# General Manual Geo-MHYDAS 12.01

These scripts are made for the Digital Landscape Representation (DLR) for MHYDAS model running with the OpenFluid platform 1.6.x or superior.

Ubuntu linux operating system and GIS GRASS software version 6.3 or superior are needed.

The two following units can be produced: Surface Units (SU) and Reach Segments (RS). For more informations about these concepts, see OpenFLUID website

For more informations, please read :

Lagacherie, P., Rabotin, M., Colin, F., Moussa, R. and Voltz, M., 2010. Geo-MHYDAS: A landscape discretization tool for distributed hydrological modeling of cultivated areas, Computers & Geosciences, 36, p.1021 – 1032. See more

The following input geographical layers can be provided for the segmentation procedures :

- DEM raster (compulsory) (e.g. ArcInfo Raster, Ascii file ...)
- land use vector layer (e.g. shapefiles ESRI, ArcInfo Coverage)
- soil use vector layer (e.g. shapefiles ESRI, ArcInfo Coverage)
- reach segments vector layer (e.g. shapefiles ESRI, ArcInfo Coverage)
- subwatersheds vector layer (e.g. shapefiles ESRI, ArcInfo Coverage)

Importing these layers in GRASS GIS software can be done by r.in.gdal command for raster data and v.in.ogr command for vector layer. If you use these scripts for the whole Digital Landscape Representation procedures and want to create the two unit types (SU and RS), at the minima the **bold** scripts are compulsories, the others are optionals.

## Processsing geographical objects

| | |
|---|---|
| m.disline | Dissolving small linear entities. |
| m.dispolyg | Dissolving small areal entities. |
| m.douglas | Smoothing areal entities (Douglas Peucker algorithm). |
| **m.network** | Hydrological network verification (rooted tree). |
| m.sbw | Subwatershed calculation program with hydrological network influence. |
| m.segline | Splitting linear entities by points. |
| m.snaplp | Snapping linear entities on areal entities. |

## Creating hydrological units by selective overlays

| | |
|---|---|
| m.colseg | Patch desired columns from the input polygon vector to the segmented output vector. |
| m.dispolygseg | Dissolving small areal entities (with hierarchical level) after segmentation. |
| **m.extractlineseg** | Linear units extraction after segmentation. |
| **m.seg** | Segmentation Procedure. |
| m.sliverpolygseg | Dissolving sliver entities with hierarchical level after segmentation (use Gravelius Index). |

## Building oriented topologies of hydrological units

| | |
|---|---|
| **m.toporeach** | Topology calculation for reach segments |
| **m.toposu** | Topology calculation for surface units |
| m.definput | Creation xml files or fluidx files for OpenFluid Engine 1.5 or superior |

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 23 January 2012

## NAME

***m.disline*** - dissolving small linear entities

## KEYWORDS

vector, line, dissolve

## SYNOPSIS

**m.disline**
**m.disline help**
**m.disline** [-**i**] **input**=*name* **output**=*name* **length**=*float* [**unit**=*string*] [**columnp**=*string*] [**valuep**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-i**
: print only statistics (no calcul) and exit

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name*
: Input vector name

**output**=*name*
: Output vector name

**length**=*float*
: min length value

**unit**=*string*
: units (meters(me), kilometers(k))
: Options: *me,k*
: Default: *me*

**columnp**=*string*
: Name of column used to treat particular entities (column must exist)

**valuep**=*string*
: value used in 'columnp' to identify the particular entities

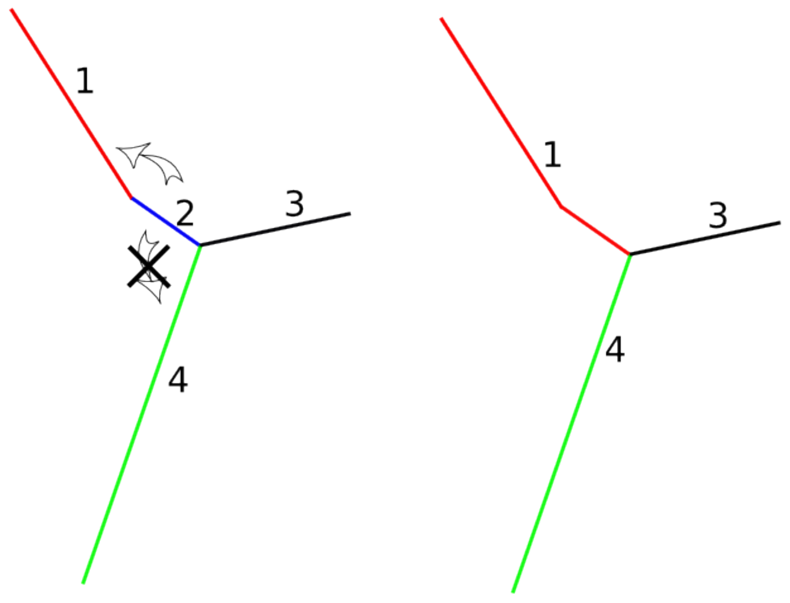
## DESCRIPTION

*m.disline* allows the user to dissolve small linear entities with user's threshold (*length* parameter). Flag i prints only statistics (see *Examples* for details). *Columnp* and *valuep* can be used to treat separately particular entities (e.g. pipe entities can be dissolve only with pipe entities), see *Examples* for details.

## NOTES

Dangle segments are removed (as in v.clean tool=rmdangle command).

Confluence integrity is preserved : in the following example, on the left figure, in the input layer, length of the segment 2 is under the user's threshold and should be merged on segment 4 which is its longest neighbour. But in order to preserve geometry confluence, segment 2 is merged on segment 1, as seen on the right figure in the output layer.



*Input layer (left) ; Output layer (right)*

## EXAMPLES

## Print Statistics

Print Statistics for dissolving linear entities for the line1 vector map (threshold is 200 meters)

```
GRASS 6.3.0 :~ > m.disline -i input=line1 output=line2 length=200 unit=me
You choose a minimum length of 200 m
 For the vector line1,
 the dissolving small linear entities operation will work
 on 8 lines for 32 total lines
 and will represent 3.5 % of the total length
```

## Dissolving linear entities

Dissolving linear entities for the line1 vector map (threshold is 200 meters)

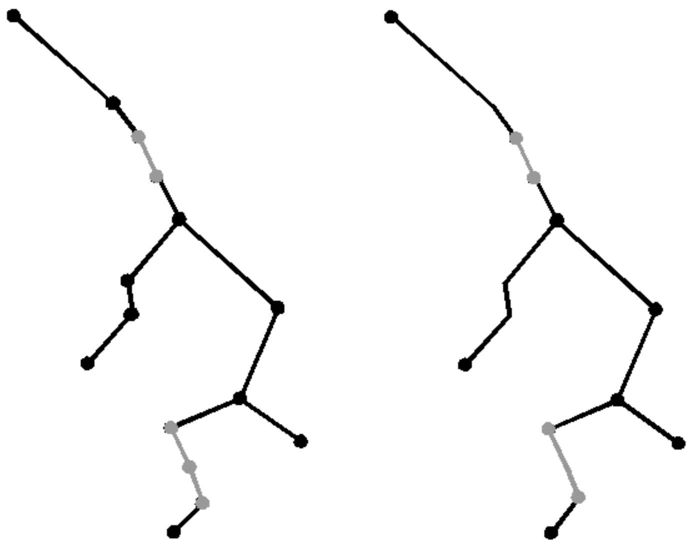
```
GRASS 6.3.0 :~ > m.disline input=line1 output=line2 length=200 unit=me
```

## Particular entities treatment

Dissolving small linear entities for the network1 vector map (threshold is 100 meters) and separate treatment for particular entities defined by the column *'type'* and the value *'pipe'*

```
GRASS 6.3.0 :~ > m.disline input=network1 output=network2 length=100 unit=me
columnp=type valuep=pipe
```

In the following figure, pipe segments are in grey color and segment nodes are represented by circles. In the output layer (on right figure), small segments with the same attribut (which length is under user's threshold) are dissolved; pipe segments can be preserved if they have no pipe neighbours.



*Input network1 layer (left) ; Output network2 layer (right)*

## SEE ALSO

m.dispolyg, m.network, m.segline, m.snaplp

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 23 January 2012

## NAME

***m.dispolyg*** - dissolving small areal entities

## KEYWORDS

vector, dissolve, areal

## SYNOPSIS

**m.dispolyg**
**m.dispolyg help**
**m.dispolyg** [-**i**] **input**=*name* **output**=*name* **area**=*float* [**unit**=*string*] [**columnp**=*string*] [**valuep**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-i**
: print only statistics (no calcul) and exit

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name*
: Input vector name

**output**=*name*
: Output vector name

**area**=*float*
: minimum area value

**unit**=*string*
: units (meters (me), kilometers (k), acres (a), hectares (h))
: Options: *me,k,a,h*
: Default: *me*

**columnp**=*string*
: Name of column used to treat particular entities (column must exist)

**valuep**=*string*
: value used in 'columnp' to identify the particular entities

## DESCRIPTION

*m.dispolyg* allows the user to dissolve small areal entities with user's threshold (*area* parameter). Flag i prints only statistics (see *Examples* for details). *Columnp* and *valuep* can be used to treat separately particular entities (e.g. road entities can be dissolved only with road entities), see *Examples* for details.

## NOTES

Small areal entities are dissolved to the neighouring entity with whom they share the longest boundary. The dissolving process is iterative (process is stopped when no more small entity is found). Small areal entities under the threshold but with no neighbours are preserved.

## EXAMPLES

### Print statistics

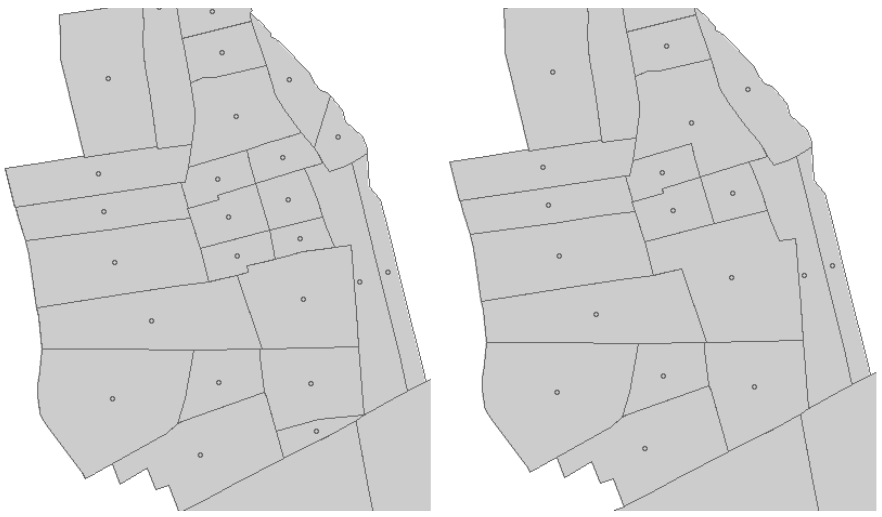
Print statistics for dissolving areal entities for the polygon1 vector map (threshold is 1200 square meters)

```
GRASS 6.3.0 :~ > m.dispolyg -i input=polygon1 output=polygon2 area=1200 unit=me
You choose a minimum area of 1200 m2
 For the vector polygon1,
 the dissolving small area entities operation will work
 on 6 polygons for 28 total polygons
 and will represent 4.3 % of the total surface
```

### Dissolving areal entities

Dissolving areal entities for the polygon1 vector map (threshold is 1200 square meters)

```
GRASS 6.3.0 :~ > m.dispolyg input=polygon1 output=polygon2 area=1200 unit=me
```

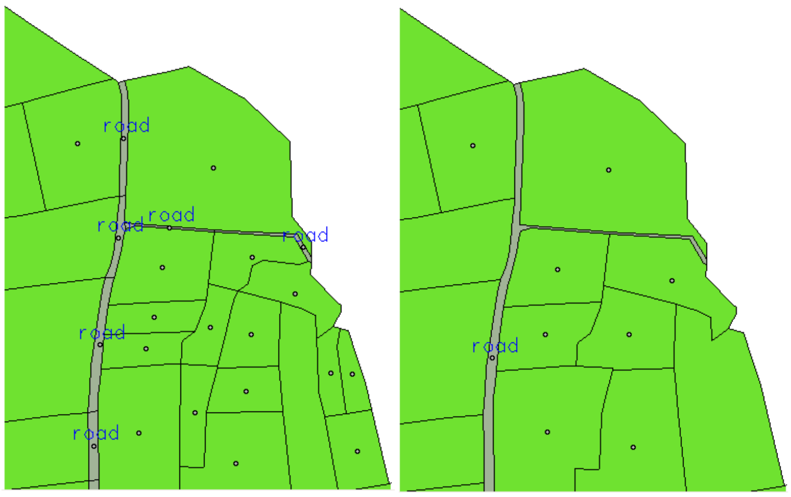


*Input polygon1 layer (left) ; Output polygon2 layer (right)*

## Particular entities treatment

Dissolving small areal entities for the plot1 vector map (threshold is 100 square meters) and separate treatment for particular entities defined by the column *'type'* and the value *'road'*

```
GRASS 6.3.0 :~ > m.dispolyg input=plot1 output=plot2 area=100 unit=me columnp=type
valuep=road
```



*Input plot1 layer (left) ; Output plot2 layer (right)*

## SEE ALSO

m.disline

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.douglas*** - smoothing areal entities (Douglas Peucker algorithm)

## KEYWORDS

vector, generalization, node, Douglas-Peucker

## SYNOPSIS

**m.douglas**
**m.douglas help**
**m.douglas input**=*name* **output**=*name* **dist**=*float* [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**--overwrite**
    Allow output files to overwrite existing files
**--verbose**
    Verbose module output
**--quiet**
    Quiet module output

## Parameters:

**input**=*name*
    Input vector map
**output**=*name*
    Output vector map
**dist**=*float*
    Douglas Peucker distance threshold (in map units)

## DESCRIPTION

*m.douglas* script allows the user to generalize areal boundaries; it is based on the well-known Douglas-Peucker algorithm (*See* Douglas & Peucker, 1973).

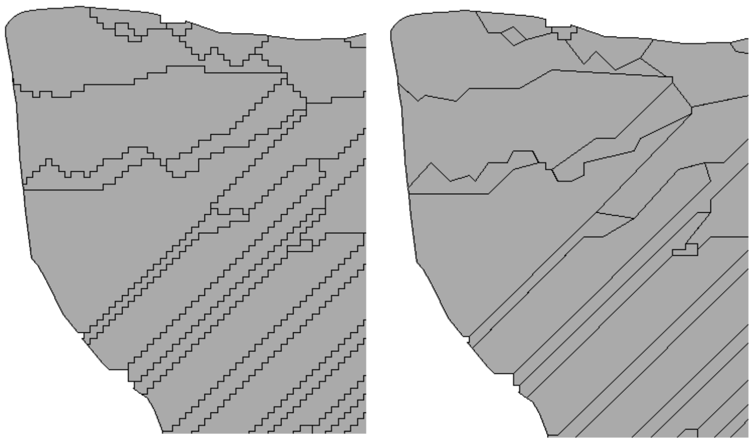
## NOTES

This script works only on lines with 90° angles (i.e. lines issued of vectorization of raster cells), all the other lines won't be modified. If the user wants to generalize all lines of a vector layer, v.generalize command should be used.

## EXAMPLES

### Areal boundaries generalized with a distance of 5

```
GRASS 6.3.0 :~ > m.douglas input=polygon1 output=polygon2 dist=5
```



*Input layer (left) ; Output layer (right)*

## SEE ALSO

v.generalize

## REFERENCES

David Douglas & Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112-122 (1973)

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed: 23* January 2012

---

Main index

## NAME

***m.network*** - hydrological network verification (rooted tree)

## KEYWORDS

vector, network, rooted tree

## SYNOPSIS

**m.network**
**m.network help**
**m.network** [-**c**] **input**=*name* **output**=*name* [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-c**
    clean vector geometry for confluence problem (default is no)
**--overwrite**
    Allow output files to overwrite existing files
**--verbose**
    Verbose module output
**--quiet**
    Quiet module output

## Parameters:

**input**=*name*
    Input vector name
**output**=*name*
    Output vector name

## DESCRIPTION

*m.network* allows the user to verify the geometry of the input line vector. As the MHYDAS model accepts only rooted tree structure for line vectors, *m.network* identifies possible confluence geometry problems, loop presence, duplicate segments and checks segment connection. See *Examples* for details.

## NOTES

Flag - *c* allows the user to clean the geometry vector and modifies the attribute table if confluence geometry problems are identified. Note that in the attribute table, a row is inserted for each new segment created (creation is made using *v.clean* *tool=break* command) but only the *cat* column is updated. See *Examples* for details.

## EXAMPLES

### Network verification

```
GRASS 6.3.0 :~ > m.network input=line1 output=line2
Writing attributes...

--> Everything seems ok !
```

### Geometry problems detected during verification

Loops detected during network verification :

```
GRASS 6.3.0 :~ > m.network input=loop1 output=loop2
WARNING: The network has 2 cycle(s)
WARNING: Please check on the loop1 vector the followings category segments
         to eliminate the cycles:
15
18
19
20
21
22
23
24
```

Confluence problems detected during network verification :

```
GRASS 6.3.0 :~ > m.network input=line3 output=line4

WARNING: <line3> has geometry problems on confluence. Check it please
```

Flag - *c* can be used to clean geometry for this problem :

```
GRASS 6.3.0 :~ > m.network -c input=line3 output=line4
Writing attributes...

--> Everything seems ok !
```

Connection problems detected during network verification :

```
GRASS 6.3.0 :~ > m.network input=disconnect1 output=disconnect2

WARNING: The network isn't connected, please check it
WARNING: Maybe check the following segments (category identifiant):
3
5
6
2
```

## SEE ALSO

m.disline, m.segline

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 23 January 2012

---

Main index

## NAME

***m.sbw*** - subwatershed calculation program with hydrological network influence

## KEYWORDS

subwatershed, network influence, stream burning method, modifying flowdir method

## SYNOPSIS

**m.sbw**
**m.sbw help**
**m.sbw** [-**bfavcis**] **dem**=*name* **watershed**=*name* **stream**=*name* **sbv**=*name* [**point**=*name*] [**file**=*string*] [**burn**=*float*] **area**=*float* [**res**=*float*] **col**=*string* **percent**=*float* [**snap**=*float*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-b**
: use stream burning method ; Value of digging in option 'burn'; default is no

**-f**
: use modifying flowdir method ; default is no

**-a**
: use of ascii coordinate point file for basin outlets; file name in option 'file' (see file option for the file format); default is no

**-v**
: use of point vector for basin outlets; vector name in option 'point'; default is no

**-c**
: smoothing corners of subwatershed features; default is no

**-i**
: use stream inlets to create basin; default is no

**-s**
: use stream confluence nodes to create basin outlets; default is no

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**dem**=*name*

Input DEM raster

**watershed**=*name*
: Input watershed raster

**stream**=*name*
: Input network vector

**sbv**=*name*
: Output subwatersheds name

**point**=*name*
: Input outlet points vector

**file**=*string*
: Name of ascii coordinate point file for basin outlets (without intitulate column, field separator is 'space', col1 x and col2 y)

**burn**=*float*
: Value of digging for the burning stream method

**area**=*float*
: min area (in map units) value for the subwatershed

**res**=*float*
: raster resolution

**col**=*string*
: column name for subwatershed identification

**percent**=*float*
: percentage of zero value accepted for subwatershed creation (area difference with watershed)

**snap**=*float*
: distance (in map units) to snap outlet point to stream (default is 1)

## DESCRIPTION

*m.sbw* allows the user to create subwaterheds with hydrological network influence. This is useful for watersheds with a man made hydrological network which does not follow the longest slope. The subwatersheds creation is forced to take care of these singularities.

## NOTES

*m.sbw* is based on r.watershed command plus several options allowing to take care the influence of existant hydrological network on subwatershed creation: using the stream burning method on DEM ( *See Mizgalewicz, P.J. & al, 1996; Saunders, W.K., & al 1996* ), or flow direction modified method ( *See Lagacherie, P. & al, 1996* ).

The stream burning method digs the DEM cells which are in contact of the network ( *Flag -b* is used), then the flow direction raster is computed and subwatersheds created.

The flow direction modified method (use *Flag -f* ) works directly on the flow direction raster and reorients raster cells which are in contact of the network. For each of these cells, orientation is modified to flag to the network. It's an iterative process which can be slow.

Because of working in cultivated landscapes, the *percent* value needs to be used for forcing subwatershed creation. **In some case of very modified landscapes, subwatersheds creation can fail.**

Subwatershed outlets can be submitted from a vector map (use *Flag -v* and *point* option), or with an ascii file (use *Flag -a* and *file* option). *snap* option allows the outlet coordinates being snapped on the hydrological network. Using *Flag -i* allows to use stream inlets for additional outlets. Using *Flag -s* allows to use confluence points to create additional outlets. Multiple use of different flags can be made to the outlet points creation.

*area*, *col* and *percent* options are compulsories:

- *area* is the minimum surface value of created subwatersheds (in map units).

- *col* will be the column name (type INTEGER) in the *output* map to identify the different subwatersheds.

- *percent* is the percentage of non value cells resulting of the difference between the whole watershed and the sum of created subwatersheds. As working in modified landscapes where DEM informations aren't the most important for flow directions, an iterative process modifies the flow direction cells in contact of subwatershed boundaries to modify their direction values.

*res* and *snap* options are optionals:

- *res* option allows the user to work with a different resolution of input DEM

- *snap* option allows the user (if *file* or *point* option is used) to snap the outlet points on the line network under this threshold (distance in map units)

## EXAMPLES

### Subwatershed creation with stream burning method, outlets submitted by point vector map

DEM is digged with 1 meter value, snap value equal to 2.5 meters, the minimum area of subwatersheds equals 1000 square meters and surface of subwatersheds must be at minimum 98% of the whole watershed (2% of cells with null value)

```
GRASS 6.3.0 :~ > m.sbw -b -v dem=dem watershed=wshed stream=stream sbv=subwshedB
burn=1 col=ident percent=2 area=1000 point=outlet snap=2.5
```

### Subwatershed creation with flow direction modified method, and smoothed corners of subwatersheds

```
GRASS 6.3.0 :~ > m.sbw -f -c dem=dem watershed=wshed stream=stream sbv=subwshedF
col=ident percent=2 area=1000
```

### Subwatershed creation with classical method, and modified DEM resolution

```
GRASS 6.3.0 :~ > m.sbw dem=dem watershed=wshed stream=stream sbv=subwshedC col=ident
percent=2 area=1000 res=50
```

## Subwatershed creation with classical method, and using confluence network for creating outlet

```
GRASS 6.3.0 :~ > m.sbw -s dem=dem watershed=wshed stream=stream sbv=subwshedC
col=ident percent=2 area=1000
```

## SEE ALSO

m.network, r.watershed

## REFERENCES

Lagacherie, P., Moussa, R., Cormary, D., AND Molenat, J. 1996. Effect of DEM data source and sampling pattern on topographical parameters and on a topography-based hydrological model. *In* "HYDROGIS'96. Application of Geographic Information System in Hydrology and Water Resources Management" (K. Kovar and H. P. Nachtnebel, eds.), pp. 191-200. IAHS, Vienna.

Mizgalewicz, P.J., Maidment, D.R., 1996. Modeling agrichemical transport in midwest rivers using geographic information systems. Center for Research in Water ressources Online Report 96-6, University of Texas, Austin, TX, 338pp.

Saunders, W.K., Maidment, D.R., 1996. A GIS assessment of nonpoint source pollution in the San Antonio- Nueces coastal basin. Center for Research in Water ressources Online Report 96-1, University of Texas, Austin, TX, 222pp.

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 23 January 2012

---

Main index

## NAME

***m.segline*** - splitting linear entities by points

## KEYWORDS

vector, line, split

## SYNOPSIS

**m.segline**
**m.segline help**
**m.segline input**=*name* **output**=*name* [**point**=*name*] [**file**=*string*] **snap**=*float* [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**--overwrite**
    Allow output files to overwrite existing files
**--verbose**
    Verbose module output
**--quiet**
    Quiet module output

## Parameters:

**input**=*name*
    Input vector name
**output**=*name*
    Output vector name
**point**=*name*
    Input point vector
**file**=*string*
    Input ascii point file (without intitulate column, field separator is 'space', col1 x and col2 y)
**snap**=*float*
    snap distance (in map units)

## DESCRIPTION

*m.segline* script allows the user to split vector lines by points. Points can be submitted from a point vector map or an ascii file (See *Examples* for details).

## NOTES

The *snap* value allows the user to snap (under this threshold) splitting points on the closest line. Note that after splitting, a row is inserted in the attribute table for each new segment created but only the *cat* column is updated.

## EXAMPLES

### Splitting line vector by vector point

With *snap* value at 2 map units

```
GRASS 6.3.0 :~ > m.segline input=line1 output=line2 point=splitPT snap=2
```

### Splitting line vector by ascii file

With *snap* value at 2 map units

```
GRASS 6.3.0 :~ > cat /tmp/splitFile
673401 1830816
GRASS 6.3.0 :~ > m.segline input=line1 output=line2 file=/tmp/splitFile snap=2
```

## SEE ALSO

m.disline, m.network

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.snaplp*** Snapping linear entities on areal entities

## KEYWORDS

vector, linear, snap, areal

## SYNOPSIS

**m.snaplp**
**m.snaplp help**
**m.snaplp** [-**b**] **input**=*name* **output**=*name* **polygon**=*name* **snap**=*float* [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-b**
: Line creation by polygone boundaries (pseudo snap because polygone boundaries are used to create a new vector line)

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name*
: Input vector map

**output**=*name*
: Output vector map

**polygon**=*name*
: Input vector polygon

**snap**=*float*
: maximum distance of snap (in map units)

## DESCRIPTION

*m.snaplp* allows the user to snap lines on areal entities in threshold (*snap* value).

## NOTES

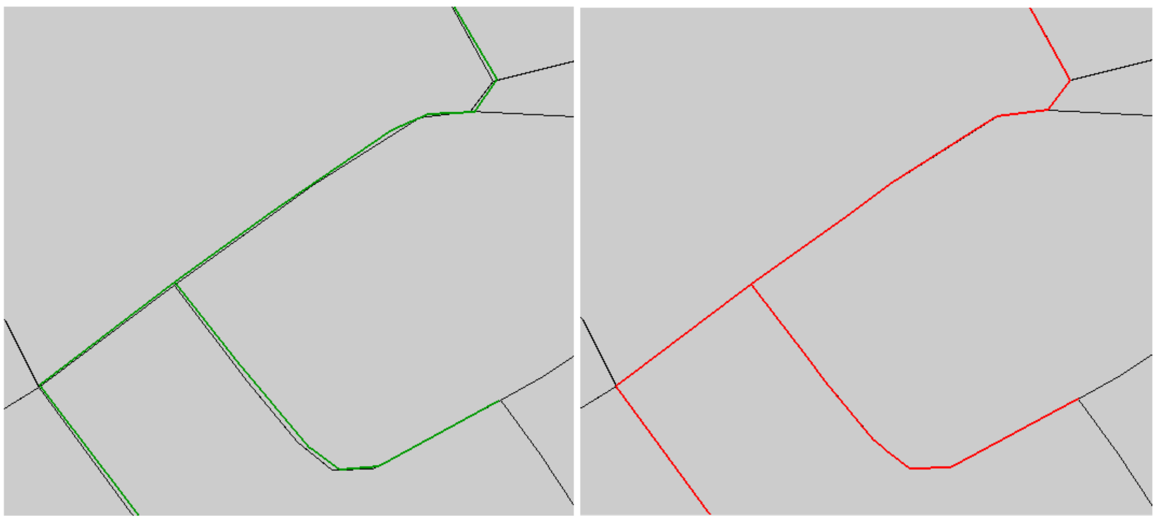
This script makes an iterative process for each line in three steps: snap line points on polygon points, snap line points on polygon boundaries and add polygon points along line. If the *snap* value is too large, geometry problems can appear. In this case, the user should clean the line vector geometry manually before using *m.snaplp* script.

If using Flag B, a more simplified method is used : closest boundaries of polygon layer are used to create a new line vector map. Each boundary used will be a new line.

## EXAMPLES

### Snapping line1 vector to polygon1 vector with a threshold of one meter

```
GRASS 6.3.0 :~ > m.snaplp input=line1 polygon=polygon1 output=line2 snap=1
```



*Input line1 layer (left; in green color) ; Output line2 layer (right; in red color)*

### Using Flag B

```
GRASS 6.3.0 :~ > m.snaplp -b input=line1 polygon=polygon1 output=line2 snap=1
```

## SEE ALSO

m.network, m.disline, m.segline

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.colseg*** - patch desired columns from input vector to the segmented output vector (for polygon or line)

## KEYWORDS

vector, segmentation, attribute table, patch

## SYNOPSIS

**m.colseg**
**m.colseg help**
**m.colseg input**=*name* **segmented**=*name* **output**=*name* **columns**=*string*[,*string*,...] **val**=*float* [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**--overwrite**
    Allow output files to overwrite existing files
**--verbose**
    Verbose module output
**--quiet**
    Quiet module output

## Parameters:

**input**=*name*
    Input  vector map to patch
**segmented**=*name*
    Input segmented vector name
**output**=*name*
    Output segmented vector name
**columns**=*string[,string,...]*
    Name of desired columns to patch from INPUT to OUTPUT (name separated by comma ',')
**val**=*float*
    distance of maximum search (in map units)

## DESCRIPTION

*m.colseg* allows the user to patch attribute columns from the input vector to the segmented output vector. Several columns can be patched with one operation. If the user wants to patch columns from several input vectors, operations must be launched for each input vector. *m.colseg* allows to patch from input line vector or input polygon vector.

## NOTES

This script is compulsory if the user wants to have attribut data from input vectors. Because, after using *m.seg*, the segmented polygon vector (or line vector) has only the ID columns in the attribute table.

## EXAMPLES

### Joint attribute data (num_parc,occ_sol) from field vector to seg vector

```
GRASS 6.3.0 :~ > m.colseg input=field segmented=seg output=seg2
columns=num_parc,occ_sol val=0.1
```

### Joint attribute data (width,Ks) from reach vector to seg_reach vector

```
GRASS 6.3.0 :~ > m.colseg input=reach segmented=seg_reach output=seg_reach2
columns=width,Ks val=10
```

## SEE ALSO

m.seg

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 23 January 2012

Main index

## NAME

***m.dispolygseg*** - dissolving small areal entities (with hierarchical level) after segmentation

## KEYWORDS

vector, selective dissolve, areal, hierarchical order

## SYNOPSIS

**m.dispolygseg**
**m.dispolygseg help**
**m.dispolygseg** [-**i**] **input**=*name* **output**=*name* **area**=*float* [**unit**=*string*] [**columnp**=*string*] [**valuep**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

### Flags:

**-i**
: print only statistics (no calcul) and exit

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

### Parameters:

**input**=*name*
: Input vector name

**output**=*name*
: Output vector name

**area**=*float*
: min area value

**unit**=*string*
: units (meters (me), kilometers (k), acres(a), hectares(h))
: Options: *me,k,a,h*
: Default: *me*

**columnp**=*string*
: Name of column used to treat particular entities (column must exist)

**valuep**=*string*
: value used in 'columnp' to identify the particular entities

## DESCRIPTION

*m.dispolygseg* allows the user to dissolve small areal entities with user's threshold (*area* parameter) and with hierarchical order. The input layer must be created by *m.seg* script. Flag i prints only statistics (see *Examples* for details). *Columnp* and *valuep* can be used to treat separately particular entities (e.g. road entities can be dissolved only with road entities), see *Examples* for details. To treat particular entities, *m.colseg* script must be used to patch attribute columns to identify entities.

## NOTES

Small areal entities are dissolved to the neighouring entitie with whom they share the longest boundary and whom have the lowest hierarchical order (selective cleaning procedure). The hierarchical order is defined by *m.seg* script and is provided by the attribute table(s)s of the input vector data. The dissolving process is iterative (process is stopped when no more small entity is found). Small areal entities under threshold but with no neighbours are preserved.

## EXAMPLES

### Print statistics for dissolving areal entities for the seg1 vector map (threshold is 1200 square meters)

```
GRASS 6.3.0 :~ > m.dispolygseg -i input=seg1 output=seg2 area=1200 unit=me
You choose a minimum area of 1200 m2
 For the vector seg1,
 the dissolving small area entities operation will work
 on 6 polygons for 28 total polygons
 and will represent 4.3 % of the total surface
```

### Dissolving areal entities

Dissolving areal entities for the seg1 vector map (threshold is 1200 square meters)

```
GRASS 6.3.0 :~ > m.dispolygseg input=seg1 output=seg2 area=1200 unit=me
```

### Particular entities treatment

Dissolving small areal entities for the seg1 vector map (threshold is 1000 square meters) and separate treatment for particular entities defined by the column *'type'* and the value *'road'* Using *m.colseg* script to patch attribute column from field vector input data.

```
GRASS 6.3.0 :~ > m.colseg input=seg1 output=seg1p polygon=field columns=type
GRASS 6.3.0 :~ > m.dispolygseg input=seg1p output=seg2 area=1000 unit=me
columnp=type valuep=road
```

## SEE ALSO

m.seg, m.sliverpolygseg, m.dispolyg, m.colseg

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.extractlineseg*** - linear units extraction after segmentation

## KEYWORDS

vector, segmentation, linear units, extraction

## SYNOPSIS

**m.extractlineseg**
**m.extractlineseg help**
**m.extractlineseg** **input**=*name* **polygon**=*name* **output**=*name* [**ID**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**--overwrite**
  Allow output files to overwrite existing files
**--verbose**
  Verbose module output
**--quiet**
  Quiet module output

## Parameters:

**input**=*name*
  Input line vector name
**polygon**=*name*
  Input segmented vector name
**output**=*name*
  Output segmented line vector name
**ID**=*string*
  Name of column for unique ID for the output layer (numerotation from 1); default is 'SELF_ID'

## DESCRIPTION

*m.extractlineseg* script allows the user to extract linear units from segmented polygon input. The input polygon layer must be created by m.seg script. Input linear layer must be provided too.

## NOTES

*m.extractlineseg* script creates a segmented linear layer from intersection between boundary nodes of the input segmented polygon layer and input linear layer. Even, if this script is able to work with complex geometry features, user should check the validity of the output linear layer produced ! Eventually, launch m.network script to verify vector geometry.

## EXAMPLES

### Creation of the seg_reach layer (with *ident* column name) from seg1 polygon layer and reachs linear layer

```
GRASS 6.3.0 :~ > m.extraclineseg input=reachs polygon=seg1 output=seg_reach ID=ident
```

## SEE ALSO

m.seg, m.colseg, m.toporeach

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.seg*** - Segmentation Procedure

## KEYWORDS

vector, segmentation, overlay, hierarchical order

## SYNOPSIS

**m.seg**
**m.seg help**
**m.seg** [-**i**] **input**=*name*[,*name*,...] **output**=*name* [**snap**=*float*] **ID**=*string*[,*string*,...] [**OUTPUT_ID**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-i**
: intersection overlay (default is union overlay)

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name[,name,...]*
: Input vector names (name separated by comma ','; in hierarchical order)

**output**=*name*
: Output vector name

**snap**=*float*
: Snap value in map units; default is 1

**ID**=*string[,string,...]*
: Column Name of the ID for each vector map (name separated by comma ','; in hierarchical order)

**OUTPUT_ID**=*string*
: ID column Name for the output vector; default is 'SELF_ID'

## DESCRIPTION

*m.seg* allows the user to overlay several input layers in order to create one overlayed layer. The input layers are the landscape objects layers (reachs, plots, subwatersheds, soil horizons). The output layer is a combined layer of these inputs, it contains two attribute tables (the first table contains informations for the areal units and the second for the linear units). After using *m.seg*, it is recommended to clean the layer with m.dispolygseg, m.sliverpolygseg scripts.

## NOTES

For additional informations concerning the overlay procedure, please read *Lagacherie, P. & al, 1996* .

*m.seg* script is able to overlay more than two vectors (and with no number limits); polygon and line vectors are accepted (but no point vectors). Vector topology must be correct to avoid geometry errors. The user must give a hierarchical order for each vector (See *Examples* for details)

Using flag *-I* allows the user to make intersection operation rather than union operation (See *Examples* to see differences between these operations)

The output vector will be a polygon vector with two attribute tables (two layers in GRASS vocabulary): layer 1 contains the column *OUTPUT_ID* ( column name to identify each output entity) and, for each input polygon vector, a column named *"ID_"* with hierarchical number (See *Examples* for details). Layer 2 contains information from the input line vector: for each input line vector, a column named *"ID_"* with hierarchical number (See *Examples* for details).

## EXAMPLES

## Segmentation procedure

The input vectors are the following in hierarchical order:

1- plots (polygon vector) : ID column name: plot_id

2- subwatersheds (polygon vector) : ID column name: subw_id

3- reachs (line vector) : ID column name: reach_id

4- soils (polygon vector) : ID columns name: soil_id

Creating seg1 output vector :

```
GRASS 6.3.0 :~ > m.seg input=plot,subwatersheds,reachs,soils output=seg1
ID=plot_id,subw_id,reach_id,soil_id OUTPUT_ID=IDENS
```

Showing columns in layer 1 (columns of the input polygon vectors):

```
GRASS 6.3.0 :~ > db.describe -c table=seg1_1
ncols: 5
nrows: 607
Column 1: cat:INTEGER:11
Column 2: plot_id1:INTEGER:11
Column 3: subw_id2:INTEGER:11
Column 4: sil_id4:INTEGER:11
Column 5: IDENS:INTEGER:11
```

Showing columns in layer 2 (columns of the input line vector):

```
GRASS 6.3.0 :~ > db.describe -c table=seg1_2
ncols: 2
nrows: 2456
Column 1: cat:INTEGER:11
Column 2: reachs_id3:INTEGER:11
```

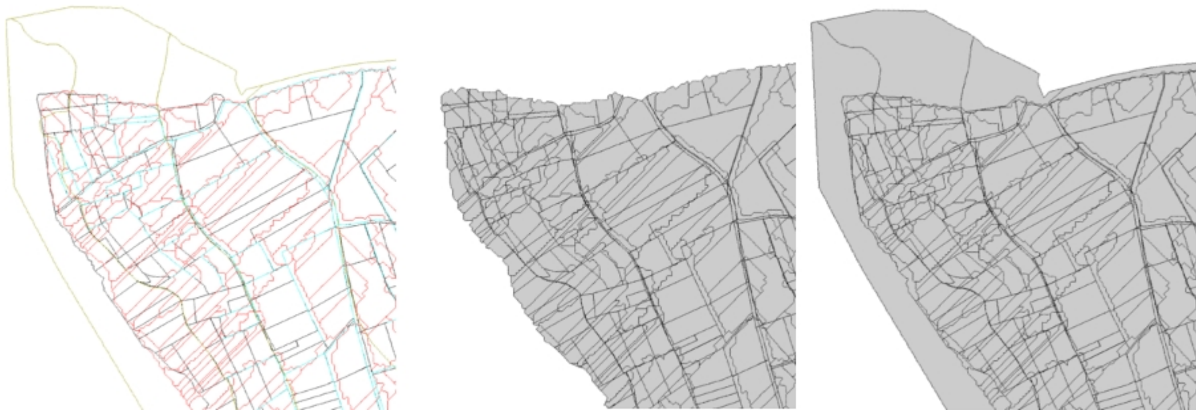
## Difference in segmentation procedure with *flag -i* or without

Creating seg1 output vector with *flag -i* (intersection operation):

```
GRASS 6.3.0 :~ > m.seg -i input=plot,subwatersheds,reachs,soils output=seg1
ID=plot_id,subw_id,reach_id,soil_id
```

Creating seg2 output vector without *flag -i* (union operation):

```
GRASS 6.3.0 :~ > m.seg input=plot,subwatersheds,reachs,soils output=seg2
ID=plot_id,subw_id,reach_id,soil_id
```



*Left: input vectors (black: plot boundaries; red: subwatershed boundaries; blue: reachs; brown: soil boundaries) ; Center: output vector seg1 (intersection operation); Right: output seg2 vector (union operation)*

## SEE ALSO

m.colseg, m.extractlineseg, m.dispolygseg, m.sliverpolygseg, m.toporeach, m.toposu

## REFERENCES

Lagacherie, P., Moussa, R., Cormary, D., AND Molenat, J. 1996. Effect of DEM data source and sampling pattern on topographical parameters and on a topography-based hydrological model. *In* "HYDROGIS'96. Application of Geographic Information System in Hydrology and Water Resources Management" (K. Kovar and H. P. Nachtnebel, eds.), pp. 191-200. IAHS, Vienna.

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

Main index

## NAME

***m.sliverpolygseg*** - dissolving sliver entities with hierarchical level after segmentation (use Gravelius Index)

## KEYWORDS

vector, selective dissolve, areal, hierarchical order, sliver

## SYNOPSIS

**m.sliverpolygseg**
**m.sliverpolygseg help**
**m.sliverpolygseg** [-**i**] **input**=*name* **output**=*name* **index**=*float* [**unit**=*string*] [**columnp**=*string*] [**valuep**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

### Flags:

**-i**
    print only statistics (no calcul) and exit
**--overwrite**
    Allow output files to overwrite existing files
**--verbose**
    Verbose module output
**--quiet**
    Quiet module output

### Parameters:

**input**=*name*
    Input vector name
**output**=*name*
    Output vector name
**index**=*float*
    gravelius INDEX max value; (in general, features with index > 1.6 are considered sliver features)
**unit**=*string*
    units (meters (me), kilometers (k), acres(a), hectares(h))
    Options: *me,k,a,h*
    Default: *me*
**columnp**=*string*
    Name of column used to protect particular entities (column must exist)
**valuep**=*string*

value used in 'columnp' to identify the particular entities

## DESCRIPTION

*m.sliverpolygseg* allows the user to dissolve sliver areal entities (using Gravelius Index) with user's threshold (*index* parameter) and with hierarchical order. The input layer must be created by m.seg script. Flag i prints only statistics (see *Examples* for details). *Columnp* and *valuep* can be used to treat separately particular entities (e.g. road entities can be dissolved only with road entities), see *Examples* for details. To treat particular entities, m.colseg script must be used to patch attribute column to identify entities.

## NOTES

Sliver areal entities are dissolved to the neighouring entity with whom they share the longest boundary and whom have the lowest hierarchical order (selective cleaning procedure). Hierarchical order is defined by m.seg script and is provided by the attribute table(s)s of the input vector data. The dissolving process is iterative (process is stopped when no more small entity is found). Sliver areal entities superior to threshold but with no neighbours are preserved.

## EXAMPLES

### Print statistics for dissolving sliver areal entities for the seg1 vector map (threshold is 2)

```
GRASS 6.3.0 :~ > m.sliverpolygseg -i input=seg1 output=seg2 index=2 unit=me
You choose a maximum gravelius index of 2
 For the vector seg1,
 the dissolving small area entities operation will work
 on 6 polygons for 28 total polygons
 and will represent 4.3 % of the total surface
```

### Dissolving sliver areal entities

Dissolving sliver areal entities for the seg1 vector map (threshold is 2)

```
GRASS 6.3.0 :~ > m.sliverpolygseg input=seg1 output=seg2 index=2 unit=me
```

### Particular entities treatment

Dissolving sliver areal entities for the seg1 vector map (threshold is 2) and separate treatment for particular entities defined by the column *'type'* and the value *'road'* . Using m.colseg script to patch attribute columns from field vector input data.

```
GRASS 6.3.0 :~ > m.colseg input=seg1 output=seg1p polygon=field columns=type
GRASS 6.3.0 :~ > m.sliverpolygseg input=seg1p output=seg2 index=2 unit=me
columnp=type valuep=road
```

## SEE ALSO

m.seg, m.dispolygseg, m.dispolyg, m.colseg

## REFERENCES

Gravelius, H. (1914) GrundriB der gesamten Gewasserkunde, Band 1: FluBkunde (Compendium of Hydrology, vol. 1: Rivers, in German). Goschen, Berlin, Germany.

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed:* 21 July 2010

---

## NAME

***m.toporeach*** - Topology calculation for reach segments

## KEYWORDS

vector, topology, linear units, MHYDAS

## SYNOPSIS

**m.toporeach**
**m.toporeach help**
**m.toporeach** [-**c**] **input**=*name* **output**=*name* [**gu**=*name*] **ID**=*string* [**IDGU**=*string*] [**colwidth**=*string*] [**colheight**=*string*] **dem**=*name* [**outlet**=*float*] [**slop_val**=*float*] [**ID_OUT**=*string*] [**UPST_OUT**=*string*] [**DNST_OUT**=*string*] [**LOW_OUT**=*string*] [**LEN_OUT**=*string*] [**WID_OUT**=*string*] [**HEIG_OUT**=*string*] [**SLOPE_OUT**=*string*] [**PCSSORD_OUT**=*string*] [**COMMENT_OUT**=*string*] [**GUID**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-c**
: use the option *outlet* to find the outlet segment (must use the column 'outlet'); otherwise DEM is used

**--overwrite**
: Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name*
: Input line vector name

**output**=*name*
: Output line vector name

**gu**=*name*
: Input ground water vector name

**ID**=*string*
: ID reach column name

**IDGU**=*string*
: ID Ground Water column name

**colwidth**=*string*

Width reach column name

**colheight**=*string*
: Height reach column name

**dem**=*name*
: Input DEM

**outlet**=*float*
: ID value of the outlet segment

**slop_val**=*float*
: Replacement value for null or negative calculated slope (must be > 0; default value is 0.0001)

**ID_OUT**=*string*
: id OUTPUT column name
: Default: *SELF_ID*

**UPST_OUT**=*string*
: UP Node OUTPUT column name
: Default: *UPST_NOD*

**DNST_OUT**=*string*
: DOWN Node OUTPUT column name
: Default: *DNST_NOD*

**LOW_OUT**=*string*
: Low RS id OUTPUT column name
: Default: *LORCH_ID*

**LEN_OUT**=*string*
: Length OUTPUT column name
: Default: *USR_LEN*

**WID_OUT**=*string*
: Width OUTPUT column name
: Default: *USR_WID*

**HEIG_OUT**=*string*
: Height OUTPUT column name
: Default: *USR_HEIG*

**SLOPE_OUT**=*string*
: Slope OUTPUT column name
: Default: *USR_SLOP*

**PCSSORD_OUT**=*string*
: Process Order OUTPUT column name
: Default: *PCSS_ORD*

**COMMENT_OUT**=*string*
: Commentary OUTPUT column name
: Default: *COMMENT*

**GUID_OUT**=*string*
: ID GU OUTPUT column name
: Default: *EXHGW_ID*

## DESCRIPTION

*m.toporeach* allows the user to calculate oriented topology of linear units (RS MHYDAS). Linear network can optionally be created by the *m.seg* and *m.extractlineseg* scripts.

## NOTES

### Input and output layers

The input layer can be created by *m.extractlineseg*, but *m.toporeach* script can also accept simple line layer (just be sure that in that case layer geometry and topology are correct; use *m.network* script to verify). The output layer will contain the following columns (columns are created during this process):

- *$ID_OUT* : unique Object ID
- *$UPST_OUT* : UpNode ID
- *$DNST_OUT* : DownNode ID (m/m)
- *$LOW_OUT* : Down Hydro Object ID
- *$LEN_OUT* : Length (m)
- *$WID_OUT* : Width (m)
- *$HEIG_OUT* : Height (m)
- *SLOPE_OUT* : Slope (m/m)
- *$PCSSORD_OUT* : Object Process Order
- *$GUID_OUT* : ID unit for underground exchanges
- *$COMMENT_OUT* : Commentary

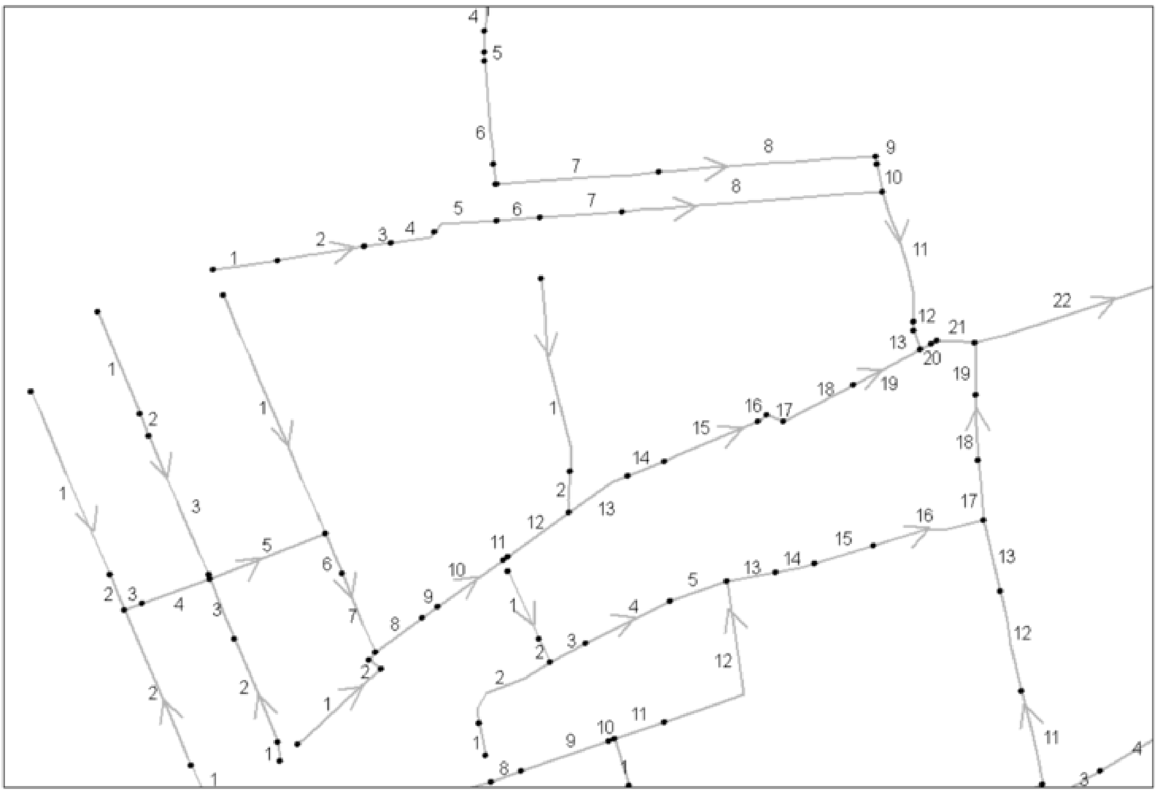
The segment outlet can be provided by user with flag *-c* and option *outlet* or automatically by DEM (algorithm will find the dangle segment which have the node with the lowest elevation). Topology with GroundWater units can be processed with *GU* and *IDGU* options. If input layer have width and height segment information, they can be provided using *colwidth* and *colheight* options; these attributes will be transfered on the output layer.
If the user wants to have xml files or fluidx files for the OpenFluid version 1.5 or superior, please use *m.definput* script to create these files.
Calculated slopes are always positive (no negative or null value); for negative or null slopes, user can manage its own value by *slop_val* option (default value is 0.0001).

## EXAMPLES

## Oriented topology calculation for seg_reachs

Outlet segment provided by Raster DEM elevation. Width and height of the reachs are provided too.

```
GRASS 6.3.0 :~ > m.toporeach input=seg_reachs output=seg_reachs2 ID=SELF_ID
dem=elevation colwidth=width colheight=height
```



*Oriented topology with Process Order labels*

## Oriented topology calculation with user's outlet provided

```
GRASS 6.3.0 :~ > m.toporeach -c input=seg_reachs output=seg_reachs2 ID=SELF_ID
dem=elevation outlet=18
```

## Oriented topology calculation with Ground Water units provided

```
GRASS 6.3.0 :~ > m.toporeach -c input=seg_reachs output=seg_reachs2 ID=SELF_ID
dem=elevation outlet=18 GU=GroundW IDGU=Ident
```

## SEE ALSO

m.seg, m.extractlineseg, m.toposu, m.definput, m.network

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed: 23* January 2012

Main index

## NAME

***m.toposu*** - Topology calculation for surface units

## KEYWORDS

vector, topology, areal units, MHYDAS

## SYNOPSIS

**m.toposu**
**m.toposu help**
**m.toposu** [-**canbsldz**] **input**=*name* **output**=*name* **dem**=*name* [**hydro**=*name*] [**inputman**=*name*] [**gu**=*name*] **ID**=*string* [**IDHYDRO**=*string*] [**POHYDRO**=*string*] [**IDGU**=*string*] [**step**=*float*] [**distreach**=*float*] [**slop_val**=*float*] [**colz**=*string*] [**ID_OUT**=*string*] [**AREA_OUT**=*string*] [**SLOPE_OUT**=*string*] [**FCDE_OUT**=*string*] [**FID_OUT**=*string*] [**FDIST_OUT**=*string*] [**PCSSORD_OUT**=*string*] [**COMMENT_OUT**=*string*] [**GUID_OUT**=*string*] [--**overwrite**] [--**verbose**] [--**quiet**]

## Flags:

**-c**
calculate altitude with one pixel on centroid for linear feature; default (no flag c) is mean feature altitude on centroid

**-a**
calculate altitude with one pixel on centroid for areal feature; default (no flag a and no flag n) is mean feature altitude on centroid

**-n**
calculate altitude with D-8-like neighbours on centroid for areal feature; default (no flag a and no flag n) is mean feature altitude on centroid

**-b**
neighbouring research with feature boundaries and one contact point ; default (no flag b) is classical (D8 modified) neighbouring research

**-s**
down neighbour choice by slope; default (no flag s) is down neighbour choice by difference altitude

**-l**
Treatment of the topology loops; default is no

**-d**
create output_dir layer (line topological direction layer between surface units centroids)

**-z**
join z attribute from INPUT for areal feature (use colz column);default (no flag a, no flag n and no flag z) is mean feature altitude on centroid

**--overwrite**

Allow output files to overwrite existing files

**--verbose**
: Verbose module output

**--quiet**
: Quiet module output

## Parameters:

**input**=*name*
: Input polygon vector name

**output**=*name*
: Output polygon vector name

**dem**=*name*
: Input DEM name

**hydro**=*name*
: Input line hydrologic vector name

**inputman**=*name*
: Input line direction vector name (manual topology)

**gu**=*name*
: Input ground water vector name

**ID**=*string*
: ID INPUT column name

**IDHYDRO**=*string*
: ID HYDRO column name

**POHYDRO**=*string*
: Column name of the Process Order of HYDRO objects

**IDGU**=*string*
: ID Ground Water column name

**step**=*float*
: distance (in meters) between each slope direction inside a SU (boundaries and one contact point topology); needed for flag b

**distreach**=*float*
: distance in meters for a possible contact between a SU and a reach

**slop_val**=*float*
: Replacement value for null or negative calculated slope (must be > 0; default value is 0.0001)

**colz**=*string*
: id z attribute column name (for flag z)

**ID_OUT**=*string*
: id OUTPUT column name
: Default: *SELF_ID*

**AREA_OUT**=*string*
: AREA OUTPUT column name
: Default: *USR_AREA*

**SLOPE_OUT**=*string*
: SLOPE OUTPUT column name
: Default: *USR_SLOP*

**FCDE_OUT**=*string*
: Flow Code OUTPUT column name
: Default: *FLOW_CDE*

**FID_OUT**=*string*
: Flow ID OUTPUT column name
: Default: *FLOW_ID*

**FDIST_OUT**=*string*
: Flow Distance OUTPUT column name

Default: *FLOW_DST*

**PCSSORD_OUT**=*string*
: Process Order OUTPUT column name
Default: *PCSS_ORD*

**COMMENT_OUT**=*string*
: Commentary OUTPUT column name
Default: *COMMENT*

**GUID_OUT**=*string*
: ID GU OUTPUT column name
Default: *EXHGW_ID*

## DESCRIPTION

*m.toposu* allows the user to calculate the oriented topology for an areal units layer. This topology is made for the MHYDAS model. Several options are possible, please read carefully the following paragraphs for choosing the right options for your calculations.

## NOTES

### Input and output layers

The input layer can be created by m.seg, but *m.toposu* script can also accept a simple polygon layer (just be sure that in that case layer geometry and topology are correct). The output layer will contain the following columns (columns are created during this process):

- *$ID_OUT* : uniq Object ID
- *$AREA_OUT* : Area (square meters)
- *$SLOPE_OUT* : Slope (m/m)
- *$FCDE_OUT* : Downstream feature type unit (R for reach segment and S for surface unit) for surface exchange
- *$FID_OUT* : ID unit for surface exchange
- *$FDIST_OUT* : Distance (m) between centroids of the unit and its downstream unit
- *$PCSSORD_OUT* : Object Process Order
- *$GUID_OUT* : ID unit for underground exchanges
- *$COMMENT_OUT* : Commentary

If the user wants to have xml files or fluidx files for the OpenFluid version 1.5 or superior, please use m.definput to create these files.
Flag *-d* allows the user to create the line topological direction layer which links surface units centroids (rooted topological tree); the layer will have *OUTPUT*_dir name.

Calculated slopes are always positive (no negative or null value); for negative or null slopes, user can manage its own value by *slop_val* option (default value is 0.0001).

*$FDIST_OUT* parameter is calculated with x,y and z coordinates of centroids of the linear or areal units (AND not only with x and y coordinates). If using Flag *-d*, differences can appear between *$FDIST_OUT* parameter and length of linear features of the topological direction layer because of taking account into the z coordinate of the points in *$FDIST_OUT* parameter calculation.

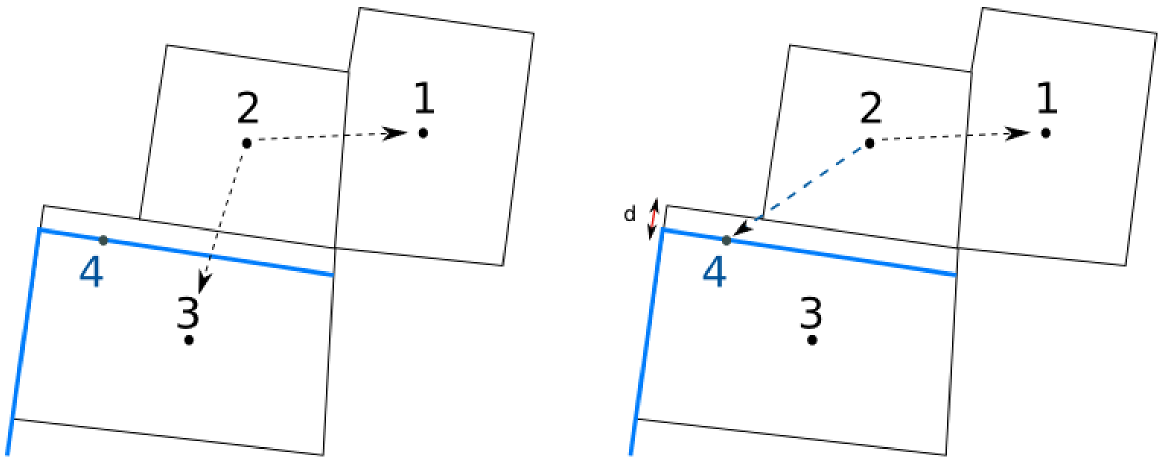
## Concerning reach network

The *HYDRO* layer is optional; if the user provides it, the layer must be created by m.toporeach. The layer must contain the following columns which are needed for this procedure:

- *IDHYDRO* : unique Hydro object ID
- *POHYDRO* : Hydro object Process Order

**DISTREACH option :**

The user can set the *distreach* value which allows to have a buffer distance between areal unit boundaries and Hydro objects. Even if a Hydro object is not a right topological neighbour, the *distreach* allows to consider it like a neighbour.

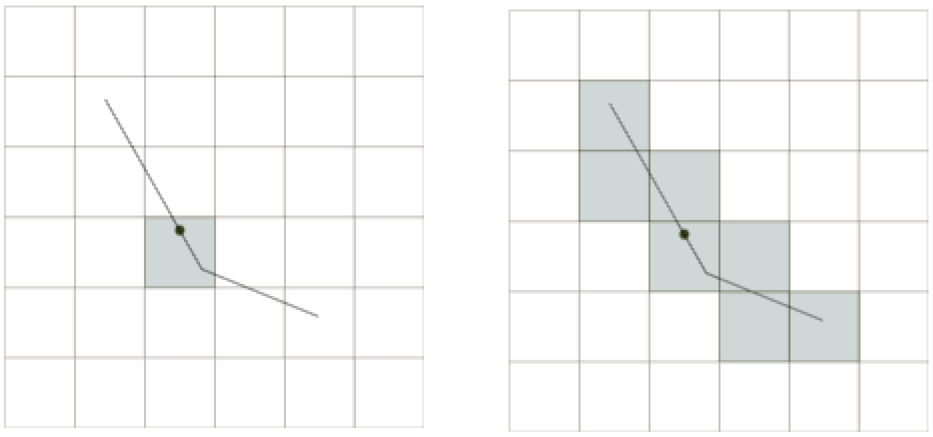


*Potential topological neighbours without distreach (left) ; Potential topological neighbours with distreach (right)*

On the left figure, without *distreach* option, the topological neighbours of the unit number 2 are the units 1 and 3. Even if the reach 4 is near the boundary of the unit 2, this reach will not be found by the algorithm because it is not a topological neighbour. On the right figure, if distance *d* (which represents the distance between reach segment and boundary of the unit 2) is smaller than the *distreach* value , reach 4 is considered by the algorithm as a topological neighbour of unit 2. The option is usefull when the geometry of these linear objects don't coincide exactly with polygon boundaries (due to different digitalizations).

**Flag C :**

*m.toposu* needs to know reach segment elevation. Two options are possible to calculate these values with the help of DEM : if flag *c* is set , just the one pixel on centroid reach is used to know the elevation of reach. If this flag is not set, all the DEM pixels which are in contact with the whole reach are used to calculate the mean elevation of the reach.



*Flag c set: pixel in contact with reach centroid is only used (left) ; Flag c not set: pixels in contact whith whole reach are used (right)*

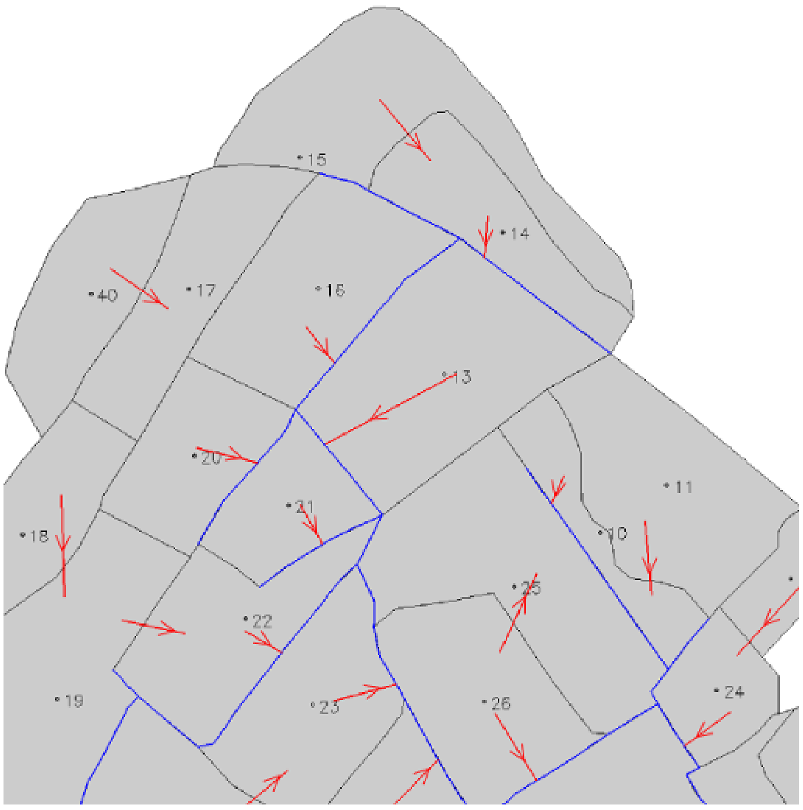
## Concerning groundwater layer

Groundwater layer is optional but if the user wants the underground-surface topology, *GU* and *IDGU* (unique ID Groundwater Object column name) must be provided. Topological relations are based on the following spatial analysis : for each SU centroid, algorithm indicates which GU polygon contains this point.

## Options for topology parametrization

**INPUTMAN option :**

This algorithm use the difference in elevation between SU to calculate oriented topology, which is based on DEM value. If the user doesn't want to use DEM value for the whole spatial domain or part of it, he can provided personal oriented topology preferences. If *INPUTMAN* layer is provided, the algorithm considers this layer in priority to create oriented topology. For SU which aren't concerned by *INPUTMAN* orientation, the DEM value is used to find the downstream neighbour. With this option, the user can have : a whole automatized oriented topology calculation with DEM (with no *INPUTMAN* layer provided), a hybrid procedure (with a manual topology for part of the spatial domain which is concerned by *INPUTMAN* orientation and with DEM values for the rest of the domain) or a whole manual procedure (if *INPUTMAN* orientations concerned the whole domain). *INPUTMAN* layer is a simple linear layer : to indicate orientation topology between two units (areal to areal or areal to linear), the user just "draws" a line which crosses the two units (see following figure for details); just be careful to the line orientations.



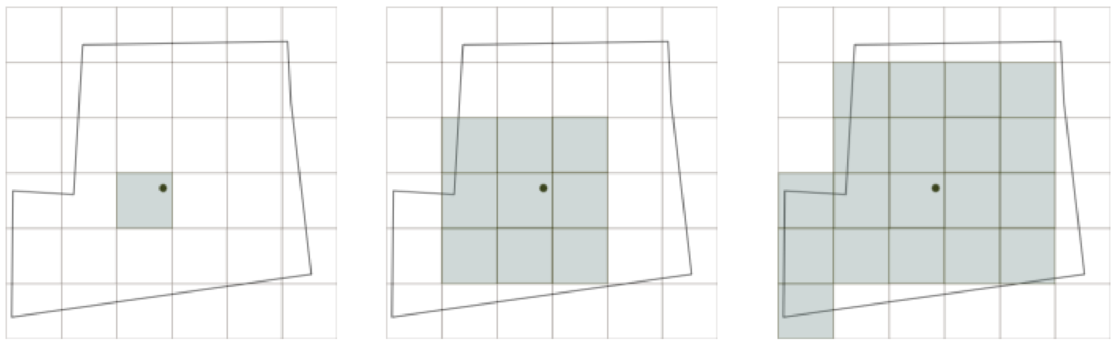
*INPUTMAN layer (in red arrows), SU layer (in grey color) and RS layer (in blue lines)*

*INPUTMAN* layer is represented by the red arrows : unit 15 has the unit 14 which is its downstream unit because a red arrow crosses these two units, and unit 14 go into the reach ( *INPUTMAN* is used with *distreach* value to find which reach segment is near the end of the arrow). Unit 17 has no arrow to find its downstream unit, so the DEM value will be used to find it.

**SU elevation using flags A, N or Z :**

Three choices to calculate SU elevation are possible (see following figure to detail) :

- Flag *A* is set : just the DEM pixel on contact with SU centroid is used to calculate SU elevation
- Flag *N* is set : the DEM pixel on contact with SU centroid and its 8 neighbours are used to calculate SU elevation
- Flag *Z* is set : using the *colz* option, the user can provided a vector with z value already stored; no altitude calculation
- No Flags set (default method) : the whole DEM pixels in contact with the SU are used to calculate the SU elevation

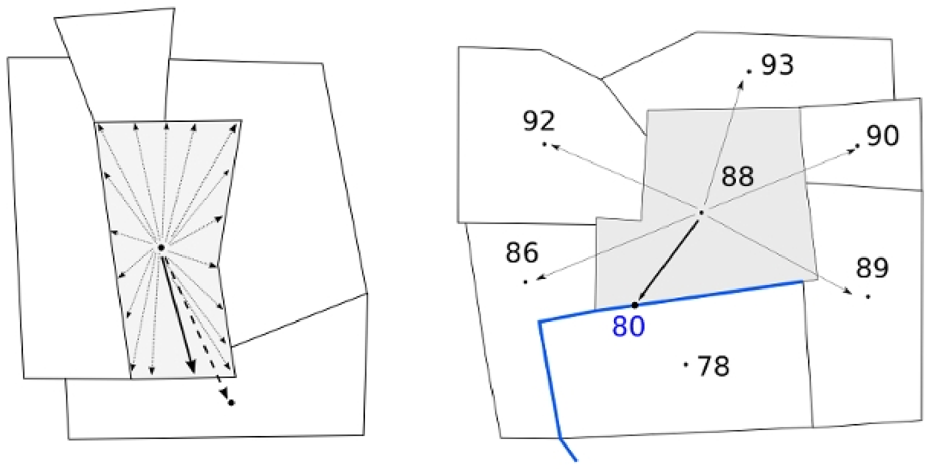


*Flag A set (left) ; Flag N set (middle) ; Flags A and N NOT set, default method (right)*

**Neighbouring research with Flag B :**

Two choices are possible to find downstream neighbours of a unit (See following figure for details):

- Flag *B* is set : For each unit, in a first step, the lowest point on unit boundary (in comparison to centroid elevation) is found, and in a second step, the algorithm searches which neighbouring unit (linear or areal) share this point. *step* option is used to create virtual points along boundaries (*step* value is the distance between each point). This method is quite slow, especially if *step* value is small.
- Flag *B* is NOT set (default method) : For each unit, the algorithm searches all neighbours (linear and areal) and choose the downstream unit which has the tallest elevation difference.



*Flag B set (left) ; Flags B NOT set, default method (right)*

On left figure, using Flag B, grey dashed arrows represent the whole potential directions, the black plain arrow the tallest elevation difference. The black dashed arrow shows the topology connection between the centroid of this unit and the centroid of its downstream unit. On right figure, when Flag B is not set, labels are elevation values : even if the tallest elevation difference could be between the grey unit and the unit with 78 m elevation; the chosen direction is between the grey unit and the blue reach segment (reach segment in this case has the priority ahead of this lowest neighbour unit)

**Downstream unit choice with Flag S :**

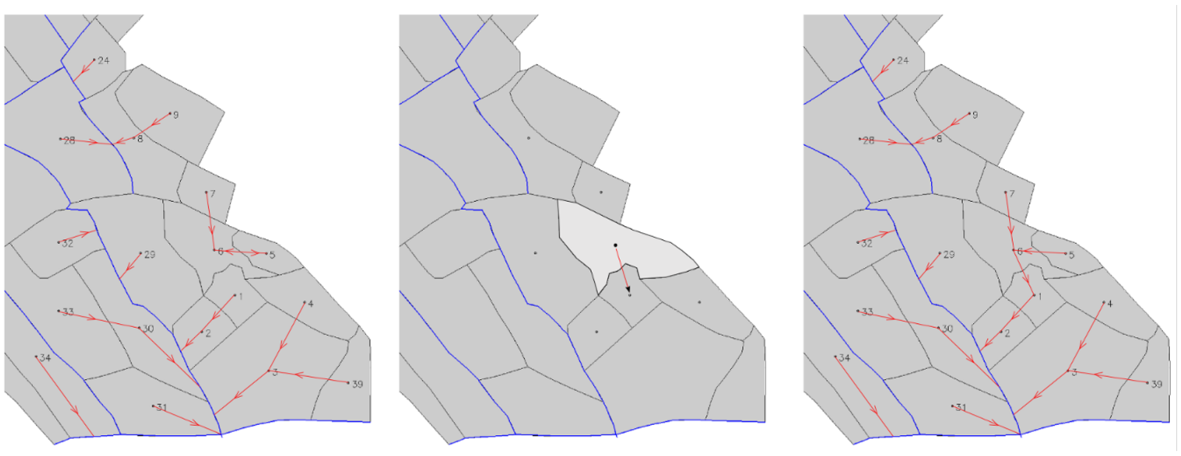
The downstream unit choice is made with two possibilities :

- Flag *S* is set : the slope between unit centroid (in case of Flag *B* no set) or between unit centroid and points on boundary (in case of Flag *B* is set) is calculated; and tallest slope is chosen to identify the downstream unit.
- Flag *S* is NOT set : downstream unit is chosen by calculating only the tallest elevation difference value (distance between points is not taken into account).

### Concerning loop treatment

Oriented topology relations need to build a perfect rooted tree. After calculting oriented topology for each areal unit, loops between units can appear (due to low quality DEM or reach network between units missed out) : these relations can be between two or more units (i.e. in the left following figure, the unit 6 has as downstream unit, the unit 5, and the unit 5 the unit 6). Using Flag *L* allows to treat these loops with the following steps :

1. Identify the clusters of areal units involved in loops (left following figure),
2. Merge these areal units into a single areal entity (middle following figure),
3. Select the neighboring unit of this entity with the greatest decrease in elevation,
4. Eliminate the loops by forcing the areal unit(s) of the cluster that share a boundary with the selected unit to flow in this unit (right following figure),
5. Repeat 1 through 4 until no loops remain.



*Topology before loop treatment (left) ; Step 2 of the loop treatment (middle); Step 4 of the loop treatment (right)*

Treated loops are known and, after treatment, if the same loop is met again, procedure is stopped with warning message for the user.

## EXAMPLES

### Using DEM value for whole spatial domain

```
GRASS 6.3.0 :~ > m.toposu -c -a -b -s -l input=surface_units output=surface_units2
directory=~/tmp/ dem=MNT hydro=reachs gu=groundW ID=SELF_ID IDHYDRO=IDH POHYDRO=POH
IDGU=IDG step=2 distreach=1
```

### Using input line direction vector, DEM value

```
GRASS 6.3.0 :~ > m.toposu -c -a -b -s -l input=surface_units output=surface_units2
dem=MNT hydro=reachs inputman=lineIN ID=SELF_ID IDHYDRO=IDH POHYDRO=POH step=2
distreach=1
```

## SEE ALSO

m.seg, m.toporeach, m.definput,

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed: 04* July 2011

---

[Main index]

## NAME

***m.definput*** - Create xml input files or fluidx files for OpenFluid Engine 1.5 or superior

## KEYWORDS

vector, OpenFluid, fluidx, xml

## SYNOPSIS

**m.definput**
**m.definput help**
**m.definput** [-**sx**] **input**=*name* **directory**=*string* **type**=*string* [**IDCOL**=*string*] [**TYPEDOWNCOL**=*string*] [**DOWNCOL**=*string*] [**GUCOL**=*string*] [**PCSSCOL**=*string*] [**COLUMN**=*string*[,*string*,...]] [**NAME**=*string*] [--**verbose**] [--**quiet**]

## Flags:

**-s**
  only show attribute columns of input and exit (no xml file creation)
**-x**
  create xml extension instead of default fluidx extension format
**--verbose**
  Verbose module output
**--quiet**
  Quiet module output

## Parameters:

**input**=*name*
  Input vector name
**directory**=*string*
  Name directory for the xml file(s)
**type**=*string*
  Unit type (SU, RS or GU)
  Options: *SU, RS, GU*
**IDCOL**=*string*
  Column name ID unit (default is SELF_ID)
**TYPEDOWNCOL**=*string*
  Column name for the TYPE downstream unit only if TYPE = SU (default is for SU TYPE : FLOW_CDE)
**DOWNCOL**=*string*
  Column name for the ID downstream unit (default is for SU TYPE : FLOW_ID, for RS

TYPE : LORCH_ID and for GU TYPE : EXHGW_ID)

**GUCOL**=*string*
: Column name for the Exchange GU connected to SU (default is EXHGW_ID)

**PCSSCOL**=*string*
: Column name for the Process Order (default is PCSS_ORD)

**COLUMN**=*string[,string,...]*
: Column names of attribut data for the Xxdefs.ddata.fluidx/xml (column name separated by comma ',')

**NAME**=*string*
: File name for XXddef.fluidx/xml and XXdefs.ddata.fluidx/xml (default is $TYPEddef.fluidx/xml and $TYPEdefs.ddata.fluidx/xml)

## DESCRIPTION

*m.definput* allows the user to create xml files or fluidx files for OpenFLUID version 1.5 or superior. Flag *s* allows to see column names for input file and exit (it's a front end of *v.info* command). The input layers must have been created for RS layer by *m.toporeach*, for SU layer by *m.toposu*; if not, input layers must contain compulsory columns (See *Notes* paragraph for details). Flag x allows to create xml files (for OpenFLUID 1.5) instead of fluidx files (for OpenFLUID 1.6)

## NOTES

If input layer is created by segmentation functions (*m.toporeach*, *m.toposu*), the following parameters are optional: *IDCOL*, *TYPEDOWNCOL*, *DOWNCOL*, *GUCOL* and *PCSSCOL*. If the user wants to create XXdefs.ddata.fluidx/xml file, *COLUMN* must be provided with column names desired. *NAME* parameter is optional; if not set, xml files will have a type name (RSddef.fluidx/xml and RSdefs.ddata.fluidx/xml for RS type, SUddef.fluidx/xml and SUdefs.ddata.fluidx/xml for SU type and GUddef.fluidx/xml and GUdefs.ddata.fluidx/xml for GU type).

## EXAMPLES

### Creation of xml file for SU layer

Creation of SUddef.xml

```
GRASS 6.3.0 :~ > m.definput -x input=SU directory=~/tmp/ type=SU
GRASS 6.3.0 :~ > cat ~/tmp/SUddef.xml
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>
    <definition>
      <unit class="SU" ID="1" pcsorder="1">
        <to class="SU" ID="2" />

      </unit>
      <unit class="SU" ID="2" pcsorder="2">
        <to class="RS" ID="1" />
      </unit>
      <unit class="SU" ID="3" pcsorder="1">
        <to class="RS" ID="2" />

      </unit>
      <unit class="RS" ID="1" pcsorder="1">
        <to class="RS" ID="2" />
      </unit>
      <unit class="RS" ID="2" pcsorder="1">
      </unit>

    </definition>
  </domain>
</openfluid>
```

Creation of SUddef.fluidx and SUdefs.ddata.fluidx

```
GRASS 6.3.0 :~ > m.definput input=SU directory=~/tmp/ type=SU COLUMN=ks,slope
```

## Creation of xml file for RS layer (not created by segmentation functions)

Showing column names

```
GRASS 6.3.0 :~ > m.definput -s input=RSmanual directory=~/tmp/ type=RS
Displaying column types/names for database connection of layer 1:
INTEGER|cat
INTEGER|Id
INTEGER|num_bief
CHARACTER|encombre_
CHARACTER|substrat
INTEGER|haut
INTEGER|bas
INTEGER|prof_
CHARACTER|infos
INTEGER|UpNode
INTEGER|DownNode
INTEGER|LowRS
DOUBLE PRECISION|LENGTH
INTEGER|ExchangeGU
INTEGER|ProcessOrd
DOUBLE PRECISION|Slope

exit

GRASS 6.3.0 :~ > m.definput -x input=RSmanual directory=~/tmp/ type=RS
IDCOL=num_bief DOWNCOL=LowRS GUCOL=ExchangeGU PCSSCOL=ProcessOrd NAME=reach
```

## SEE ALSO

m.toporeach, m.toposu

## AUTHORS

Michael Rabotin, UMR LISAH, Montpellier, France

rabotin@supagro.inra.fr

*Last Changed: 23* January 2012

---