Developer Kit

ASCII file data formats

This chapter covers the GOCAD data file format. It is adapted from the GOCAD Developer's Guide. If you have any questions or need more information, please contact us.

A.1 Basics

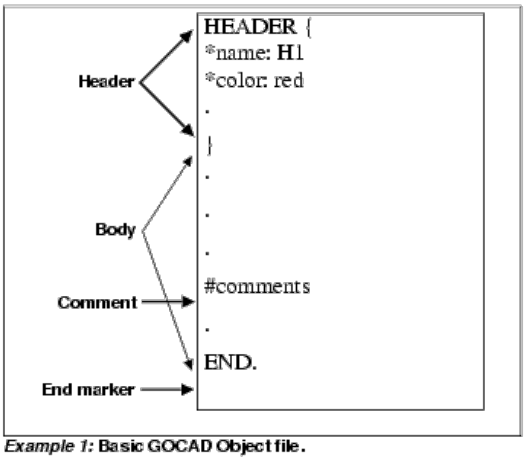
A GOCAD Object file consists of three parts: the Header, the Body and the End marker. A GOCAD Object file may also contain Comments that start with a # sign on each comment line. .

A.1.1 Header

The header gives the Object Type name (GOCAD TSurf) and the version number (1.0).

The case of the Object Type name matters. The version number is not necessary. The Object Type name and version number are used to call up the correct GOCAD Object Reader when the file is loaded.

Also stored in the Header section are Attribute values, which include the Object's name (HEADER{...}).



Example 1: **Basic GOCAD Object file.**

A.1.2 Body

The Body contains the geometrical and Property information of the Object.

Since GOCAD Objects are inheriting in nature, so are the Object files.

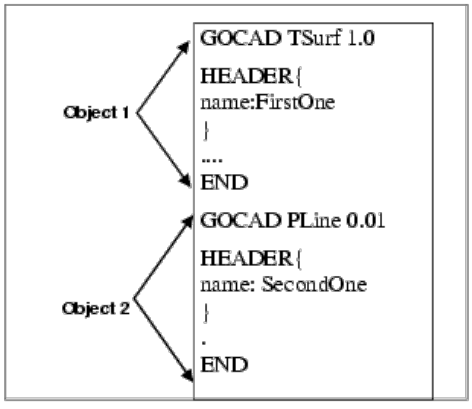
For example, TSurf (Surface) is derived from the Atomic Object Class (Type), so a TSurf file contains all the file elements of an Atomic Object, plus some extras.

All GOCAD Objects are derived from the Object Class, therefore all GOCAD Objects contain the file elements described in Object (PageA4).

When describing the ASCII format of the different Classes (Types) of GOCAD objects, only the extra elements are described to avoid redundancy.

A.1.3 END

The trailer, END, provides an end-of-object marker. Because of the existence of the End marker, multiple Objects can be stored in a single file. See Figure2:



Example 2: **Multiple GOCAD Objects file.**

A.1.4 Comments

GOCAD allows comments in any part of an Object file, as long as the first column of the line is a # sign.

A.1.5 Units

Units are not fully implemented in the current version of GOCAD, but theoretically, Units may be specified as shown below:

seconds meters
meters/seconds^2
meters/seconds
seconds*meters
seconds
3600*seconds

0.44704*meters/seconds
0.3048*meters/seconds
0.3048*seconds*meters

Restrictions: at most one scalar in front, at most one division.

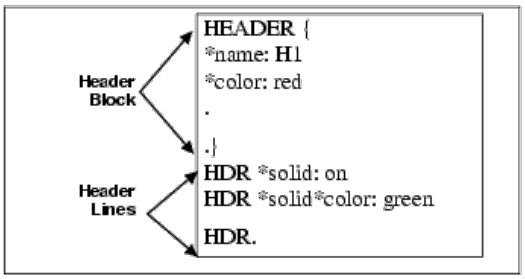
A.2 Object

The Object file elements define the Object style and the Object coordinate space and Units. Style Attributes

The HEADER block, delimited by {and}, contains the Style definition. Each line defines an Attribute of an Object. Attributes are strictly optional, except for the name of the Object. An Object must have a name.

If preferred, an Attribute can be given on a separate line outside of the HEADER block; in that case the line should begin with the HDR keyword as shown in Example 3 (page 4).

An example of Object Style element is:



Example 3: **Object Style file element.**

A.2.1 Coordinate System

The Object Coordinate System paragraph is optional for all objects. If the paragraph is not present, the object is loaded in the default coordinate system in the current session.

The coordinate system description must be given after the HEADER block and before any other keyword giving the geometry of the object such as VRTX, PVRTX, ORIGIN, AXIS ....

The Coordinate System must be given as follows:

GOCAD_ORIGINAL_COORDINATE_SYSTEM
NAME name
AXIS_NAME name_u name_v name_w
AXIS_UNIT unit_u unit_v unit_w
ZPOSITIVE keyword
END_ORIGINAL_COORDINATE_SYSTEM

It is imperative that the Coordinate System definition is enclosed between the keywords GOCAD_ORIGINAL_COORIDNATE_SYSTEM and END_ORIGINAL_COORDINATE_SYSTEM.

The NAME and AXIS_NAME values are information string which gives the name of the local object coordinate system and its axes. The values must be specified by strings of characters enclosed in double-quotes, i.e. AXIS_NAME "X" "Y" "Z" or AXIS_NAME "X" "Y" "T"

The AXIS_UNIT specifies the unit for each axes. The axis unit for the U and V axes must be identical, the axis unit for the W axis can be different from the unit of the U and V. It is possible to define units as AXIS_UNIT "m" "m" "m" or AXIS_UNIT "m" "m" "ft" but a definition such as AXIS_UNIT "m" "ft" 'km" is invalid. The AXIS_UNIT values must be specified by strings of characters enclosed by double-quotes.

The ZPOSITIVE value indicates the direction of the Z axis. The values allowed for the keyword after ZPOSITIVE are either Depth (Z is increasing downwards) or Elevation (Z is increasing upwards)

The coordinates are read in double precision converted to simple precision after coordinate system conversion. When an object is loaded into Gocad, its local coordinate system is compared with the current global coordinate system: if needed, the object geometrical coordinates are converted internally. When the coordinates are displayed, for example, for a picking in the 3D camera, the coordinates displayed are in the session coordinate system. When the object is saved back to a stand alone file, its geometrical coordinates are restored in its original coordinates system.

A.3 Atomic

The Atomic data format defines the points, locations and Property values, of an Atomic (currently, this includes PointsSet, Curve, Surface, Solid and GShape). The Atomic inherits all of the Object file format elements, plus Atomic data.

A.3.1 Atomic

This section describes the recommended Atomic file format.

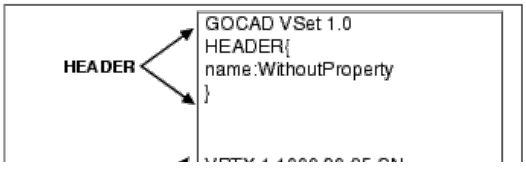
Since Property values are optional, a basic Atomic data line consists of three parts: Type, ID number, and data:

VRTX ID X Y Z [CN]

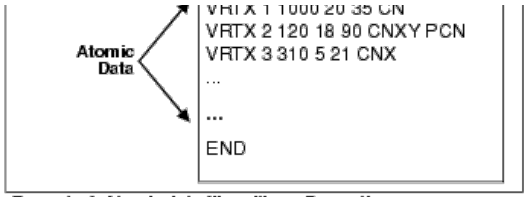
where the XYZ describes the location of that point. Additional information maybe attached at the end of each VRTX line to specify the Interpolation Restriction on that Atom (Control Node information).For example:

A.3.1.1 Properties definition

If there are Properties, they must be declared after the HEADER block and before the first point data line as:



PROPERTIES Pname1 Pname2...
PROPERTY_CLASSES class_name1 class_name2 (optional, but if you declare one, you must declare all)
UNITS unit1 unit2...(optional, but if you declare one, you must declare all)
NO_DATA_VALUES v1 v2...(optional, but if you declare one, you must declare all)
ESIZES esize_1 esize_2... (optional, but if you declare one, you must declare all)



Example 4: **Atomic data file, with no Properties.**

The PROPERTIES line list all the properties that will be attached to the point. The ordering of the properties on the line correspond to the ordering of the properties values associated to the point definition.

The PROPERTY_CLASSES line lists the property class names for each property defined in the PROPERTIES list. If one property class must be listed all property classes must be listed.

The NO_DATA_VALUES line lists the no-data-values for each of the property defined in the property list. For vectorial properties, there is only one no-data-value and each property vector element should be equal to this value if it is to be treated as a no-data-value vectorial property.

The UNITS line lists the units for each of the property defined in the property list (See Units for format specification of Units).

The ESIZES line lists the dimension of each property defined in the property list. By default the dimension of a property is 1, but for example a property representing a 3D vector will be of dimension 3. For example. if you have a set of points with a 1 dimensional property (called "porosity") and a 3D vectorial property called "throw", the definition of the properties and one point will look like:

PROPERTIES porosity throw ESIZES 1 3 PVRTX 1 X Y Z porosity_value throw_x throw_y throw_z

where throw_x, throw_y and throw_z are the three components of the throw vector at the specified location.

A.3.1.2 Point location and properties definition

The data line which defines a point location and its properties look likes:

PVRTX ID X Y Z PV1 PV2...

where PV1, PV2... are Property values. For example.

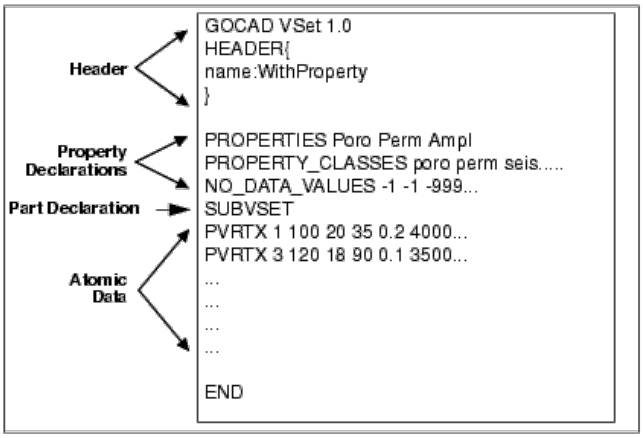
A.3.2 Parts (Subset)

An Atomic Object can have its VRTX grouped into subsets. In a VSet, the keyword is SUBVSET; in PLine, it is ILINE; in TSurf, it is TFACE; and in TSOLID, it is TVOLUME.

A.3.3 Atomic-Old from C-GOCAD

This section describes an older Atomic format, mainly made to be compatible with C-GOCAD file format.

The format consists of two sections, the location section and the Property section. The Property section is optional.



Example 5: **Atomic data file, with Properties.**

The location section is identical to the new format:

VRTX ID X Y Z

The Property value section contains a header and the list of Property values associated to a point defined in the previous section. The header defines the names of the DataPack Fields (Properties) associated with the Atomic. The following example Example 5 (page 8)declares three properties.

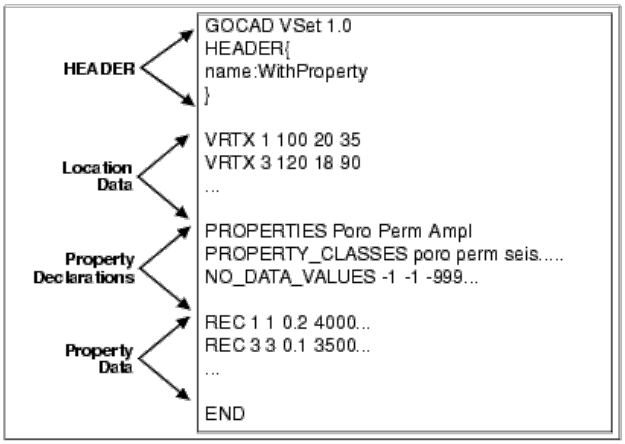
FIELDS Pname1 Pname2
(or PROPERTIES Pname1 Pname2...)
PROPERTY_CLASSES class_name1 class_name2 (optional, but if you declare one, you must declare all)
UNITS unit1 unit2...(optional, but if you declare one, you must declare all)
NO_DATA_VALUES v1 v2...(optional, but if you declare one, you must declare all)
ESIZES esize_1 esize_2... (optional, but if you declare one, you must declare all)

The Property section define, for each point defined in the location section, the Property values associated with the Atomic's DataPackFields.

REC ID1 ID2 PV1 PV2...

Associated with a REC definition are two identifiers needed to be specified for compatibility reasons with earlier version. The two identifiers must be identical.

An example of a Object file identical to the one given in Example 5 (page 8), but in this old data format is given below:.



***Example 6:* Old Atomic data file, with Properties.**

A.4 TSurf (Surface)

The data file of a TSurf includes the inherited file elements of a Object and of an Atomic. The Atoms (Vertices) of the TSurf must be defined before the triangles.

A.4.1 Geologic Information

Before defining vertices and triangles, geologic information can be given.Geological information consists of GEOLIGICAL_TYPE, GEOLOGICAL_FEATURE and STRATIGRAPHIC_POSITION.

The geological type of a TSurf can be specified using the following format:

GEOLOGICAL_TYPE geological_type_name

where geological_type_name can be one the following: `top`, `intraformational`, `fault`, `unconformity`, `intrusive`, `topography`, `boundary`, and `ghost`. The information is used during layer construction.

The stratigraphic position is specified as follow:

STRATIGRAPHIC_POSITION age time

where `age` is the name of a stratigraphic unit. `time` is the numerical value of age (35000, etc.). This information is used during layer computation.

The geological feature is specified as follow:

GEOLOGICAL_FEATURE geologic_feature_name

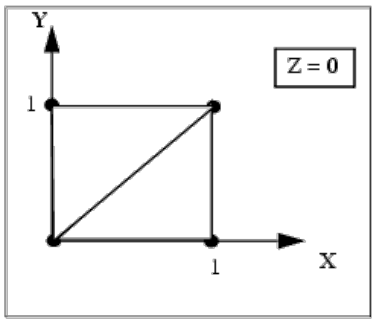
where `geologic_feature_name` is the name of the geologic feature that the surface represents. This information is used to associate surface to well markers (the geologic feature should have the same name than the well marker).

A.4.2 Triangles

Triangles definition must occur after vertices definitions. Each Triangle is then defined by its three Vertices (Atoms) in the following format:

TRGL id1 id2 id3

where ids are the IDs of the already-defined Atoms (Vertices). For example, the ascii file corresponding to a TSurf named SQUARE, containing four vertices, and two triangles is shown in Figure1.




```
GOCAD TSurf 1
HEADER{
name:SQUARE
}
GEOLOGICAL_TYPE top
TFACE
VRTX 1  0.  0.  0.
VRTX 2  1.  0.  0.
VRTX 3  1.  1.  0.
VRTX 4  0.  1.  0.
TRGL 1  2  3
TRGL 1  3  4

END
```

FIGURE 1. An example of a TSurf and its data file.

A.4.3 Special Points: VRTX vs. ATOM

In addition to the VRTX or PVRTX point/vertex descriptor find in the Atomic format definition, on can find the additional line inside a TSurf ascii file:

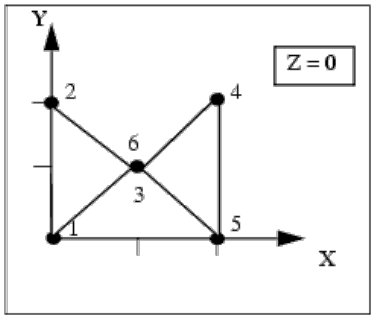
ATOM id1 id2 (where id1 > id2)

where `id1` is the index of the new ATOM and `id2` is the index of the already existing VRTX node which XTZ this Atom shares.

The goal of the ATOM keyword is to create a new ATOM but which shares an already existing vertex. An Atom node has its one Property values but it is spatially linked to the existing VRTX node. In other word, an ATOM and its referenced VRTX are collocated, but not connected. Triangles construction will use the vertex id or the atom id.

An example of the use of such Atom record is given below:

```
GOCAD TSurf 1
HDR name:s1
VRTX 1 0 0 0
VRTX 2 0 2 0
VRTX 3 1 1 0
VRTX 4 2 2 0
VRTX 5 2 0 0
TRGL 1 2 3
TRGL 3 4 5
END
```




```
GOCAD TSurf 1
HDR name:s1
TFACE
VRTX 1 0 0 0
VRTX 2 0 2 0
VRTX 3 1 1 0
TRGL 1 2 3
TFACE
VRTX 4 2 2 0
VRTX 5 2 0 0
ATOM 6 3
TRGL 4 5 6
END
```

FIGURE 2. An example of a shared vertex TSURF.

The input Tsurf file on the left will create a Surface shown at the center. The Surface has two TFaces, because two Triangles need to share an edge to be considered connected. When the Surface is saved, GOCAD will output the file on the right. It recognizes that the two Triangles both have one vertex at an identical location, but topologically, they can not be the same point. So, GOCAD creates a new point, ATOM 6, that shares the identical location of VRTX 3, but is an independent point in the sense that it has its own Property values (not in this case). However, had you have two more triangles in your input file, TRGL 2 4 3 and TRGL 6 5 1, the ATOM line would not have been created because the four triangles are then connected through direct or indirect edge-sharing.

A.4.4 Parts/TFace

The TFACE keyword is proposed to ensure identical indexing of the VRTX every time you save an Surface whose mesh you have not modified.

In Figure2 the file on the right (the one output by GOCAD) not only has the extra ATOM line, it also two TFACE lines to separate VRTX (and ATOM) and TRGL from different TFaces. They are
currently commented out for their full implementation is still to be determined by the GOCAD Committee.

GOCAD will accept the simpler format shown on the right of Figure2 (where all the triangles and VRTX definition of all parts are all merged into one block) but will output the file shown on the right of Figure2.

A.4.5 Borders

An edge of a TSurf can be a single piece or can be divided into multiple pieces. Each piece is called a Border, and it is separated from its neighboring Border by a Border Extremity. A Border Extremity is a designated node (VRTX, ATOM) on an edge of a TSurf that separates two Borders.

Borders definition should appear at the end of the TSurf file format, in the format shown below:

BSTONE atom_vrtx_id
BORDER border_id bstone_id_1 bstone_id_2

The BSTONE line defines an ATOM/VRTX as a Border Extremity.

The BORDER line defines the border, border_id, as starting at vrtx_id1, which must have been declared as a Border Extremity in proceeding BSTONE lines, and continuing in the direction of vrtx_id2, which must be adjacent to vrtx_id1 (this border ends when it runs into another Border Extremity).

The Border Extremities are always included in the list of points along the border.

A.5 PLine (Curve)

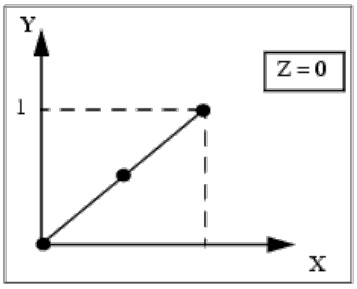
The Ascii PLine format inherits from the Object and from the Atomic formats. As for the TSurf, geologic information can be specified (See Geologic Information (PageA10)). For PLine geologic feature is used to relate multiple PLine to the same horizon.

The Atoms of the Lines must be defined first. Each Segment in the PLine is then defined by its 2 apices in the following format:

```
SEG id1 id2
```

where ids are the IDs of the already-defined Atoms.

For example, the ascii file corresponding to he PLine represented in Figure3 may look like:




```
GOCAD PLine 1
HEADER{
name:TwoSeg
}
ILINE
VRTX 1  0.  0.  0.
VRTX 2  1.  1.  0.
VRTX 3  0.5  0.5  0.
SEG 1 3
SEG 3 2

END
```

FIGURE 3. An example of a PLine and its data file.

If no segments information is given, the atoms are assumed to form an open line, from the first atom to the last atom in the file order.

If the PLine is composed of multiple parts, the parts can be combined or can be separated into ILines (or Isolated Lines). Each ILine is separated by a ILINE keyword and each ILine specify its own points and segments. In the same manner as for the TSurf, an ILINE keyword is proposed to ensure identical indexing of the VRTX every time you save a line which topology has not modified.

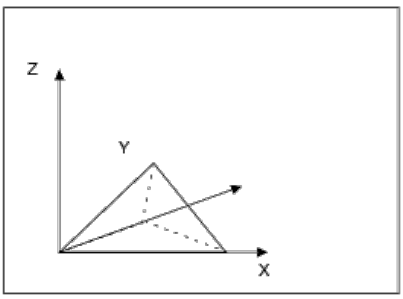
A.6 TSolid (Solid)

The Ascii TSolid format inherits from the Object and from the Atomic formats. The Atoms of the Atomic must be defined first. Each Tetrahedron in the Solid is then defined by its 4 apices in the following format.:

```
TETRA id1 id2 id3 id4
```

where ids are the IDs of the already-defined Atoms.

For example, the ascii file corresponding to he Solid represented in Figure4 may look like:




```
GOCAD TSolid 1
HEADER{
name: Pyramid
}
TVOLUME
VRTX 1  0.  0.  0.
VRTX 2  1.  0.  0.
VRTX 3  0.  1.  0.
VRTX 3  0.5  0.5  0.8
TETRA 1 2 3 4
END
```

FIGURE 4. An example of a TSolid and its data file.

If the TSolid is composed of multiple parts, the parts can be combined or can be separated into TVolumes (or Tetrahedralize volumes). Each TVolume is separated by a TVOLUME keyword and each TVolume specify its own points and tetrahedras.

A.7 Well

The Well ASCII format contains (in addition to the sections relative to an Object definition) 4 sections: a header section, a wellpath section, a well curves (log) section and a zone/marker section.

A.7.1 WellPath

The WellPath is composed of a header and a coordinate section: The header defines the reference point and the datum plane. The coordinate section defines the well path by giving either a series of PATH definition or VRTX definition.

```
WREF X Y Z
DPLN datum_plane
PATH Zm Z dX dY
VRTX X Y Z
```

WREF gives the X and Y coordinates of the reference point for the well path; usually the surface location of the well. The Z value of the WREF is used as the depth for drawing the derrick and the 0 position of the Well Name. In GOCAD we do not have TVD, and the measured-Z and true-Z (SSTVD) are explicitly defined in every PATH statement and cannot be modified by changing the WREF-z. In other applications when one adjusts the KB value, the entire measured-depth vs. True-z realtionship is modified, NOT in GOCAD. when you change the WREF-z, the only thing that changes is where the derrick is displayed. Therefore WREF-z is not used as KB by GOCAD.

PATH describes a point of the WellPath: giving a measured depth, Zm; a real depth (with KB or RT elevation already taken out; the "real world" Z coordinate), Z; and x, y deviations, dX and dY (relative to the X and Y of the WREF point). The Z and dX, dY of the PATH is used to draw the well path, while the Zm is needed to interpolate a Curve point (REC, which is given in measured depth; seeWellCurve Internal Data (PageA20)) position. The PATH format is the preferred format (GOCAD's output format).

VRTX describes a point of the WellPath given in absolute (real world) coordinates. In other words, VRTX_X=PATH_dX+WREF_X; VRTX_Y=PATH_dY+WREF_Y; VRTX_Z=PATH_Z. If the Well Path is given in the VRTX format, the WREF point is used as the Zm=0 point to construct measured depths Well Curves and other information that is given in measured depth format.

New path descriptions

Two new terms have been introduce two specify the WellPath: TVD_PATH and TVSS_PATH.

- TVSS_PATH

  ```
  TVSS_PATH Zm Zss dX dY
  ```

  TVSS_PATH is almost the same as the original PATH statement. GOCAD expects 4 values following this keyword, Zm Zss dX dY.

  However the sign of Zss is still to be determined by the GOCAD committee.

- TVD_PATH

  ```
  TVD_PATH Zm Ztvd dX dY
  ```

  This format is designed to accommodate well path information that does not have KB or other elevation values substracted from its directional survey. When GOCAD encounters the keyword TVD_PATH, it expects 4 values following this keyword, Zm Ztvd dX and dY. TVD is the true vertical depth. To find the sub-sea depth, GOCAD substracts the WREF Z value from Ztvd.

- PATH_CURVE_X

  The WellPath can also be specified as a special three special WellCurves. Each coordinates is contained in a WellCurve. Values in these curves must be in real world coordinates. The WellPath is specified as follow:

  ```
  WREF X Y Z
  PATH_CURVE_X name_of_well_curve_containing_X_information.
  PATH_CURVE_Y name_of_well_curve_containing_Y_information.
  PATH_CURVE_Z name_of_well_curve_containing_Z_information.
  ```

A.7.2 WellZone and WellMarkers

WellZone and WellMarkers data are both optional, and are given in the following format:

```
ZONE name Zm1 Zm2 index
MRKR Marker_name measured_flag Zm
```

```
[DIP azimuth dip]
[NORM X Y Z]
[MREF horizon_name]
[UNIT >rock_layer_name]
```

Additional information (between [and]) can be attached to each Marker. This extra information includes the orientation (given as DIP or NORM) of the Marker, the Marker Reference Horizon (currently not utilized by GOCAD) and the Rock Unit (currently not utilized by GOCAD).

The DIP information is given in *Grads.* What are Grads? It is an ancient French measurement of angles; there are 100 grads in a right angle (as opposed to 90 degrees in a right angle).

The NORM information is always given in XYZ, representing the normal vector of the Marker. This is the preferred representation for Marker orientation (this is the way GOCAD will output Marker orientation information, instead of DIP).

The optional information can be on the same line as the marker itself, or on lines that immediately follow the marker ascii code (MRKR).

New WellMarker dip specifications

DIP angles are to be given in *Grads*, which is not a unit a lot of people are familiar with. GOCAD introduced a new dip specification DIPDEG where the two angles are given in *degree*.

```
DIPDEG azimuth dip
```

A.7.3 WellCurves

The WellCurves section consists of two parts, the WellCurve Header section and the WellCurve Internal Data section. The header section defines the format of the Well Curves. The WellCurve data can be given in the same file, or in an external binary or ascii file.

External Binary Data Declaration

If the WellCurve data are stored in an external binary file, you must provide the name of the file, before any WellCurve Header information, using the following statement:

```
BINARY_DATA_FILE filename
```

The above statement gives the name of the binary file in which Well Curves are stored as a series of Z (measured depth) values and data values:

```
P1Z1, P1Z2,...,P1Z20, P1V1, P1V2,...P1V20, P2Z1, P2Z2,...,P2Z12, P2V1, P2V2,..., P2V12, P3Z1,...etc.
```

External ASCII Data Declaration

If the data are stored in an external ASCII file, you must provide the following information before any WellCurve Header information:

```
ASCII_DATA_FILE filename
DEPTH_COLUMN index
NCOLUMNS ncol
NROWS nrow
```

The data are read in as a matrix of floating-point numbers and records are separated by blanks. The dimension of the matrix is defined by NCOLUMN and NROWs.Each column is a Property and NROWS specifies the number of data points per Property. The DEPTH_COLUMN specifies which column contains the measured depth data, which also means that in the external ASCII format, different Properties must all have data values at the same measured depth point.

For example, the following file, named AsciiExample.wl.dat,

```
P1V1 P2V1 P3V1 Z1 P4V1
P1V2 P2V2 P3V2 Z2 P4V2
P1V3 P2V3 P3V3 Z3 P4V3
.....
P1200 P2V200 P3V200 Z200 P4V200
```

should be specified as

```
ASCII_DATA_FILE AsciiExample
DEPTH_COLUMN 4
NCOLUMNS 5
NROWS 200
```

WellCurve Header

Each WellCurve is defined inside a block beginning with WELL_CURVE and ending with END_CURVE. The format of the header is:

```
WELL_CURVE
PROPERTY type_name
[UNITS z_unit f_unit]
[PROPERTY_CLASS property_class_name]
[INTERPOLATION {Linear,Block}]
[PROP_SAMPLE_STATS n x x2 min max]
[PROP_NO_DATA_VALUE nodatavalue]
.....
END_CURVE
```

- The PROPERTY defines the name of the well log.
- The UNITS statement defines the unit of the measured depth and the unit of the Property (See Units for the format description). The units are used to convert the wellcurve data into the units of the property class associated to the property.
- You can also define the Property Class of the Property using the following statement:

  PROPERTY_CLASS property_class_name

  Defining the Property Class places the Property in its proper statistical database for future interactive manipulations inside GOCAD. Furthermore, the data will be transformed into the unit of the Property Class. If the Property Class is not given, the data will keep in the unit defined in the UNITS specification.

- The interpolation method used to interpolate intermediate values in the Well Curve is defined using the following statement:

  [INTERPOLATION {Linear,Block}]

  If INTERPOLATION is set to Linear, linear interpolation between the two nearest WellCurve values will be performed. If INTERPOLATION is set to Block the value of any point in the interval between two values will be set to the upper value (lower zm). The default is Linear.

- The Well Curve property statistics can be defined using the following statement:

  [PROP_SAMPLE_STATS n x x2 min max]

  n is the number of samples, x is the mean, x2 is the variance, min is the minimum of max is the maximum value of the property in the well curve.

- An optional no_data_value can be given to describe holes or invalid values. The format is given below:

  NO_DATA_VALUE no_data_value

WellCurve Internal Data

If the WellCurve data are given in the same file as the WellCurve Header, the data values are given in the following format:

```
REC Zm V
```

where Zm is the measured depth and V is the Property (log) value at that point.

If V is equal to the no_data_value (which must be declared before any Well Curve Header), the data is treated as missing. Alternatively, you can specify a HOLE, i.e. a discontinuity in the well curve. For example:

REC 3. 1 HOLE REC 10. 1 indicates that there is a hole (no data) in the curve between depth 3. and 10 and GOCAD will not interpolate this particular Property between those tow point. See Single Well File Example.

WellCurve External ASCII Data

If a WellCurve data is defined in an external ASCII File, the WellCurve Data portion comprises only one line:

```
COLUMN ncol
```

which tells GOCAD which column in the ASCII file contains the data points of this Property. This information along with the DEPTH_COLUMN information,

```
DEPTH_COLUMN index
```

will be used to retrieve data from the ASCII file to create the WellCurve. See Well with an External ASCII Data File Example.

WellCurve External Binary Data

If a WellCurve is defined in an external binary file, the user must specify the starting point (in the binary file) of this Property and the number of data points in this Property by including the following statement in the WellCurve Header:

```
SEEK seekpos
```

NPTS npts

This information will be used to read npts z values and npts data values from the binary file starting at Byte position seekpos. See Well with an External Binary Data File Example.

A.7.4 Single Well File Example

The first example is a well file that includes the WellCurve data:

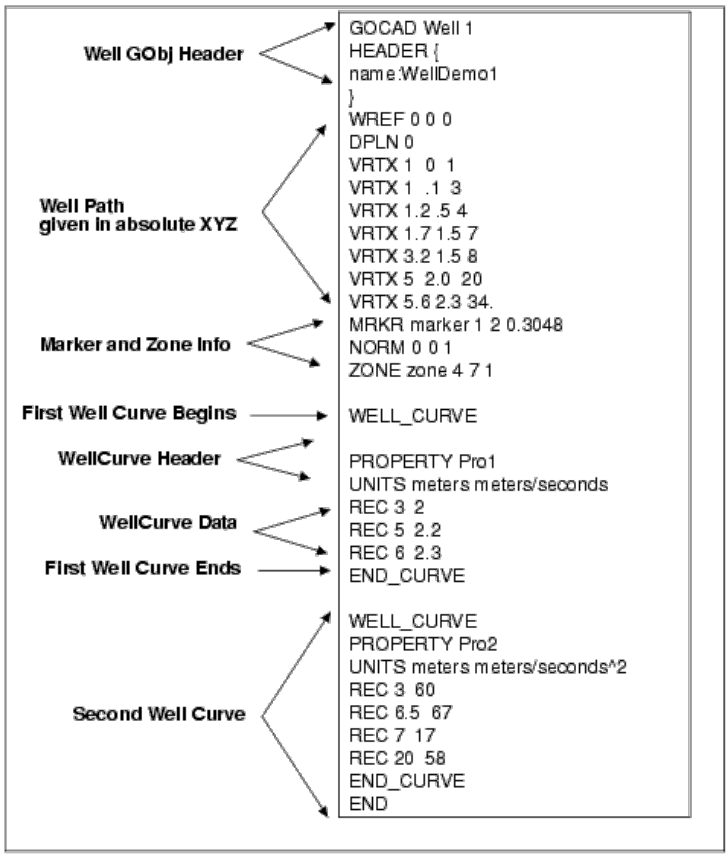


FIGURE 5. Single Well File format..

A.7.5 Well with an External Binary Data File Example

The second example is a well with the WellCurve data stored in the external binary file described in External Binary Data Declaration:

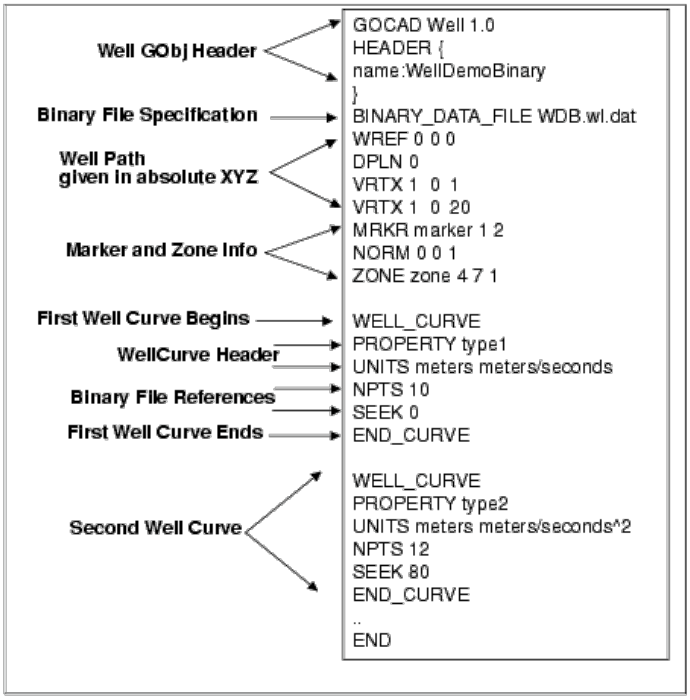


FIGURE 6. Well with an External Binary Data File.

A.7.6 Well with an External ASCII Data File Example

The third example is a well with the WellCurve data stored in the external ASCII file described in External Binary Data Declaration:

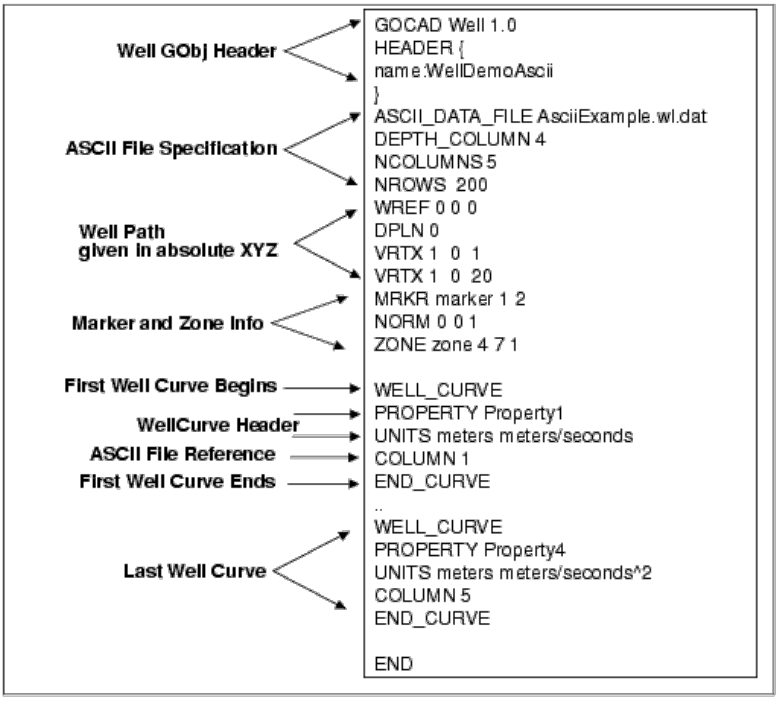


FIGURE 7. Well with an External ASCII Data File.

A.8 Grid3d or Voxet

A Voxet is a rigid 3D Grid Object; it can carry multiple Properties. A Property in a Voxet is often referred to as a grid3d. It is derived from the Object Class.

A common confusion comes from not realizing that UVW can mean the (i,j,k) indexing of the Voxet Nodes, but it can also mean the coordinate system in which the Voxet resides.

In addition to its inherited Object file elements, a Voxet is further defined in two parts: the header section, which defines the geometry, and the grid3d sections which defines the Properties (grid3ds) and the Region section (which gives Region storage and Region information). Each Property in the Voxet has its own grid3d section that defines the Property and Property values.

A.8.1 Geometry Section

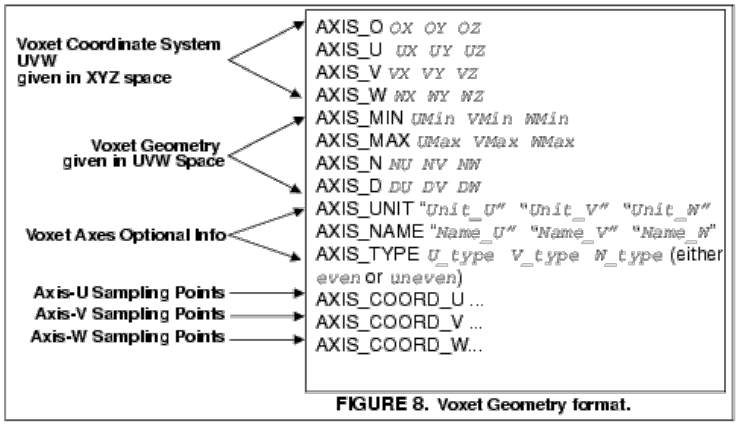


FIGURE 8. Voxet Geometry format.

UVW Coordinate System and Voxet Dimension

A Voxet is never directly defined in the XYZ space. The user must first define the UVW coordinate system (which may be identical to the XYZ space) in terms of the XYZ coordinate system, then define the Voxet geometry in the UVW space. See Figure8 and Figure9.

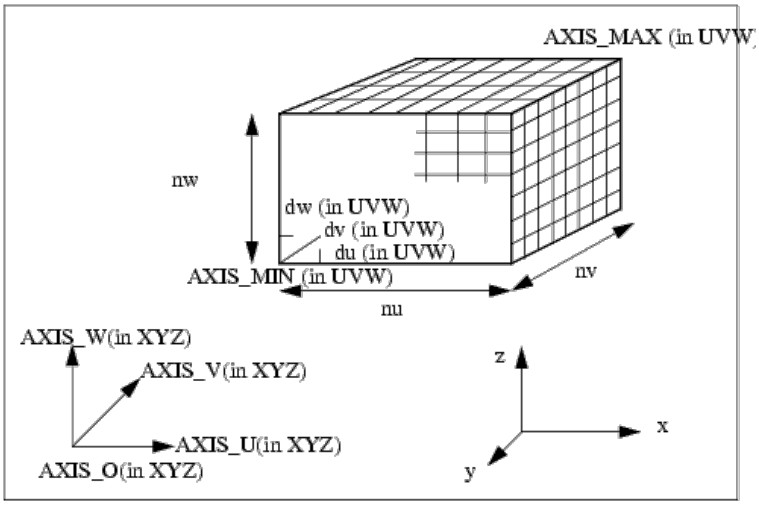


FIGURE 9. Voxet File geometry specifications.

AXIS_O, AXIS_U, AXIS_V, AXIS_W are required definitions. They define the coordinate system of the Voxet:

AXIS_O represents the origin of the Voxet coordinate system, not the Voxet the origin.

AXIS_U, AXIS_V, AXIS_W represents the direction and length of the vector, U, V and W respectively in the (X,Y,Z)

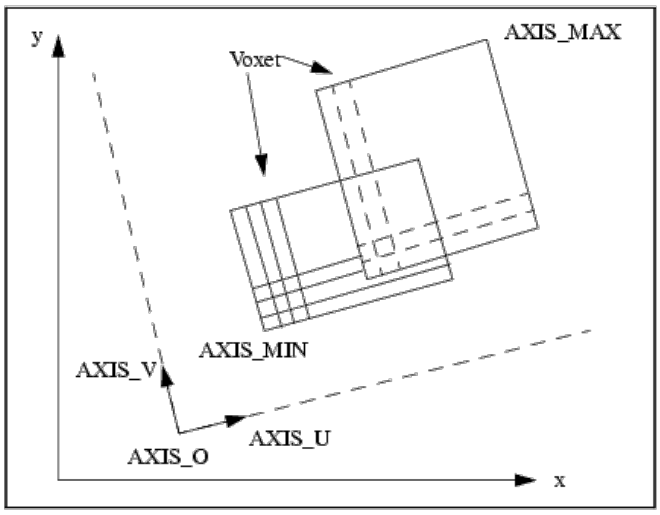


FIGURE 10. Voxets in MapView

coordinate system (See Figure10).
For example, in Seismic AXIS_U represents typically the time or depth axis. An example will be: AXIS_U 0. 0.0004 (in seconds).

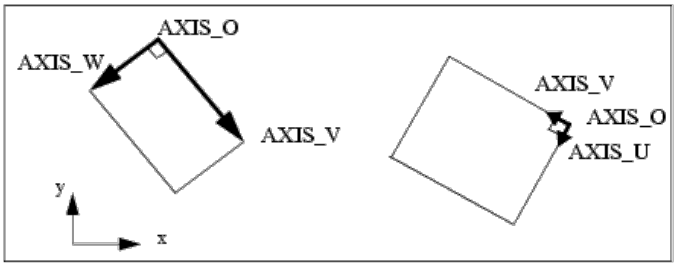


FIGURE 11. Other examples of Voxet axes specifications

A common confusion comes from not realizing that this set of vectors, AXES_OUVW, only defines the UVW coordinate system in which the Voxet resides; they do NOT define the dimension of the Voxet. The dimension of a Voxet is defined by another set of parameters, AXES MIN-MAX.

AXIS_MIN is by default 0., 0., 0., AXIS_MAX is by default 1,.1.,1.

AXIS_MIN defines in the (U,V,W) coordinate space, the origin of the Voxet.

AXIS_MAX defines where, in the (U,V,W) coordinate space, the maximum point of the Voxet is.

AXIS_MIN and AXIS_MAX define the bounding box of the Voxet in the (U,V,W) space.

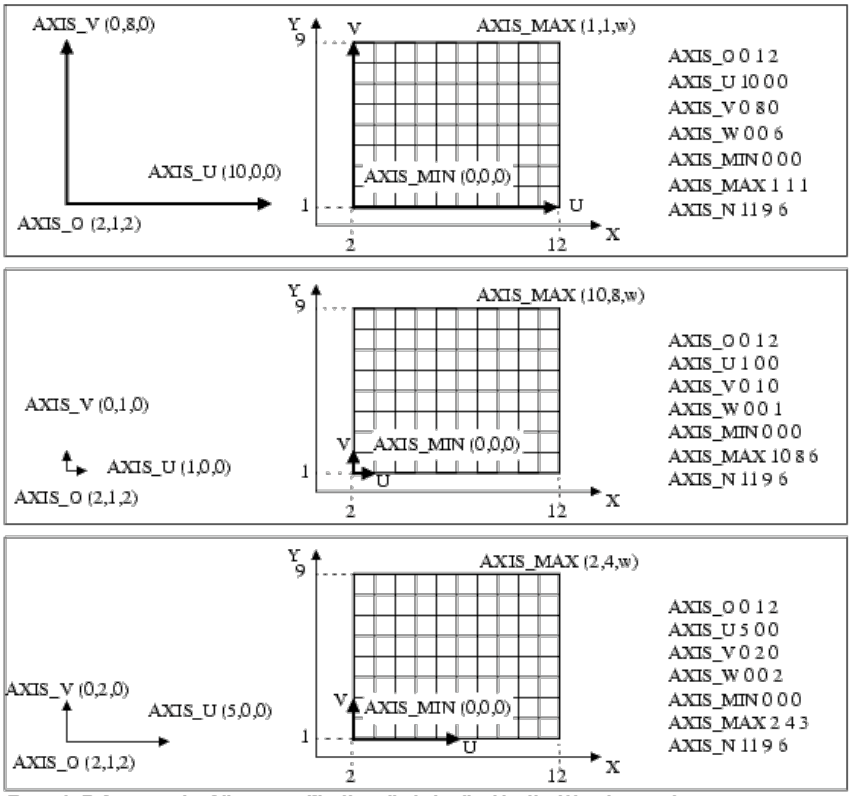
If AXIS_U, AXIS_V and AXIS_W represents the full box then AXIS_MAX should be equal to (1.,1.,1.). If AXIS_U, AXIS_V, AXIS_W represents the dimension of one cell then AXIS_MAX should be equal to AXIS_N. (See Figure7)

These two sets of specifications, AXES UVW and AXES MIN-MAX, are not unique. For example, you can specify:

- AXIS_UVW to be (1, 0, 0) (0,1, 0)(0, 0,1) and AXIS_MIN and MAX to be (0,0,0) (10, 8, 6);
- Or you can specify AXIS_UVW to be (10, 0, 0) (0, 8, 0)(0, 0, 6) and AXIS_MIN and AXIS_MAX to be (0,0,0) (1, 1, 1);
- Or you can specify AXIS_UVW to be (5, 0, 0) (0, 2, 0)(0, 0, 2) and AXIS_MIN and AXIS_MAX to be (0,0,0) (2, 4, 3).

  These three specifications are identical (in terms of the Voxet dimension). See Figure7.

  For practical reasons (less confusing), most users prefer to set the AXES UVW to be the dimension of a Cell (data spacing), or to be the dimension of the entire Voxet volume; as shown in the two upper examples in Figure7. The third example in Figure7 is rarely utilized.



Example 7: **An example of three specifications that give the identical Voxet geometry.**

The upper two are the most common cases; i.e. the AXES UVW either define the entire volume dimension or they define a cell dimension. When GOCAD outputs a Voxet file, it uses the upper most format; i.e. the UVW axes define the entire volume.

Do not confuse Voxet grid lines with Voxet cells. Grid lines connect Voxet nodes while Voxet cells are centered at each Node. Shown above are grid lines, not cells.

Voxet Grid Density

AXIS_D and AXIS_N are redundant. Only one need to be provided.

AXIS_N (NU,NV, NW) defines the number of samples of the Voxet in each direction U, V, and W.

AXIS_D (DU,DV,DW) defines the node-spacing (cell-dimension, sampling rate) along the three axes of the Voxet in the UVW space.

This following equation describes the relationship between these two parameters:

(NU-1)*DU = AXIS_MAX (UMax) - AXIS_MIN (UMax)

One thing to remember is that Voxet cells are centered at each data point, so the number of cells along each axis is the number of data points along each axis, NOT one less.

Each axis could be unevenly sampled and the sampling points coordinates defined then in the AXIS_COORD_{U,V,W} section. However, uneven sampling is not implemented yet.

A.8.2 Voxet Regions

Currently, Region information **must** be given before Property information.

Region and connectivity information in a Voxet is saved on a per data point basis (values in the file increase first in U, then V, then W) in an external binary file with the default name "__flags@@". For each Region, a point is either in it or not in it.

In the main file itself, there are two sections describing the Region-related information.

- The first section defines how the Region information for each Voxet point is stored in the external file.

  FLAGS_ARRAY_LENGTH length_of_the _array FLAGS_BIT_LENGTH number_of_bits (never less than 7, the first 6 bits for connectivity info) FLAGS_ESIZE number_of_bytes (should equal (number_of_bits+1)/8 rounded up) FLAGS_OFFSET offset_inside_the_binary_file_for_the_first_data_point FLAGS_FILE binary_file_name (values in the file increase first in U, then V, then W)

- The second section gives the name of each Region and its flag bit position.

REGION region_name region_bit_marker_position_number (must be less than number_of_bits)

An example of an Voxet ASCII file with Region information is given below:

```
....
AXIS_N 10 10 10

FLAGS_ARRAY_LENGTH 1000
FLAGS_BIT_LENGTH 10
FLAGS_ESIZE 2
FLAGS_OFFSET 0
FLAGS_FILE v1__flags@@

REGION RegionExample 6
REGION HighPorosity 7
REGION LowPerm 8
REGION BadBuy 9

END
```

A.8.3 Grid3d/Property Section

The grid3d section defines the property. Currently, Region information must be given before Property information.

For each property there is a unique identifier, id, which is used to relate different Property statements to the same Property. The data can be in an external file in a binary format, or inside the file in ascii format.

For each Property, the declaration must be the first line (PROPERTY id "property name") and the Property file name (PROP_FILE id filename) or data (DATA) must be the last line.

```
PROPERTY id "property name"
PROPERTY_CLASS id "property-class_name"
PROP_UNIT id "unit"
PROP_LEGAL_RANGE id min max (where min/max is a float or **none**
PROP_SAMPLE_STATS id n x x2 min max
PROP_NO_DATA_VALUE id value
PROP_SAMPLE_STATS id n x x2 min max
PROP_ETYPE id data_type (either IEEE or IBM)
PROP_FORMAT id file_format (either RAW or SEGY)
PROP_ESIZE id element_size (either 1 or 4)
PROP_FILE id filename
PROP_OFFSET id offset
DATA
```

PROP_OFFSET indicates that the array of float value begins at the given offset (in bytes, = number of data points x PROP_ESIZE) in the PROP_FILE.

PROP_ETYPE specifies the type of floating point value that is in the file. It can be IBM floating point value or IEEE floating point value.

PROP_FORMAT specifies the format of the file. SEGY format implies that the data resides in a standard SEGY file. RAW specifies that the data is formatted as a C array where the fast axis is the Axis_U.

The token DATA, if present, indicates that the Property data is to be read following that token. If this is the case, the floating point array of data must be in ascii RAW format.

A.8.4 Example

An example of a Voxet file with an external Property file:

```
GOCAD Voxet 0.01
AXIS_O 2128403. -79200. 0.
AXIS_U 0. 0. 55.
AXIS_V 0. 220. 0.
AXIS_W 220. 0. 0.
AXIS_MIN 0.0 0.0 0.0
AXIS_MAX 549. 227. 149.
AXIS_N 550 228 150
AXIS_D 1. 1. 1.
AXIS_NAME "Z" "Y" "X"
AXIS_UNIT "m" "m" "m"
AXIS_TYPE even even even

PROPERTY 1 "Seismic"
PROP_UNIT 1 " "
PROP_ESIZE 1 4
PROP_ETYPE 1 IEEE
PROP_FILE 1 /tmp/seismic.nohdr
```

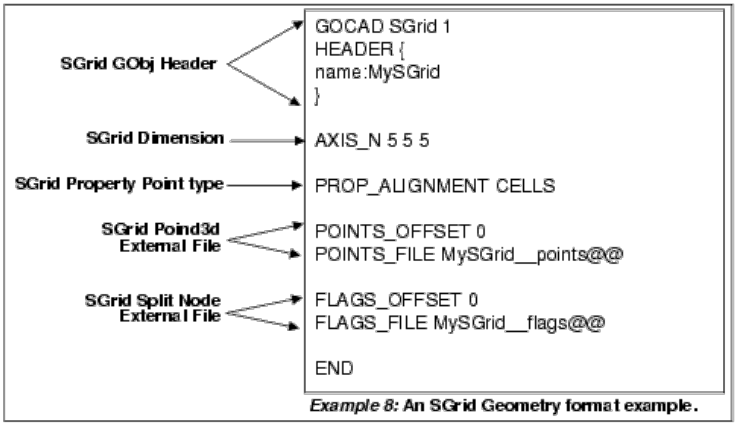
A.9 Stratigraphic Grid

The geometry of an SGrid is defined by a cube of Point3d (a set of points in space) and its Properties by a Cube of floating point values. In addition to the inherited Object file elements, an SGrid file consists of three sections: the header section, which defines the geometry storage information, the region sections (which defines Region storage information and Region information), and the property sections (which defines Property and Property storage information).

Be aware of the deadly differences between SGrid terminology and Voxet terminology!

- Header Section
- SGrid Region Format
- Property Section
- Ascii External File
- Split Nodes
- Examples

A.9.1 Header Section



Example 8: **An SGrid Geometry format example.**

The statements

AXIS_N NX NY NZ

specifies the dimension of the stratigraphic grid. This dimension will be the dimension of the Point3d array, which specifies the location of each node of the SGrid and of the float arrays which specifies properties.

The Point3d array is read in from a file specified by the POINTS_FILE, and the starting position of that Point3d array is given by POINTS_OFFSET.

POINTS_OFFSET offset POINTS_FILE filename

The statements

FLAGS_OFFSET offset FLAGS_FILE filename

specifies the filename and offset in the file where the flags array lies. The flags array contains the split-node information. This is different from a Voxet where the FLAGS information is related to the Regions.

The statement

PROP_ALIGNMENT (CELLS or POINTS)

specifies whether the Stratigraphic Grid is cell -centered or corner-point. If the SGrid is cell-centered the size of the point array will be NX*NY*NZ but the size of the Property arrays will be (NX-1)*(NY-1)*(NZ-1). If the SGrid is a corner-point sgrid, the size of the property array will be NX*NY*NZ.

A.9.2 SGrid Region Format

Region information **must** be given before properties information.

Region information in an SGrid is saved on a per data point basis (values in the file increase first in U, then V, then W) in an external binary file with the default name "__region_flags@@". For each Region, a point is either in it or not in it.

In the main file itself, there are two sections describing the Region-related information

- The first section gives the name of each Region and its flag bit position in the external file.

  REGION region_name region_bit_marker_position_number (must be less than number_of_bits)

- The second section defines how the Region information for each Voxet point is stored in the external file.

REGION_FLAGS_ARRAY_LENGTH length_of_the _array REGION_FLAGS_BIT_LENGTH number_of_bits (at least 1 greater than the number of Regions) REGION_FLAGS_ESIZE number_of_bytes (should equal (number_of_bits+1)/8 rounded up) REGION_FLAGS_OFFSET offset_inside_the_binary_file_for_the_first_data_point REGION_FLAGS_FILE binary_file_name (values in the file increase first in U, then V, then W)

An example of an SGrid file with Region information is given below:

```
AXIS_N 41 21 31
.....
FLAGS_FILE MySGrid__flags@@

REGION Reg_top_1 0
REGION Reg_1_2 1
REGION Reg_2_bot 2
REGION Facies_Region_1 3
REGION Facies_Region_2 4
REGION Facies_Region_3 5
REGION Facies_Region_4 6
REGION Facies_Region_5 7
REGION Facies_Region_6 8
REGION BM_Inactive_Region 9
REGION_FLAGS_ARRAY_LENGTH 26691
REGION_FLAGS_BIT_LENGTH 10
REGION_FLAGS_ESIZE 2
REGION_FLAGS_OFFSET 0
REGION_FLAGS_FILE MySGrid__region_flags@@

PROPERTY 1 "Geol_Facies_1"
PROPERTY_CLASS 1 "facgm"
.....
```

A.9.3 Property Section

The Property section defines the property at each node. Currently, Region information **must** be given before Property information.

Each Property has its unique identifier, id, which is used to relate different Property statements to that Property. The data can be in an external file in a ascii or binary format.

For each Property, the declaration must be the first line (PROPERTY id "property name") and the Property file name (PROP_FILE id filename) or data (DATA) must be the last line.

The definitions of the Property format is identical to Voxet. Please see Grid3d/Property Section (PageA31).

```
PROPERTY id "property name"
PROPERTY_CLASS id "property_class_name"
PROP_UNIT id "unit"
PROP_LEGAL_RANGE id min max (where min/max is a float or none)
PROP_NO_DATA_VALUE id value
PROP_SAMPLE_STATS id n x x2 min max
PROP_ETYPE id data_type (must be IEEE, IBM, or SEGY)
PROP_ESIZE id element_size (either 1 or 4)
PROP_FILE id filename
PROP_NO_DATA_VALUE id no_data_value
PROP_OFFSET id offset
DATA
```

The fast axis is Axis_U. The data are specified as a C array.

If the SGrid is cell-centered the size of the point array will be NX*NY*NZ but the size of the Property arrays will be (NX-1)*(NY-1)*(NZ-1).

A.9.4 Ascii External File

The geometry and the properties can both be read from an ASCII data file. First this file must be specified in the header as follows:

```
ASCII_DATA_FILE filename
```

The format of that file is:

```
x y z p1 p2 p3... flag u v w
```

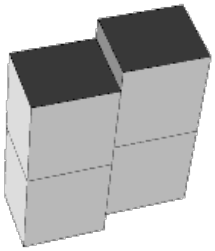
where x, y, z specifies the location of that node; p1, p2, p3,... are the Property values at that node; flag specifies the connectivity flag of that node; and u, v, w specifies the index of the node.

A.9.5 Split Nodes

Introduction

A node is a corner of a cell that is usually shared between 8 cells except when a fault affect this area (and on the

border where less cells are available).Faulting introduces split nodes.The ascii or binary file describing the geometry contains one x,y,z location. When a grid is faulted the nodes affected by the faulting have several x,y,z ; additional x,y,z information (the first one is in the ascii or binary file describing the geometry) is stored in the SGrid header file (usually defined with the extension ?s .sg ?t). The following set of examples are linked with the following SGrid depicted in Figure10



Example 9: **Faulted SGrid**

Format

Here is an example of split node description :

SPLIT 59 19 3 -225.76 177.902 1407.1 36 1 0 0 0 1 0 0 0 1

This corresponds to the following information :

SPLIT U V W X Y Z id Cell(u,v,w) Cell(u-1,v,w) Cell(u,v-1,w) Cell(u-1,v-1,w), Cell(u,v,w-1) Cell(u-1,v,w-1) Cell(u,v-1,w-1) Cell(u-1,v-1,w-1)

This split node is attached to the grid regular node U V W. The first occurrence of a split node U V W means that the regular node is split. There can be up to 7 SPLIT lines per U V W.

All the Cell(u,v,w) are boolean values (0 or 1).

If Cell(u-1,v-1,w) is equal to 1, then this node (SPLIT) is a corner of the Cell(u_1,v-1,w).

Remark : when a node is on the external border of the grid, it does not have 8 neighbor cells. For the missing cells the boolean value is set to 1

Split Face

Nodes are split along faces. When a face is split, split faces can be gathered in FaceSets. A FaceSet is described as:

FACET_SET name number_of_faces cell_uvw_1 face_dir_1.... cell_uvw_2 face_dir_2

where cell_uvw represents the cell number (U+V*NU+W*NU*NV) and face_dir is the face indicator (U=0, V=1, W=2).

A.9.6 Examples

A.9.6.1 External Binary files

An example of an SGrid with one Property and external binary files (lines that start with an * are optional Attribute lines):

```
GOCAD SGrid 0.01
HEADER {
name:s3
}
AXIS_N 32 32 32
PROP_ALIGNMENT CELLS
POINTS_OFFSET 0
POINTS_FILE s3__points@@
FLAGS_OFFSET 0
FLAGS_FILE s3__flags@@

PROPERTY 1 "SGS_simulation_1"
PROPERTY_CLASS 1 "porndx"
PROP_UNIT 1 none
PROP_SAMPLE_STATS 1 62883 0.24624 0.000661522 0 0.415
PROP_ESIZE 1 4
PROP_ETYPE 1 IEEE
PROP_ALIGNMENT 1 CELLS
PROP_OFFSET 1 0
PROP_FILE 1 s3_SGS_simulation_1@@
END
```

A.9.6.2 External Ascii File

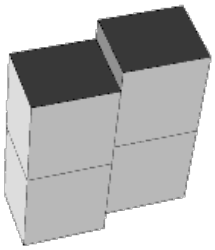
Below is the header file for the SGrid with one property.

GOCAD SGrid 0.01 HEADER { name:s3 *painted:on } AXIS_N 32 32 32 PROP_ALIGNMENT CELLS ASCII_DATA_FILE s3__ascii@@ PROPERTY 1 "SGS_simulation_1" PROPERTY_CLASS 1 "porndx" PROP_UNIT 1 none PROP_SAMPLE_STATS 1 29791 0.247145 0.000618586 0.154 0.414999 PROPERTY_CLASS_HEADER 1 "porndx" { *low_clip:0.183077 *high_clip:0.311214 *pclip:99 } END

Below are the first lines off an ASCII external file

* * X Y Z SGS_simulation_1 Flag I J K * 145745 9416230 -3458.3147 0.229000002 1031 0 0 0 145995 9416230 -3459.58105 0.229790032 1039 1 0 0 146245 9416230 -3460.82251 0.229000002 1039 2 0 0 146495 9416230 -3462.18555 0.228938624 1039 3 0 0

A.9.6.3 Split Nodes



Example 10: **Faulted SGrid**

Below is the SGrid header file for a faulted SGrid.

```
GOCAD SGrid 1
HEADER {
name:my_sgrid
}
AXIS_N 2 3 3
PROP_ALIGNMENT CELLS
ASCII_DATA_FILE my_sgrid__ascii@@

SPLIT 0 1 0 0.0260121 0.022876 0.640355 1 0 0 1 1 0 0 1 1
SPLIT 1 1 0 -0.153246 0.022876 0.640355 2 0 0 1 1 0 0 1 1
SPLIT 0 1 1 0.0260121 0.022876 0.890355 3 0 0 1 1 0 0 1 1
SPLIT 1 1 1 -0.153246 0.022876 0.890355 4 0 0 1 1 0 0 1 1
SPLIT 0 1 2 0.0260121 0.022876 1.14036 5 0 0 1 1 0 0 1 1
SPLIT 1 1 2 -0.153246 0.022876 1.14036 6 0 0 1 1 0 0 1 1

FACE_SET face_set_0 2
8 1 2 1
END
```

A.10 GSurf (2D Grid)

```
GOCAD GSurf 1
HEADER {
name:G1
}
```
A.10.1.1

```
ORIGIN or AXIS_O X Y Z
AXIS_U X Y Z
AXIS_V X Y Z
AXIS_W X Y Z
AXIS_MIN Min_x Min_y Min_z
AXIS_MAX Max_x Max_y Max_z
AXIS_N Nu Nv
TYPE POINTS or CELLS
```
A.10.1.2

```
PROPERTY 1 "W"
PROPERTY_CLASS 1 "w"
PROP_UNIT 1 none
PROP_NO_DATA_VALUE 1 -9999.99
PROP_SAMPLE_STATS 1 135838 0.410638 0.0565332 0 1
PROPERTY_CLASS_HEADER 1 "w" {
}
PROP_ESIZE 1 4
PROP_TYPE 1 IEEE
PROP_ALIGNMENT 1 POINTS
PROP_OFFSET 1 0
PROP_FILE 1 G1_W@@
```
A.10.2.1

```
PROPERTY Property_index "Prop"
PROPERTY_CLASS Property_index "prop"
PROP_UNIT Property_index none
PROP_NO_DATA_VALUE Property_index -99999
PROPERTY_CLASS_HEADER Property_index "prop" {
}
PROP_ESIZE Property_index 4
PROP_TYPE Property_index IEEE
PROP_ALIGNMENT Property_index POINTS
PROP_OFFSET Property_index 0
PROP_FILE Property_index TestPoint100_Prop@@
```
A.10.2.3

```
END
```

A 2D-Grid, also called gridded surface is defined by an Origin, delta-X, delta-Y and a series of Z values. It can carry multiple Properties. A 2D grid can be compare to a TSurf (Surface), but its triangulation and the distribution of its Atoms show a regular gridded pattern.

A 2D-Grid is defined into two major parts: the header section, which defines th geometry and the property sections (which defines Property and Proeprty storage information)

- Header
- Property Sections

A.10.1 Header

- Header

- Geometry Parameters

A.10.1.1 Header

The GSurf ASCII file starts with the common header for all GOCAD file format. The type of object is identified by the "GSurf" keyword following th GOCAD initial identifier.

The "HEADER" section contains information on the 2D Grid such as graphical attributes: the only parameter needed is the name of the Object.

In the example, the 2D Gris is named "G1".

A.10.1.2 Geometry Parameters

This section lists the geometrical parameters of the 2D Grid. The GSurf basis is equivalent to a 3D Voxet Grid basis.

The origin is marked by one of the two keywords "ORIGIN" or "AXIS_0" and expects a 3D points. Typically, if the grid is defined with a reference to the reference altitude 0, the origin is defined as: ORIGIN X Y 0

The AXIS_U, AXIS_V, and AXIS_W define the length and the orientation of the 2D Grid axes. The three keywords expect 3D vector giving the length and orientation of the Grid axes as shown in Figure 1-1 (page A-42).

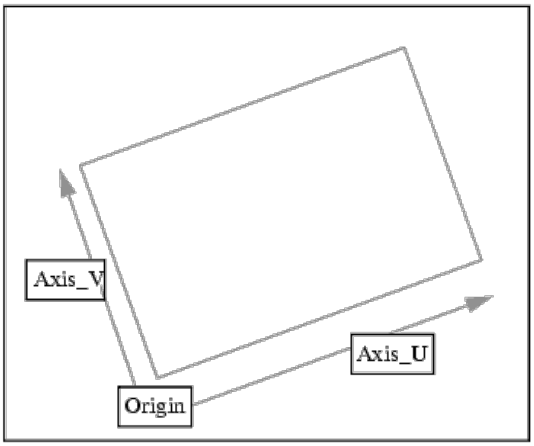


Figure 1-1 2D Grid orientation and origin

Typically, if the 2D Grid is defined with altitude or barthymetry data, AXIS_W is 0. 0. 1.AXIS_U and AXIS_V can not be parallel to the X and Y axes.

The AXIS_MIN and AXIS_MAX are 3d grid points which control the extensions of the basis like Voxet. Typically, these are 0. 0. 0. for AXIS_MIN and 1. 1. 1. for AXIS_MAX: the 2D Grid is defined on the full extension of the basis.

Axis_N is the keyword defining the number of row/columns on the U and V axis: it expects 2 integer values. TYPE is the keyword defining the 2D Grid property type: either corner points POINTS or cell-centered CELLS.

A.10.2 Property Sections

- Geometry Property Definition
- Property Definition
- ASCII values for Geometry and Property

A.10.2.1 Geometry Property Definition

This section is always present and describes the specifications of the 2D Grid Geometry.

GSurf always carries a property named "W". The "W" property is used to specify the real world coordinate to the GSurf points. It is equivalent to Z when the Origin Z coordinate is 0. and the Axis_W coordinates are 0. 0. 1. Otherwise, the XYZ real world position can be obtained by the formula:

(EQ 1) XYZ=Basis Point (U, V, 0) + W*Axis_W

All keywords are followed by the property index. For the GSurf W geometry, the index is always 1.

The first seven lines are common property description which gives the Property name, class, unit, no-data-value, sample statistics and header attributes. For the GSurf geometry, the no-data-value defines which value will be used to get holes and missing points in the Grid.

PROP_ESIZE and PROP_TYPE describe the type of values stored in the external binary file. Right now, GSurf supports only 4 bytes (float) IEEE values for geometry and property. Support for other types and sizes of property might be added in the future.

PROP_ALIGNMENT is the location type of property: corner-point or cell-centered. GSurf geometry is always corner-point like Stratigraphic grid's geometry.

PROP_OFFSET gives the bytes offset which may be applied to shift the read inside the external binary file given by the PROP_FILE keyword

The external binary file is expected to contain 32 bits floating point values ordered so that U is the fast axis.

To read a GSurf external binary file, the following code can be used:

```
FILE* external_binary_file = fopen( "G1_W@@", "rb" );
fseek( external_binary_file , prop_offset, SEEK_SET);
for( GridCoord v = 0; v < nv; ++v ) {
    for( GridCoord u = 0; u < nu; ++u ) {

        float v;
        fread( &v, 4, 1, external_binary_file );

    }
}
```

A.10.2.2 Property Definition

This section is optional and is it only present when the 2D Grid carries additional properties. There are as many property sections as there are properties on the 2D Grid. The section is nearly identical to the Geometry section except for the PROP_ALIGNMENT keyword which must match the value given in the Geometry parameters section.

A.10.2.3 ASCII values for Geometry and Property

Instead of using an external binary file for getting the property values, an external ASCII file or a DATA section inside the GSurf file can be used.

The Geometry/Property section are slightly different

PROPERTY *Property_index* "Prop"
PROPERTY_CLASS *Property_index* "prop"
PROP_UNIT *Property_index* none
PROP_NO_DATA_VALUE *Property_index* -99999
PROPERTY_CLASS_HEADER *Property_index* "prop" {
}
PROP_ESIZE *Property_index* 4
PROP_TYPE *Property_index* **IEEE**
PROP_ALIGNMENT *Property_index* **POINTS**
PROP_FORMAT *Property_index* **ASCII** or **BINARY**
PROP_ASCII_MODE *Property_index* **U_FAST_AXIS** or **V_FAST_AXIS** or **INDEXES** or **COORDINATES**

PROP_FORMAT indicates whether the property is specified as ASCII or binary. If the keyword is not present, it is assumed that the property is stored in an external binary file.

If the PROP_FORMAT is specified as ASCII and there is no PROP_FILE line, the values are expected to be stored inside the main 2D Grid file starting with DATA keywords. If an external file is specified, the DATA keyword must not be present in the external ASCII file.

The PROP_ASCII_MODE keyword specifies how the values are ordered in the DATA section or in the external FILE

1 U_FAST_AXIS

Expected format: Single column of floating point values. Data is read with a double loop

```
for( GridCoord v = 0; v < nv; ++v ){
    for( GridCoord u = 0; u < nu; ++u ){

        fscanf( ... );

    }
}
```

2 V_FAST_AXIS

Expected format: Single column of floating point values. Data is read with a double loop

```
for( GridCoord u = 0; u < nu; ++u ){
        for( GridCoord v = 0; v < nv; ++v ){

                fscanf( ... );

        }
}
```

For U_FAST_AXIS and V_FAST_AXIS, all values must be recorded.

3 INDEXES

Expected format: Two columns of U and V coordinates, one column of floating point values. Data values are given with their corresponding U and V coordinates in the 2D Grid basis space.

Example

```
DATA 0 0 -99999.
DATA 2 2 200. ...
```

4 COORDINATE

Expected format: Two columns of X and Y coordinates, one column of floating point values. Data values are given with their corresponding X and Y coordinates in the world basis space.

Example

```
DATA 1000. 100. -99999.
DATA 1010. 110. 200.
```

For INDEXES and COORDINATES, it is possible not to record all the values. The values not recorded in the file will be assumed to be no-data-values.

A.11 Model3d

A Model file contains the description of its Model Members and its Property (Property Model). A model file containes a valid 3D model, i.e. a model where all surfaces have been already made topologically coherents (i.e. they have been all intersected by themselves and the other surfaces of the model). When the model is loaded only regions and layers will be reconstructed from their definition.

A.11.1 Model Descriptor Elements

A Model3D consists of Surfaces, TFaces, Regions, Layers, Faultblocks and Property-Model database. The ASCII description of the Model therefore consists of 5 parts:

- TSURF description TSURF tsurf_name
- TFACE description TFACE geological_type surface_name X1 Y1 Z1 X2 Y2 Z2 X3 Y3 Z3
  X1,Y1,Z1,... describes a key triangle of the TFace. The TFace will be then referenced inside the Regions description by the unique identifier. Once the model is read, the model is build without recomputing intersections. The key triangle and the region definition will be used to match newly constructed region with saved regions so that region names can be reassigned correctly.
  geological_type is the geological type of the TSurf containing the TFace and surface_name is the name of the same TSurf.
- REGION description REGION identifier {[+-]TFACE_identifier] } 0
  Each Region is given a unique identifier, and the list of the oriented TFaces composing this Region. The list of tfaces ends with a 0. The orientation is specified by either a + or a - sign.
- LAYER description LAYER layer_name {region_identifier} 0
  Each Layer is given a name and the list of Regions which compose the layer. The list ends with a 0.
- FAULTBLOCK description FAULTBLOCK faultblock_name {region_identifier} 0
  A faultblock is given a name and the list of regions which compose the faultblock. The list ends with a 0.

A.11.2 PROPERTY_MODEL_DB description

Definitions of a ModelProperty and its Variables are given in a block that begins with BEGIN_PROPERTY_MODEL_DB_{ , ends with }_END_PROPERTY_MODEL_DB.

Since a Property in a Model3d Object is defined on a per Property per Layer basis, the DataBase is described using a Style syntax. The Style syntax allows us to easily describe the scope of the definition. The rules are as follows:

DB : the following definitions are global to the DataBase.
*LDB*layer_name* : the following definitions are given for all Properties in the specified Layer.
*PDB*property_name* : the following definitions are valid for the particular property in all layers.
*PLDB*propery_name*layer_name*: the following definitions are defined for a particular Property in a particular Layer.

For example:

*DB*V0*type: constant
*DB*V0*value: 1800.0

defines the variable V0 inside the DataBase to be a constant equal to 1800. Similarly,

*LDB*Layer1*Vs*type: constant
*LDB*Layer1*Vs*value: 1800.0

defines the variable Vs inside the Layer Layer1 to be a constant equal to 1800.

A.11.2.1 Variables type

There are currently 7 types of variables:

- constant
- grid
- shoot
- top_shoot, bottom_shoot
- interpolation
- linear
- c_script

We will be describing each of them in the next paragraphs.

constant

The following two lines

*DB*V0*type: constant
*DB*V0*value: 1800.0

define a constant variable V0 of value 1800.

grid

The following three lines:

*DB*V0*type: grid
*DB*V0*grid: name of an object capable of giving property values at X,Y,Z
*DB*V0*property: property_name

define a variable which gets its value from another object. The object is what is called in GOCAD a geometrical point property server, i.e., the object can give a value at any point in space where the object is defined. Examples of such objects include Voxet, SGrid, TSolid and 3D models.

shoot

The following lines define a variable that extracts the value at a given point by shooting on a surface and interpolating the value at the impact point. They are a lot of varieties of shooting variables which all differ from the initial location of the shoot. This has some importance with multi-valued surfaces.

*DB*V0*type: shoot
*DB*V0*target: name of the target surface
*DB*V0*property: property_name
*DB*V0*way: down or up

The target gives the name of the surface, and the shoot can be done downward or upward. The nearest impact point will be used to compute the property value.

top_shoot, bottom_shoot

The following lines define a variable that extracts the value at a given point by shooting on a surface and interpolating the value at the impact point. The initial position of the ray will be on the outside of the bounding box of the target surface either from the top or from the bottom:

*DB*V0*type: top_shoot or bot_shoot
*DB*V0*target: name of the target surface

*DB*V0*property: property_name

interpolation

The following lines define a variable that extracts the value at a given point by shooting on a top and bottom surface, getting the value at both impact points and interpolating the value at the initial shooting location using a Z linear scale.

*DB*V0*type: interpolation
*DB*V0*top: name of the top target surface
*DB*V0*bottom: name of the bottom target surface
*DB*V0*property: property_name

linear

A linear variable used three other variables V0, K, Z0 in the built-in formula: V = V0 + K * (Z - Z0); The names of the variable need not to be V0, K, Z0 but must be declared in this order in the definition of the variable.

```
*DB*Vp*type: linear
*DB*Vp*arguments: V0 K Z0
```

c_script

A c_script variable is used to compute the value of a property inside a region using a script. The script is defined using a C like syntax. The function defined in the script assigns the value of the property variable to a formula.

```
*LDB*Layer2*Vp*type: c_script
*LDB*Layer2*Vp*arguments: V0 K Z0 Z
*LDB*Layer2*Vp*value: {Vp=V0+K*(Z-Z0);}
```

Other examples of formula could be:

```
if( V1 < 1234 ) Vp = sqrt(V2) ; else Vp = 0;
```

A script in GOCAD accepts every C expressions and if statements. Standard math functions are available. Arguments to the function must be given so that the script can interpret correctly the variable names. Undefined variable names are assumed to be temporary variables.

A.11.3 Example

In this example, the surfaces that compose the model have been written into separate files. This is indicated by the style attribute `*geo_file_is_separate' in the header of the Model. In that case, each surfaces of the model will be saved/read to/from separate files. Each surface file name will be generated by concatenating the surface name with the suffix defined by the `*geo_file_suffix' attribute and by prefixing the name with the prefix `*geo_file_prefix'. The files will be placed in the same directory as the model file.

If the attribute `*separate_file' is set to false or unset (which is the default) the surfaces will be saved at the end of the model file.

```
GOCAD Model3d 0.01
HEADER {
name:saltmodel
geo_file_is_separate:true
geo_file_suffix:.ts
}
TSURF Box
TSURF F1
TSURF H1
TSURF H2
TSURF H3
TSURF H4
TSURF H5
TSURF H6
```

faces description

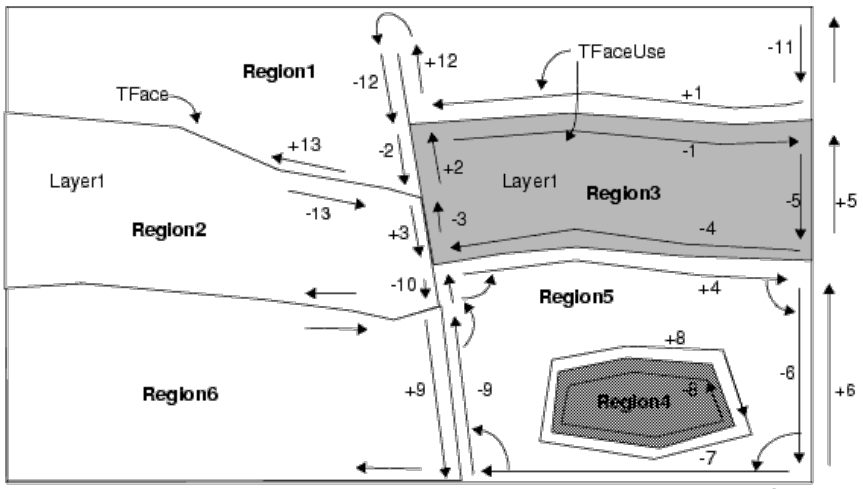
Each Face is described using a key triangle. The key triangle is a substitute for a name for the TFace. We associate the key triangle of a TFace to a number. This number will be used in the region description. See TFACE description (PageA46).

```
TFACE 1 none Box
17014.9824 17460.1855 -2548.08936
16134.4531 17460.1855 -1782.04761
17014.9824 17460.1855 -1016.00592
TFACE 2 none Box
18776.043 15775.0977 5112.32764
18776.043 17460.1855 5112.32764
```

17895.5117 16617.6406 5112.32764
.....etc. ... etc...

regions description:

Each region is described giving the list of the oriented TFaces bounding it. See REGION description (PageA46). The + or - sign are here to define one side vs. another side of the TFace. They are arbitrary but must be consistent so that two regions can not be bounded by the same side.



Example 11: **Region definitions**

Using the diagram below here are some of the definition of the region:

REGION 3
-1 -5 -4 -3 +2 0
REGION 5
+4 -6 -7 -9 +10 +8 0
REGION 1
+1 -11 +13 -2 -12 +12 0
.....etc.... etc. ...

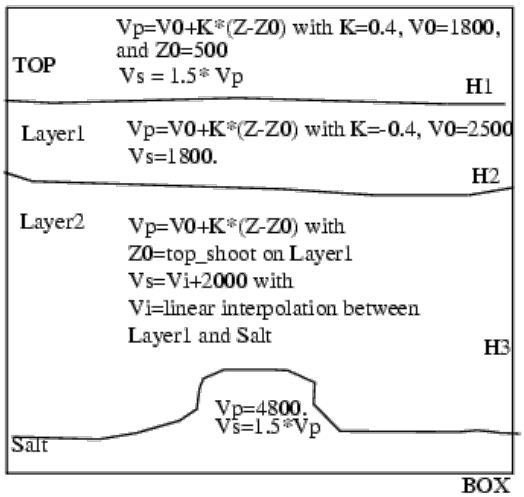
layers description

See LAYER description (PageA46). For the regions drawn in Example 11 (page 51)

LAYER Layer1
2 3 0
LAYER TOP
1 0
.....
FAULT_BLOCK no_fault_block_name
21 22 23 24 25
26 27 28 29 30 0
....

property database

PROPERTY Vp PROPERTY_CLASS vp
PROPERTY Vs PROPERTY_CLASS vs
BEGIN_PROPERTY_DB_{
*DB*variables: Vs Vp K V0 Z0
*DB*Vs*type: c_script
*DB*Vs*arguments: Vp
*DB*Vs*value: {Vs=1.5*Vp;}
*DB*K*type: constant
*DB*K*value: 0.4
*DB*V0*type: constant
*DB*V0*value: 1800.0
*DB*Z0*type: constant
*DB*Z0*value: 500.0
*DB*Vp*type: linear
*DB*Vp*arguments: V0 K Z0
*LDB*Layer1*variables: V0 K Vs
*LDB*Layer1*V0*type: constant
*LDB*Layer1*V0*value: 2500.0



```
*LDB*Layer1*K*type: constant
*LDB*Layer1*K*value: -0.400000
*LDB*Layer1*Vs*type: constant
*LDB*Layer1*Vs*value: 1800.0
*LDB*Layer2*variables: Vp Z0
*LDB*Layer2*Z0*type: top_shoot
*LDB*Layer2*Z0*target: Layer1
*LDB*Layer2*Z0*property:Z
*LDB*Layer2*Vp*type: c_script
*LDB*Layer2*Vp*arguments: V0 K Z0 Z
*LDB*Layer2*Vp*value: {Vp=V0+K*(Z-Z0);}
*LDB*Salt*variables: Vp
*LDB*Salt*Vp*type: constant
*LDB*Salt*Vp*value: 4800.0
*LDB*Shell2*variables: Vp
*LDB*Shell2*Vp*type: constant
*LDB*Shell2*Vp*value: 4800.0
*PDB*Vs*variables: Vi
*PDB*Vs*Vi*type: interpolation
*PDB*Vs*Vi*top: Layer1
*PDB*Vs*Vi*bottom: Salt
*PDB*Vs*Vi*property: Vp
*PLDB*Vs*Layer2*variables: Vs
*PLDB*Vs*Layer2*Vs*type: c_script
*PLDB*Vs*Layer2*Vs*arguments: Vi
*PLDB*Vs*Layer2*Vs*value: {Vs=Vi+2000.0;}
}_PROPERTY_MODEL_DB

END
```

FIGURE 12. Illustration of the example given in Example *(Page A50).*

A.12 GShape

A GShape is a Curve with Sections defined by Ribs along the Atoms of the Curve. Its file format therefore consists of a PLine (Curve) and

```
NBRIB nbribs
TOP rib_begin rib_end
SEC atom_id x0 y0 z0 x2 y2
z2 .... x_nbribs-1
y_nbribs-1 z_nbribs-1
```

NBRIBS defines the number of ribs at each Atom of the BackBone (the Curve).

TOP defines the top portion of the GShape and is optional information.

SEC gives the XYZ coordinates of the vectors that define the profile of the GShape at the specified Atom of the BackBone.

At the end of each SEC line, additional information maybe added:

CSEC specifies that the Section is a Control Section.

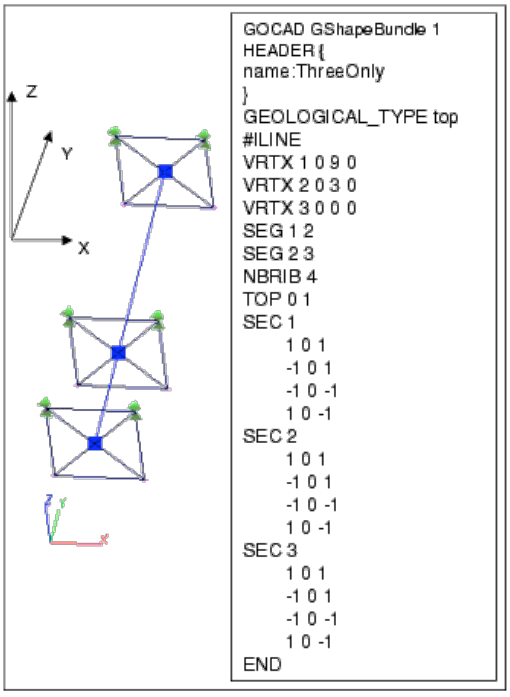


FIGURE 13. An example of a GShape and its data file.

A.13 Group

The format of the Ascii file for Object Group is given by different examples.

Heterogeneous group:

```
GOCAD HeterogeneousGroup 1.0
HEADER {
name:gp1
}
BEGIN_MEMBERS
GOCAD TSurf 0.01
....
END
GOCAD PLine 0.01
....
END
END_MEMBERS
END
```

Homogeneous Group

```
GOCAD HomogeneousGroup 1.0
HEADER {
name:gp2
}
TYPE TSurf
BEGIN_MEMBERS
```

```
GOCAD TSurf 0.01
....
END
GOCAD TSurf 0.01
....
END
END_MEMBERS
END
```

Group and its Member Objects into separate files:

```
GOCAD HeterogeneousGroup 1.0
HEADER {
name:gp1
separate_files:on
}
FILE gp2.gp
FILE pl2.pl
END
```

A.14 Geostatistical Files

GOCAD's goal is to provide geostatistical applications for users who are already familiar with the terminology and concepts of geostatistics. Therefore we assume that the user does not need detailed definitions of the variables used in this section, such as CUT_OFF, COVARIANCE_MODEL, etc.

Almost all geostatistical applications in GOCAD requires one or more ASCII files that contain variogram and/or other geostatistical information.

This section is based on information provided by Tom Tran. If you have any questions about this or other GOCAD geostatistical applications, please contact him directly at jtttr@chevron.com.

These file formats change rapidly as the geostatistical applications are being modified and developed rapidly. If your files are not accepted by GOCAD during an application, please contact Tom Tran at jtttr@chevron.com or Arben Shtuka at shtuka@ensg.u-nancy.fr.

A.14.1 An Example of a Variogram File

```
================================================================================
# coordinate system can either be XYZ, XYW, or UVW
# in this example, XYW is chosen; therefore, the areal correlation
# ranges are in real-world coordinates and the vertical correlation
# range is in normalized (0 to 1) coordinate
COORDINATE_SYSTEM XYW
# not used
KRIGING_TYPE 0
# maximum number of nearby data for kriging
MAX_CLOSE 16
# 0 = simple kriging; 1 = ordinary kriging
KRIGING_OPTION 0
# search ellipsoid
# generally, one should make the ranges of the search ellipsoid
# larger than the correlation ranges
# format: angle1 angle2 angle3 range1 range2 range3
SEARCH_ELLIPSOID 0. 0. 0. 4000. 4000. 0.4
# covariance model
# format COVARIANCE_MODEL sill number_of_nested_structures
# model_type angle1 angle2 angle3 range1 range2 range3 contribution
# model type can either be SPHERICAL EXPONENTIAL GAUSSIAN or POWER
# in this example, there is only 1 nested structure, so only one model line
# is needed
COVARIANCE_MODEL 1. 1
SPHERICAL 0. 0. 0. 2000 2000 0.2 1.
END
================================================================================
```

A.14.2 An Example of an Indicator_Variogram File

```
====================================================================
# comments line start with this symbol
# coordinate system can either be XYZ, XYW, or UVW
# in this example, XYW is chosen; therefore, the areal correlation
# ranges are in real-world coordinates and the vertical correlation
```

```
# range is in normalized (0 to 1) coordinate
COORDINATE_SYSTEM XYW
# not used
KRIGING_TYPE 0
# maximum number of nearby data for kriging
MAX_CLOSE 16
# 0 = simple kriging; 1 = ordinary kriging
KRIGING_OPTION 0
# 3 cutoffs
NB_CUT_OFFS 3
# first cutoff is 0.268 with cumulative probability of 25%
# second cutoff is 0.300 with cumulative probability of 50%
# third cutoff is 0.332 with cumulative probability of 75%
CUT_OFFS 0.268 0.25 0.300 0.50 0.332 0.75
# search ellipsoids corresponding to each cutoff
# generally, one should make the ranges of the search ellipsoids
# larger than the correlation ranges
# format: angle1 angle2 angle3 range1 range2 range3
SEARCH_ELLIPSOID 45. 0. 0. 5000. 2500. 0.2
SEARCH_ELLIPSOID 0. 0. 0. 5000. 5000. 0.2
SEARCH_ELLIPSOID 135. 0. 0. 5000. 2500. 0.2
# covariance model for each cutoff
# format COVARIANCE_MODEL sill number_of_nested_structures
# model_type angle1 angle2 angle3 range1 range2 range3 contribution
# model type can either be SPHERICAL EXPONENTIAL GAUSSIAN or POWER
# in this example, there is only 1 nested structure, so only one model line
# is needed
# covariance for first cutoff
COVARIANCE_MODEL 1. 1
SPHERICAL 45. 0. 0. 3000. 500. 0.1 1.
# covariance for second cutoff
COVARIANCE_MODEL 1. 1
SPHERICAL 0. 0. 0. 2500. 2500. 0.1 1.
# covariance for third cutoff
COVARIANCE_MODEL 1. 1
SPHERICAL 135. 0. 0. 3000. 250. 0.1 1.
END
```

A.14.3 An Example of a Column_Average_Map File

This file is needed in Block-Kriging and Simulated Annealing. In this example, an sgrid with dimension 138x33x100. The first two lines must start with NX and NY which specify the number of cells in u and v direction, followed by NX*NY lines of data (one data per line) representing the column averages.

```
NX 138
NY 33
0.23
...
0.30
```

A.14.4 An Example of a Scattergram File

This file is needed in Cloud Transform (w/ P-Field). This file has no header lines, just a series of data lines in the format of
Independent_Variable Dependent_Variable

```
0.1 1.5
0.1 1.3
0.1 1.6
...
1.2 3.9
```

A.14.5 An Example of an External_Histogram File

This file is needed in Simulated Annealing and Continuous Histogram Correction. This file has no header lines, just a serious of data lines in the format of
Data_Value
GOCAD then constructs the Histogram using the data read in from the file.

```
1.0
0.9
...
1.2
```

A.14.6 An Example of a Facies_Map File

This file is needed in Fill From Facies Map. This file has a header line for the user's own identification (not used in algorithm), followed by a serious of data lines in the format of
X_coord Y_coord Facies_value

```
========================================================================
1000000.0 23455066.0 2
1230303.0 39393939.0 1
...
3030303.0 10393030.0 3
========================================================================
```

A.14.7 An Example of a Annealing_Schedule File

This file is needed in Simulated Annealing.

```
========================================================================
# comments line start with this symbol
#but there is no real comments because if the variables
#are not obvious to the users, it would take too long to define them
# comments
INITIAL_TEMPERATURE 0.001
REDUCTION_FACTOR 0.1
# comments
MAX_PERTURB 1000.
MAX_PERTURB_PER_TEMP 10.
# comments
MAX_SUCC_PERTURB_PER_TEMP 3.
MIN_OBJECTIVE 0.001
# comments
REPORT_INTERVAL 0.2
STOPPING_NUMBER 5
========================================================================
```

A.15 Other Text Files

A.15.1 Color File

A color file contains color assignments for Well Marker and Well Zone.

Not all Well Markers or Zones in the GOCAD session that calls this file need to be listed in the file and not all listed Markers or Zones need to exist in the GOCAD session that calls this file. The color can be in RGB code, or an ASCII name that GOCAD recognizes. If a Marker is assigned a color that GOCAD does not recognize, such as pacific sunset, it will be displayed in black.

```
name_1 color_1
name_2 color_2
Top_of_Mesozoic green
Jurrasic_Interval 0.3 0.42 0.7
...
```

Where name_n is the name of a Marker or a Zone, and color_n is its assigned color either as RGB code, e.g. 0.3 0.42 0.7, or an English name, such as green.

A.15.2 ASCII Curve File

GOCAD allows Well Curves to be added to an existing Well (during a GOCAD session) from an ASCII file. Such a file has 2 header lines followed by data lines:

```
Number_of_Fields
field_1_name field_2_name I_am_Z field_4_name...
data_1_1 data_2_1 data_3_1 data_4_1...
data_1_2 data_2_2 data_3_2 data_4_2...
...
```

Where Number_of_Fields specifies how many fields (columns) in each data line, and field_name_n is the Proeprty name of each field (column), which will be used to name the Well Curve created from that field (column) of data.

One of these fields (columns) must contain the Z value (depth) for that data line. In the GOCAD command Add Curve From Ascii dialog, the user will be asked to enter the Property (field) name that contains the Z values., e.g. I_am_Z. Z values can be in either measured depth or true depth, which is also specified by the user in the Add Curve From Ascii command dialog.