Chapter 6

Map & Table Calculation

Note: This chapter is written as an enhancement to Chapter 6: Map & Table calculation in the ILWIS 2.1 Reference Guide. It does *not* completely replace the mentioned chapter: instead you are invited to use both as one.

6.1 Map calculation

6.1.1 Logical operators

Truth table of logical AND
The result of A and B is true only if both expressions A and B are true. If either expression A or expression B is true, false is returned. If A is true and B is undefined, or vice versa, then undefined is returned.

The truth table of the logical AND is:

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

Truth table of logical OR
The result of A or B is true if one or both of the expressions a and b is true.

The truth table of the logical OR is:

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

Truth table of logical XOR
The result of A xor B is true if only one of the expressions A or B is true. When both expressions are true or when both expressions are false, false is returned. When either expression A or B is undefined, undefined is returned.

The truth table of the logical XOR is:

XOR	T	F	?
T	F	T	?
F	T	F	?
?	?	?	?

Truth table of logical NOT
The result of not A is true if expression A is false. If expression A is true, false is returned. If expression A is undefined, undefined is returned.

The truth table of the logical NOT is:

NOT	
T	F
F	T
?	?

6.2 Table calculation

6.2.1 Aggregating values

Several 'functions' are available to aggregate values of a value column. Aggregation means that you get one aggregate value, for instance the average or the sum, of a whole column, or one value per group of class names. In this way, you can for instance calculate the total area of each class.

The following aggregation 'functions' are available:

- average,
- count,
- minimum,
- median,
- maximum,
- predominant,
- standard deviation, and
- sum.

These functions can be used:

- to aggregate values of a *whole column* (using any aggregation function). You will obtain one output value; for all records, the same aggregation answer appears.
- to aggregate values of a column per group, i.e. aggregate values by the classes or ID of another column (using any aggregation function): use a 'group by' column. For all records that have the same class or ID in the selected 'group by' column, the same aggregation answer will appear. The 'group by' column is usually a column with a class, or ID domain.

- to aggregate values while taking into account weights: use a weight column. In this way, you can calculate weighted averages, etc. The weight column is a column with a value domain.
  - When you do not select a 'group by' column, you will obtain one output value.
  - When you do select a 'group by' column, you will obtain answers per group.

Aggregation results can be written either:
- into the same table, in a new column, or
- in a new table, in a new column (only possible through the menu), or
- in another existing table, in a new column.

Aggregations can be performed by choosing the Aggregation from the Columns menu in a table window, or by typing an expression on the command line of a table window.

Aggregation through a dialog box
Generally, you will aggregate columns via the menu in the table window. To aggregate the values of a column:
- In a table window which contains the column which values you want to aggregate, open the Columns menu and choose the Aggregation command. The Aggregate Column dialog box appears.
- Fill out the Aggregate Column dialog box:
  - Select the value column that contains the data you want to aggregate.
  - Select the aggregation function that you want to use: Average, Count, Maximum, Median, Minimum, Predominant, Standard Deviation or Sum.
  - If you want to aggregate values by classes or IDs in a Group By column, select the Group By check box, and select the 'group by' column; else the values of the complete column will be aggregated.
  - If you want to use weight values during the aggregation, select the Weight check box and select the weight column; else all records are treated equally.
  - If you want the aggregation results written into another table; select the Output Table check box and type a table name; else the results will be written into a column in the current table.
  - Type a name for the output column that will contain the results of the aggregation.

For examples of aggregations, click the links in the text on aggregations from the command line as given below.

Aggregation through the command line
To aggregate the values of a column you may also type a statement on the command line of a table window.

The syntax for aggregation on the command line is:
`ColumnAggregateAvg(`*`col`*`)`

calculates the average value of values in column *col*

`ColumnAggregateAvg(`*`col, g`*`)`
: calculates the average value of values in column *col* per group *g*

`ColumnAggregateAvg(`*`col, g, w`*`)`
: calculates the average value of values in column *col* per group *g* using weights *w*

`ColumnAggregateAvg(`*`col , , w`*`)`
: calculates the average value of values in column *col* using weights *w*

`ColumnAggregateCnt(`*`col, g`*`)`
: counts the number of times that column *col* is not undefined, optionally per group *g*

`ColumnAggregateMax(`*`col, g`*`)`
: determines the maximum value of column *col*, optionally per group *g*

`ColumnAggregateMed(`*`col, g, w`*`)`
: calculates the median value of column *col*, optionally per group *g*, and optionally using weights *w*

`ColumnAggregateMin(`*`col, g`*`)`
: determines the minimum value of column *col*, optionally per group *g*

`ColumnAggregatePrd(`*`col, g, w`*`)`
: determines the predominant value of column *col*, optionally per group *g*, and optionally using weights *w*

`ColumnAggregateStd(`*`col, g, w`*`)`
: calculates the standard deviation of column *col*, optionally per group *g*, and optionally using weights *w*

`ColumnAggregateSum(`*`col, g`*`)`
: calculates the sum of column *col*, optionally per group *g*

Notes

- The use of a group column *g* and/or a weight column *w* is optional:
  - Aggregations with parameters (*col*) return one aggregation result for all entries of column *col*.
  - Aggregations with parameters (*col, g*) return one aggregation result for all entries with the same class name or ID in group column *g*.
  - Aggregations with parameters (*col, g, w*) return one weighted aggregation result for all entries with the same class name or ID in group column *g*.
  - Aggregations with parameters (*col , , w*) return one weighted aggregation result for all entries of column *col*.
- Parameter *col* refers to nothing else than a column name. This means that within the brackets no other expressions can be used.
- Parameter *g* is a column with a class or ID domain; parameter *w* is a column with a value domain.
- For ILWIS 1.41 users: argument *g* can be considered as the key column.

Aliases

Instead of the long '`ColumnAggregate`*`AggFunc`*', you can also use the following aliases (the links provide examples):

`AGGAVG(`*`col, g, w`*`)`	calculates the average value of *col*, optionally per group *g* ,and optionally using weights *w*
`AGGCNT(`*`col, g`*`)`	counts the number of times that column *col* is not undefined, optionally per group *g*
`AGGMAX(`*`col, g`*`)`	determines the maximum value of *col*, optionally per group *g*
`AGGMED(`*`col, g, w`*`)`	calculates the median value of *col*, optionally per group *g*, and optionally using weights *w*
`AGGMIN(`*`col, g`*`)`	determines the minimum value of *col*, optionally per group *g*
`AGGPRD(`*`col, g, w`*`)`	determines the predominant value of *col*, optionally per group *g*, and optionally using weights *w*
`AGGSTD(`*`col, g, w`*`)`	calculates the standard deviation of *col*, optionally per group *g*, and optionally using weights *w*
`AGGSUM(`*`col, g`*`)`	calculates the sum of *col*, optionally per group *g*

Example 1
Two simple aggregations are:

```
Avg1 = AGGAVG(Area)
Avg2 = AGGAVG(Area,Landuse)
```

Parcel	Landuse	Area	Avg1	Avg2
00123	Residential	4000	10000	5000
00124	Residential	3500	10000	5000
00125	Commercial	17500	10000	17500
00126	Residential	7500	10000	5000
00127	Industrial	20000	10000	20000
01272	Institutional	12500	10000	12500
04625	Residential	5000	10000	5000

Column `Avg1` contains the average of the area of all parcels.

Column `Avg2` contains the averages of parcel areas per land use class: Residential, Commercial, Industrial and Institutional. For class Residential:
(4000+3500+7500+5000) / 4 = 5000

Example 2 (advanced)
When you are currently in one table (Province) and you want to retrieve aggregated values from another table (Municip), you may use:

```
Population = AGGSUM(Municip.pop, municip.prov)
```

Province.tbt		**Municip.tbt**		
Dom. Province	any info	Dom. Municip	Population	Province
Prov1	1000	Municip1	9920	Prov1
Prov2	2000	Municip2	4131	Prov1
Prov3	5000	Municip3	2161	Prov2
Prov4	4000	Municip4	4918	Prov1
Prov5	2000	Municip5	10461	Prov3
....		....		

Province.tbt after aggregation

Dom. Province	any info	Population
Prov1	1000	18969
Prov2	2000	2161
Prov3	5000	10461
Prov4	4000	?
Prov5	2000	?
...		

This expression will only work when the GroupBy column (Municip.prov) has the same domain as the current table (Province). It is usually easier to perform a join operation through the Join dialog box. This will give the same result as the expression above, and is even more flexible. For more information, see Table calculation : Join columns.

Example 3 (advanced)
To write output values from a current table (Parcel) into in a new column of another existing table (Landuse), use the following syntax:

```
Landuse.AvgParcelSize = AGGAVG(Parcel.Area, Parcel.Landuse)
```

Parcel.tbt

Dom. Parcel	Landuse	Area
00123	Residential	4000
00124	Residential	3500
00125	Commercial	17500
00126	Residential	7500
00127	Industrial	20000
01272	Institutional	12500
04625	Residential	5000
...		

Landuse.tbt

Dom. Landuse	any info
Residential	1000
Commercial	2000
Industrial	5000
Institutional	4000
...	

Landuse.tbt after aggregation

Dom. Landuse	any info	AvgParcelSize
Residential	1000	5000
Commercial	2000	17500
Industrial	5000	20000
Institutional	4000	12500
...		

The average of `Area` in table `Parcel` grouped by Landuse, is written into column `AvgParcelSize` in table `Landuse`.

When you want to retrieve for instance area values from a histogram, you have to specify the extension of the histogram as *tablename.ext.column*.

6.2.2 Joining columns of other tables

The join operation enables you to read a column from a second table and join it into the current table. To do so, you need a link between the tables. This link is made via the domain of the tables or via the domain of columns in the tables. When a column is used to make the link, this column is called a *key column*. When the domain of a table is used to make the link, you do not have to specify it.

Joining two tables use the same domain
Do not specify key columns. The tables have a one to one relation.

Joining column in the current table uses the same domain as the second table
Only specify a key column from the current table, i.e. key1 or the *Key column*. The relation between the column of the current table and the second table is a many to one relation.

Joining current table uses the same domain as a column in the second table
Only specify a key column from the second table, i.e. key2:

- when the classes or IDs in the key column of the second table are not unique (i.e. one to many relation), then the values that you want to join need to be aggregated and you have to specify key2 as the *Group By column* by which the values to be joined will be grouped during aggregation. Optionally, you can select a column from the second table as a weight column.
- when the classes or IDs in the key column of the second table are unique (i.e. one to one relation): specify key2 as the *Via Key column*.

Joining column in the current table and second table use the same domain
Specify a key column from the current table, i.e. key1, as well as a key column from the second table, i.e. key2:

- when the classes or IDs in the key column of the second table are not unique (i.e. many to many relation), then the values that you want to join need to be aggregated and you have to specify key2 as the *Group By column* by which the values to be joined will be grouped during aggregation. Optionally, you can select a column from the second table as a weight column.
- when the classes or IDs in the key column of the second table are unique (i.e. many to one relation): specify key2 as the *Via Key column*.

Notes

- When you need to specify a key column from the first table, i.e. key1; this is called the Key column.
- When you need to specify a key column from the second table, i.e. key2, you can either do this by specifying a Group By column (you will aggregate the values to be joined) or by specifying a Via Key column.
- Links between tables are always through class or ID domains.
- For more information on column aggregations, refer to Table calculation : aggregating values.

Joining through a dialog box
Generally, you will join columns via the menu in the table window. To join a column from another table into the current table:

- In a table window which displays the table into which you want to join one or more columns of another table, open the Columns menu and choose the Join command. The Join Column dialog box appears.
- Fill out the Join Column dialog box:
  - Select the table from which you want to join a column into the current table.

- Select a column from the second table, i.e. the column, which you want to join into the current table.
- If necessary, select the Key check box and fill out a column name of the current table (key1) which will be used to make a link to the second table; else deselect the Key check box.
- If necessary, select the Aggregation check box and select a column from the second table for the Group By column (key2). Furthermore, select an aggregation function and, optionally, select a weight column from the second table which contains the weight values to be used during the aggregation. Otherwise, fill out a column name from the second table as the Via Key (key2 and no aggregation).
- Type the name of the output column that will contain the joined values.

Joining through the command line
To join a column from another table into the current table you may also type a statement on the command line of a table window.

The syntax of the join operation on the command line is:

(1)	OutColName =	`ColumnJoin`(TableName, ColumnName)
(2)	OutColName =	`ColumnJoin`(TableName, ColumnName, Key1)
(3)	OutColName =	`ColumnJoin2ndKey`(TableName, ColumnName, ViaKey)
(4)	OutColName =	`ColumnJoin2ndKey`(TableName, ColumnName, Key1, ViaKey)
(5)	OutColName =	`ColumnJoin`*`AggFunc`*(TableName, ColumnName, GroupBy)
(6)	OutColName =	`ColumnJoin`*`AggFunc`*(TableName, ColumnName, GroupBy, Weight)
(7)	OutColName =	`ColumnJoin`*`AggFunc`*(TableName, ColumnName, GroupBy, Weight, Key1)

where:

OutColName	is the output column name. Usually, this is the same as the column name that was chosen to be joined into the current table.
`ColumnJoin`	is the command to start the Join operation.
`ColumnJoin2ndKey`	is the command to start the Join operation using a column from the second table to make a link to the current table.
TableName	is the name of the second table from which you want to join a column into the current table.
ColumnName	is a column name from the second table, i.e. the column that you want to join into the current table.
Key1	is a column from the current table; the domain of this column is either the same as the domain of the second table, or as the domain of a column in the second table.
ViaKey	is a column from the second table in which the classes or IDs are unique; the domain of this column is either also the domain of the current table or the same as the domain of a column in the current table. No aggregation will be performed.

`ColumnJoin`*`AggFunc`*
: is the command to start the Join operation with aggregation, i.e. joining while using an aggregation function. Type directly after `ColumnJoin` one of the following aggregation functions: `Avg` | `Cnt` | `Max` | `Med` | `Min` | `Prd` | `Std` | `Sum`. The *AggFunc* part should thus be replaced by one of the aggregation functions.

GroupBy
: is a column from the second table in which the classes or IDs are not unique; the values to be joined will be aggregated according to this GroupBy column.

Weight
: is an optional parameter to specify a column from the second table which contains weight values for the aggregation.

Formula 1 represents the case where both tables have the same domain.
Formula 2 represents the case where a *column* of the current table has the same domain as the second table (i.e. many to one relation).
Formula 3 represents the case where the current table uses the same domain as a *column* in the second table and when the classes or IDs in that column in the second table are unique (one to one relation).
Formula 4 represents the case where a *column* in the current table uses the same domain as a *column* in the second table and when the classes or IDs in that column in the second table are unique (many to one relation).
Formula 5 represents the case where the current table uses the same domain as a column in the second table and when the classes or IDs in that *column* in the second table are not unique (one to many relation). The values will be aggregated during the join.
Formula 6 represents the case where the current table uses the same domain as a *column* in the second table and when the classes or IDs in that column in the second table are not unique (one to many relation). The values will be aggregated during the join while using a weight column.
Formula 7 represents the case where a *column* in the current table uses the same domain as a *column* in the second table and when the classes or IDs in that column in the second table are not unique (many to many relation). The values will be aggregated during the join while using a weight column.

Example 1
The domain of the current table is the same as the domain of another table.

Table Landuse contains landuse classes and also lists the commercial value of these landuse classes.

The (raster or polygon) histogram of the landuse map contains the area of each landuse class.
The column Area from the histogram will be joined into attribute table Landuse.

Landuse.tbt		**Landuse.his/.hsa**		**Landuse.tbt after joining**		
Dom Landuse	Commval	Dom Landuse	Area	Dom Landuse	Commval	Area
Residential	1000	Residential	9920800	Residential	1000	9920800
Commercial	2000	Commercial	4131200	Commercial	2000	4131200
Industrial	5000	Industrial	2161600	Industrial	5000	2161600
Institutional	4000	Institutional	4918400	Institutional	4000	4918400
Agricultural	2000	Agricultural	10461600	Agricultural	2000	10461600

Open table Landuse as the current table. From the Columns menu in the table, choose Join.
In the Join Columns dialog box:

- Select histogram `Landuse` as the second table,
- Select column `Area` as the column you want to join into the current table,
- Type a name for the output column to contain the joined values, e.g. Area.

Or open table Landuse as the current table, and type the following expression on the command line of the table window:

```
 Area = ColumnJoin(Landuse.his.Area)
 Area = ColumnJoin(Landuse.hsa.Area)
```

With the first formula, the column `Area` from the raster histogram `Landuse.his` is joined into the attribute table.
With the second formula, the `Area` column from polygon histogram `Landuse.hsa` is joined into the attribute table.

Example 2
The domain of a column in the current table is the same as the domain of the second table.
Table Municip contains a number of municipalities, the population of each municipality and a column indicating whether the municipalities are considered large, medium or small (column MunClass).
Table MuniSubs contains information for large, medium and small municipalities. It contains a column Subsidy, which represents for instance expected subsidy figures for types of municipalities.
The subsidy figures in table MuniSubs will be joined into the Municipality table.

Municip.tbt

Dom. Municip	Population	MunClass
Municip1	99208	MunLarge
Municip2	41312	MunMedium
Municip3	21616	MunSmall
Municip4	49184	MunMedium
Municip5	104616	MunLarge
...		

MuniSubs.tbt

Dom.MunClass	Subsidy
MunSmall	1000
MunMedium	2000
MunLarge	5000
...	

Municip.tbt after joining

Dom. Municip	Population	MunClass	Subsidy
Municip1	99208	MunLarge	5000
Municip2	41312	MunMedium	2000
Municip3	21616	MunSmall	1000
Municip4	49184	MunMedium	2000
Municip5	104616	MunLarge	5000
...			

Open table Municip as the current table. From the Columns menu in the table, choose Join.

In the Join Columns dialog box:

- Select table `MuniSubs` as the second table,
- Select column `Subsidy` as the column to join into the current table,
- Select the Key check box, select column `MunClass`.
- Type a name for the output column to contain the joined values, e.g. Subsidy.

Or open table Municip as the current table, and type the following expression on the command line of the table window:

```
 Subsidy = ColumnJoin(RateMuni, Subsidy, MunClass)
```

Example 3

The domain of your current table is the same as the domain of a column in the second table.
Table Province lists for each province some information. The domain of this table is Province; the Province domain contains all provinces of a certain country.
Table Municip contains population figures for each municipality. Furthermore, for each municipality it is known in which province it is. The column Population in the table has a value domain. The column Province has domain Province.
Column Population in table Municip will be joined into the Province table. As a province usually contains more than one municipality, the municipal population figures need to be aggregated during the join.

Province.tbt

Dom. Province	any info
Prov1	1000
Prov2	2000
Prov3	5000
Prov4	4000
Prov5	2000
....	

Municip.tbt

Dom. Municip	Population	Province
Municip1	9920	Prov1
Municip2	4131	Prov1
Municip3	2161	Prov2
Municip4	4918	Prov1
Municip5	10461	Prov3
....		

Province.tbt after joining

Dom. Province	any info	Population
Prov1	1000	18969
Prov2	2000	2161
Prov3	5000	10461
Prov4	4000	?
Prov5	2000	?
....		

Open table Province as the current table. From the Columns menu in the table, choose Join.
In the Join Columns dialog box:

- Select table `Municip` as the second table,
- Select column `Population` as the column to join into the current table,
- The Aggregation check box is already selected:
  - select for the Aggregation function: Sum,
  - select for the GroupBy column: column Province,
- Type a name for the output column to contain the joined values, eg Population.

Or open table Province as the current table, and type the following expression on the command line of the table window:

```
Population = ColumnJoinSum(Municip, Pop, Prov)
```

Example 4
The domain of a column in the current table is the same as the domain of a column in the second table.
From the Columns menu in the table, choose Join. Choose the table name, which you want to use and the column to join into your current table. Then, select a key column in your current table and a column in the second table to group your data. Note that these two columns must have the same domain. You may choose to perform an aggregation on the values. The join will take place as described before.

6.2.3 Creating and running scripts (example)

By creating and applying a script, you can perform a series of ILWIS operations. This is comparable to using batch files in ILWIS version 1.4.

With a script, MapCalc and TabCalc expressions can be performed, and any ILWIS operation. Further, some extra commands are possible to show objects, or to perform some file management. For more information on script syntax, see Appendices : operators and functions in MapCalc and TabCalc, Appendices : ILWIS expressions and Appendices : ILWIS script language (syntax).

To create a script
- In the Main window, open the File menu, and choose Create Script, or
- In the Operation-list, double-click the item New Script.

The Create Script dialog box appears in which you can type your script expressions.

Example
To calculate a slope maps in percentages and in degrees:
1. Create a script (e.g. 'Slopes')
2. In the dialog box where the script is defined: type, to insert a comment line:
   ```
   // script to calc slope maps in percentages and degrees
   ```
3. To use operation InterpolContour to create an interpolated height map from segment contour lines; type:
   ```
   %2 = MapInterpolContour(%1,geo)
   ```
   Perform a contour interpolation on segment map `%1`, use existing georeference 'geo', and write the output to map `%2`.
   Advanced users may wish to define a georeference corners with a script command; see Appendices : ILWIS script language (syntax).
4. To use filter dfdx on the interpolated contour map to calculate height differences in X-direction; type:
   ```
   %3 = MapFilter(%2, dfdx)
   ```
   Filter map `%2` with the `dfdx` filter and write the output to map `%3`.
5. To use filter dfdy on the interpolated contour map to calculate height differences in Y-direction; type:
   ```
   %4 = MapFilter(%2, dfdy)
   ```
   Filter map `%2` with the `dfdx` filter and write the output to map `%4`.

6. To calculate a slope map from these, type:
   ```
   %5 = 100 * HYP(%3,%4) / PIXSIZE(%2)
   ```
   HYP is an internal Mapcalc/Tabcalc function;
   `%3` and `%4` are the output maps from the filtering;
   function `PIXSIZE` returns the pixel size of raster map `%2`;
   `%5` is the output map name of the map containing slope value in percentages.
7. To convert the percentage values into degrees, type:
   ```
   %6 = RADDEG(ATAN(%5/100))
   ```
   Function ATAN and RADDEG are internal MapCalc/TabCalc functions.
8. After running the script (see step 9), the output maps will be available as dependent maps. The expression by which a map is created is stored in the map's object definition files. The data file for an output map will be calculated when you double-click an output map in the Catalog.
   To have the script calculate the data files for the output maps, you may add the following lines to your script:
   ```
   calc %2.mpr
   calc %5.mpr
   calc %6.mpr
   ```
   In fact, by adding only `calc %6.mpr`, all maps, which are part of the process to calculate map `%6` will be calculated as well.
9. To run the script, type on the command line in the Main window:
   ```
     run Slopes Contour.mps DEM.mpr DX DY SlopePct SlopeDeg
   ```
   In script Slopes, `%1` is filled out as Contour.mps, `%2` as DEM.mpr, `%3` as DX, `%4` as DY, `%5` as SlopePct, and `%6` as SlopeDeg.

Of course, you can also use objects names inside the script instead of parameters `%1` (Contour, segment map), `%2` (DEM, Digital Elevation Model), etc. For more information on running scripts, see also How to run scripts.

The result of running this script are maps SLOPEPCT and SLOPEDEG which are slope maps in percentages and in degrees.

Mind: the following slope values are the same: 30°= 58%, 45°= 100%, 60°= 173%, 80°= 567%. As you see, slope values in the SLOPEPCT map can be greater than 100%.

Additionally, you can prepare representations for both maps with the Representation Value/Gradual editor.
You can also create two domain Groups to classify both output maps, e.g.:
classes 0-10%, 10-25%, 25-50%, 50-100%, >100% for the slope map in percentages and
classes 0-6°, 6-12.5°, 12.5-22.5°, 22.5-45°, >45° for the slope map in degrees.

Use these domain groups in the Slicing operation.