFolXYZ, 2))
        dbX = dbFolXYZ(1, UBound(dbFolXYZ, 2)) / dbIDWsum
        dbY = dbFolXYZ(2, UBound(dbFolXYZ, 2)) / dbIDWsum
        dbZ = dbFolXYZ(3, UBound(dbFolXYZ, 2)) / dbIDWsum

    End Select

  ' calculating the Dip Angle, the Dip Direction and the length of the "average-vector"
    dbRadius = Sqr(dbX ^ 2 + dbY ^ 2 + dbZ ^ 2)
    dbDipAn = Arcsin(-dbZ / dbRadius) * 180 / PI
    If dbX >= 0 Then
      dbDipDir = Arccos(dbY / (dbRadius * Cos(dbDipAn * PI / 180))) * 180 / PI
    Else
      dbDipDir = 360 - (Arccos(dbY / (dbRadius * Cos(dbDipAn * PI / 180))) * 180 / PI)
    End If
    FillFolFields = CStr(Round(dbDipDir, 0)) & " " & CStr(Round(dbDipAn, 0))
  Else
    FillFolFields = ""
  End If

End Function

 ' calculates the Arcus Sinus Private Function Arcsin(x As Double)
  If x = -1 Then
    Arcsin = PI / -2
  Else
    Arcsin = Atn(x / Sqr(-x * x + 1))
  End If
End Function

 ' calculates the Arcus Cosinus Private Function Arccos(x As Double)
  If x = -1 Then
    Arccos = PI
  Else
    Arccos = Atn(-x / Sqr(-x * x + 1)) + 2 * Atn(1)
  End If
End Function

Private Function ApparentDipCalc(DipDir As Double, DipAn As Double)
  Dim dbXDD As Double                ' X-component of dip direction
  Dim dbYDD As Double                ' Y-component of dip direction
  Dim dbDeltXprfl As Double          ' Difference in X-coord along profile
  Dim dbDeltYprfl As Double          ' Difference in Y-coord along profile
  Dim dbRadiusPrfl As Double         ' length of the profile
  Dim dbGamma_ As Double             ' angle between dip direction and strike of profile
  Dim dbResult As Double             ' apparent dip
  Dim blAngleCorr As Boolean         ' specifies if the dip angle has to be corrected because
                                     ' of an angle between profile and dip direction which
                                     ' is bigger than PI/2 (i.e. 90)

  blAngleCorr = False
  If DipAn = 90 Then DipAn = 89.99999
  DipDir = DipDir * PI / 180         ' convert angles in degree to radians
  DipAn = DipAn * PI / 180
  dbDeltXprfl = lgXend - lgXstart
  dbDeltYprfl = lgYend - lgYstart
  dbRadiusPrfl = Sqr(dbDeltXprfl ^ 2 + dbDeltYprfl ^ 2)
  dbXDD = Sin(DipDir)
  dbYDD = Cos(DipDir)

 ' applying scalar product rule
  dbGamma_ = Arccos((dbXDD * dbDeltXprfl + dbYDD * dbDeltYprfl) / dbRadiusPrfl)

  If dbGamma_ > PI / 2 Then
    dbGamma_ = PI - dbGamma_
    blAngleCorr = True
  End If
  dbResult = Atn(Cos(dbGamma_) * Tan(DipAn)) * 180 / PI
  Select Case blAngleCorr
  Case False
    ApparentDipCalc = Round(dbResult, 0)
  Case True
    ApparentDipCalc = 180 - Round(dbResult, 0)
  End Select
End Function

 ' returns the spatial reference of the geology polygons
Private Function GetSpatialReference()

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntMp As IMap
  Dim aoiUID As UID
  Dim aoiEnumLyrs As IEnumLayer
  Dim aoiCrntLyr As IFeatureLayer
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFtCrs As IFeatureCursor
  Dim aoiCrntFt As IFeature
  Dim aoiGeometry As IGeometry

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntMp = aoiCrntDoc.FocusMap
  Set aoiUID = New UID
  aoiUID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"

  Set aoiEnumLyrs = aoiCrntMp.Layers(aoiUID, True)
  aoiEnumLyrs.Reset
  Set aoiCrntLyr = aoiEnumLyrs.Next

  Do While Not aoiCrntLyr Is Nothing
    If aoiCrntLyr.Name = strGeoLayer Then
      Exit Do
    End If
    Set aoiCrntLyr = aoiEnumLyrs.Next
  Loop

  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)
  Set aoiCrntFt = aoiFtCrs.NextFeature
  Set aoiGeometry = aoiCrntFt.Shape
  Set GetSpatialReference = aoiGeometry.SpatialReference

End Function

 ' Terminates the execution of the module

Public Sub Killer()
  Unload FrmExprtPrfl
  End
End Sub
```

B.3.2 Spatial averaging

The name of this form in the VBA© programming environment in ArcMap© is *FrmOrntAvrg*!

```
Private Sub cmdBtRun_Click()

  If IsNumeric(FrmOrntAvrg.txtBxXstp.Value) And _
    IsNumeric(FrmOrntAvrg.txtBxYstp.Value) And _
    IsNumeric(FrmOrntAvrg.txtBxRds.Value) Then
      Call modOrntAvrg.Main
  Else:
    MsgBox "Please specify appropriate parameters!", , "Check parameters!"
    Exit Sub
  End If

End Sub

Private Sub UserForm_Activate()

 Dim aoiFields As esriCore.IFields
 Dim lgFldCntr As Long
 Dim aoiField As esriCore.IField
 Dim aoiCrntDoc As esriCore.IMxDocument
 Dim aoiCrntLayer As esriCore.IFeatureLayer
 Dim aoiCrntFtCls As esriCore.IFeatureClass

  FrmOrntAvrg.Height = 310
  Set aoiCrntDoc = ThisDocument
  Set aoiCrntLayer = aoiCrntDoc.SelectedLayer
  Set aoiCrntFtCls = aoiCrntLayer.FeatureClass

 ' loop through all fields of the feature class
 ' that has been found above

  Set aoiFields = aoiCrntFtCls.Fields
  For lgFldCntr = 0 To (aoiFields.FieldCount - 1)
    Set aoiField = aoiFields.Field(lgFldCntr)
    If aoiField.Name <> "Shape" Then
      With FrmOrntAvrg
        .lstBxDipDir.AddItem aoiField.Name
        .lstBxDipAn.AddItem aoiField.Name
      End With
    End If
  Next lgFldCntr

End Sub
```

The name of this module in the VBA© programming environment in ArcMap© is *modOrntAvrg*!

```
Option Explicit

Dim dbNRows As Double Dim dbNCols As Double

Public Sub CheckLayerSelection()

  Dim aoiCrntDoc As esriCore.IMxDocument
  Set aoiCrntDoc = ThisDocument
  If Not aoiCrntDoc.SelectedLayer Is Nothing Then

    If TypeOf aoiCrntDoc.SelectedLayer Is IFeatureLayer Then
      Dim aoiCrntLayer As esriCore.IFeatureLayer
      Dim aoiCrntFtCls As esriCore.IFeatureClass
      Set aoiCrntLayer = aoiCrntDoc.SelectedLayer
      Set aoiCrntFtCls = aoiCrntLayer.FeatureClass

      If aoiCrntFtCls.ShapeType = esriGeometryPoint Then
        FrmOrntAvrg.Show
      Else
        MsgBox ("The selected layer doesn't contain appropriate data!")
      End If

    Else
      MsgBox ("The selected layer is not a feature layer!")
    End If

  Else
    MsgBox ("Please select the layer containing your planar measuremets!")
  End If

End Sub

Public Sub Main()

  Dim dbCoords() As Double

  Call ShapeConstruct
  Call CalcCoords(dbCoords)
  Call ShapeFiller(dbCoords)
  Call NewShapeLooper

  Unload FrmOrntAvrg

End Sub

Private Sub ShapeFiller(dbCoords() As Double)

  Dim strShpPth As String
  Dim strPntShpName As String

  strShpPth = FrmOrntAvrg.txtBxShpPth.Value
  strPntShpName = FrmOrntAvrg.txtBxShpName.Value

  Dim aoiFWS As IFeatureWorkspace
  Dim aoiWorkspaceFactory As IWorkspaceFactory
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFeat As IFeature

  Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
  Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strShpPth, 0)
  Set aoiCrntFtCls = aoiFWS.OpenFeatureClass(strPntShpName)

  Dim i As Long
  Dim j As Long
  Dim aoiFolPt As IPoint

  FrmOrntAvrg.prgB.Value = 0
  FrmOrntAvrg.prgB.Max = dbNRows - 1

  For i = 0 To dbNRows - 1
    For j = 0 To dbNCols - 1

      Set aoiFolPt = New Point
      aoiFolPt.PutCoords dbCoords(j, i, 0), dbCoords(j, i, 1)
      Set aoiFeat = aoiCrntFtCls.CreateFeature
      Set aoiFeat.Shape = aoiFolPt
      aoiFeat.Store

    Next j

    FrmOrntAvrg.prgB.Value = i
    FrmOrntAvrg.prgB.Refresh

  Next i

End Sub

Private Sub ShapeConstruct()

On Error GoTo Err

  FrmOrntAvrg.Height = 340
  FrmOrntAvrg.Repaint

  Dim strShpPth As String
  Dim strPntShpName As String
  Const strShapeFieldName As String = "Shape"

  strShpPth = FrmOrntAvrg.txtBxShpPth.Value
  strPntShpName = FrmOrntAvrg.txtBxShpName.Value

' Open the folder to contain the shapefile as a workspace

  Dim aoiFWS As IFeatureWorkspace
  Dim aoiWorkspaceFactory As IWorkspaceFactory
  Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
  Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strShpPth, 0)

' Set up a simple fields collection

  Dim aoiFields As IFields
  Dim aoiFieldsEdit As IFieldsEdit
  Set aoiFields = New esriCore.Fields
  Set aoiFieldsEdit = aoiFields
  Dim aoiField As IField
  Dim aoiFieldEdit As IFieldEdit

' Make the shape field including its spatial reference

  Set aoiField = New esriCore.Field
  Set aoiFieldEdit = aoiField

' specify that you deal with a shape file

  aoiFieldEdit.Name = strShapeFieldName
  aoiFieldEdit.Type = esriFieldTypeGeometry
  Dim aoiGeomDef As IGeometryDef
  Dim aoiGeomDefEdit As IGeometryDefEdit
  Set aoiGeomDef = New GeometryDef
  Set aoiGeomDefEdit = aoiGeomDef
  With aoiGeomDefEdit
    .geometryType = esriGeometryPoint
    .GridCount = 1
    .GridSize(0) = 10
    .AvgNumPoints = 2
    .HasM = False
    .HasZ = False

 ' calls a function to find out the spatial reference of the
 ' point shape layer on which the calculations will be based
```

```
    Set .SpatialReference = GetSpatialReference()

  End With
  Set aoiFieldEdit.GeometryDef = aoiGeomDef
  aoiFieldsEdit.AddField aoiField

  Set aoiField = New esriCore.Field
  Set aoiFieldEdit = aoiField
  With aoiFieldEdit
      .Length = 10
      .Name = "ID"
      .Type = esriFieldTypeInteger
  End With
  aoiFieldsEdit.AddField aoiField

  Set aoiField = New esriCore.Field
  Set aoiFieldEdit = aoiField
  With aoiFieldEdit
      .Length = 5
      .Name = "DipDir"
      .Type = esriFieldTypeInteger
  End With
  aoiFieldsEdit.AddField aoiField

  Set aoiField = New esriCore.Field
  Set aoiFieldEdit = aoiField
  With aoiFieldEdit
      .Length = 5
      .Name = "DipAn"
      .Type = esriFieldTypeInteger
  End With
  aoiFieldsEdit.AddField aoiField

 ' Create the shapefile - i.e. a new Feature Class

  Dim aoiFeatClass As IFeatureClass
  Set aoiFeatClass = aoiFWS.CreateFeatureClass(strPntShpName, aoiFields, Nothing, _
                                          Nothing, esriFTSimple, strShapeFieldName, "")

Exit Sub

Err:
  MsgBox Err.Description & Chr$(13) & "Error Number is: " & Err.Number _
  & Chr(13) & "The macro will terminate !", vbCritical, _
  "An error occurred - sorry for the inconvenience!"

  End

End Sub

 ' returns the spatial reference of the current map

Private Function GetSpatialReference()

  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntLyr As IFeatureLayer
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiCrntFt As IFeature
  Dim aoiGeometry As IGeometry
  Dim aoiFtCrs As IFeatureCursor

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntLyr = aoiCrntDoc.SelectedLayer
  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)
  Set aoiCrntFt = aoiFtCrs.NextFeature
  Set aoiGeometry = aoiCrntFt.Shape
  Set GetSpatialReference = aoiGeometry.SpatialReference

End Function

Private Sub CalcCoords(dbCoords() As Double)

  Dim aoiCrntDoc As esriCore.IMxDocument
  Dim aoiCrntLayer As esriCore.IFeatureLayer
  Dim aoiCrntFtCls As esriCore.IFeatureClass
  Dim dbXmin As Double
  Dim dbXmax As Double
  Dim dbYmin As Double
  Dim dbYmax As Double
  Dim dbIncrX As Double
  Dim dbIncrY As Double

  Set aoiCrntDoc = ThisDocument
  Set aoiCrntLayer = aoiCrntDoc.SelectedLayer

  dbXmin = aoiCrntLayer.AreaOfInterest.xmin
  dbXmax = aoiCrntLayer.AreaOfInterest.xmax
  dbYmin = aoiCrntLayer.AreaOfInterest.ymin
  dbYmax = aoiCrntLayer.AreaOfInterest.ymax
  dbNCols = FrmOrntAvrg.txtBxXstp.Value
  dbNRows = FrmOrntAvrg.txtBxYstp.Value

  Dim i As Integer
  Dim j As Integer

  ReDim dbCoords(dbNCols, dbNRows, 2)
  dbIncrX = (dbXmax - dbXmin) / dbNCols
  dbIncrY = (dbYmax - dbYmin) / dbNRows

  For i = 0 To dbNRows - 1
    For j = 0 To dbNCols - 1
      dbCoords(j, i, 0) = (j + 0.5) * dbIncrX + dbXmin
      dbCoords(j, i, 1) = (i + 0.5) * dbIncrY + dbYmin
    Next j
  Next i

End Sub

Private Sub NewShapeLooper()

  FrmOrntAvrg.lblPrgBar.Caption = "Filling Shape-File ..."
  FrmOrntAvrg.Repaint

  Dim strShpPth As String
  Dim strPntShpName As String
  Dim aoiFWS As IFeatureWorkspace
  Dim aoiWorkspaceFactory As IWorkspaceFactory
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiFtCrs As IFeatureCursor
  Dim i As Long
  Dim aoiCrntFt As IFeature
  Dim aoiFolPt As IPoint
  Dim strOrient As String
  Dim strDipDir As String
  Dim strDipAn As String

  strShpPth = FrmOrntAvrg.txtBxShpPth.Value
  strPntShpName = FrmOrntAvrg.txtBxShpName.Value

  Set aoiWorkspaceFactory = New ShapefileWorkspaceFactory
  Set aoiFWS = aoiWorkspaceFactory.OpenFromFile(strShpPth, 0)
  Set aoiCrntFtCls = aoiFWS.OpenFeatureClass(strPntShpName)
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)

  FrmOrntAvrg.prgB.Value = 0
  FrmOrntAvrg.prgB.Max = aoiCrntFtCls.FeatureCount(Nothing) - 1

  For i = 0 To aoiCrntFtCls.FeatureCount(Nothing) - 1

    Set aoiCrntFt = aoiFtCrs.NextFeature
    Set aoiFolPt = aoiCrntFt.Shape
    strOrient = CalcOrient(aoiFolPt)
    If strOrient <> "" Then
      strDipDir = Split(strOrient, " ")(0)
      strDipAn = Split(strOrient, " ")(1)
      aoiCrntFt.Value(aoiCrntFt.Fields.FindField("DipDir")) = strDipDir
      aoiCrntFt.Value(aoiCrntFt.Fields.FindField("DipAn")) = strDipAn
      aoiCrntFt.Store
    End If

    FrmOrntAvrg.prgB.Value = i
    FrmOrntAvrg.prgB.Refresh

  Next i

End Sub

Private Function CalcOrient(aoiFolPt As IPoint)

On Error GoTo Err

  Const PI As Double = 3.14159265358979
  Dim strOrient As String
  Dim aoiCrntDoc As IMxDocument
  Dim aoiCrntLyr As IFeatureLayer
  Dim aoiCrntFtCls As IFeatureClass
  Dim aoiCrntFt As IFeature
  Dim aoiPtBuffer As IGeometry
  Dim aoiFtCrs As IFeatureCursor
  Dim aoiTopOpPt As ITopologicalOperator
  Dim aoiRelOpPt As IRelationalOperator
  Dim intFtCntr As Long
  Dim aoiPnt As IPoint
  Dim dbFolXYZ() As Double
  Dim dbX As Double
  Dim dbY As Double
  Dim dbZ As Double
  Dim dbIDWsum As Double
  Dim dbDipAn As Double
  Dim dbDipDir As Double
  Dim dbRadius As Double
  Dim blIDW As Boolean
  Dim strDipDirName As String
  Dim strDipAnName As String

  blIDW = FrmOrntAvrg.chkBxIDW.Value
  Set aoiCrntDoc = ThisDocument
  Set aoiCrntLyr = aoiCrntDoc.SelectedLayer
  Set aoiTopOpPt = aoiFolPt
  Set aoiPtBuffer = aoiTopOpPt.Buffer(FrmOrntAvrg.txtBxRds.Value)
  Set aoiRelOpPt = aoiPtBuffer
  Set aoiCrntFtCls = aoiCrntLyr.FeatureClass
  Set aoiFtCrs = aoiCrntFtCls.Search(Nothing, False)
  ReDim dbFolXYZ(3, 0)
  strDipDirName = FrmOrntAvrg.lstBxDipDir.Value
  strDipAnName = FrmOrntAvrg.lstBxDipAn.Value

  For intFtCntr = 0 To aoiCrntFtCls.FeatureCount(Nothing) - 1
```

```
   Set aoiCrntFt = aoiFtCrs.NextFeature
   Set aoiPnt = aoiCrntFt.Shape

   If aoiRelOpPt.Contains(aoiPnt) Then
     dbDipDir = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipDirName)) * PI / 180
     dbDipAn = aoiCrntFt.Value(aoiCrntFt.Fields.FindField(strDipAnName)) * PI / 180

' calculate the XYZ-coords from dip angle and dip direction

     dbFolXYZ(0, UBound(dbFolXYZ, 2)) = Sqr((aoiFolPt.x - aoiPnt.x) ^ 2 + (aoiFolPt.y - aoiPnt.y) ^ 2)
     dbFolXYZ(1, UBound(dbFolXYZ, 2)) = Cos(dbDipAn) * Sin(dbDipDir)
     dbFolXYZ(2, UBound(dbFolXYZ, 2)) = Cos(dbDipAn) * Cos(dbDipDir)
     dbFolXYZ(3, UBound(dbFolXYZ, 2)) = -Sin(dbDipAn)
     ReDim Preserve dbFolXYZ(3, UBound(dbFolXYZ, 2) + 1)
   End If

 Next intFtCntr

' storing the components' sum in the "last" record of the array
' if the buffer was not empty and regarding IDW if necessary

 If UBound(dbFolXYZ, 2) <> 0 Then
   Select Case blIDW

 ' without IDW
   Case False

 ' filling the last fields of the array with the sum of X-, Y- and Z-components
     For intFtCntr = 0 To UBound(dbFolXYZ, 2) - 1
       dbFolXYZ(1, UBound(dbFolXYZ, 2)) = dbFolXYZ(1, UBound(dbFolXYZ, 2)) _
                                          + dbFolXYZ(1, intFtCntr)
       dbFolXYZ(2, UBound(dbFolXYZ, 2)) = dbFolXYZ(2, UBound(dbFolXYZ, 2)) _
                                          + dbFolXYZ(2, intFtCntr)
       dbFolXYZ(3, UBound(dbFolXYZ, 2)) = dbFolXYZ(3, UBound(dbFolXYZ, 2)) _
                                          + dbFolXYZ(3, intFtCntr)
     Next intFtCntr

     dbX = dbFolXYZ(1, UBound(dbFolXYZ, 2))
     dbY = dbFolXYZ(2, UBound(dbFolXYZ, 2))
     dbZ = dbFolXYZ(3, UBound(dbFolXYZ, 2))

 ' with IDW
   Case True
 ' filling the last fields of the array with the sum of X-, Y- and Z-components
 ' and weighting it by division through the distance
       For intFtCntr = 0 To UBound(dbFolXYZ, 2) - 1
         dbFolXYZ(0, UBound(dbFolXYZ, 2)) = dbFolXYZ(0, UBound(dbFolXYZ, 2)) _
                                            + dbFolXYZ(0, intFtCntr) ^ -1
         dbFolXYZ(1, UBound(dbFolXYZ, 2)) = dbFolXYZ(1, UBound(dbFolXYZ, 2)) _
                                            + dbFolXYZ(1, intFtCntr) / dbFolXYZ(0, intFtCntr)
         dbFolXYZ(2, UBound(dbFolXYZ, 2)) = dbFolXYZ(2, UBound(dbFolXYZ, 2)) _
                                            + dbFolXYZ(2, intFtCntr) / dbFolXYZ(0, intFtCntr)
         dbFolXYZ(3, UBound(dbFolXYZ, 2)) = dbFolXYZ(3, UBound(dbFolXYZ, 2)) _
                                            + dbFolXYZ(3, intFtCntr) / dbFolXYZ(0, intFtCntr)
       Next intFtCntr

 ' normalising back for IDW
       dbIDWsum = dbFolXYZ(0, UBound(dbFolXYZ, 2))
       dbX = dbFolXYZ(1, UBound(dbFolXYZ, 2)) / dbIDWsum
       dbY = dbFolXYZ(2, UBound(dbFolXYZ, 2)) / dbIDWsum
       dbZ = dbFolXYZ(3, UBound(dbFolXYZ, 2)) / dbIDWsum

   End Select

 ' calculating the Dip Angle, the Dip Direction and the length of the "average-vector"
   dbRadius = Sqr(dbX ^ 2 + dbY ^ 2 + dbZ ^ 2)
   dbDipAn = Arcsin(-dbZ / dbRadius) * 180 / PI
```

```
    If dbX >= 0 Then
      dbDipDir = Arccos(dbY / (dbRadius * Cos(dbDipAn * PI / 180))) * 180 / PI
    Else
      dbDipDir = 360 - (Arccos(dbY / (dbRadius * Cos(dbDipAn * PI / 180))) * 180 / PI)
    End If
    CalcOrient = CStr(Round(dbDipDir, 0)) & " " & CStr(Round(dbDipAn, 0))

  Else
    CalcOrient = "-9999 -9999"
  End If

Exit Function

Err:
  MsgBox Err.Description & Chr$(13) & "Error Number is: " & Err.Number _
  & Chr(13) & "The macro will terminate", vbCritical, _
  "An error occurred - sorry for the inconvenience!"

  End

End Function

 ' calculates the Arcus Sinus
Private Function Arcsin(x As Double)
  Const PI As Double = 3.14159265358979
  If Abs(Fix(x)) <> 1 Then
    Arcsin = Atn(x / Sqr(-x * x + 1))
  Else:
    Arcsin = Sgn(x) * PI / 2
  End If
End Function

 ' calculates the Arcus Cosinus
Private Function Arccos(x As Double)
  Const PI As Double = 3.14159265358979
  If Abs(Fix(x)) <> 1 Then
    Arccos = Atn(-x / Sqr(-x * x + 1)) + 2 * Atn(1)
  Else:
    Arccos = 0.5 + (-x / 2) * PI
  End If
End Function
```