



**User Manual**  
**Working With Raster Data**  
**Version 1.0**  
**08.02.2014**

**Author: Dirk Zacher**



Faculty for Mathematics and Computer Science

Database Systems for New Applications

58084 Hagen, Germany



## Table of Contents

1	Introduction .....	1
2	Quick Start – learning by doing.....	3
3	Reference Manual.....	29
3.1	Tile algebra data types .....	29
3.1.1	Grid data types .....	29
3.1.2	Spatial data types.....	30
3.1.3	Moving spatial data types .....	31
3.1.4	Instant spatial data types.....	32
3.2	Tile algebra operators .....	33
3.2.1	atinstant.....	33
3.2.2	atlocation.....	34
3.2.3	atperiods.....	34
3.2.4	atrange .....	35
3.2.5	bbox .....	36
3.2.6	CELL1 .....	36
3.2.7	CELL2 .....	37
3.2.8	CELLS .....	37
3.2.9	compose .....	38
3.2.10	deftime .....	38
3.2.11	fromline .....	39
3.2.12	fromregion .....	39
3.2.13	getgrid.....	40
3.2.14	inst .....	40
3.2.15	map .....	41
3.2.16	map2 .....	42
3.2.17	matchgrid .....	43
3.2.18	maximum .....	43
3.2.19	minimum.....	44
3.2.20	t2mt.....	44
3.2.21	tiles .....	45
3.2.22	toraster2 .....	46
3.2.23	toregion.....	47
3.2.24	val .....	47
A	References .....	49



## 1 Introduction

SECONDO is an extensible database management system (DBMS) and is developed at field of teaching “database systems for new applications” at the University of Hagen. It provides a generic environment for the implementation of database components and has a modular, extensible architecture. The system frame of SECONDO can contain different implementations of DBMS data models. In contrast to the extensibility of commercial database systems an exchange of the kernel data model is also possible in SECONDO.

The architecture of SECONDO contains three important components:

- Kernel
- Optimizer
- Graphical User Interface (GUI)

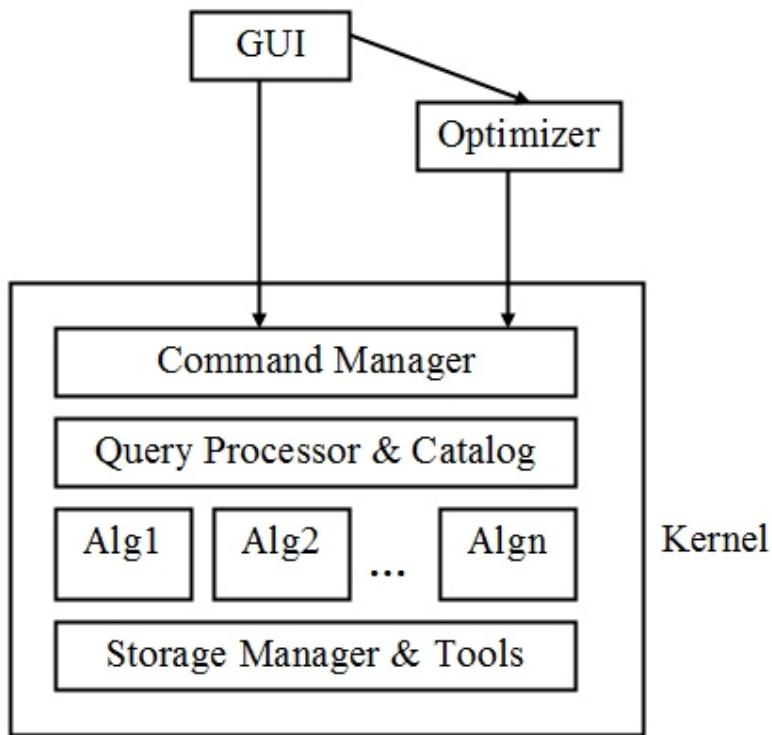


Figure 1.1: Architecture of extensible database system SECONDO [SEC14]

The kernel implements specific data models, provides query processing over all algebras of the SECONDO system and is easily extensible by an algebra module concept, using a well defined interface. Base functionality of the kernel can be extended by implementations of user defined algebras. In these algebras application oriented data types and operators can be defined and implemented. The kernel is written in C++.

The optimizer offers conjunctive query optimization, provides a SQL-like language and is extensible by optimization rules and cost functions [GBA+04]. It is written in PROLOG.

The graphical user interface (GUI) is extensible by data type specific viewers, application specific viewers and display functions. It is written in Java.

The source code of SECONDO is generally available, for example by download from SECONDO homepage [SEC14].

SECONDO contains two algebras supporting work with raster data, Raster2 algebra and Tile algebra. Both algebras provide similar data types and operators for raster data. The main difference between Raster2 algebra and Tile algebra lies in the implementation and usage of data types of these algebras.

In Raster2 algebra an object contains many raster data, for example height data at coordinates North 50 East 7 through North 52 East 10 in Germany. In contrast, a Tile algebra object normally contains less raster data than a Raster2 algebra object, because Tile algebra data types were implemented as attribute data types. Thus Tile algebra data types can be used in relations in contrast to Raster2 algebra data types.

Display of Raster2 algebra data types and Tile algebra data types in the SECONDO GUI was implemented identically to ensure comparable visualisation.

It is also possible to work with objects of Raster2 algebra and objects of Tile algebra at the same time. For this purpose Tile algebra provides operators **tiles** and **toraster2**. These operators allow the conversion of a Raster2 algebra object to a corresponding Tile algebra object and the reverse conversion of a Tile algebra object to a corresponding Raster2 algebra object.

## 2 Quick Start – learning by doing

The following chapter describes how to work with the SECONDO system and particularly how to work with raster data in SECONDO.

Work through the following steps:

1. If you do not have a SECONDO installation, go to the SECONDO download page <http://dna.fernuni-hagen.de/Secondo.html> and select an installation suitable for your platform, for example Linux, Mac OS X or Microsoft Windows.

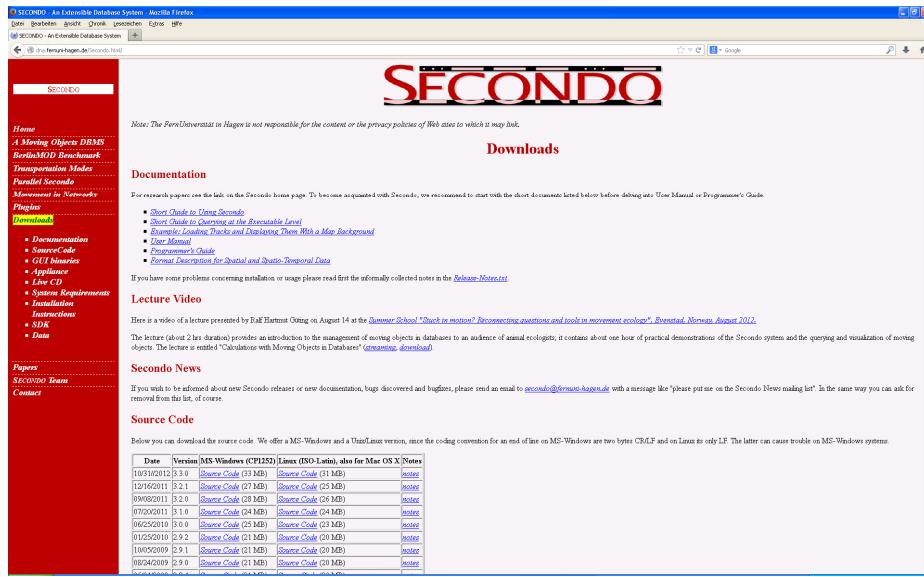


Figure 2.1: SECONDO download page

2. If necessary, get the latest version of SECONDO from the web site and extract it, preferably into the directory \$HOME/secondo.

3. Open a shell window and change into the SECONDO directory by command  
cd secondo.



Figure 2.2: Change into SECONDO directory

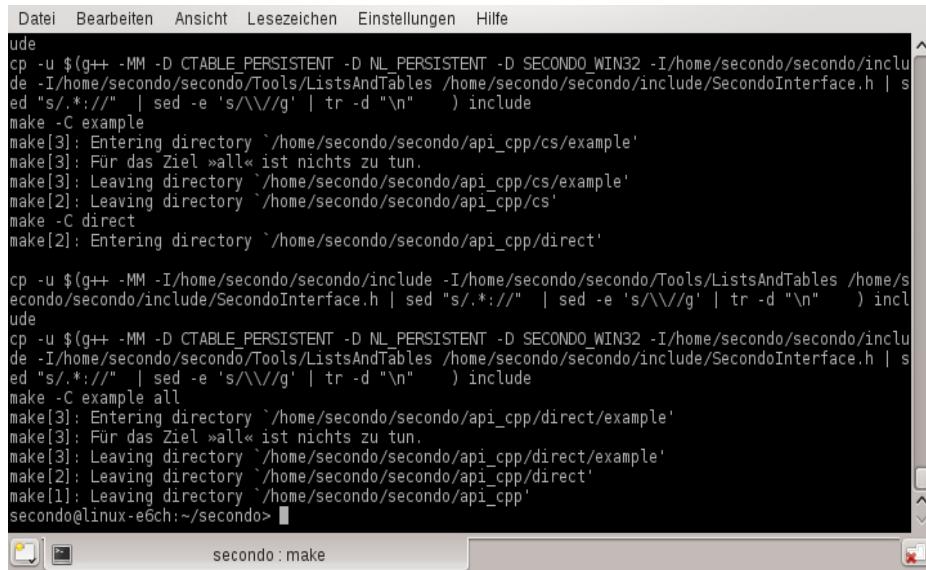
4. If the location of SECONDO system is not the directory \$HOME/seconde, you need to set an environment variable to this directory.  
On Windows installations this can be done by command `setvar $PWD`, on Linux use command `secroot`.  
In any case what happens is `export SECONDO_BUILD_DIR=$PWD`.

5. Compile your SECONDO system by command make.



```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
seconde@linux-e6ch:~/cd secondo
seconde@linux-e6ch:~/seconde> make
```

Figure 2.3: Begin of SECONDO compilation



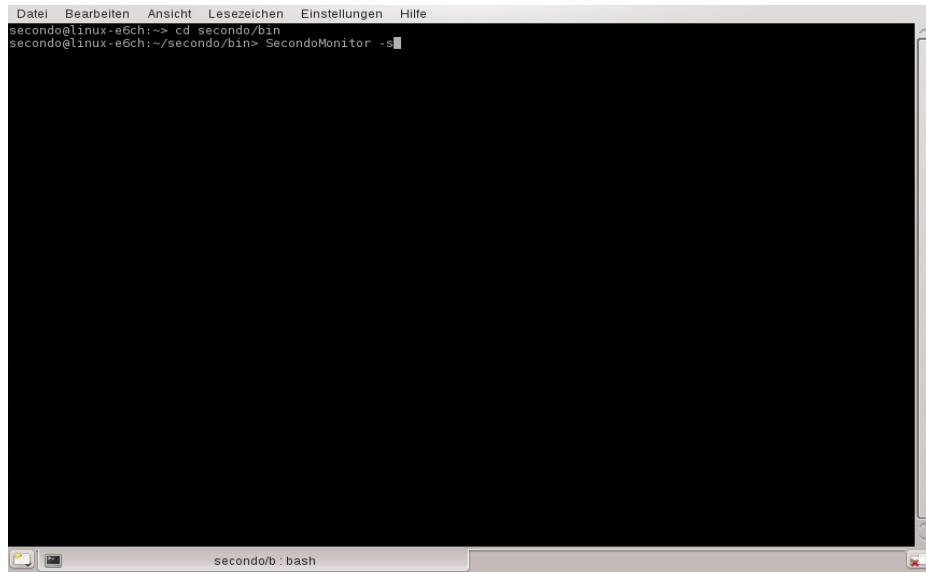
```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
ude
cp -u $(g++ -MM -D CTABLE_PERSISTENT -D NL_PERSISTENT -D SECONDO_WIN32 -I/home/seconde/seconde/include -I/home/seconde/seconde/Tools/ListsAndTables /home/seconde/seconde/include/SecondoInterface.h | sed "s/.*/://" | sed -e 's/\//\//g' | tr -d "\n" ) include
make -C example
make[3]: Entering directory `/home/seconde/seconde/api_cpp/cs/example'
make[3]: Für das Ziel »all« ist nichts zu tun.
make[3]: Leaving directory `/home/seconde/seconde/api_cpp/cs/example'
make[2]: Leaving directory `/home/seconde/seconde/api_cpp/cs'
make -C direct
make[2]: Entering directory `/home/seconde/seconde/api_cpp/direct'

cp -u $(g++ -MM -I/home/seconde/seconde/include -I/home/seconde/seconde/Tools/ListsAndTables /home/seconde/seconde/include/SecondoInterface.h | sed "s/.*/://" | sed -e 's/\//\//g' | tr -d "\n" ) include
cp -u $(g++ -MM -D CTABLE_PERSISTENT -D NL_PERSISTENT -D SECONDO_WIN32 -I/home/seconde/seconde/include -I/home/seconde/seconde/Tools/ListsAndTables /home/seconde/seconde/include/SecondoInterface.h | sed "s/.*/://" | sed -e 's/\//\//g' | tr -d "\n" ) include
make -C example all
make[3]: Entering directory `/home/seconde/seconde/api_cpp/direct/example'
make[3]: Für das Ziel »all« ist nichts zu tun.
make[3]: Leaving directory `/home/seconde/seconde/api_cpp/direct/example'
make[2]: Leaving directory `/home/seconde/seconde/api_cpp/direct'
make[1]: Leaving directory `/home/seconde/seconde/api_cpp'
seconde@linux-e6ch:~/seconde>
```

Figure 2.4: End of SECONDO compilation

After compile a runnable SECONDO system is available.

6. Open a new shell window, change into directory secondo/bin by command cd secondo/bin and execute command SecondoMonitor -s.

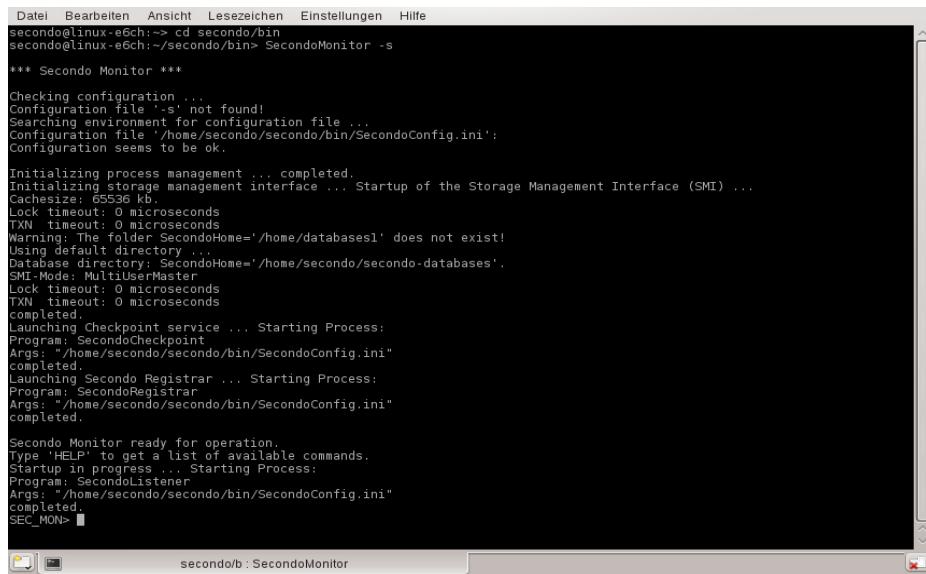


A screenshot of a terminal window titled "seondo/b : bash". The window shows a command-line interface with the following text:

```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe  
seondo@linux-e6ch:~> cd secondo/bin  
seondo@linux-e6ch:~/seondo/bin> SecondoMonitor -s
```

Figure 2.5: Start of SecondoMonitor

Command SecondoMonitor -s starts a process listening for user interfaces to register with the kernel.



A screenshot of a terminal window titled "seondo/b : SecondoMonitor". The window shows a command-line interface with the following text:

```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe  
seondo@linux-e6ch:~> cd secondo/bin  
seondo@linux-e6ch:~/seondo/bin> SecondoMonitor -s  
  
*** Seconde Monitor ***  
Checking configuration ...  
Configuration file 's' not found!  
Searching environment for configuration file ...  
Configuration file '/home/seondo/seconde/bin/SecondoConfig.ini':  
Configuration seems to be ok.  
  
Initializing process management ... completed.  
Initializing storage management interface ... Startup of the Storage Management Interface (SMI) ...  
CacheSize: 20536  
Lock timeout: 0 microseconds  
TXN timeout: 0 microseconds  
Warning: The folder SecondoHome='/home/databases1' does not exist!  
Using default directory ...  
Database directory: SecondoHome='/home/seondo/seconde-databases'.  
SMI-Mode: MultiUserMaster  
Lock timeout: 0 microseconds  
TXN timeout: 0 microseconds  
completed.  
Launching Checkpoint service ... Starting Process:  
Program: SecondoCheckpoint  
Args: "/home/seondo/seconde/bin/SecondoConfig.ini"  
completed.  
Launching Seconde Registrar ... Starting Process:  
Program: SecondoRegistrar  
Args: "/home/seondo/seconde/bin/SecondoConfig.ini"  
completed.  
  
Seconde Monitor ready for operation.  
Type 'HELP' to get a list of available commands.  
Startup in progress ... Starting Process:  
Program: SecondoListener  
Args: "/home/seondo/seconde/bin/SecondoConfig.ini"  
completed.  
SEC_MON>
```

Figure 2.6: Execution of SecondoMonitor

7. Open a new shell window, change into directory secondo/Optimizer by command cd secondo/Optimizer and execute command StartOptServer.

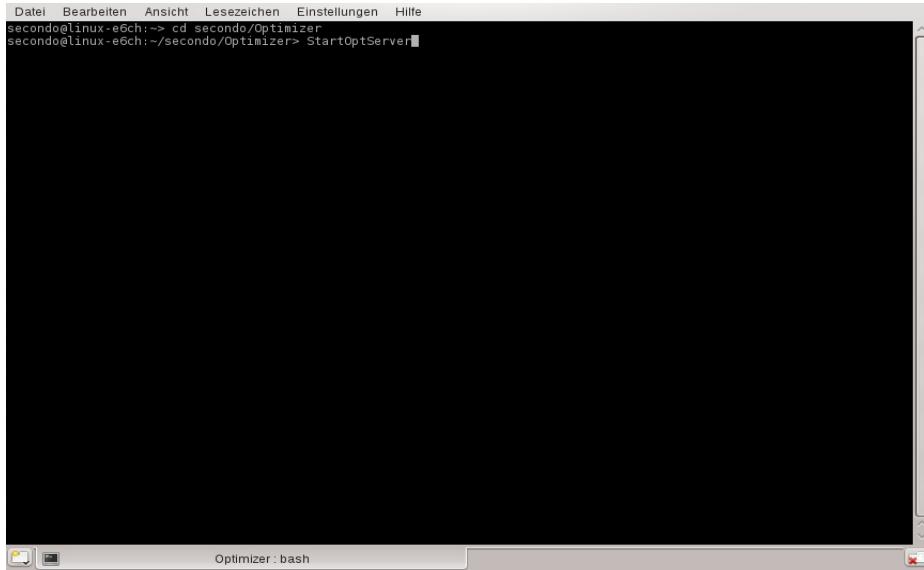


Figure 2.7: Start of Optimizer Server

Command StartOptServer starts the optimizer process.

The screenshot shows a terminal window with a dark background and light-colored text. It displays a large list of configuration options for the optimizer, each with a brief description. At the bottom of the list, it says 'Type 'loadOptions.' to load the saved option configuration.' and 'waiting for requests'. The window title is 'Optimizer : sh'.

```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
[ ]  largeQueries(qdgm): Large predicate set optimization (query graph decomposition and materialization)
[ ]  largeQueries(aco): Large predicate set optimization (ascending cost order)
[ ]  pathTiming: Prompt time used to find a best path.
[ ]  dynamicSample: Use dynamic instead of static (saved) samples.
[x]  autoSamples: Automatically determine sample sizes.
[ ]  eagerObjectCreation: Create all samples and small objects at 'open databases'.
[ ]  rewriteMacros: Allow for macros in queries.
[ ]  rewriteInference: Add inferred predicates to where clause.
[ ]  rewriteIndexRules: Infer predicates from R-tree indices.
[ ]  rewriteNonempty: Handle 'nonempty' in select statements.
[ ]  rewriteCSE: Extend with attributes for CSE values.
[ ]  rewriteCSEAll: Extend with attributes for ALL_CSEs.
[ ]  rewriteRemove: Remove attributes as early as possible.
[x]  debug: Execute debugging code. Also use 'toggleDebug'.
[x]  autosave: Autosave option settings on 'halt'.
[ ]  noprogress: Do not send predicate data for progress estimation.
[ ]  subqueries: Process subqueries.
[ ]  subqueryUnnesting: Apply unnesting algorithms to subqueries.
[ ]  correlations: Derive joint probabilities for selection predicates before computing a best plan
[ ]  joinCorrelations: Additionally derive joint probabilities for join predicates (nested-loop)
[ ]  joinCorrelations2: The same behavior as joinCorrelations but use product-filter
[ ]  adaptiveJoin: Allow usage of adaptive join operators.
[ ]  entropy: Use entropy maximization together with an exploration query on a small sample database
[ ]  nestedRelations: Support for nested relations.
[ ]  memoryAllocation: Assign memory to operators.

Type 'loadOptions.' to load the saved option configuration.
Type 'saveOptions.' to save current option configuration to disk.
Type 'defaultOptions.' to restore the default options.
Type 'setoption(X)' to select option X.
Type 'deloption(X)' to unselect option X.
Type 'showoptions.' to view this option list.

Type 'helpMe.' to get an overview on user level predicates.

% calloptimizer compiled 0.23 sec, 1.553.568 bytes
waiting for requests
opt-server > optserver >
```

Figure 2.8: Execution of Optimizer Server

8. Open a new shell window, change into directory secondo/Javagui by command cd secondo/Javagui and execute command sgui.

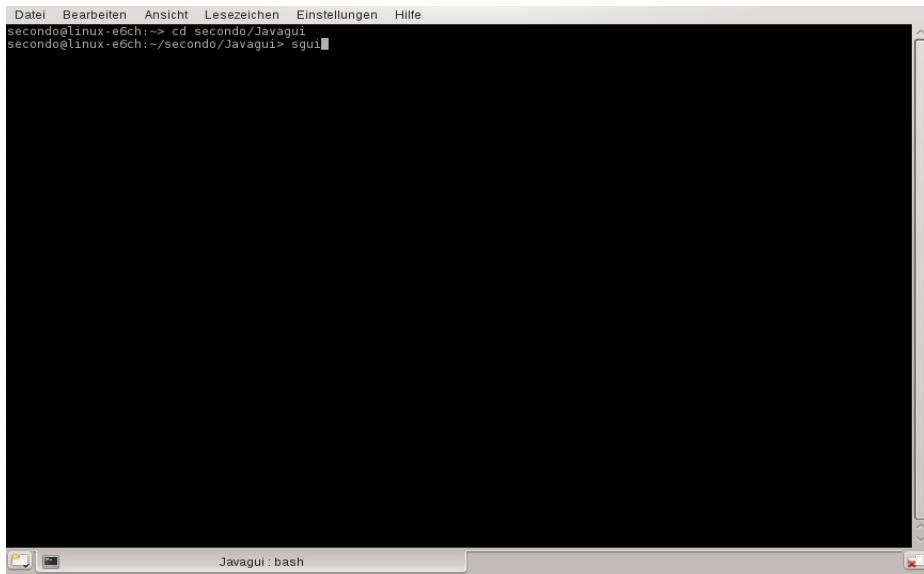


Figure 2.9: Start of SECONDO GUI

Command sgui starts the graphical user interface of SECONDO.

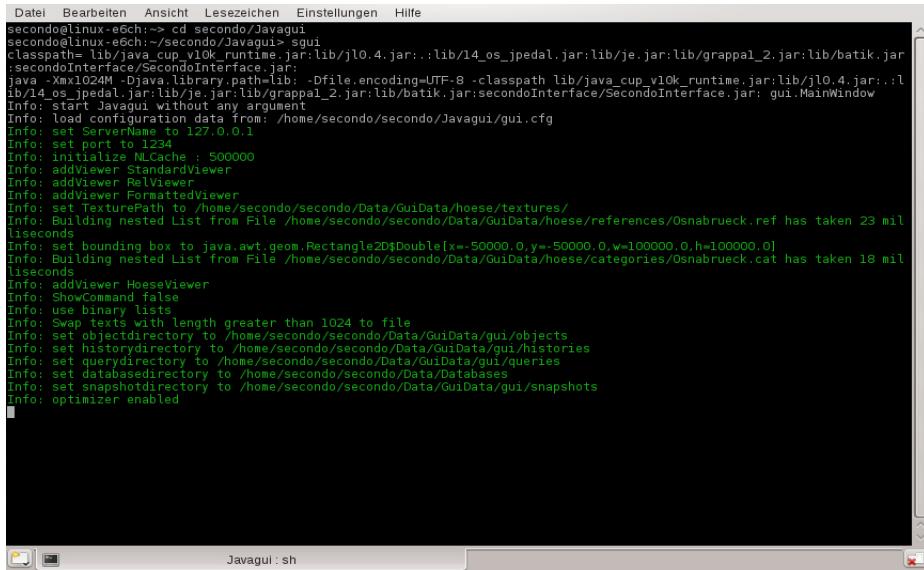


Figure 2.10: Execution of SECONDO GUI

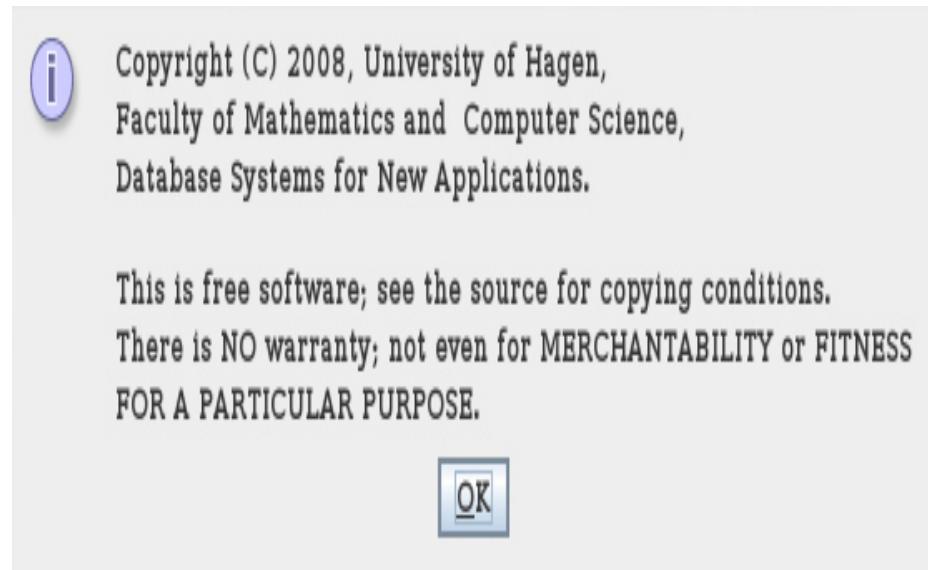


Figure 2.11: Copyright Information of SECONDO GUI

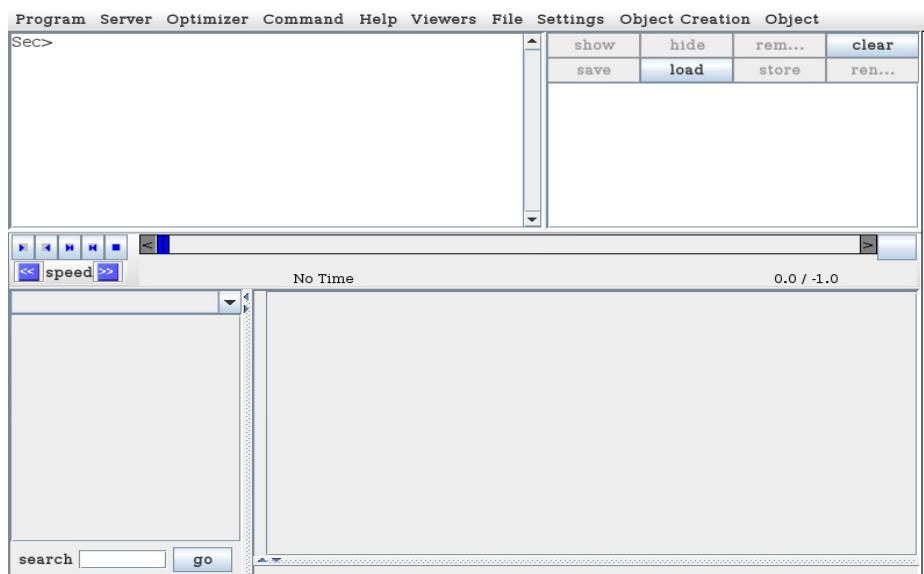


Figure 2.12: SECONDO GUI after start of application

9. At the command window in the upper left area of the SECONDO GUI execute command create database RasterDatabase.

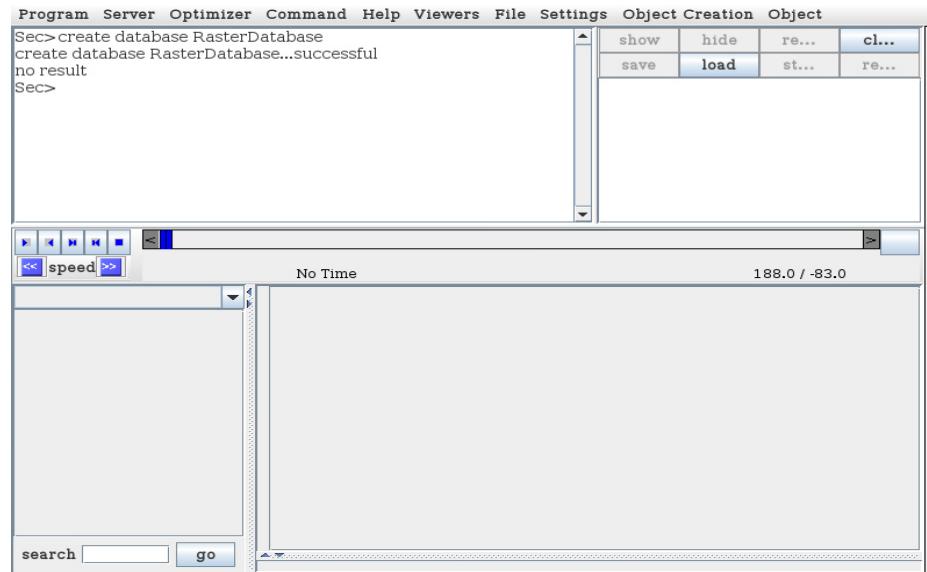


Figure 2.13: SECONDO GUI after creation of RasterDatabase

10. Open database RasterDatabase by command  
open database RasterDatabase.

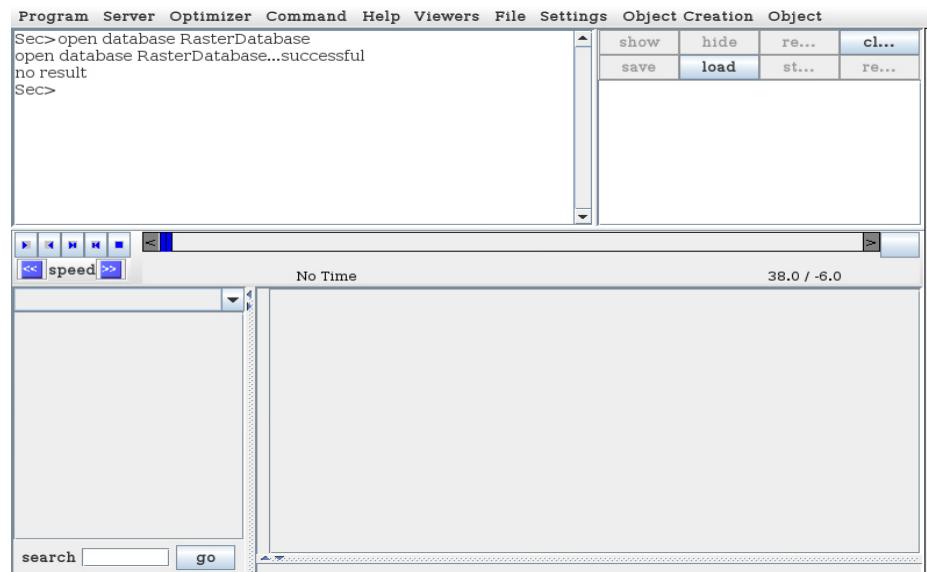


Figure 2.14: SECONDO GUI after opening of RasterDatabase

11. Import raster data of hgt format from file secondo/Data/Raster2/N51E007.hgt into a Raster2 algebra object of type sint by command  
let N51E007 = '../Data/Raster2/N51E007.hgt' feed  
importHgt.

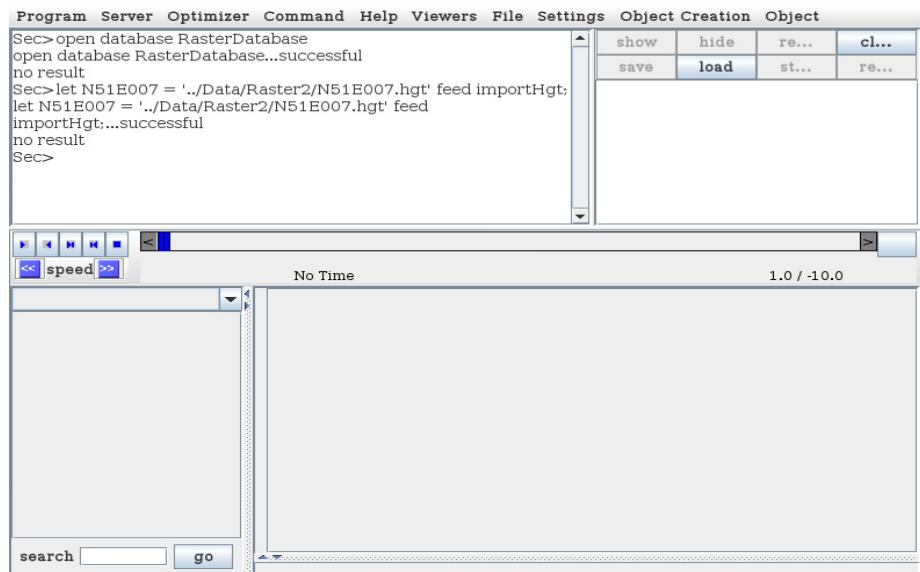


Figure 2.15: SECONDO GUI after import of hgt data

12. Display Raster2 algebra N51E007 object of type sint in Hoesse Viewer by command  
query N51E007.

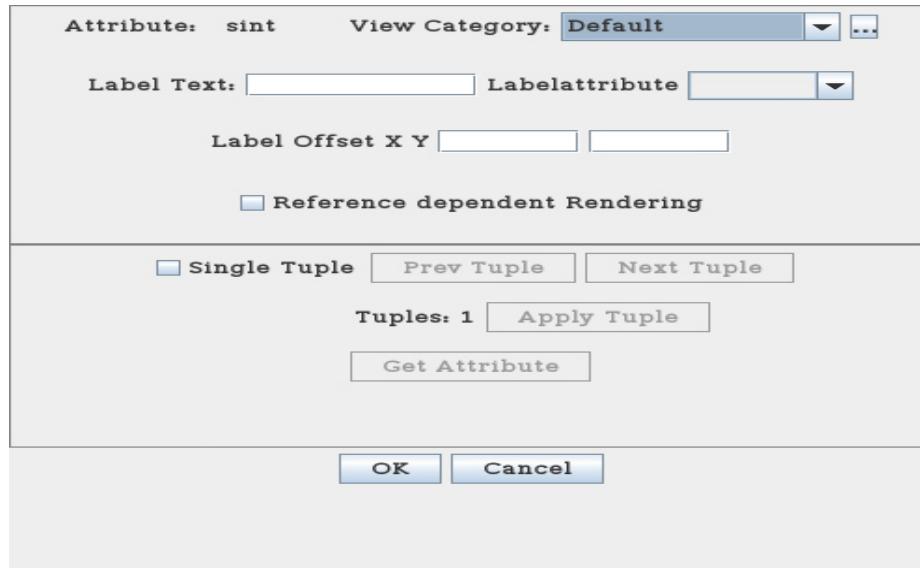


Figure 2.16: Display properties of N51E007 object

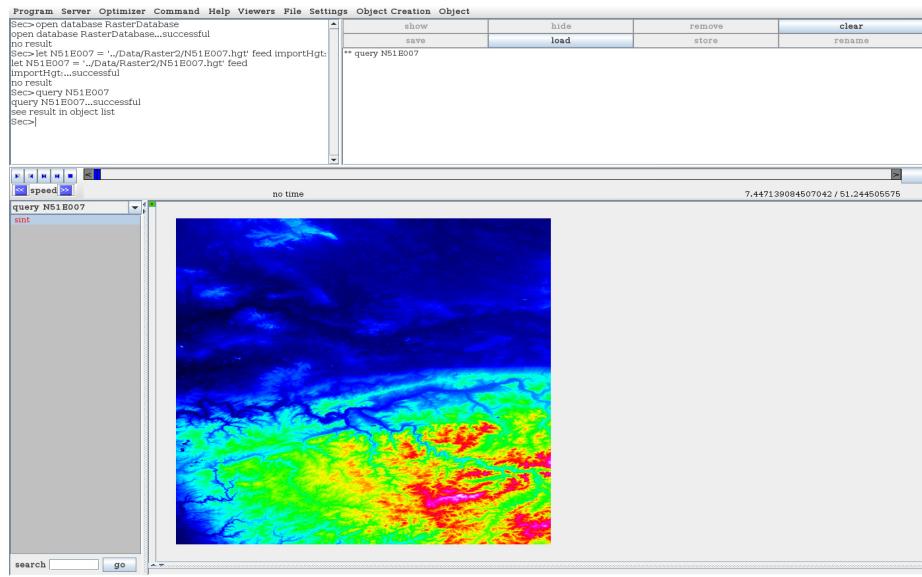


Figure 2.17: Visualisation of N51E007 object

13. Import raster data of esri raster format from file secondo/Data/Raster2/N52E006.asc into a Raster2 algebra object of type sreal by command

```
let N52E006 = './Data/Raster2/N52E006.asc' feed
importEsriRaster.
```

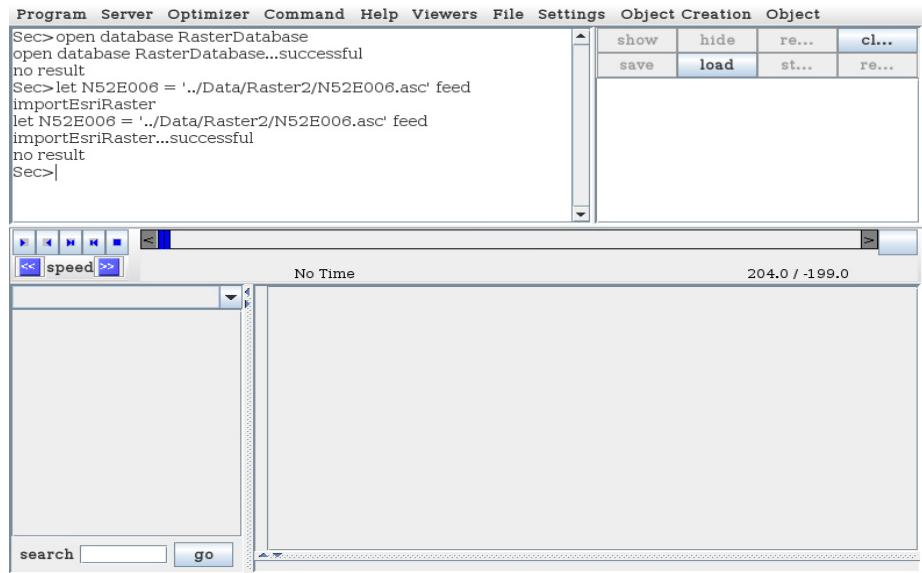


Figure 2.18: SECONDO GUI after import of esri raster data

14. Display Raster2 algebra N52E006 object of type *sreal* in Hoes View by command  
query N52E006.

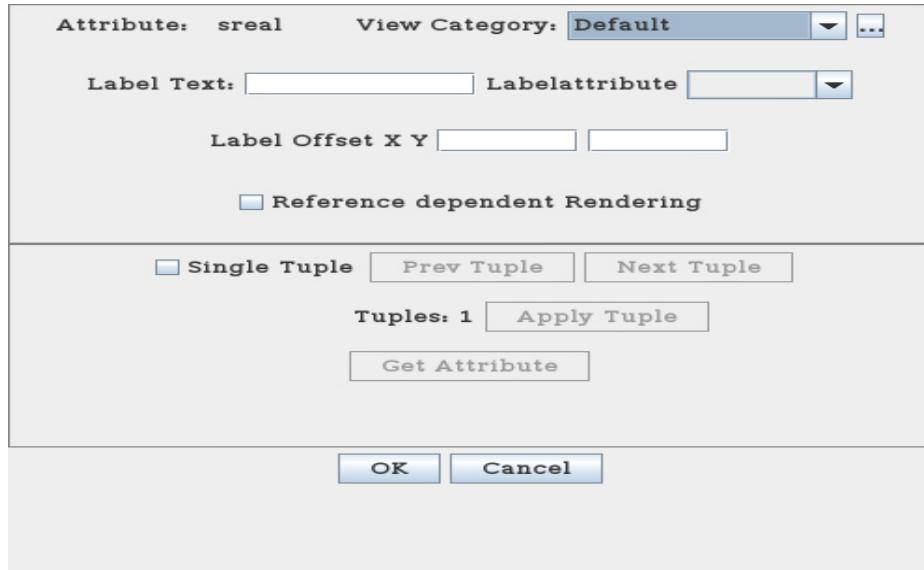


Figure 2.19: Display properties of N52E006 object

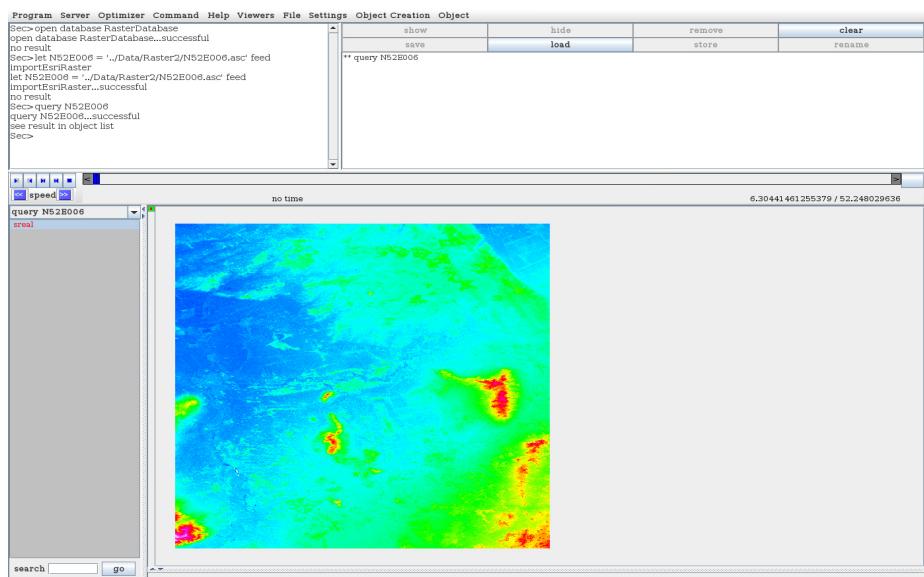


Figure 2.20: Visualisation of N52E006 object

15. Convert Raster2 algebra N51E007 object of type sint into a relation of Tile algebra tint objects and save result in N51E007Relation object by command  
let N51E007Relation = tiles(N51E007)  
namedtransformstream[No] consume.

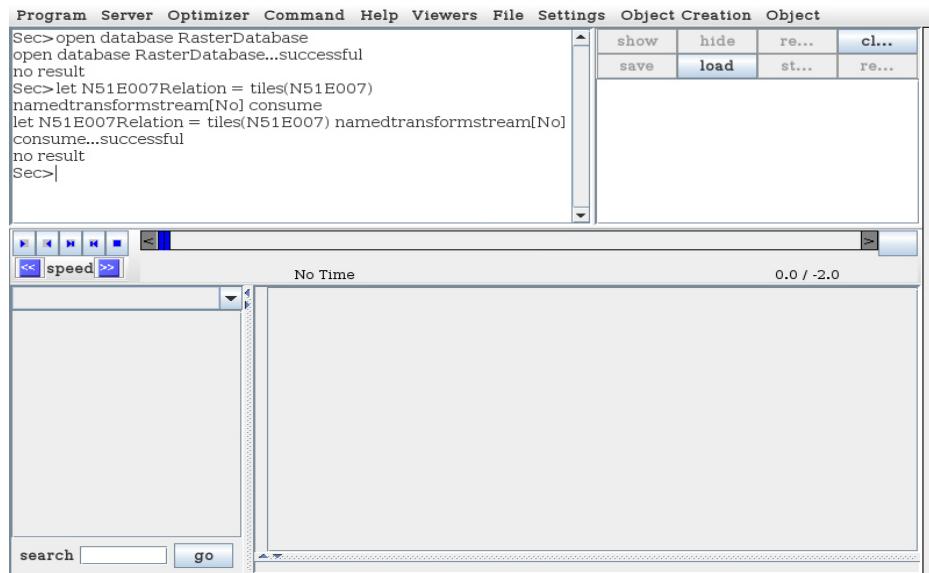


Figure 2.21: Conversion of N51E007 object into N51E007Relation

The operator namedtransformstream can be used to transform a stream of values into a stream of tuples. It creates for each value in the input stream one tuple with a single attribute that can be specified in a SECONDO command.

16. Display relation N51E007Relation of Tile algebra tint objects in Hoese Viewer by command query N51E007Relation.

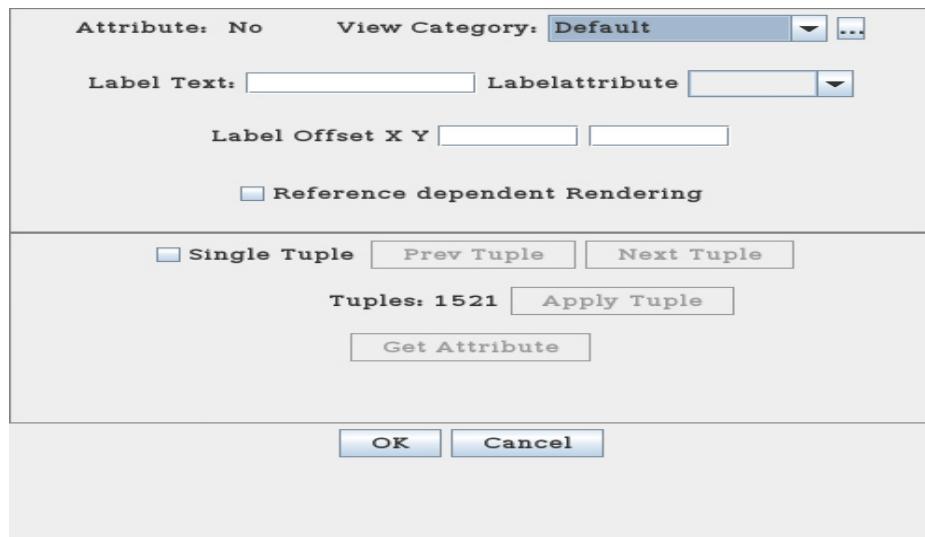


Figure 2.22: Display properties of N51E007Relation object

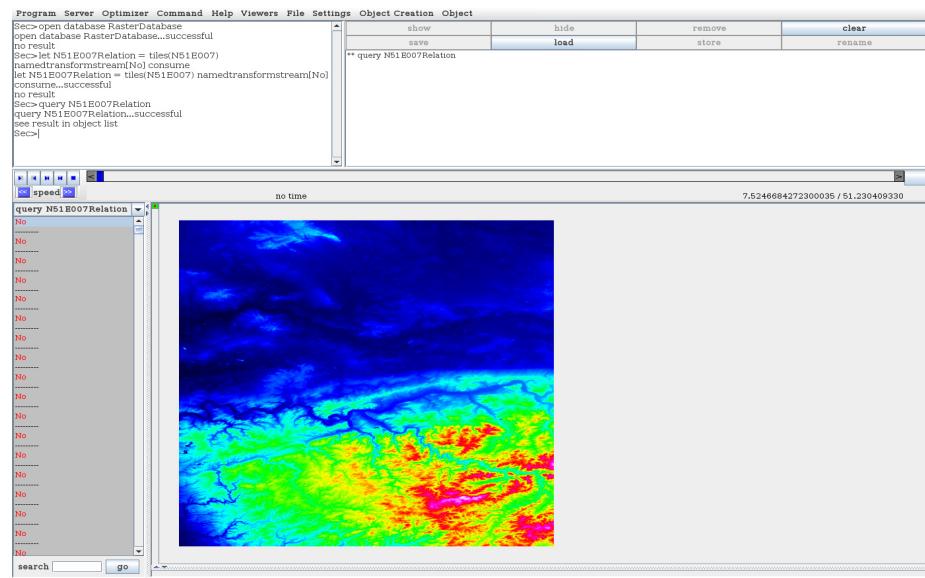


Figure 2.23: Visualisation of N51E007Relation object

17. Convert Raster2 algebra N52E006 object of type sreal into a relation of Tile algebra treal objects and save result in N52E006Relation object by command

```

let N52E006Relation = tiles(N52E006)
namedtransformstream[No] consume.

```

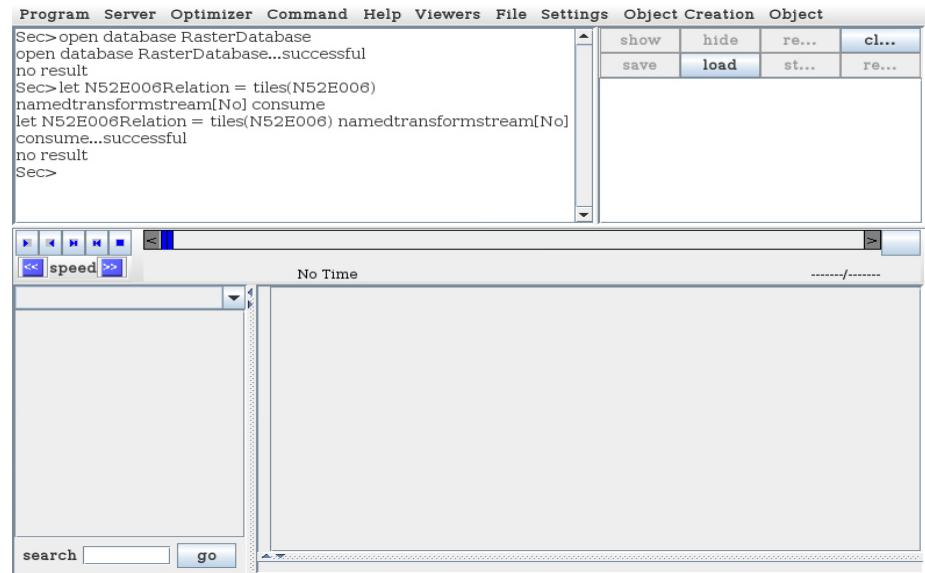


Figure 2.24: Conversion of N52E006 object into N52E006Relation

18. Display relation N52E006Relation of Tile algebra *treal* objects in Hoese Viewer by command query N52E006Relation.

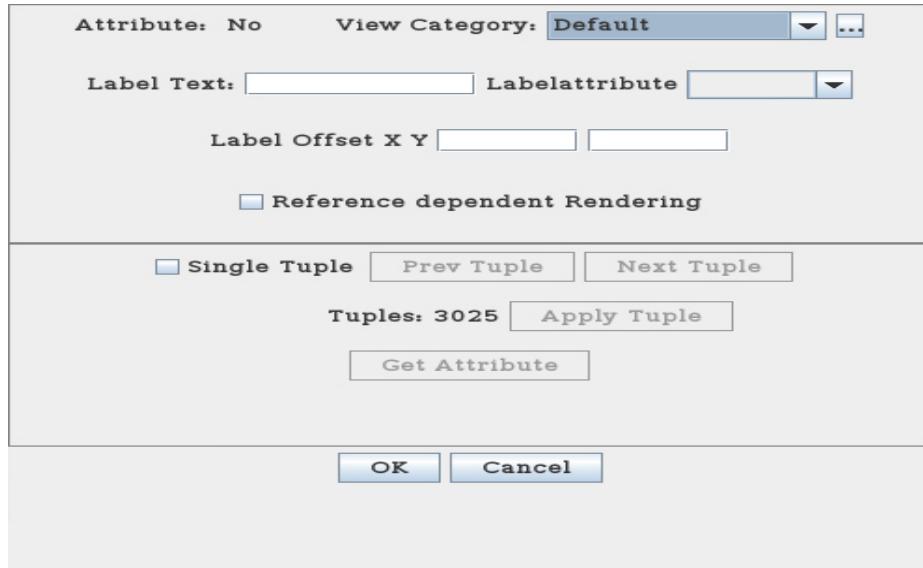


Figure 2.25: Display properties of N52E006Relation object

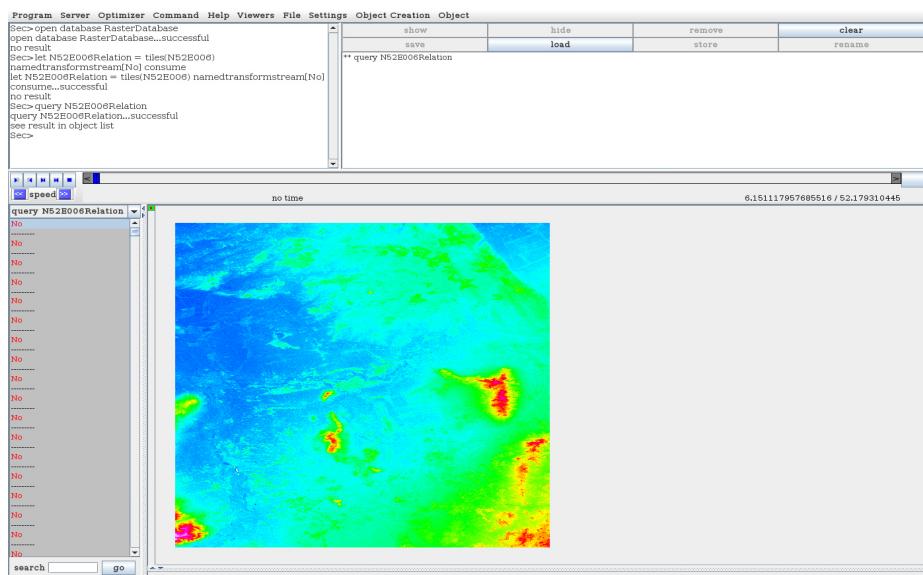


Figure 2.26: Visualisation of N52E006Relation object

19. Convert Raster2 algebra N51E007 object of type sint into a relation of Tile algebra tint objects and display the first 50 tiles by command  
query tiles(N51E007) namedtransformstream[No] head[50] consume.

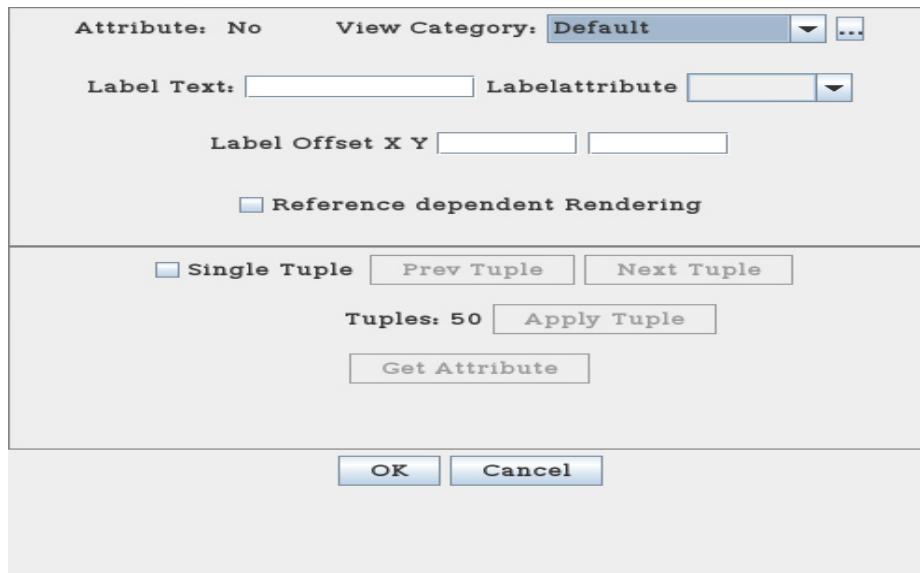


Figure 2.27: Display properties of the resulting relation containing the first 50 tuples

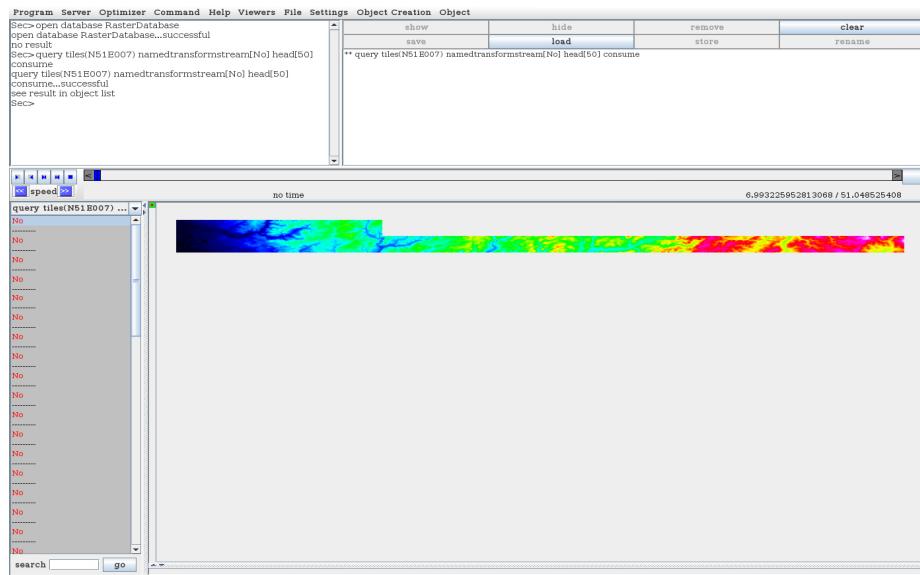


Figure 2.28: Visualisation of the resulting relation containing the first 50 tuples

20. Convert Raster2 algebra N51E007 object of type sint into a relation of Tile algebra tint objects and display the last 400 tiles by command  
query tiles(N51E007) namedtransformstream[No] tail[400]  
consume.

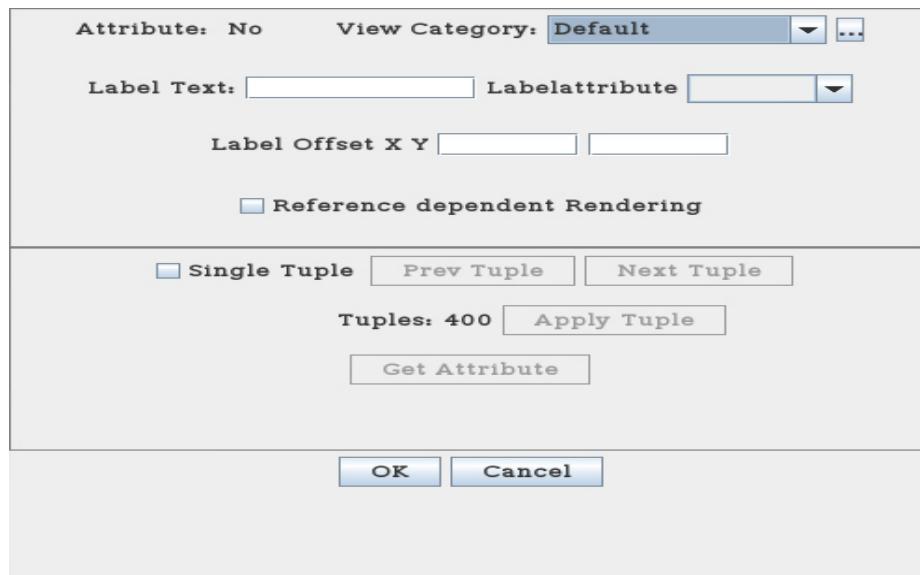


Figure 2.29: Display properties of the resulting relation containing the last 400 tuples

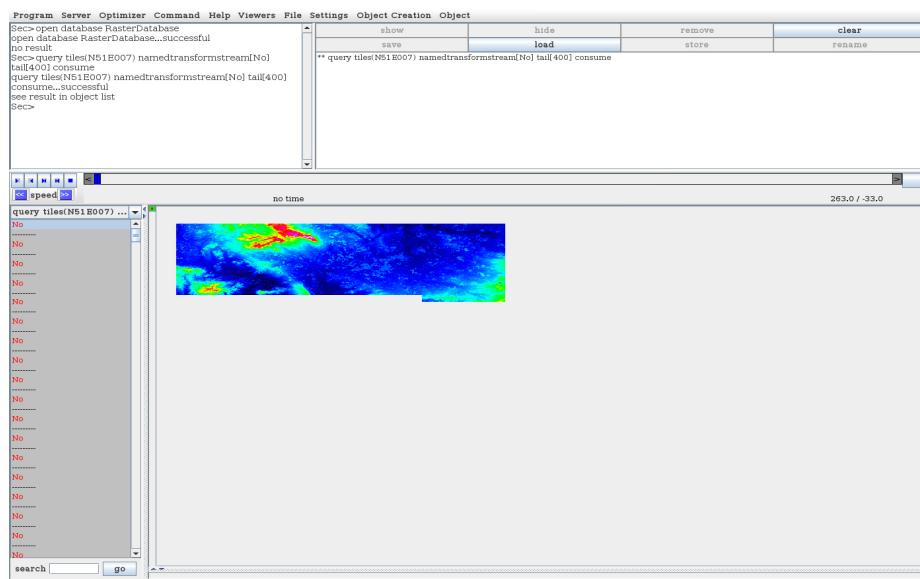


Figure 2.30: Visualisation of the resulting relation containing the last 400 tuples

21. Display all tiles of N51E007Relation of Tile algebra *tint* objects whose minimum is greater than 50 by command

```
query N51E007Relation feed filter[minimum(.No) > 50]
consume.
```

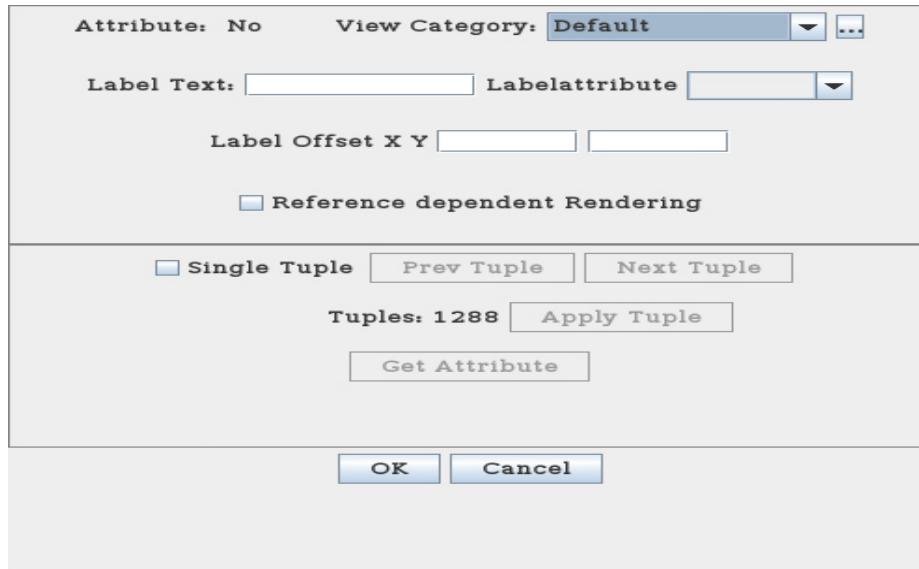


Figure 2.31: Display properties of N51E007Relation tuples with a minimum greater than 50

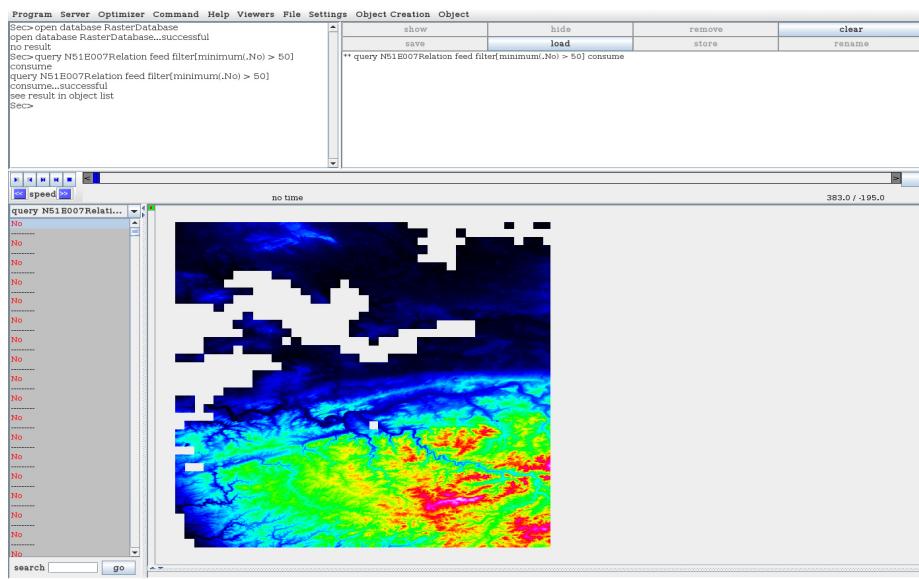


Figure 2.32: Visualisation of N51E007Relation tuples with a minimum greater than 50

22. Display all tiles of N51E007Relation of Tile algebra tint objects whose maximum is less than 400 by command

```
query N51E007Relation feed filter[maximum(.No) < 400]
consume.
```

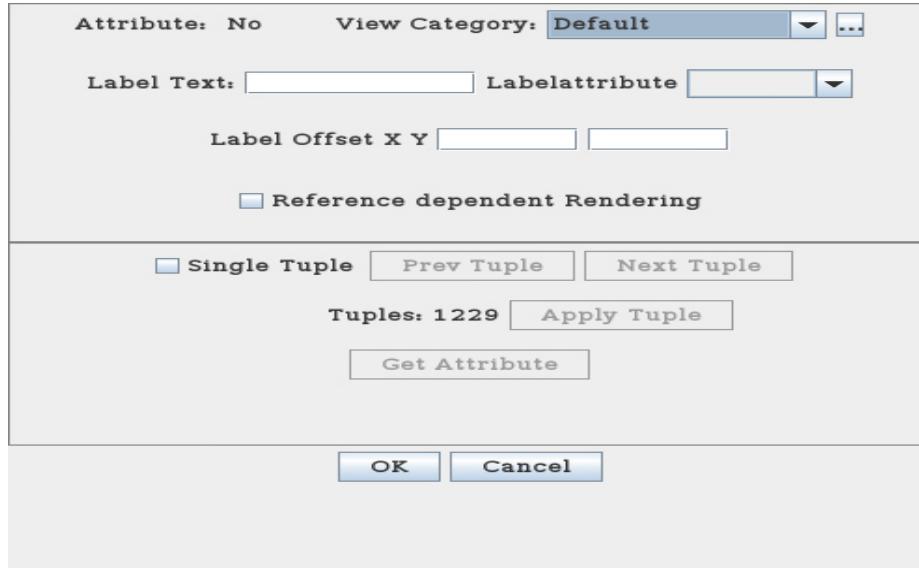


Figure 2.33: Display properties of N51E007Relation tuples with a maximum less than 400

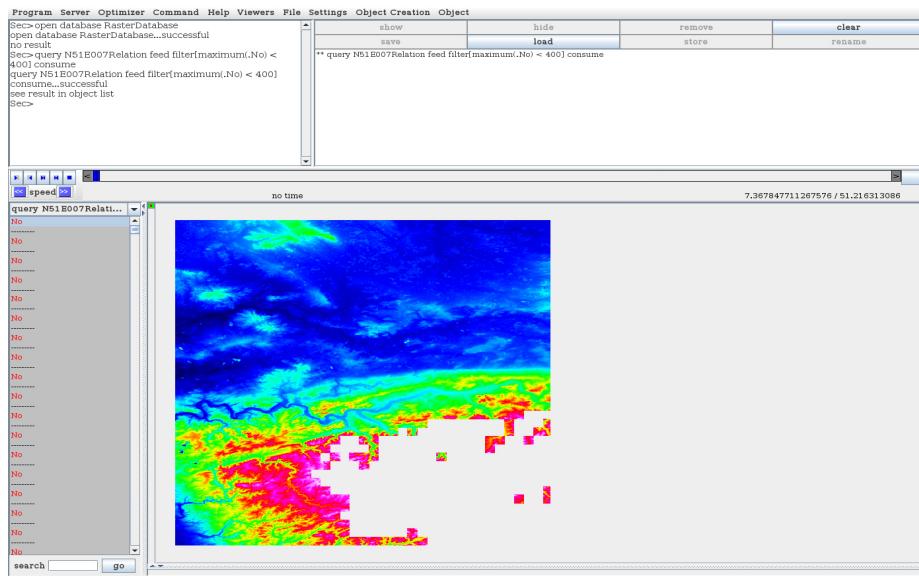


Figure 2.34: Visualisation of N51E007Relation tuples with a maximum less than 400

23. Display all tiles of N51E007Relation of Tile algebra *tint* objects whose minimum is greater than 50 and maximum is less than 400 by command

```
query N51E007Relation feed filter[(minimum(.No) > 50) and  
(maximum(.No) < 400)] consume.
```

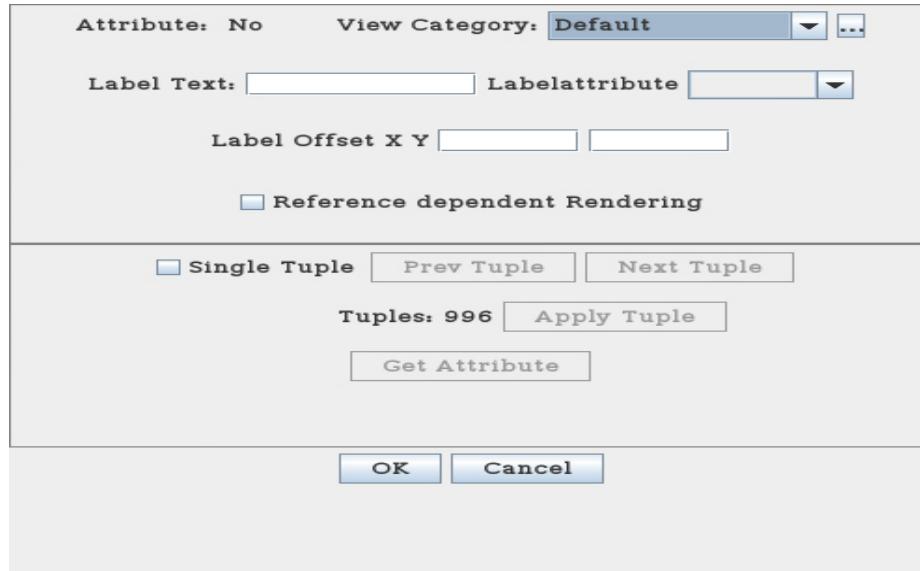


Figure 2.35: Display properties of N51E007Relation tuples with a minimum greater than 50 and a maximum less than 400

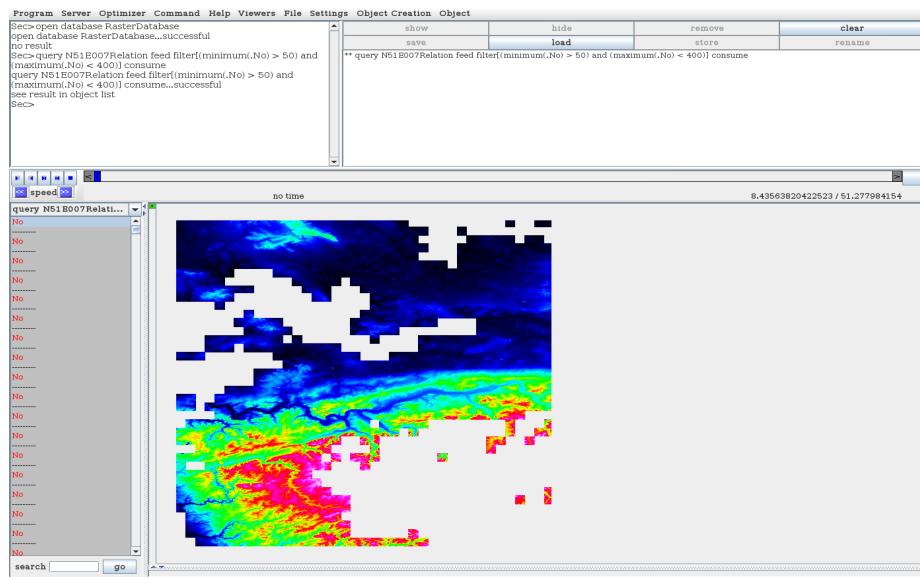
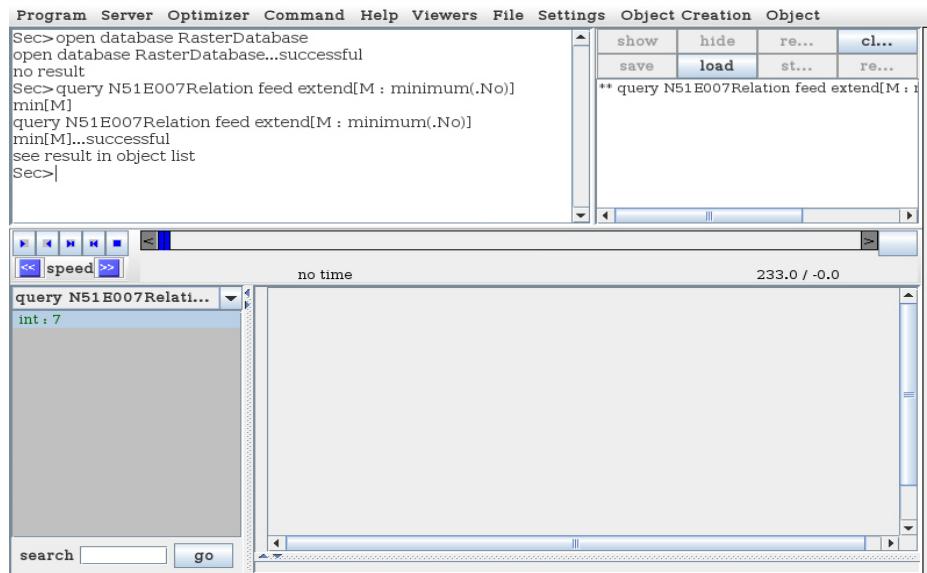


Figure 2.36: Visualisation of N51E007Relation tuples with a minimum greater than 50 and a maximum less than 400

24. Display minimum value of all tiles of N51E007Relation by command  
query N51E007Relation feed extend[M : minimum(.No) ]  
min[M].



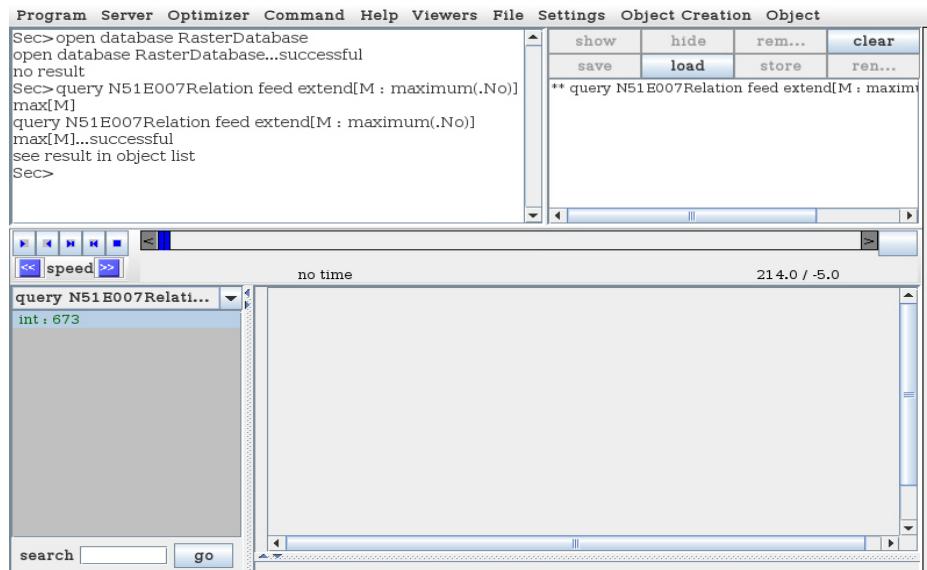
The screenshot shows a software interface with a menu bar (Program, Server, Optimizer, Command, Help, Viewers, File, Settings, Object Creation, Object) and a toolbar with buttons for show, hide, save, load, etc. The main window displays a command-line session:

```
Sec>open database RasterDatabase
open database RasterDatabase...successful
no result
Sec>query N51E007Relation feed extend[M : minimum(.No)]
min[M]
query N51E007Relation feed extend[M : minimum(.No)]
min[M]...successful
see result in object list
Sec>
```

Below the command window is a status bar with "speed" and "no time" and a timer showing "233.0 / -0.0". A results pane on the right shows the output of the query.

Figure 2.37: Visualisation of the minimum value of N51E007Relation

25. Display maximum value of all tiles of N51E007Relation by command  
query N51E007Relation feed extend[M : maximum(.No) ]  
max[M].



The screenshot shows a software interface with a menu bar (Program, Server, Optimizer, Command, Help, Viewers, File, Settings, Object Creation, Object) and a toolbar with buttons for show, hide, save, load, etc. The main window displays a command-line session:

```
Sec>open database RasterDatabase
open database RasterDatabase...successful
no result
Sec>query N51E007Relation feed extend[M : maximum(.No)]
max[M]
query N51E007Relation feed extend[M : maximum(.No)]
max[M]...successful
see result in object list
Sec>
```

Below the command window is a status bar with "speed" and "no time" and a timer showing "214.0 / -5.0". A results pane on the right shows the output of the query.

Figure 2.38: Visualisation of the maximum value of N51E007Relation

26. Display grid values of all tiles of N51E007Relation by command query N51E007Relation feed extend[G : getgrid(.No)] consume.

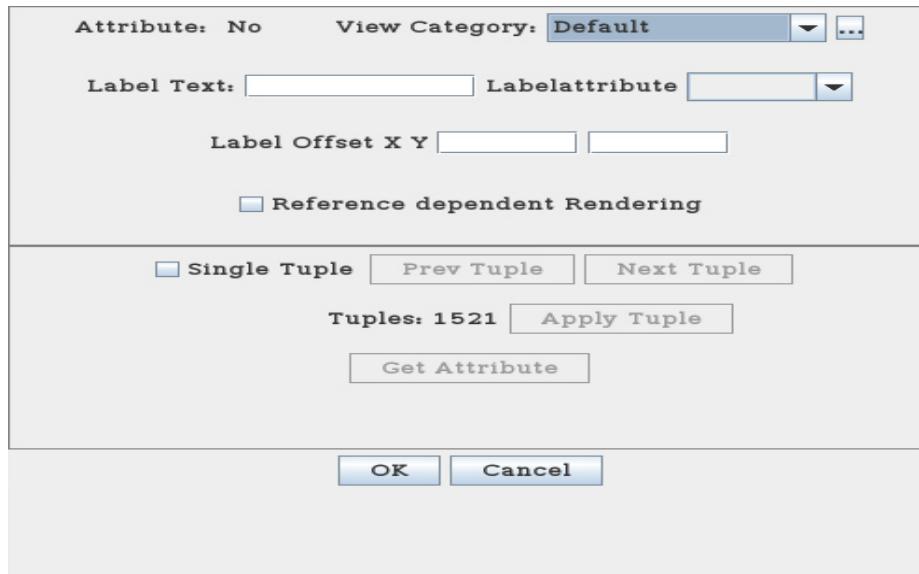


Figure 2.39: Display properties of grid values of N51E007Relation

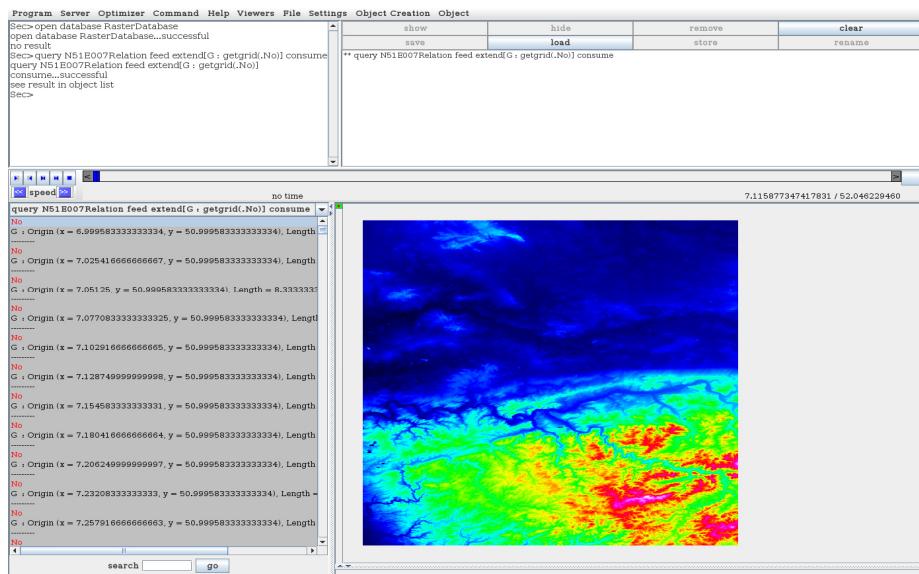


Figure 2.40: Visualisation of grid values of N51E007Relation

27. Display tile of N51E007Relation that contains a defined value at location  $x = 7.5$  and  $y = 51.5$  and display this value by command

```
query N51E007Relation feed filter[isdefined(.No atlocation  
[[const point value(7.5 51.5)]]]] extend[AL :  
.No atlocation [[const point value(7.5 51.5)]]] consume.
```

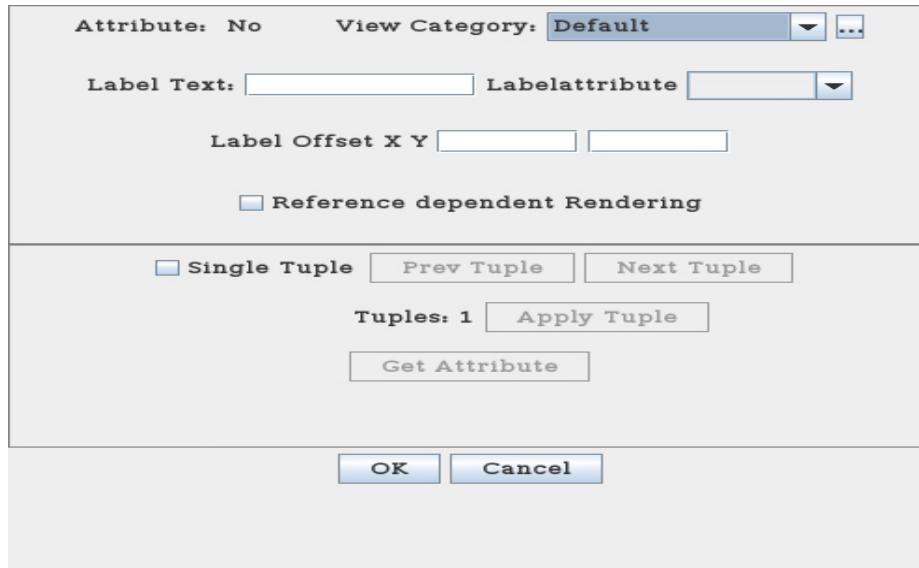


Figure 2.41: Display properties of a tile of N51E007Relation that contains a defined value at location  $x = 7.5$  and  $y = 51.5$

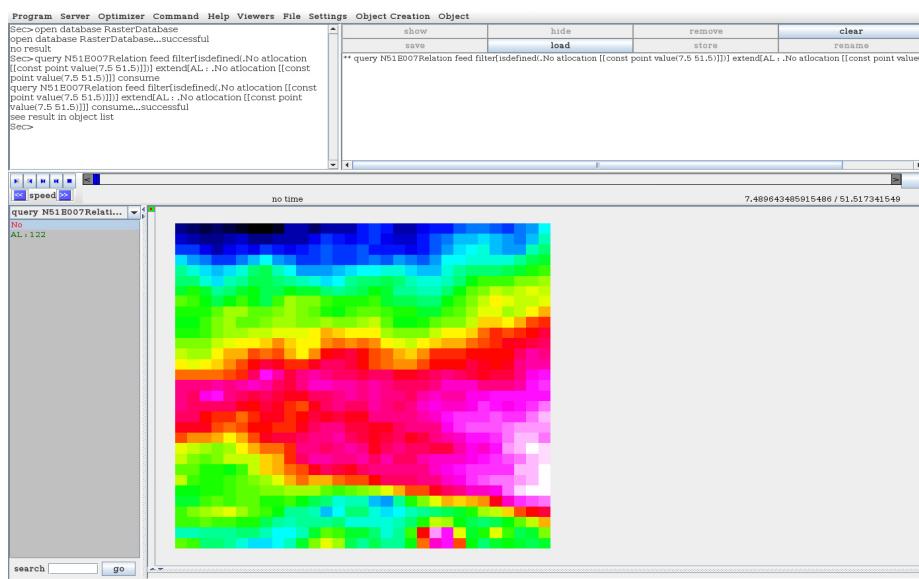


Figure 2.42: Visualisation of a tile of N51E007Relation that contains a defined value at location  $x = 7.5$  and  $y = 51.5$

28. Display all tiles of N51E007Relation and corresponding bounding boxes by command  
query N51E007Relation feed extend[BB : bbox(.No) ] consume.

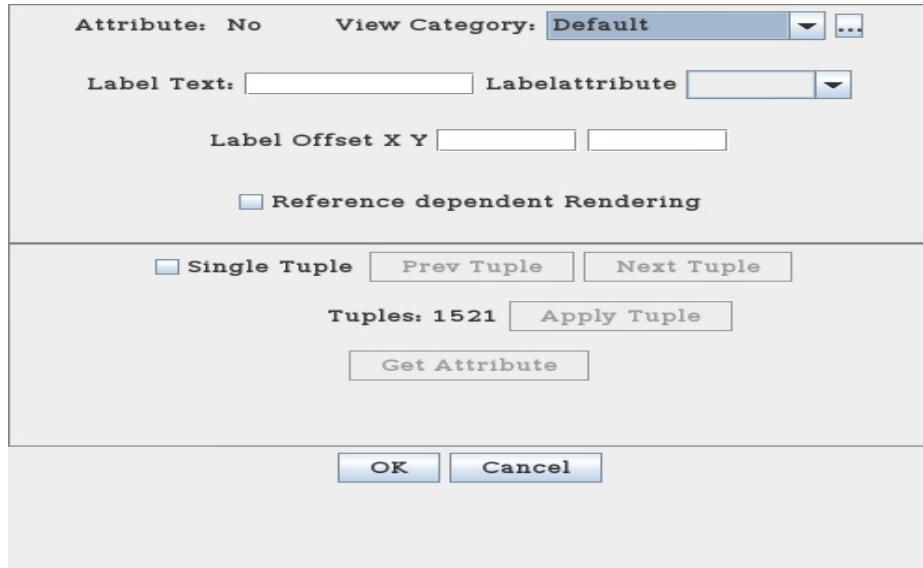


Figure 2.43: Display properties of all tiles of N51E007Relation

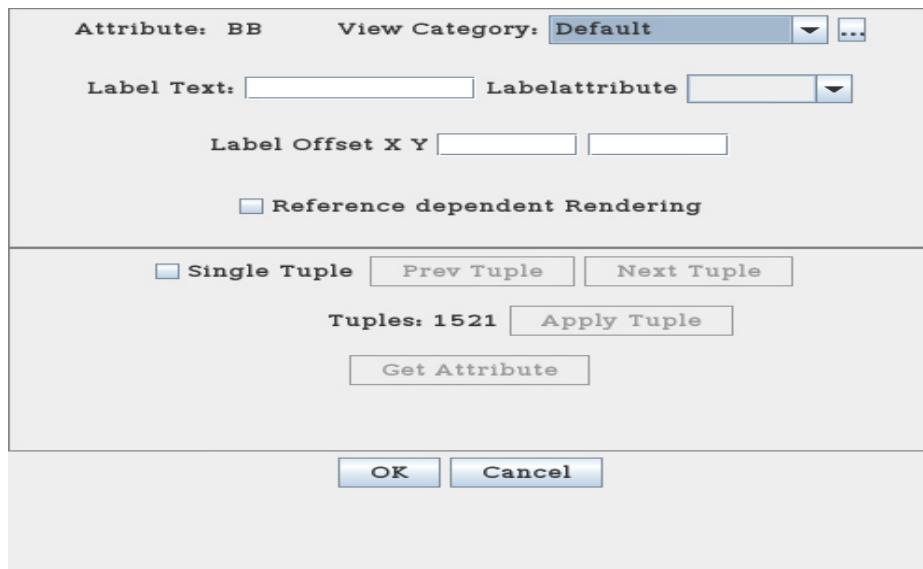


Figure 2.44: Display properties of corresponding bounding boxes  
of all tiles of N51E007Relation

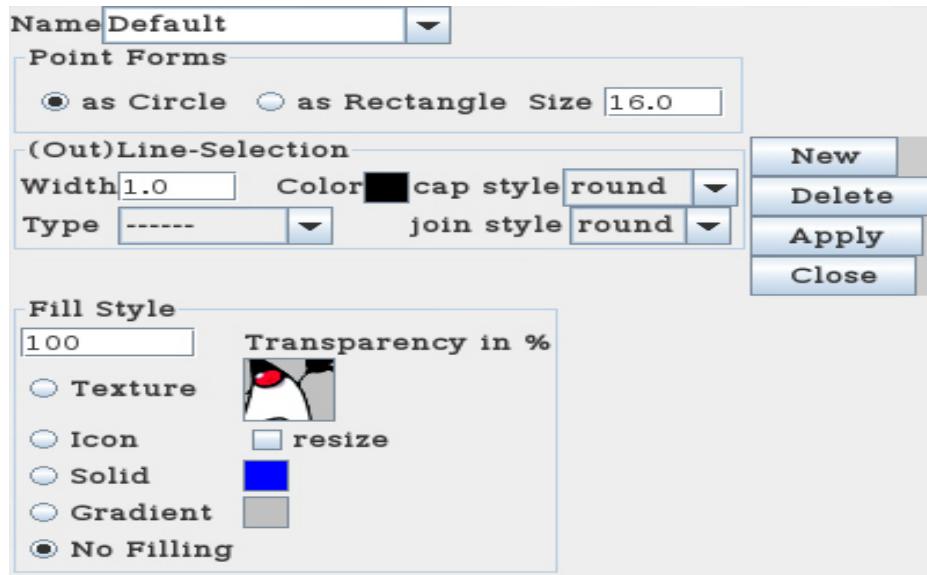


Figure 2.45: View Category properties of corresponding bounding boxes of all tiles of N51E007Relation

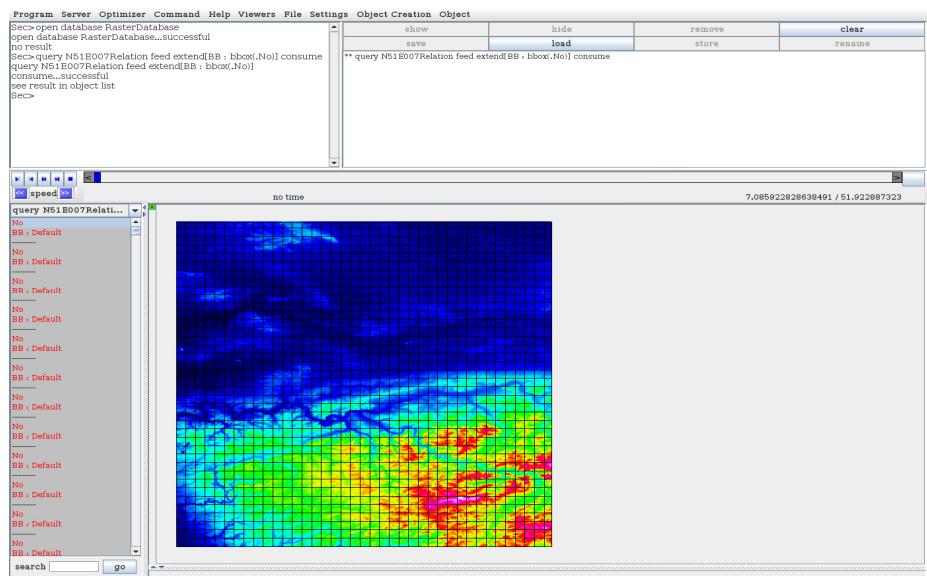


Figure 2.46: Visualisation of all tiles of N51E007Relation and corresponding bounding boxes

29. To get more information about the SECONDO system and available algebras, types and operations, execute the following commands at the command interface in the SECONDO GUI:

```
list algebras
list algebra Raster2Algebra
list algebra TileAlgebra
list type constructors
list operators
```

It is also possible to find a particular operator, for example operator **atlocation**, by command

```
query SEC2OPERATORINFO feed filter[.Name contains "atlocation"] consume
```

30. Terminate SECONDO GUI by main menu command Program → Exit, SECONDO optimizer by command quit in the optimizer shell window and SECONDO Monitor by command quit followed by yes in the SECONDO Monitor shell window.

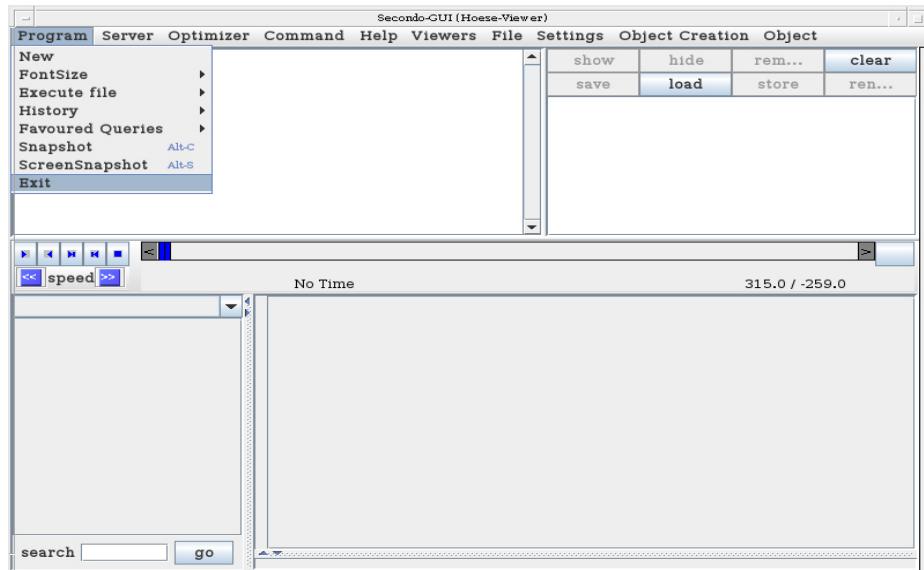


Figure: 2.47: Termination of SECONDO GUI

```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
WARNING: Maximum stack size for global stack is 128 MB
execute query: secondo('close database')
Command failed with error code : 6
and error message
Secondo: No database open.

Secondo: No database open.
no solution found for 'secondo('close database')' /l

opt-server > connection ended normally

opt-server >
bye client
number of clients is :0

opt server > quit
execute query: halt( )
#
# A fatal error has been detected by the Java Runtime Environment:
#
# SIGSEGV (0xb) at pc=0xb7790869, pid=6747, tid=3062012784
#
# JRE version: 6.0_27 b27
# Java VM: Java HotSpot(TM) Server VM (20.0-b12 mixed mode linux-x86 )
# Derivative: IcedTea-1.12.4
# Distribution: Dummy Product (i586), package suse-32.1-i386
# Problematic frame:
# C [libpthread.so.0+0xb69] pthread_kill+0x9
#
# An error report file with more information is saved as:
# ./home/secondo/secondo/Optimizer/hs_err_pid6747.log
#
# If you would like to submit a bug report, please include
# instructions how to reproduce the bug, and visit:
# http://java.sun.com/webapps/bugzilla/
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
./StartOptServer: Zeile 113: 6747 Abgebrochen      $cmd
secondo@linux-e6ch:~/secondo/Optimizer>
```

Figure 2.48: Termination of Optimizer Server

```
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
Launching Checkpoint service ... Starting Process:
Program: SecondoCheckpoint
Args: "/home/secondo/secondo/bin/SecondoConfig.ini"
completed.
launching Secondo Registrar ... Starting Process:
Program: SecondoRegistrar
Args: "/home/secondo/secondo/bin/SecondoConfig.ini"
completed.

Secondo Monitor ready for operation.
Type 'HELP' to get a list of available commands.
Startup in progress ... Starting Process:
Program: SecondoListener
Args: "/home/secondo/secondo/bin/SecondoConfig.ini"
completed.
SEC_MON Starting Process:
Program: SecondoDBB
Args: "-srw" "/home/secondo/secondo/bin/SecondoConfig.ini" --socket=5
Redirecting server output to file SecondoServer.msg
Starting Process:
Program: SecondoDBB
Args: "-srw" "/home/secondo/secondo/bin/SecondoConfig.ini" --socket=5
Redirecting server output to file SecondoServer.msg
quit
Really shutdown the system and quit (confirm with 'y' or 'yes')?
SEC_MON> yes
*** Signal SIGTERM (15) caught! Calling default signal handler ...
Shutdown in progress ... completed.
Secondo Listener terminated with return code 0.

Terminating Secondo Monitor ...
*** Signal SIGTERM (15) caught! Calling default signal handler ...
Terminating Secondo Registrar ... completed.
Secondo Registrar terminated with return code 0.
Terminating Checkpoint Service ... completed.
Checkpoint service terminated with return code 256.
SecondoMonitor terminated.
secondo@linux-e6ch:~/secondo/bin>
```

Figure 2.49: Termination of SecondoMonitor

## 3 Reference Manual

### 3.1 Tile algebra data types

The following chapter describes all Tile algebra data types.  
These data types can be classified into four data type categories:

grid data types  
spatial data types  
moving spatial data types  
instant spatial data types

#### 3.1.1 Grid data types

Tile algebra contains the following grid data types:

*tgrid*  
*mtgrid*

Data type *tgrid* represents a 2-dimensional grid definition with x-origin, y-origin and the length of a grid cell.

Data type *mtgrid* represents a 3-dimensional grid definition with x-origin, y-origin, length of a grid cell and a *duration* value for the time dimension.

Operator **toraster2** is executable on grid data types.

### 3.1.2 Spatial data types

Spatial data types contain values characterized by a x-coordinate and a y-coordinate. Square size of x-dimension and y-dimension of a spatial data types is restricted by the size of a memory page.

Tile algebra contains the following spatial data types:

*tint*  
*treal*  
*tbool*  
*tstring*

Data type *tint* represents a tile of integer values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value and a maximum value.

Data type *treal* represents a tile of double values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value and a maximum value.

Data type *tbool* represents a tile of char values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value and a maximum value.

Data type *tstring* represents a tile of string values of size 31 x 31 which are stored optimized as unique strings in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value and a maximum value.

Operators **atlocation**, **astrange**, **bbox**, **CELL1**, **CELL2**, **CELLS**, **getgrid**, **map**, **map2**, **matchgrid**, **maximum**, **minimum** and **t2mt** are executable on spatial data types.

### 3.1.3 Moving spatial data types

Moving spatial data types contain values characterized by a x-coordinate, a y-coordinate and a time-coordinate. Square size of x-dimension and y-dimension of a moving spatial data types is restricted by the size of a memory page. Size of time-dimension of a moving spatial data type has the constant value of 10 time units.

Tile algebra contains the following moving spatial data types:

*mtint*  
*mtreal*  
*mtbool*  
*mtstring*

Data type *mtint* represents a tile of integer values of size 31 x 31 x 10 which are stored in a Flob and contains a *mtgrid* object as 3-dimensional grid definition, a minimum value and a maximum value.

Data type *mtreal* represents a tile of double values of size 31 x 31 x 10 which are stored in a Flob and contains a *mtgrid* object as 3-dimensional grid definition, a minimum value and a maximum value.

Data type *mtbool* represents a tile of char values of size 31 x 31 x 10 which are stored in a Flob and contains a *mtgrid* object as 3-dimensional grid definition, a minimum value and a maximum value.

Data type *mtstring* represents a tile of string values of size 31 x 31 x 10 which are stored optimized as unique strings in a Flob and contains a *mtgrid* object as 3-dimensional grid definition, a minimum value and a maximum value.

Operators **atinstant**, **atlocation**, **atperiods**, **atrangle**, **bbox**, **CELL1**, **CELL2**, **CELLS**, **deftime**, **getgrid**, **maximum**, **map**, **map2**, **matchgrid** and **minimum** are executable on moving spatial data types.

### 3.1.4 Instant spatial data types

Instant spatial data types contain values characterized by a x-coordinate and a y-coordinate at corresponding instant value. Square size of x-dimension and y-dimension of an instant spatial data types is restricted by the size of a memory page.

Tile algebra contains the following instant spatial data types:

*itint*  
*itreal*  
*itbool*  
*itstring*

Data type *itint* represents a tile of integer values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value, a maximum value and a corresponding instant value.

Data type *itreal* represents a tile of double values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value, a maximum value and a corresponding instant value.

Data type *itbool* represents a tile of char values of size 31 x 31 which are stored in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value, a maximum value and a corresponding instant value.

Data type *itstring* represents a tile of string values of size 31 x 31 which are stored optimized as unique strings in a Flob and contains a *tgrid* object as 2-dimensional grid definition, a minimum value, a maximum value and a corresponding instant value.

Operators **inst** and **val** are executable on instant spatial data types.

## 3.2 Tile algebra operators

The following chapter describes all Tile algebra operators.

### 3.2.1 atinstant

Operator **atinstant** returns the values of a moving spatial data type object for an instant time point in the form of an instant spatial data type object.

syntax:

\_ atinstant \_

signatures:

$\underline{mtT} \times \underline{\text{instant}} \rightarrow \underline{itT}$  for  $T$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$

example:

```
query [const mtint value ((0.0 0.0 1.0 2.0) (2 2 2) (0 0 0 (0  
1 2 3 4 5 6 7)))] atinstant create_instant(2.0);
```

### 3.2.2 atlocation

Operator **atlocation** returns the value at a location point of a spatial data type object or the value(s) at a location point of a moving spatial data type object.

syntax:

\_ atlocation [\_, \_]

signatures:

$tT \times \underline{point} \rightarrow T$	for $T$ in { <u>int</u> , <u>real</u> , <u>bool</u> , <u>string</u> }
$\underline{mt}T \times \underline{point} \rightarrow \underline{m}T$	for $T$ in { <u>int</u> , <u>real</u> , <u>bool</u> , <u>string</u> }
$\underline{mt}T \times \underline{point} \times \underline{instant} \rightarrow T$	for $T$ in { <u>int</u> , <u>real</u> , <u>bool</u> , <u>string</u> }

examples:

```
query [const tint value ((0.0 0.0 1.0) (2 2) (0 0 (0 1 2 3))  
(29 29 (4 5 6 7)))] atlocation [[const point value (1.5  
1.5)]];  
  
query [const mtint value ((0.0 0.0 1.0 1.0) (2 2 2) (0 0 0 (0  
1 2 3 4 5 6 7)))] atlocation [[const point value (1.5 1.5)]];  
  
query [const mtint value ((0.0 0.0 1.0 1.0) (2 2 1) (0 0 0 (0  
1 2 3)) (29 29 9 (4 5 6 7)))] atlocation [[const point value  
(1.5 1.5)], [const instant value "2000-01-03-00:00:00.001"]];
```

### 3.2.3 atperiods

Operator **atperiods** restricts the values of a moving spatial data type object to given periods.

syntax:

atperiods(\_)

signatures:

$\underline{mt}T \times \underline{periods} \rightarrow \underline{m}T$	for $T$ in { <u>int</u> , <u>real</u> , <u>bool</u> , <u>string</u> }
---	---

example:

```
query [const mtint value ((2.0 2.0 1.0 2.0) (2 2 1) (0 0 0 (0  
1 2 3)) (29 29 9 (4 5 6 7)))] atperiods [const periods value  
(("2000-01-03" "2000-01-05" TRUE FALSE))];
```

### 3.2.4 atrange

Operator **atrange** returns the values at range of a rectangle of a spatial data type object, the values at range of a rectangle of a moving spatial data type object or the values at range of a rectangle, a begin instant value and an end instant value of a moving spatial data type object.

syntax:

$\_ \text{atrange} [\_, \_, \_]$

signatures:

$xT \times \underline{\text{rect}} \rightarrow xT$  for  $x$  in  $\{\underline{t}, \underline{mt}\}$ ,  $T$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$   
 $\underline{mt}T \times \underline{\text{rect}} \times \underline{\text{instant}} \times \underline{\text{instant}} \rightarrow \underline{mt}T$  for  $T$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$

examples:

```
query [const tint value ((2.0 2.0 1.0) (2 2) (0 0 (0 1 2 3))  
(29 29 (4 5 6 7)))] atrange [[const rect value (3.0 31.0 3.0  
31.0)]];  
  
query [const mtint value ((2.0 2.0 1.0 1.0) (2 2 1) (0 0 0 (0  
1 2 3)) (29 29 9 (4 5 6 7)))] atrange [[const rect value (3.0  
31.0 3.0 31.0)]];  
  
query [const mtint value ((2.0 2.0 1.0 1.0) (2 2 1) (0 0 0 (0  
1 2 3)) (29 29 9 (4 5 6 7)))] atrange [[const rect value (2.5  
31.5 2.5 31.5)], [const instant value "2000-01-03"], [const  
instant value "2000-01-04"]];
```

### 3.2.5 bbox

Operator **bbox** returns the bounding box rectangle of a spatial data type object or the bounding box rectangle of a moving spatial data type object.

syntax:

`bbox(_)`

signatures:

$\underline{t}T \rightarrow \underline{rect}$  for  $T$  in  $\{\underline{int}, \underline{real}, \underline{bool}, \underline{string}\}$   
 $\underline{mt}T \rightarrow \underline{rect3}$  for  $T$  in  $\{\underline{int}, \underline{real}, \underline{bool}, \underline{string}\}$

examples:

```
query bbox([const tint value ((1.1 2.2 1.0) (2 2) (0 0 (0 1 2  
3)) (28 28 (4 5 6 7))))];  
  
query bbox([const mtint value ((1.0 1.0 1.0 1.0) (2 2 1) (0 0  
0 (0 1 2 3)) (8 8 9 (4 5 6 7))))];
```

### 3.2.6 CELL1

Operator **CELL1** is a type mapping operator.

signatures:

$\underline{x}T \times \dots \rightarrow T$  for  $x$  in  $\{t, mt\}$ ,  $T$  in  $\{\underline{int}, \underline{real}, \underline{bool}, \underline{string}\}$

example:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1)))] map  
[fun(n: CELL1) n * 2];
```

### 3.2.7 CELL2

Operator **CELL2** is a type mapping operator.

signatures:

$\underline{x}T \times \underline{y}U \times \dots \rightarrow U$  for  $x, y$  in  $\{\underline{t}, \underline{mt}\}$ ,  $T, U$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$

example:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ]
[const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ] map2
[fun(n: int, m: CELL2) n + m];
```

### 3.2.8 CELLS

Operator **CELLS** is a type mapping operator.

signatures:

$\underline{x}T \times \dots \rightarrow \underline{\text{rel}}(\underline{\text{tuple}}([Elem : T]))$  for  $x$  in  $\{\underline{t}, \underline{mt}\}$ ,  $T$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$

example:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ]
matchgrid[[const tgrid value (0.0 0.0 0.5)], fun(c: CELLS) c
feed avg[Elem], FALSE];
```

### 3.2.9 compose

Operator **compose** merges a *mpoint* object and a spatial data type object into a moving data type object.

syntax:

$\_ \text{compose} \_ \text{ }$

signatures:

$\text{mpoint} \times \underline{x}T \rightarrow \underline{m}T$  for  $x$  in  $\{t, \underline{mt}\}$ ,  $T$  in  $\{\text{int}, \text{real}, \text{bool}, \text{string}\}$

example:

```
query [const mpoint value (((("2000-01-03-00:00:00.000" "2000-01-05-00:00:00.000" TRUE FALSE) (0.5 0.5 1.5 1.5)) ((("2000-01-05-00:00:00.000" "2000-01-07-00:00:00.000" TRUE FALSE) (10.5 10.5 10.0 10.0)) ((("2000-01-07-00:00:00.000" "2000-01-09-00:00:00.000" TRUE FALSE) (1.0 1.0 0.5 0.5))))] compose [const tint value ((0.0 0.0 1.0) (2 2) (0 0 (0 1 2 3)) (2 2 (4 5 6 7)))];
```

### 3.2.10 deftime

Operator **deftime** returns the defined periods of a moving spatial data type object.

syntax:

$\text{deftime}(\_)$

signatures:

$\underline{m}T \rightarrow \underline{periods}$  for  $T$  in  $\{\text{int}, \text{real}, \text{bool}, \text{string}\}$

example:

```
query deftime([const mtint value ((0.0 0.0 1.0 1.0) (1 1 1) (0 0 0 (1)) (1 1 1 (1))))];
```

### 3.2.11 fromline

Operator **fromline** creates a stream of tbool objects from a line object and a tgrid object.

syntax:

fromline( \_, \_ )

signature:

line × tgrid → stream(tbool)

example:

```
query fromline([const line value ((64.0 14.2 194.0 44.2))],  
[const tgrid value (10.0 10.0 2.0)]) count;
```

### 3.2.12 fromregion

Operator **fromregion** creates a stream of tbool objects from a region object and a tgrid object.

syntax:

fromregion( \_, \_ )

signature:

region × tgrid → stream(tbool)

example:

```
query fromregion([const region value (((0.0 1.0) (1.0 0.0)  
(2.0 0.0) (3.0 1.0) (3.0 2.0) (2.0 3.0) (1.0 3.0) (0.0 2.0)  
(0.0 1.0)))), [const tgrid value (0.0 0.0 1.0)]) count
```

### 3.2.13 getgrid

Operator **getgrid** returns the *tgrid* object of a spatial data type object or the *mtgrid* object of a moving spatial data type object.

syntax:

getgrid(\_)

signatures:

<i>lT</i> → <i>tgrid</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
<i>mtT</i> → <i>mtgrid</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }

examples:

```
query getgrid([const tint value ((0.0 1.0 2.0) (2 2) (0 0 (0 1  
2 3)) (29 29 (4 5 6 7))))];  
  
query getgrid([const mtint value ((0.0 1.0 2.0 3.0) (2 2 1) (0  
0 0 (0 1 2 3)) (29 29 9 (4 5 6 7))))];
```

### 3.2.14 inst

Operator **inst** returns the instant value of an instant spatial data type object.

syntax:

inst(\_)

signatures:

<i>itT</i> → <i>instant</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
-----------------------------	--

example:

```
query inst([const itint value ((instant "2013-05-13") ((0.0  
0.0 1.0) (2 2) (0 0 (0 1 2 3)) (29 29 (4 5 6 7))))];
```

### 3.2.15 map

Operator **map** maps a spatial data type object to another spatial data type object or a moving spatial data type object to another moving spatial data type object.

syntax:

$\_ \text{map}[\_]$

signatures:

$\underline{x}T \times (T \rightarrow U) \rightarrow \underline{x}U$  for  $x$  in  $\{\underline{t}, \underline{mt}\}$ ,  $T, U$  in  $\{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$

example:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1)))] map  
[. * 2];
```

### 3.2.16 map2

Operator **map2** combines a spatial data type object and a spatial data type object to a new spatial data type object, a spatial data type object and a moving spatial data type object to a new moving spatial data type object, a moving spatial data type object and a spatial data type object to a new moving spatial data type object or a moving spatial data type object and a moving spatial data type object to a new moving spatial data type object.

syntax:

`-- map2[ ]`

signatures:

$\underline{t}T \times \underline{t}U \times (T \times U \rightarrow V) \rightarrow \underline{t}V$	for $T, U, V$ in <code>{int, real, bool, string}</code>
$\underline{t}T \times \underline{mt}U \times (T \times U \rightarrow V) \rightarrow \underline{mt}V$	for $T, U, V$ in <code>{int, real, bool, string}</code>
$\underline{mt}T \times \underline{t}U \times (T \times U \rightarrow V) \rightarrow \underline{mt}V$	for $T, U, V$ in <code>{int, real, bool, string}</code>
$\underline{mt}T \times \underline{mt}U \times (T \times U \rightarrow V) \rightarrow \underline{mt}V$	for $T, U, V$ in <code>{int, real, bool, string}</code>

examples:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ]
[const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ] map2
[fun(n: int, m: int) n + m];

query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1))) ]
[const mtint value ((0.0 0.0 1.0 1.0) (1 1 1) (0 0 0 (1)) (0 0
9 (1))) ] map2 [fun(n: int, m: int) n + m];

query [const mtint value ((0.0 0.0 1.0 1.0) (1 1 1) (0 0 0
(1)) (0 0 9 (1))) ] [const tint value ((0.0 0.0 1.0) (1 1) (0 0
(1))) ] map2 [fun(n: int, m: int) n + m];

query [const mtint value ((0.0 0.0 1.0 1.0) (1 1 1) (0 0 0
(1)) (0 0 9 (1))) ] [const mtint value ((0.0 0.0 1.0 1.0) (1 1
1) (0 0 0 (1)) (0 0 9 (1))) ] map2 [fun(n: int, m: int) n + m];
```

### 3.2.17 matchgrid

Operator **matchgrid** applies a user function to a spatial data type object or a moving spatial data type object.

syntax:

$\_ \text{matchgrid} [\_, \_, \_]$

signatures:

$$\begin{aligned} \underline{t}T \times \underline{tgrid} \times (\underline{\text{rel}}(\underline{\text{tuple}}([Elem : T])) \rightarrow U) \times \underline{\text{bool}} \rightarrow \underline{t}U \\ \text{for } T, U \text{ in } \{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\} \\ \underline{mt}T \times \underline{mtgrid} \times (\underline{\text{rel}}(\underline{\text{tuple}}([Elem : T])) \rightarrow U) \times \underline{\text{bool}} \rightarrow \underline{mt}U \\ \text{for } T, U \text{ in } \{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\} \end{aligned}$$

examples:

```
query [const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1)))]
matchgrid[[const tgrid value (0.0 0.0 0.5)], fun(cell:
    rel(tuple([Elem : int]))) cell feed avg[Elem], FALSE];

query [const mtint value ((0.0 0.0 1.0 2.0) (1 1 1) (0 0 0
(1)))] matchgrid[[const mtgrid value (0.0 0.0 0.5 (duration (1
0)))], fun(cell: rel(tuple([Elem : int]))) cell feed
avg[Elem], FALSE];
```

### 3.2.18 maximum

Operator **maximum** returns the maximum value of a spatial data type object or a moving spatial data type object.

syntax:

$\text{maximum}(\_)$

signatures:

$$\underline{x}T \rightarrow T \quad \text{for } x \text{ in } \{\underline{t}, \underline{mt}\}, T \text{ in } \{\underline{\text{int}}, \underline{\text{real}}, \underline{\text{bool}}, \underline{\text{string}}\}$$

examples:

```
query maximum([const tint value ((0.0 0.0 1.0) (2 2) (0 0 (0 1
2 3)) (29 29 (4 5 6 7))))];

query maximum([const mtint value ((0.0 0.0 1.0 1.0) (2 2 1) (0
0 0 (0 1 2 3)) (29 29 9 (4 5 6 7))));
```

### 3.2.19 minimum

Operator **minimum** returns the minimum value of a spatial data type object or a moving spatial data type object.

syntax:

`minimum(_)`

signatures:

$\underline{x}T \rightarrow T$  for  $x$  in  $\{t, \underline{mt}\}$ ,  $T$  in  $\{\underline{int}, \underline{real}, \underline{bool}, \underline{string}\}$

examples:

```
query minimum([const tint value ((0.0 0.0 1.0) (2 2) (0 0 (0 1  
2 3)) (29 29 (4 5 6 7))));
```

```
query minimum([const mtint value ((0.0 0.0 1.0 1.0) (2 2 1) (0  
0 0 (0 1 2 3)) (29 29 9 (4 5 6 7))));
```

### 3.2.20 t2mt

Operator **t2mt** adds a time dimension to a spatial data type object and returns resulting moving spatial data type object.

syntax:

`t2mt(_, _, _, _)`

signatures:

$\underline{t}T \times \underline{duration} \times \underline{instant} \times \underline{instant} \rightarrow \underline{mt}T$  for  $T$  in  $\{\underline{int}, \underline{real}, \underline{bool}, \underline{string}\}$

example:

```
query t2mt([const tint value ((0.0 0.0 1.0) (1 1) (0 0 (1)))),  
[const duration value (0 43200000)], [const instant value  
"2000-01-03"], [const instant value "2000-01-04"]);
```

### 3.2.21 tiles

Operator **tiles** converts a Raster2 algebra *grid2* object to a Tile algebra *tgrid* object, a Raster2 algebra *grid3* object to a Tile algebra *mtgrid* object, a Raster2 algebra spatial data type object to a *stream* of Tile algebra spatial data type objects, a Raster2 algebra moving spatial data type object to a *stream* of Tile algebra moving spatial data type objects or a Raster2 algebra instant spatial data type object to a stream of Tile algebra instant spatial data type objects.

syntax:

`tiles(_)`

signatures:

<i>grid2</i> → <i>tgrid</i>	
<i>grid3</i> → <i>mtgrid</i>	
<i>sT</i> → <i>stream(tT)</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
<i>msT</i> → <i>stream(mtT)</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
<i>isT</i> → <i>stream(itT)</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }

examples:

```
query tiles([const grid2 value (1.2 3.4 5.6)]);  
  
query tiles([const grid3 value (1.2 3.4 5.6 (duration (7  
0))))];  
  
query tiles([const sint value ((0.0 0.0 1.0) (2 2) (0 0 (0 1 2  
3)) (60 60 (4 5 6 7))))] count;  
  
query tiles([const msint value ((0.0 0.0 1.0 1.0) (2 2 1) (0 0  
0 (0 1 2 3)) (60 60 9 (4 5 6 7))))] count;  
  
query tiles([const isint value ((instant "2013-05-13") ((0.0  
0.0 1.0) (2 2) (0 0 (0 1 2 3)) (60 60 (4 5 6 7))))] count;
```

### 3.2.22 toraster2

Operator **toraster2** converts a Tile algebra *tgrid* object to a Raster2 algebra *grid2* object, a Tile algebra *mtgrid* object to a Raster2 algebra *grid3* object, a *stream* of Tile algebra spatial data type objects to a Raster2 algebra spatial data type object, a *stream* of Tile algebra moving spatial data type objects to a Raster2 algebra moving spatial data type object or a *stream* of Tile algebra instant spatial data type objects to a Raster2 algebra instant spatial data type object.

syntax:

toraster2(\_)

signatures:

<i>tgrid</i> → <i>grid2</i>	
<i>mtgrid</i> → <i>grid3</i>	
<i>stream</i> ( <i>tT</i> ) → <i>sT</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
<i>stream</i> ( <i>mtT</i> ) → <i>msT</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }
<i>stream</i> ( <i>itT</i> ) → <i>isT</i>	for <i>T</i> in { <i>int</i> , <i>real</i> , <i>bool</i> , <i>string</i> }

examples:

```
query toraster2([const tgrid value (1.2 3.4 5.6)]);  
  
query toraster2([const mtgrid value (1.2 3.4 5.6 (duration (7  
0))))]);  
  
query toraster2(tiles([const sint value ((-10.0 -10.0 1.0) (1  
1) (0 0 (0)) (31 0 (1)) (0 31 (2)) (61 61 (3))))];  
  
query toraster2(tiles([const msint value ((-10.0 -10.0 1.0  
1.0) (1 1 1) (0 0 0 (0)) (10 0 0 (1)) (0 10 0 (2)) (19 19 9  
(3))))];  
  
query toraster2(tiles([const isint value ((instant "2013-05-  
13") ((-10.0 -10.0 1.0) (1 1) (0 0 (0)) (31 0 (1)) (0 31 (2))  
(61 61 (3))))]);
```

### 3.2.23 toregion

Operator **toregion** maps a *tbool* object to a *region* object.

syntax:

\_ toregion

signature:

*tbool* → *region*

### 3.2.24 val

Operator **val** returns the spatial data type object of an instant spatial data type object.

syntax:

val(\_)

signatures:

*itT* → *tT*    for *T* in {*int*, *real*, *bool*, *string*}

example:

```
query val([const itint value ((instant "2013-05-13") ((0.0 0.0  
1.0) (2 2) (0 0 (0 1 2 3)) (29 29 (4 5 6 7))))];
```



## References

- [BT12] Betreuerteam, Rasterdaten in Secondo, Aufgabenbeschreibung Phase 2, FernUniversität Hagen, 2012
- [G11] Ralf Hartmut Güting, A Short Guide to Writing Executable Queries in SECONDO, FernUniversität Hagen, 2011
- [GAB+12] Ralf Hartmut Güting, Dirk Ansorge, Thomas Behr, Christian Düntgen, Simone Jandt, Markus Spiekermann, SECONDO User Manual, Version 3.1, FernUniversität Hagen, 2012
- [GBA+04] Ralf Hartmut Güting, Thomas Behr, Victor Teixeira de Almeida, Zhiming Ding, Frank Hoffmann, Markus Spiekermann, SECONDO: An Extensible DBMS Architecture and Prototype, FernUniversität Hagen, Informatik-Report 313, 2004
- [GBJ+12] Ralf Hartmut Güting, Thomas Behr, Simone Jandt, Fabio Valdés, Fachpraktikum Erweiterbare Datenbanksysteme, Wintersemester 2012/13, Aufgabe der Phase 2 „Rasterdaten”, FernUniversität Hagen, 2012
- [SEC14] SECONDO-Homepage, 2014  
<http://dna.fernuni-hagen.de/Secondo.html/index.html>
- [ST11] SECONDO Team, A Database System for Moving Objects, FernUniversität Hagen, 2011