

# Getting Started



# Building and Using Queries



with

**TNTmips®**

**TNTedit™**

**TNTview®**

---

# Before Getting Started

This booklet introduces you to the use of database queries in the TNT products and shows you how to construct queries to utilize the attribute information attached to your vector, CAD, and TIN objects. The query process may seem complex at first, but this series of step-by-step exercises leads you through the required structure and syntax of queries, progressing from simple one-line examples to queries with multiple condition statements and processing loops.

**Prerequisite Skills** This booklet assumes that you have completed the exercises in *Getting Started: Displaying Geospatial Data* and *Getting Started: Navigating*. Those exercises introduce essential concepts and skills that are not covered again here. Please consult those booklets and the TNTmips reference manual for any review you need.

**Sample Data** The exercises presented in this booklet use sample data that is distributed with the TNT products. If you do not have access to a TNT products CD, you can download the data from MicroImages' web site. In particular, this booklet uses sample files in the QUERY data collection. Make sure that you have a copy of the sample data on your hard drive so changes can be saved when you use the objects in these files.

**More Documentation** This booklet is intended only as an introduction to the use of database queries. Consult the section entitled "Display Vector, CAD, and TIN by Query" in the Display volume of the TNTmips reference manual for more information on the Query Editor, and Appendix 2: Database Queries for a complete reference on query commands and functions.

**TNTmips® and TNTlite®** TNTmips comes in two versions: the professional version and the free TNTlite version. This booklet refers to both versions as "TNTmips." If you did not purchase the professional version (which requires a hardware key), TNTmips operates in TNTlite mode, which limits object size and does not allow export.

Database queries can also be used to control the display of geospatial objects in TNTview, and to select elements for editing in TNTedit. All exercises in this booklet can be completed in TNTlite using the sample geodata provided.

*Randall B. Smith, Ph.D., 17 September 2001*

It may be difficult to identify the important points in some illustrations without a color copy of this booklet. You can print or read this booklet in color from MicroImages' web site. The web site is also your source for the newest Getting Started booklets on other topics. You can download an installation guide, sample data, and the latest version of TNTlite.

**<http://www.microimages.com>**

---

# Welcome to Building and Using Queries

TNTmips gives you great flexibility to use the database attributes of vector, CAD, and TIN objects to control the display and printing of the object, or to select elements for use in various processes. Database queries provide the most complete and versatile means to utilize this attribute information.

A **database query** is a set of instructions defining attribute criteria that are used to select records from a database. The specific spatial elements (such as lines or polygons) to which those records are attached are then automatically selected for the current process. A query applies to a specific element type, and you can simultaneously use separate queries for different element types in an object. In Spatial Data Display, the Style By Script option allows you to specify display parameters for selected elements on the basis of attributes. The attribute information you refer to in queries can be qualitative (such as a class name), or quantitative (such as crop yield values).

Queries must use a standard “grammar and syntax” that TNTmips understands. The query language used is a subset of the TNTmips Spatial Manipulation Language (SML). You compose queries in the Query Editor window, which has menus that simplify construction of valid queries by letting you choose fields from the available database tables and insert symbols and functions from list windows that outline the correct syntax. The Query Editor also provides a syntax checker to help you find errors before applying the query.

The exercises in this booklet use queries to select or style elements in a vector object for display. Keep in mind, though, that queries can be used in any process that selects component elements in vector, CAD, or TIN objects. In addition, you can use queries in some raster processes to select cell values for processing.



## STEPS

- make sure that you have a copy of the sample data in the QUERY data collection on your hard drive
- launch TNTmips
- launch the Spatial Data Display process and select New 2D Group from the toolbar

The exercises on pages 4-9 introduce the structure of simple query statements, comparison operators, and some useful tools for building and checking queries. Pages 10-17 cover query structures involving compound statements, the use of variables and comments, and using queries to check database record attachments. Styling by Script is introduced on pages 18-19, along with the use of conditional “if-then-else” constructions. Examples of queries based on the spatial and topological attributes of vector objects are found on pages 20-23. Pages 24-25 show scripts to create computed fields in database tables, while pages 26-27 provide examples of queries that are useful in editing vector objects.

# Select by Querying a Single Field

## STEPS

- click the Add Vector icon button and select Add Vector Layer from the dropdown menu
- select the CBSOILS\_LITE object from the CB\_SOILQ Project File in the QUERY data collection
- in the Vector Object Display Controls window, check that the Select option on the Lines tabbed panel is set to All
- set the Select option on the Labels panel to None
- check that the Style option on the Polygons panel is set to By Attribute



The simplest form of query selects a specific type of spatial element (such as polygons, lines, or points in a vector object) on the basis of the values for a single database attribute. In this exercise you enter a simple query that selects soil map polygons in a vector object for display. Each soil type has associated values for maximum potential yield for several crops. The query selects polygons for which the potential crop yield for wheat is greater than 35 bushels per acre. The query statement has the form:

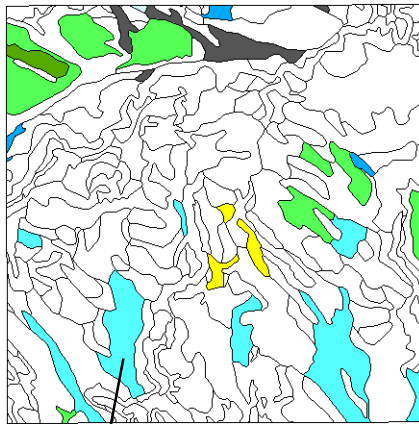
Attribute	Comparison Operator	Value
-----------	---------------------	-------

The query must specify which database table contains the attribute information, and in which field it is found. This “attribute location” information must be entered in the form TABLE.FIELD. The value in

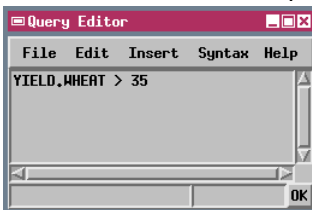
this example is a simple numeric value, and the comparison operator is the “Greater than” operator (>).



- set the Select option on the Polygons panel to By Query and click on the adjacent Specify button
- in the Query Editor window, type the following text exactly (including capitalization):  
YIELD.WHEAT > 35
- click [OK] in the Query Editor window



Since line selection was set to All, the outlines of all soil polygons are drawn. The polygons selected by the query are filled with colors and fill patterns that are based on the soil type and that were previously set up for display By Attribute. Several soil types meet the wheat yield selection criterion. Unselected polygons remain unfilled.




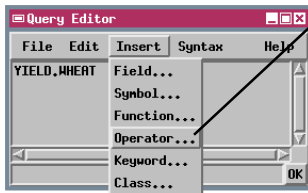
- click [OK] in the Vector Object Display Controls window to accept the display settings and display the vector object

# Using the Insert Operator Option

The previous query selected soil polygons belonging to several soil type classes. Let's refine the selection criterion so that the query selects only those polygons with a potential wheat yield of exactly 38 bushels per acre. To specify this selection criterion, use the "Equal to" operator (==, double equal sign) in the query statement. The Insert / Operator menu option in the Query Editor opens the Insert Operator window, which lets you choose the operator from a scrolled list and insert it into the query statement at the current cursor location.

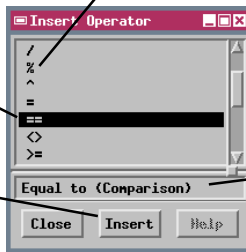
## STEPS

- click on the Vector icon in the Layer icon row to open the Vector Object Display Controls window 
- click on Select: [Specify] in the Polygon Options panel
- highlight "> 35" in the Query Editor window and press <Delete>
- open the Insert menu and select Operator
- in the Insert Operator window, scroll down and select the "==" operator; click [Insert], then [Close]
- in the Query Editor window, type 38 to the right of the operator, then click [OK]
- click [OK] in the Vector Object Display Controls window to accept the display settings and display the vector object



Choose Operator from the Insert menu to open the Insert Operator window...

...which lists the available query operators.

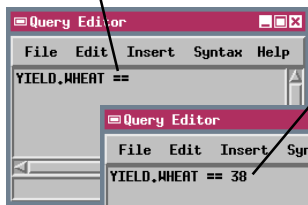


Select the "Equal to" operator.

Click [Insert]...

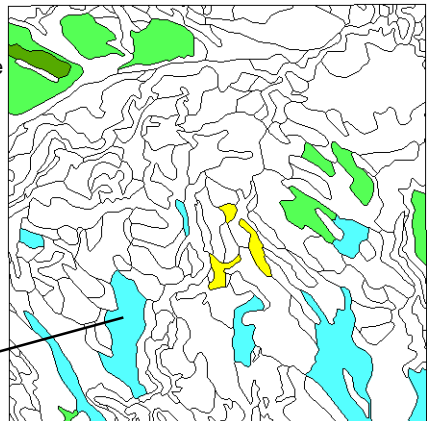
...to enter the operator in the query statement.

The text field describes the function of the selected operator.



Type the value to be matched.

Fewer soil classes meet the more restrictive selection criterion in the revised query statement.



# Using the Insert Field Option

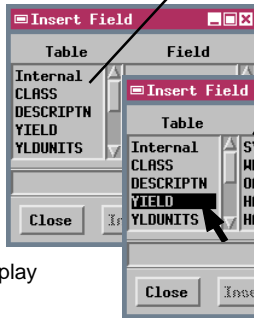
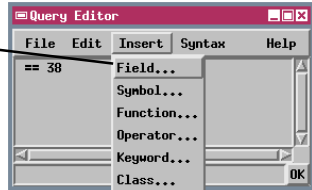
## STEPS

- ☑ open the Vector Object Display Controls window and the Query Editor window
- ☑ select YIELD.WHEAT in the existing query and press <Delete>
- ☑ open the Insert Menu and select Field
- ☑ in the Insert Field window that opens, click on YIELD in the Table list
- ☑ click on OATS in the Field list, then click [Insert]
- ☑ click [Close] on the Insert Field window
- ☑ change the value on the right side of the query statement to 43, then click [OK]
- ☑ click [OK] in the Vector Object Display Controls window

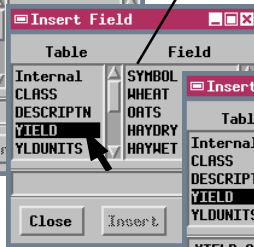
You can also use the Insert/Field menu option in the Query Editor to help construct or modify queries. This option opens the Insert Field window, from which you can choose the Table and Field and automatically insert the attribute location information into your query statement in the correct form.

Choose Field from the Insert menu to open the Insert Field window...

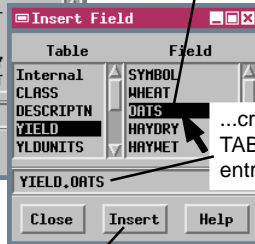
...which lists all available database tables for the selected element type.



Choosing a Table also displays a list of its fields.

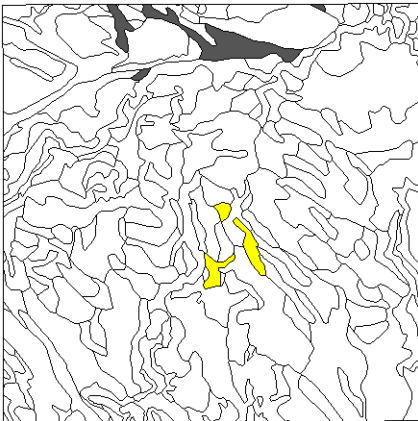


Choosing a field...



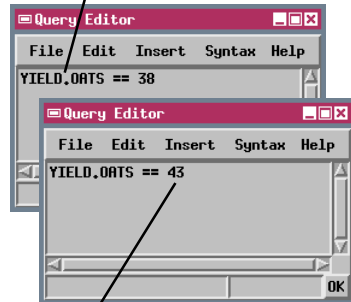
...creates the TABLE.FIELD entry.

This query selects several of the same soil classes selected by the first query (page 4).



Click the Insert button...

... to insert the TABLE.FIELD entry into your query statement.



Change the value on the right side of the statement to 43.

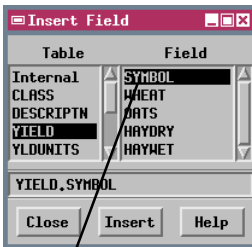
## Querying a String Field

The query language used in TNTmips is case-sensitive. If the table CLASS contains a field called Class, the TABLE.FIELD entry must read CLASS.Class; if you enter it as CLASS.CLASS, the query process will not find the field and will indicate there is an error in the query. Using the Insert Field procedure helps you avoid this type of problem.

The database fields you have used in your queries so far have contained numeric data. The YIELD table for CBSOILS\_LITE also contains a field named SYMBOL with soil type symbols in String format. The term “string” is short for “character string,” which means that the field is not evaluated numerically, and can contain text and other nonnumeric characters. String fields may contain numerals (for example, CLASS1), but they are read as characters rather than as numbers. String values in query statements must be enclosed in double quotes, and are also case-sensitive.

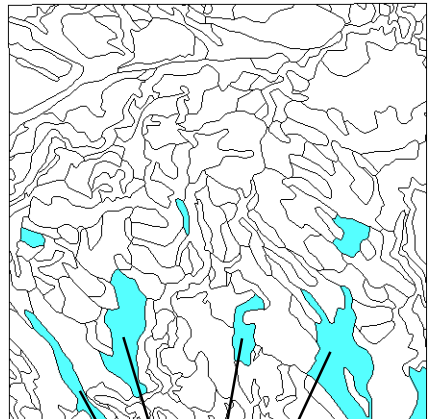
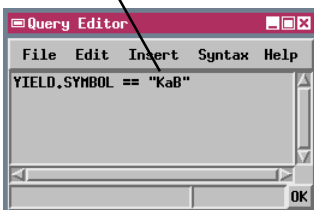
### STEPS

- open the Vector Object Display Controls window and the Query Editor window
- select YIELD.SYMBOL in the existing query and press <Delete>
- use the Insert Field procedure to insert YIELD.SYMBOL on the left side of the query statement
- change the value on the right side of the query statement to "KaB" (including the double quotes), then click [OK]
- click [OK] in the Vector Object Display Controls window



The SYMBOL field contains string values.

Enclose a string value in double quotes.



Selected soil polygons belonging to class KaB.

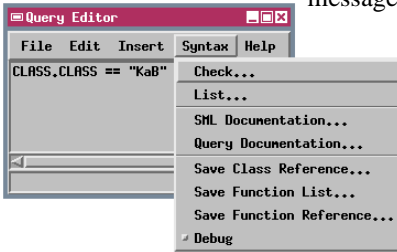


# Checking Query Syntax

## STEPS

- ☑ open the Vector Object Display Controls window and the Query Editor window
- ☑ manually change the left side of the existing query statement to CLASS.CLASS (all capitals)

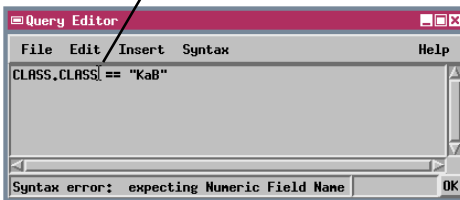
The rules concerning capitalization and use of quotes for string values are examples of the syntax of the TNTmips query language. Query syntax is checked automatically when you click [OK] to execute the query, and when you choose Close from the File menu to close the Query Editor. (If the query contains a syntax error when you try to close the Query Editor, the window remains open and an error message is displayed.)



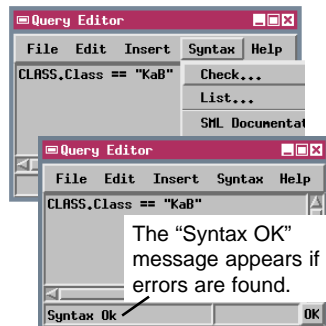
- ☑ choose Check from the Syntax menu
- ☑ note the error message, then click [OK] on the Message window.
- ☑ change the left side of the query statement to CLASS.Class
- ☑ choose Check from the Syntax menu
- ☑ note the "Syntax OK" message

The process starts checking at the beginning of the query. If no syntax errors are encountered, the message line at the bottom of the Query Editor window reads "Syntax OK." If a syntax error is detected, the text cursor is placed at the end of the first component of the statement that the computer could not interpret, and an error message appears in the message line. In this example, the query checker detected that there is no database field named CLASS in table CLASS, so the cursor is placed at the end of the CLASS.CLASS entry. After correcting a syntax error, you can use the Syntax option again to check for errors in the remainder of the query.

In this example the cursor is placed at the end of the invalid TABLE.FIELD entry.



In more complex queries, the syntax error may actually be the result of something missing after the point where the cursor is placed, such as a closing parenthesis or the 'end' statement of a begin / end loop.



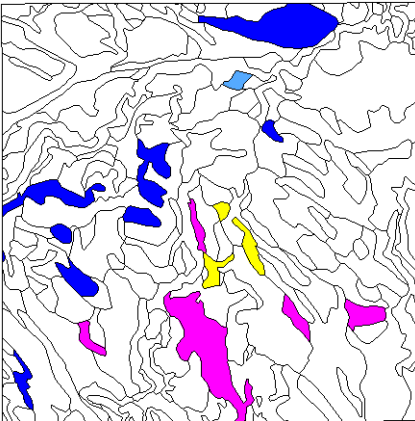


## Using Calculations in Queries

The value on the right side of a query statement can also be provided by a database field or a calculation involving a database field. Calculations in queries can use standard arithmetic operations: addition (+), subtraction (-), multiplication (\*), and division (/). You can enter the operation symbols from the Insert Operator window if you wish. The Insert / Function option provides access to

trigonometric and other mathematical functions that can also be used in query statements. The sample query for this exercise selects soil polygons for which the potential yield for oats is exactly 5 bushels per acre greater than the yield for wheat.

By now you have probably noticed that the last query used for a particular object and element type is automatically stored with the object and is opened the next time you select the same By Query option. If you wish to store several queries for the same object for future use, you can use the Save and Save As options on the Query Editor's File menu. These options allow you to save the query currently displayed in the Query Editor window as a text file with a .QRY file extension, or as a text object in a Project File. To recall a stored query, use the Open option on the File menu.



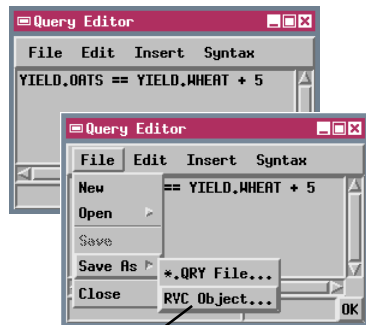
### STEPS

- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query statement:

```
YIELD.OATS == YIELD.WHEAT + 5
```

- choose Save As from the File menu, then \*.QRY File from the menu that opens
- use the standard File Selection window to name a new file to contain the query
- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

Usually a query will work only with a specific object because of a reference to a unique database field. If you have a series of objects with identical database formats, or the query refers only to fields in standard tables created by TNTmips, then you can use the same query for any of the objects.



Choose RVC Object to store the query as a text object in a Project File.

# Compound Queries

## STEPS

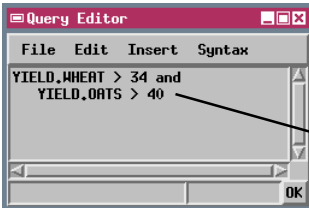
- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

```
YIELD.WHEAT > 34 and YIELD.OATS > 40
```

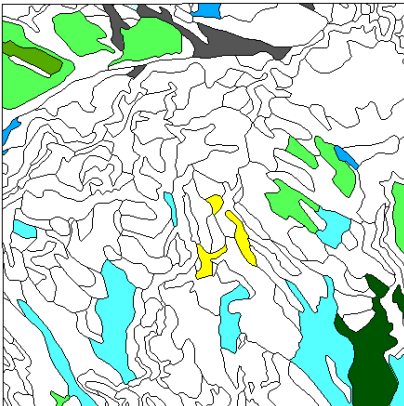
- click [OK] in the Query Editor window and in the Vector Object Display Controls window
- repeat above steps but substitute “or” for “and” in the query statement

Each of the queries used in the previous exercises employs a single selection comparison to choose polygons for display. In many cases you may need to select elements using a combination of several criteria. A series of selection comparisons in a query statement must be related to each other by one or more logical operators from set theory, such as “and,” “or,” and “not.” These operators must be entered in all lowercase letters. You can insert logical operators in query statements using the Insert Operator procedure.

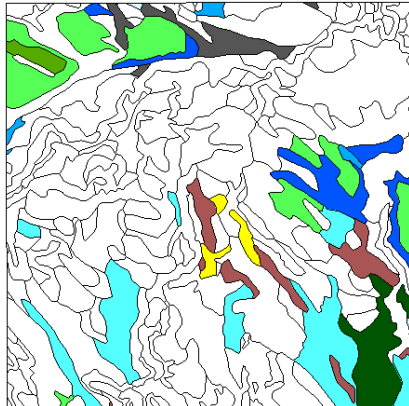
When two comparisons are linked by the logical “and” operator, both comparisons must be true in order to make the entire query statement true and select the element. When two comparisons are linked by the logical “or” operator, the query statement is true if either of the individual comparisons is true. Elements meeting either criterion are selected.



A long query statement may continue onto additional lines, though you may wish to indent subsequent lines to make it clear that they are part of one statement.



Polygons for which potential wheat yield is over 34 bushels per acre, and potential oat yield is over 40 bushels per acre.



Polygons for which either potential wheat yield is over 34 bushels per acre, or potential oat yield is over 40 bushels per acre.

## Using the “not equal to” Operator

Most of the soil types in the Crow Butte area have a higher potential yield for oats than for wheat, but wheat usually brings a higher price than oats when the crop is sold. Let's assume the crop prices per bushel are \$3.25 for oats and \$4.00 for wheat. This example query is used to identify soil types for which the total potential crop price per acre for oats (potential yield in bushels per acre times price per bushel) is greater than or equal to that for wheat.

### STEPS

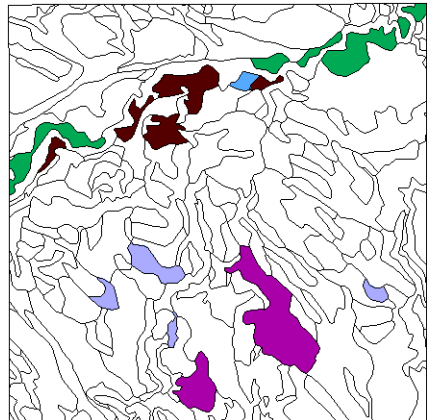
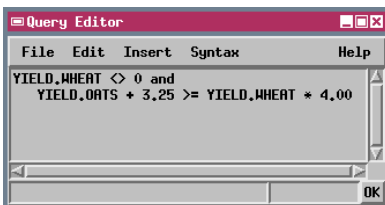
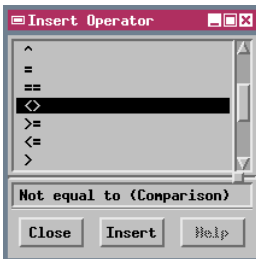
- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

This query is complicated by the fact that the potential crop

```
YIELD.WHEAT <> 0 and
YIELD.OATS * 3.25 >= YIELD.WHEAT * 4.00
```

yield for soil types that cannot be cultivated is 0, and such soil types would satisfy the selection comparison in the second line of the query. The first line of the query excludes the zero-yield soil types, and illustrates the use of the “not equal to” operator (<> or !=). This statement selects polygons for which the potential yield for wheat is not equal to 0. Only these polygons are subjected to the price comparison in the second line.

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window



Polygons for which a crop of oats would bring a higher price per acre than wheat, assuming the potential crop yields and the stated prices per bushel.

# Using Comments and Variables

## STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

```
dollars = 129 # define variable for
              # required crop
              # price per acre

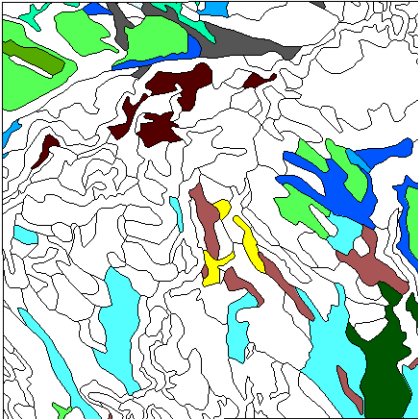
# select polygons based on crop price
YIELD.OATS * 3.25 > dollars or
YIELD.WHEAT * 4.00 > dollars
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

You can enhance the readability and later usefulness of queries by including comments. A comment begins with the “#” symbol, and may be on a line by itself or at the end of a statement. You can use comments within the query to explain individual statements, and an introductory comment to provide an explanation of the intended use of the query and what object it applies to.

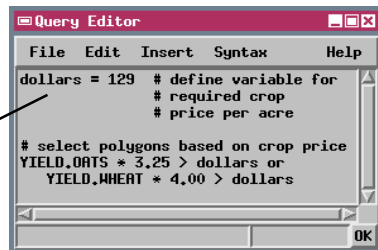
The TNTmips query process also allows you to name and assign values to variables for use in a query. This example query selects soil polygons that exceed a required potential crop price per acre for either oats or

wheat. The first line of the query is an assignment statement that defines a numeric variable called “dollars” to store the required price, and gives it a value of 129. The “=” symbol is used to assign a value to a variable (which is why “==” must be used for the “Equal to” operator).



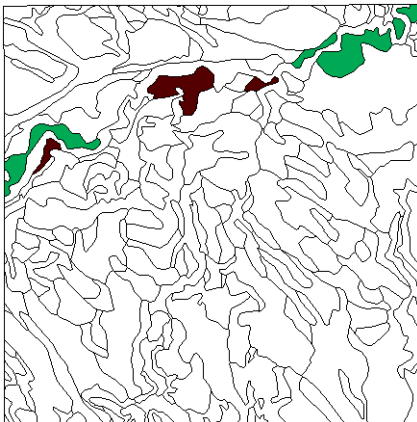
Variables are useful when the same value is used more than once in the query. If you want to run this query again with a different required price, you only need to change the single value assigned to the variable “dollars”. If the query were written without the variable, you would need to change two actual numeric values in the selection statement.

Variable names must always be in lower-case characters, and must begin with a letter. A variable name cannot be the same as a query command or a database table or field name.



## Using String Variables

You can also define variables to contain string values. The name you create for a string variable must end in a \$. The query in this exercise defines a string variable name\$, which is assigned the value "Glenberg". The query selects a subset of soil polygons belonging to the Glenberg soil series, which includes two soil types in the Crow Butte area. Instead of using the two class symbols to select the polygons, this query takes advantage of the fact that the NAME field in the DESCRIPTN table provides a soil description that begins with the name "Glenberg" for both classes. The query uses the "contains" operator, which selects elements for which a specified character string matches all or part of a string field. In this case the character string to be matched ("Glenberg") is stored in the name\$ variable. Polygons meeting this selection comparison are then screened on the basis of their area (in square meters), which is stored in the Area field in the standard POLYSTATS table. (A POLYSTATS table is present only if standard attributes have been calculated for the vector object.)

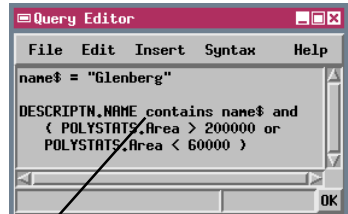


### STEPS

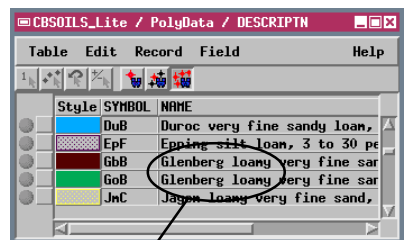
- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

```
name$ = "Glenberg"
DESCRIPTN.NAME contains name$ and
( POLYSTATS.Area < 60000 or
POLYSTATS.Area > 200000 )
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window



The "contains" operator selects polygons for which the character string in the name\$ variable matches any part of the DESCRIPTN.NAME string field.



The text string "Glenberg" is included in the DESCRIPTN.NAME field for both soil types belonging to the Glenberg series.

# Using the Logical “not” Operator

## STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

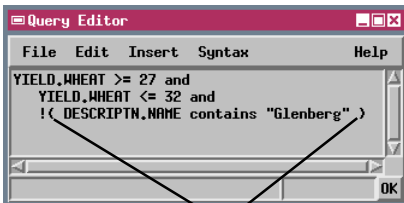
```
YIELD.WHEAT >= 27 and  
YIELD.WHEAT <= 32 and  
!( DESCRIPTN.NAME contains "Glenberg" )
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

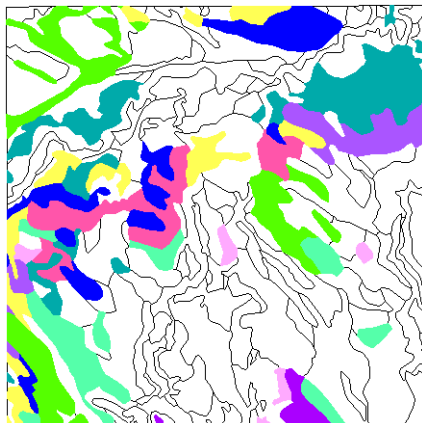
A number of soils in the Crow Butte area have potential wheat yields comparable to those of the two Glenberg soils (27 and 32 bushels per acre). The query in this exercise selects all of the soil types within this range of wheat yield values *except* the Glenberg soils.

The first two lines of the query select polygons for which the potential wheat yields fall within the designated range. The third line of the query begins with the logical “not” operator (!), which reverses the result of the variable, operator, or expression that follows it. In this case, the expression following the “not” operator would only select polygons belonging to the two Glenberg soils. The “not” operator reverses this result and selects all polygons meeting the previous yield requirements except the Glenberg soil polygons.

The “not” operator is especially useful when there is a large set of values you do want to select and a smaller, more easily specified set of values that you don’t want to select.



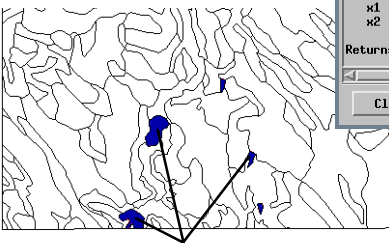
The ‘not’ operator reverses the next element that follows it (including variables or other operators). If you want the ‘not’ operator to apply to an entire expression (as in this example), the expression must be enclosed in parentheses.



## Checking Record Attachments

The query used in this exercise differs from those used in all previous exercises. Instead of querying specific attribute values, it is based on the number of records in a particular database table that are attached to a polygon. The query employs the `SetNum()` set function, which returns the number of items in a set. In this case the set is provided by an expression that has the form `Table[*]`, which stands for all records attached to an element in the named table.

This query identifies all soil class polygons that have no attached records in the YIELD table. The converse query `[ SetNum(YIELD[*]) > 1 ]` would identify all elements with more than one record attached. You can use queries with this form to identify polygons that have not yet had attributes assigned, or that somehow had extra records attached. In this example, all of the polygons with no attached records in the YIELD table belong to the class WATER. They represent small lakes and ponds, and therefore have no potential crop yield.



Polygons with no attached records in the YIELD database table all belong to class WATER. You can also select unattached elements by right-clicking on the icon for the corresponding table in the Group Controls or Layout Controls window and choosing Select All Unattached Elements from the popup menu.

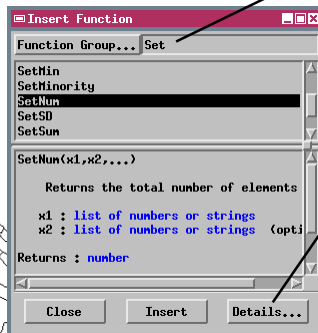
### STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

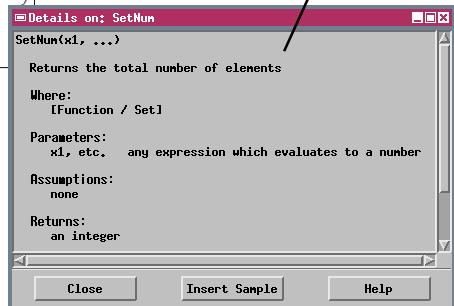
```
SetNum(YIELD[*]) < 1
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

Use the Function Group option in the Insert Function window to select which group of functions to list in the window.



Click [Details] on the Insert Function window to see further information on the currently selected function and an example of its use.





## Select using Multiple Attached Records

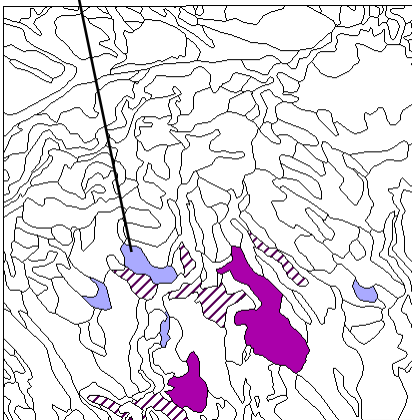
### STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

```
"WB" in LAYER[*].texture
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

Soil class polygons that have weathered bedrock as part of their typical soil profile.



The LAYER table for the soil class polygons in the CBSOILSQ object contains information on the different layers in a typical soil profile for each soil. There is a separate record for each layer in the profile, and thus multiple records attached to each soil polygon. Selecting elements on the basis of attributes among multiple attached records requires a special query syntax.

In this exercise, for example, we want to select soil types that include weathered bedrock in any part of the soil profile. This attribute is coded by the string “WB” in the texture field. If you try using the conventional selection query `LAYER.texture == “WB”`, you will find that no polygons are selected, even though some soils do have weathered bedrock in the lower part of the profile. This query structure only checks the *first* attached record in the table for each polygon, which in this case is usually the layer 1 record, containing attributes for the topmost soil layer. Subsequent records for the deeper soil layers are ignored.

To query the texture field of *all* of the attached records, we must use the expression `LAYER[*].texture`, which returns a set that lists the contents of the texture field from each record attached to the current polygon. We then need to determine if any of the members of the set correspond to the desired attribute “WB”. The easiest way to do this is to use the keyword “in” as a logical operator. The query is true if the variable preceding the operator is an exact match to any of the elements in the set produced by the expression following the operator. This construction can be used with either string or numeric fields.

## Find Island Polygons

A vector polygon that is wholly enclosed within a larger polygon is called an **island polygon**. Because island polygons often have different attributes than the enclosing polygon, processes that alter the topology or attribute assignments of a vector object must keep track of island polygon relationships.

The Internal table for polygons includes several fields that contain information pertaining to island polygons. You can query these fields to select island polygons or polygons containing islands. The Internal.Inside field contains the element number of the enclosing polygon, if any. All island polygons will have a non-zero value in this field. The first query therefore selects all island polygons. The NumIslands field shows the number of islands contained by each polygon. The second query in this exercise selects polygons that have a NumIslands value greater than 0, corresponding to all polygons that contain islands.

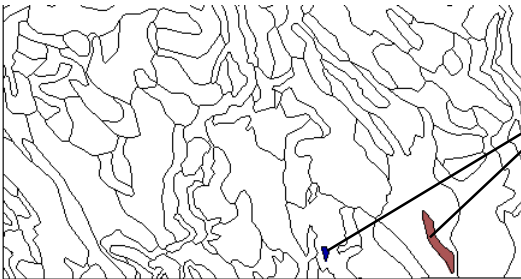
### STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the following query:

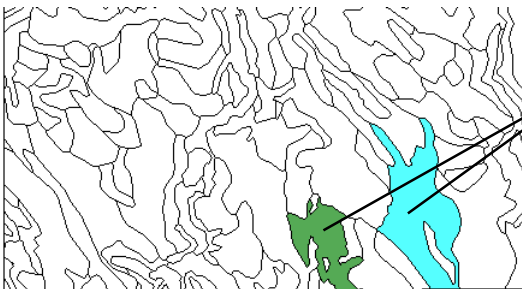
```
Internal.Inside > 0
```

- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window
- repeat the above steps using the following query:

```
Internal.NumIslands > 0
```



Island polygons selected by the first query. Each island belongs to a different soil class than its enclosing polygon.



Two polygons in the CBSOILS\_LITE vector object include island polygons, and were therefore selected by the second query.

# Styling by Script

## STEPS

- open the Vector Object Display Controls window
- in the Lines tabbed panel, turn on the Draw Lines Before Polygons toggle button



- in the Polygons tabbed panel, set the Style option to By Script, and click the adjacent Specify button
- use the Insert Symbol procedure (Numeric and String options) to create the following style script:

```
FillInside = 1
FillColor$ = "100 50 0"
DrawBorder = 1
DrawColor$ = "red"
```

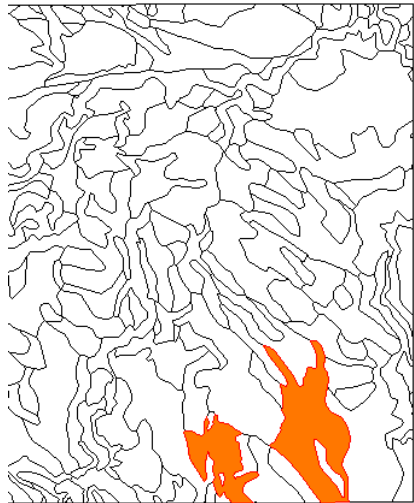
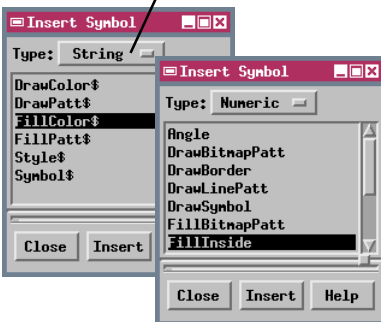
- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

The Style by Script option allows you to specify display characteristics for subsets of the selected elements on the basis of their attributes. To introduce the style options, this exercise retains the previous selection query, but uses a style script to set new display parameters for all selected polygons.

(Normally you would use the All Same style option to accomplish this.)

When you are setting styles by script, the Insert menu on the Query Editor window provides access to additional variables which are used to set display characteristics. FillInside and DrawBorder are numeric variables which are assigned a value of 1 to fill selected polygons and draw a border around them. FillColor\$ and DrawColor\$ are string variables which are used to set the color for the polygon fill and polygon border, respectively. The value assigned to them (enclosed in double quotes) can be either a color name (red, green, blue, black, white, yellow, orange, brown, cyan, magenta, or gray), or a set of RGB values (each from 0 to 100%).

Use the option button on the Insert Symbol window to access the different symbol lists: constants or the different types of variable (including numeric, string, and vector, among others).



## Compound Style Scripts

In this exercise all soil class polygons are selected for display, and a style script is used to define two different sets of polygon display parameters on the basis of the polygon area.

When you want to specify alternative actions in a query or style script, you must use “if-then-else” commands to explicitly define the logic. The statements in this script translate to “**if** a polygon has an area greater than 200000 square meters, **then** fill it with yellow, **else** (otherwise) fill it with a bitmap pattern (‘BitmapPatt4’)”. When more than one related statement follows a “then” or “else” command (as in this example), the group of statements must be enclosed within begin/end commands. Omitting the begin/end commands after “else” in this query would not produce a syntax error.

However, in that case only the first statement would be applied as the alternative to the “then” action; the remaining statements would be interpreted as applying globally to **all** selected polygons (like the first two lines of the query), overriding the style parameters defined earlier.

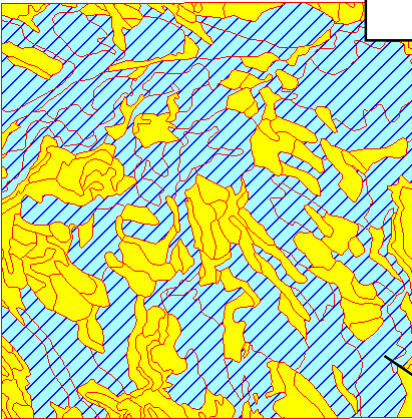
### STEPS

- open the Vector Object Display Controls window
- in the Polygons tabbed panel, set the Select option to All, leave the Style option set to By Script, and click the adjacent Specify button
- choose New from the File menu
- use the Insert procedures and / or manual entry to create the query shown below
- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

```

DrawBorder = 1
DrawColor$ = "red"
if ( POLYSTATS.Area < 200000 ) then
  begin
    FillInside = 1
    FillColor$ = "yellow"
  end
else
  begin
    FillInside = 1
    FillBitmapPatt = 1
    FillPatt$ = "BitmapPatt2"
  end

```



In order to use a bitmap fill pattern in a script, the pattern assigned to the FillPatt\$ variable must reside in the User Set of defined patterns for the object. In order to have the pattern drawn, variables FillInside and FillBitmapPatt must both be set to 1. See the booklet *Getting Started: Creating and Using Styles* for information on creating fill patterns and other styles.

Polygons with an area of 200,000 square meters or greater are filled with the stripe pattern defined in BitmapPatt2. Smaller polygons are filled with yellow.

## Find the Polygon Enclosing a Point

### STEPS

- open the Vector Object Display Controls window
- in the Polygons tabbed panel, set the Style option to By Attribute
- set the Select option to By Query, and click the adjacent Specify button
- choose New from the File menu in the Query Editor window
- use the Insert procedures and / or manual entry to create the query shown below
- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

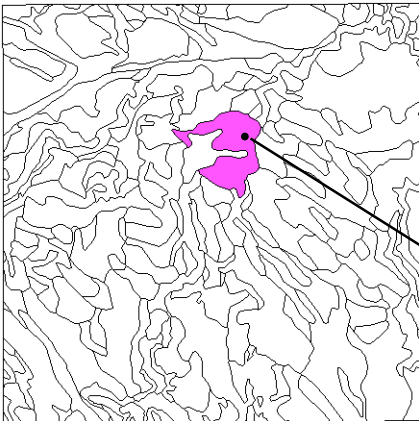
The query process incorporates a number of spatial functions that can be used to select elements. The FindClosestPoly( ) function (in the Vector function group) returns the element number of the polygon that encloses a point with the specified x and y coordinates. The required parameters for the function include the vector object to query, the x and y coordinate values, and the object number of the georeference subobject to use in processing the coordinate values. The georeference object number is provided by the function GetLastUsed-GeorefObject( ) (in the Georeference Function Group), which is used in the assignment statement for the variable “georef”. The predefined variable “Vect” (in the Vector list in the Insert Symbol window) is used to indicate the current vector object. The first two lines of the query define variables

```
xvar = -103.33861
yvar = 42.73583
georef = GetLastUsedGeorefObject( Vect )
c_poly = FindClosestPoly( Vect, xvar, yvar, georef )
Internal.ElemNum == c_poly
```

containing values for the x and y coordinates of the point.

The FindClosestPoly( ) function is used in an assignment statement that stores the element number of the enclosing polygon in a numeric variable (c\_poly in this example). The final statement of the

query compares the element number of each polygon in the object (Internal.ElemNum) to the number stored in c\_poly to find the matching polygon for display.



Location of the point specified by coordinates xvar (longitude) and yvar (latitude). Values used in the query must be in the same coordinate system as the specified georeference subobject. Latitude / Longitude coordinates must be expressed in decimal degrees.

## Polygon Adjacency Query: Logic

A selection query can also make use of the topological information associated with a vector object. Each line in a vector object has a beginning node and an ending node, which define a left and right side for the line. Each polygon is made up of specific line elements, and the Internal table for lines includes fields that contain the element numbers of the polygons that lie on either side of the line. The `GetVectorPolyAdjacentPolyList()` function (in the Vector list in the Insert Function window) uses this information to determine which polygons are adjacent to the current polygon. This function can be used in a query to select polygons that are adjacent to specific polygon classes.

As an example, let's examine a query for the `CBSOILS_LITE` vector object that selects polygons belonging to soil class "SrD" that are also adjacent to polygons of class "Sa." To be considered adjacent, the polygons must share a common line boundary, not just a common node. The general strategy used in such a query is as follows:

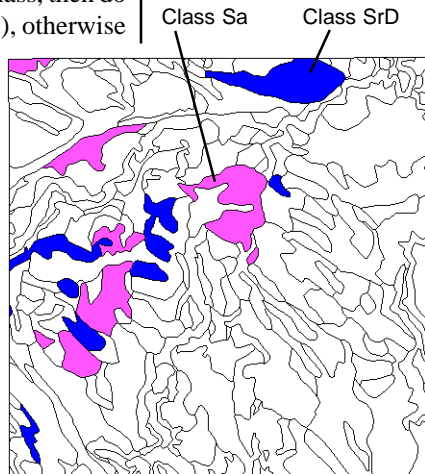
- 1) Define the class to select.
- 2) If a polygon belongs to the selected class, then do the subsequent steps (test for adjacency), otherwise reject it.
- 3) Get the list of polygons that are adjacent to the current polygon.
- 4) Check the class assignment for each adjacent polygon. If any of them match the defined adjacent class, select the current polygon for display. If none match, reject it.

The syntax for this query is shown and explained on the next page.

### STEPS

- open the Vector Object Display Controls window and the Query Editor window
- choose New from the File menu
- use the Insert procedures and / or manual entry to enter the query shown on the next page
- click [OK] in the Query Editor window
- click [OK] in the Vector Object Display Controls window

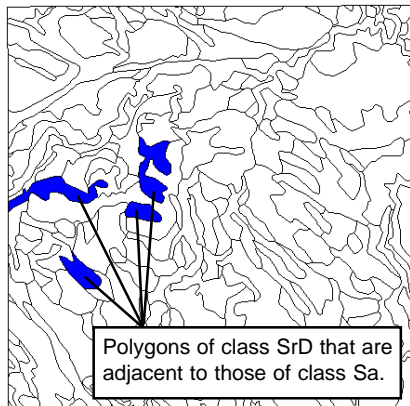
Vector object `CBSOILS_LITE` with all polygons of classes SrD and Sa selected (for comparison with illustration on the next page).



## Polygon Adjacency Query: Syntax

```
1  if (CLASS.Class == "SrD") then
2      begin
3          array polylist [10]
4          numpolys = GetVectorPolyAdjacentPolyList(Vect, polylist)
5          for i = 1 to numpolys begin
6              polynum = polylist[i]
7              if (Vect.poly[polynum].CLASS.Class$ == "Sa")then
8                  return 1
9          end
10         return 0
11     end
12 else return 0
```

1. Conditional selection of class SrD polygons for subsequent testing.
2. Begins the processing loop to check the class of adjacent polygons.
3. Defines a one-dimensional array called "polylist" to hold a list of the element numbers of polygons that are adjacent to the current polygon. Initializes the array size at 10 elements (it is resized automatically by the function in the next statement).
4. Defines a variable "numpolys" that is assigned a value equal to the number of polygons adjacent to the current polygon. This value is returned by the GetVectorPolyAdjacentPolyList function, which also finds the element numbers of the adjacent polygons and stores them in the "polylist" array. The predefined variable "Vect" is used to indicate the current vector object.
5. Begins a processing loop to examine the class of each polygon in the array. The loop is run once for each element in the array, beginning with the first position (array index 1) and continuing to the last position (specified by the current value of variable "numpolys"). In each loop the variable "i" is assigned the value of the current array index for use in the next statement.
6. Assigns the number of the current adjacent polygon (specified by its index in the array) to the variable "polynum".
7. Looks up the class of the current adjacent polygon and compares it to the specified adjacent class. The database specification is in the form "Object.database[record#.table.field". (The "\$" at the end of the database specification indicates that the target field is a string field.) If the classes match, then...
8. The "return 1" statement explicitly states that the query is true for a polygon satisfying the above condition, so the polygon will be selected for display.
9. End of array processing loop.
10. If all adjacent polygons fail the class test above, the "return 0" statement states that the query is false.
11. End of polygon adjacency loop.
12. States that the query is false for a polygon not meeting the initial class selection condition in statement 1.







## Census Boundaries in TIGER Data

Vector objects imported from the US Census Bureau's TIGER / Line files are made up of line segments representing natural and manmade physical features as well as governmental and census boundaries. The boundaries of census tracts (and equivalent Block Numbering Areas, or BNA's) and their component census blocks usually coincide with other map features and are not explicitly identified by a Census Feature Class Code (CFCC) like the basic map features.

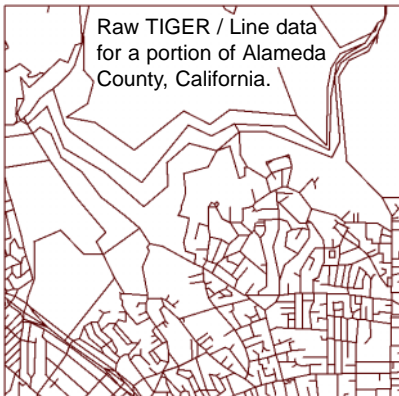
Census block boundary lines can be selected for display or extraction using a query that selects lines for which the block numbers on the left and right side are not the same. Blocks that have been subdivided retain the same block number, but are identified by different letters in left and right block suffix fields; the second set of statements in the sample query selects these boundaries. Finally, blocks in adjacent BNA's can have the same number, so the final statement selects lines separating different BNA's.

### STEPS

- remove the layer used in the previous exercise 
- click the Add Vector icon and select Add Vector Layer from the dropdown menu 
- select the ALAMEDA object from the TIGER Project File
- in the Vector Object Display controls window, set the Style option in the Lines panel to All Same and the Select option to By Query
- open the Query Editor window and choose New from the File menu
- use the Insert procedures and / or manual entry to create the query shown below
- click [OK] in the Query Editor window
- click [OK] in the Vector


Object Display Controls window

```
Basic_Data.Block_Left <> Basic_Data.Block_Right
or ( Basic_Data.Block_Left == Basic_Data.Block_Right
and Basic_Data.BlockSuff_Left <>
Basic_Data.BlockSuff_Right )
or Basic_Data.BNANum_Left <> Basic_Data.BNANum_Right
```



# Computed Fields from Multiple Records

## STEPS

- ☑ minimize the Display windows from the previous exercise
- ☑ choose Edit / Attribute Databases from the TNTmips main menu
- ☑ navigate to and select the CBSOILS\_LITE object in the CBSOILSQ Project File
- ☑ turn on the Polygon radio button in the Select window
- ☑ in the Database Editor window, right-click on the box for Class table and select Edit Definition
- ☑ in the definition window for the Class table, click the Add Field icon button 
- ☑ in the field list, highlight the default name for the new field and type ClassArea
- ☑ on the Field info panel, select Computed from the Field Type menu and click Edit Expression
- ☑ enter the query shown below in the Query Editor window

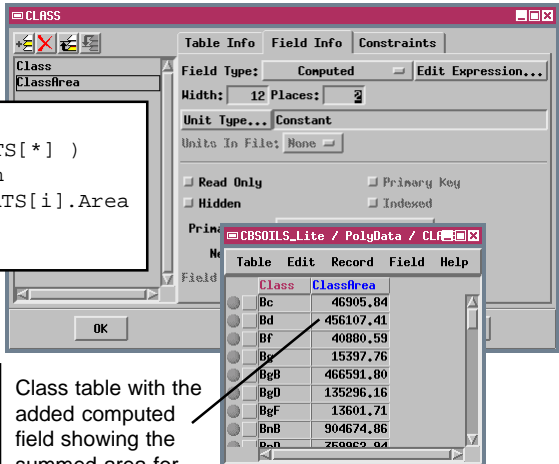
```
sum = 0.0
num = SetNum( POLYSTATS[*] )
for i = 1 to num begin
    sum = sum + POLYSTATS[i].Area
end
return sum
```

- ☑ click [OK] in the Query Editor window
- ☑ enter 12 in the Width text box and 2 in the Places text box
- ☑ click [OK] in the definition window
- ☑ double-click on the box for the Class table to open it

Scripts can also be used to define the values for computed fields in database tables. In many cases these scripts need only create simple arithmetic combinations of other fields in the same record. The task in this exercise is more complex: to create a computed field in the polygon Class table for CBSOILS\_LITE that shows the total area for each soil type.

Polygon areas are stored in the POLYSTATS table for individual polygons, but we are creating the computed field in the Class table, which has one record for each soil type. The script shown here is designed to sum the polygon areas for each soil class and return that sum as the computed field value.

The script defines a numeric variable “sum” that is used to sum the areas in the POLYSTATS.Area field. This variable must be initially reset to a value of 0.0 for each class. The variable “num” is assigned a value (for the current soil class) equal to the number of attached records in the POLYSTATS table. This variable is used to set the number of iterations of the loop that sums the areas.



Class table with the added computed field showing the summed area for each soil type.

- ☑ choose Table / Close to close the Class table

# String Expression Fields

A string expression field is a special type of computed field in a database table. The simplest use of a string expression field is to copy the contents of a string field in another linked table into the current table. The expression in that case is simply the appropriate TABLE.FIELD reference. You can also use string expressions to merge the contents of several string fields into one new field. For example a table called NAME could have separate fields for first and last names. You can use the “+” (add) operator to merge these strings. The expression

NAME.FIRST + “ ” + NAME.LAST

would produce entries with the form “John Doe”. The expression must include any separating characters (spaces, commas) in quotes, as shown. You can use a merged string expression field to provide text for more informative Data Tips or labels.

The expression you use in this exercise employs the `sprintf( )` function, which allows you to format complex string expressions more easily. The first function argument is a control string (in quotes), which is followed by the string field references. Each of the “%s” entries in the control string stands for one of the listed string field references. The control string can also incorporate inserted text, spaces, or punctuation.


```
sprintf( "Sec %s Twp %s Rng %s", Sections.Section,
Sections.Township, Sections.Range )
```



Section	Township	Range	SecTwpRng
34	31N	51W	Sec 34 Twp 31N Rng 51W
35	31N	51W	Sec 35 Twp 31N Rng 51W
36	31N	51W	Sec 36 Twp 31N Rng 51W
13	32N	51W	Sec 13 Twp 32N Rng 51W
14	32N	51W	Sec 14 Twp 32N Rng 51W
15	32N	51W	Sec 15 Twp 32N Rng 51W
16	32N	51W	Sec 16 Twp 32N Rng 51W

Formatted text created by the string expression.

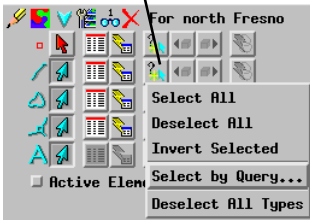
## STEPS

- choose File / Open Database on the Database Editor window
- navigate to and select the CBSECT object in the CB\_SECT Project File
- turn on the Polygon radio button in the Select window
- in the Database Editor window, right-click on the box for the Sections table and select Edit Definition
- in the definition window for the Sections table, click on the Range field in the list and click the Add  Field icon button
- in the field list, highlight the default name for the new field and type SecTwpRng
- select String Expression from the Field Type menu and click Edit Expression
- enter the query shown below in the Query Editor window

- click [OK] in the Query Editor window
- enter 25 in the Width text box
- click [OK] in the definition window
- double-click on the box for the Sections table to open it
- choose File / Close from both Database Editor windows when you have completed this exercise

# Queries to Check Digitizing Artifacts

Click the Select / Deselect icon button and choose Select by Query to open the standard Query Editor window so that you can enter a query for that element type.

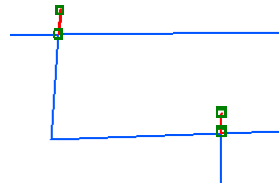


Selection queries can also be useful when you are creating or editing a vector object using the TNTmips Spatial Data Editor. Complex vector objects can contain digitizing errors such as line overshoots, unclosed polygons, and sliver polygons. Many of these flaws are not visible except at high zoom levels, which makes manual checking difficult and time-consuming. You can speed up the search for potential topology problems by using selection queries such as the examples below. The Spatial Data Editor window provides icons that allow you to create and apply a selection query for a particular element type.

## OVERSHOOTS

Overshoots are short line segments that incorrectly extend beyond a line intersection. If you have run the Standard Attributes process for the vector object, you can use a selection query based on line length to select all very short lines for examination and possible removal:

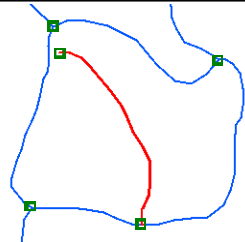
**LINESTATS.Length < [ your length value ]**



## UNCLOSED POLYGONS

In a vector object containing a network of polygons, a gap between two lines that should intersect may leave a single polygon where two separate polygons should exist. Lines that fail to close a polygon can be found by query because they have the same polygon on both sides:

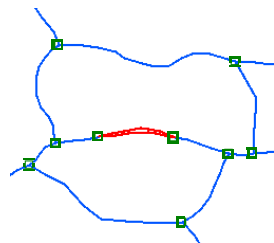
**Internal.LeftPoly == Internal.RightPoly**



## SLIVER POLYGONS

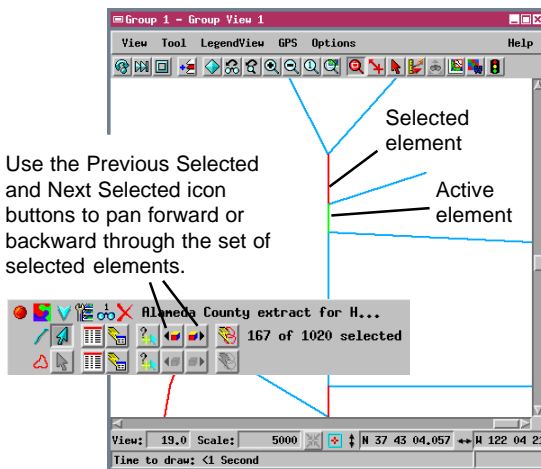
Double-tracing polygon boundaries can create extraneous sliver polygons along the boundary of two contiguous polygons. Sliver polygons usually have a much smaller area than the main polygons, and are usually highly elongate (with a high Compactness Ratio). Use a combined query on the Area and CompactRatio fields in the POLYSTATS table to select sliver polygons:

**POLYSTATS.Area < [ your area value ] or  
POLYSTATS.CompactRatio > 3.00**






## Pan by Query


A query executed from the Element Selection window in the Spatial Data Editor or in Spatial Data Display often selects more than one element. One of these selected elements is designated the “active” element; the active and selected elements are highlighted in different colors. Editing operations can be applied to either the active or the selected elements. You can use the Previous Selected and Next Selected icon buttons on the Element Selection window to step forward and backward through the selected set of elements, making each one active in turn. The view is automatically repositioned (if necessary) to display the current active element. This “pan by query” feature allows you to remain zoomed in to examine (and perhaps edit) each element while easily stepping through the selected set.



### STEPS

- restore the View and Group Controls windows
- open the Vector Object Display Controls window
- set the Select option in the Lines panel to All and click [OK]
- open the Options menu in the Spatial Data Display View window and make sure that the Show Scale / Position option is turned on
- type “5000” in the Scale text box at the bottom of the View window, and press <Enter>
- click the Show Details icon  button on the Layer icon row
- click the Select icon button for lines 
- click the Select / Deselect icon  button for lines, and choose Select by Query from the dropdown menu
- enter the following query in the Query Editor:

```
LINESTATS.Length < 50
```

- click [Apply] on the Select by Query window
- click the Next Selected icon  button

The exercises in this Getting Started booklet have introduced the fundamentals of the structure and syntax of database queries for use in TNTmips, TNTedit, and TNTview. The query language is a subset of the Spatial Manipulation Language (SML) used in TNTmips, and shares the same syntax. In addition to the documentation on queries cited on page 2, you may wish to consult the SML documentation in TNTmips Reference Manual for additional programming hints to expand your query capabilities.

# Advanced Software for Geospatial Analysis

MicroImages, Inc. publishes a complete line of professional software for advanced geospatial data visualization, analysis, and publishing. Contact us or visit our web site for detailed product information.

**TNTmips** TNTmips is a professional system for fully integrated GIS, image analysis, CAD, TIN, desktop cartography, and geospatial database management.

**TNTedit** TNTedit provides interactive tools to create, georeference, and edit vector, image, CAD, TIN, and relational database project materials in a wide variety of formats.

**TNTview** TNTview has the same powerful display features as TNTmips and is perfect for those who do not need the technical processing and preparation features of TNTmips.

**TNTatlas** TNTAtlas lets you publish and distribute your spatial project materials on CD-ROM at low cost. TNTAtlas CDs can be used on any popular computing platform.

**TNTserver** TNTserver lets you publish TNTAtlases on the Internet or on your intranet. Navigate through geodata atlases with your web browser and the TNTclient Java applet.

**TNTlite** TNTlite is a free version of TNTmips for students and professionals with small projects. You can download TNTlite from MicroImages' web site, or you can order TNTlite on CD-ROM.

## Index

arithmetic operations.....	9	element selection (continued)	
assignment statement.....	12	sliver polygons.....	26
comments.....	12	undershoots.....	26
comparison operators.....	4	Insert Field window.....	16
equal to, ==.....	5	Insert Operator window.....	5
greater than, >.....	4	logical operators (and, or, not).....	10,14
not equal to, <>.....	11	opening a query.....	9
contains.....	13	pan by query.....	27
compound queries.....	10,19	selecting <i>See</i> element selection	
computed fields.....	24,25	saving a query.....	9
database query, defined.....	3	SetNum function.....	15
element selection		string field.....	7
adjacent polygons.....	21,22	string expression field.....	25
by query.....	3	string variables.....	13
island polygons.....	17	style by script.....	3,18,19
multiple attached records.....	16	syntax, checking.....	8
overshoots.....	26	Table[*] expression.....	15
no attached records.....	15	TIGER data.....	23
polygon enclosing point.....	20	variables.....	12,13



**MicroImages, Inc.**

11th Floor - Sharp Tower  
206 South 13th Street  
Lincoln, Nebraska 68508-2010 USA

Voice: (402) 477-9554  
FAX: (402) 477-9559

email: [info@microimages.com](mailto:info@microimages.com)  
internet: [www.microimages.com](http://www.microimages.com)