QUERIES

Tutorial

Building and Using Queries

with

TNTmips®

TNTedit™

TNTview®

Before Getting Started

This booklet shows you how to construct queries in the TNT products to utilize the attribute information attached to your vector, CAD, shape, and TIN objects. 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 the tutorial booklets *Displaying Geospatial Data* and *TNT Product Concepts*. Those exercises introduce essential concepts and skills that are not covered again here. Please consult those booklets for any review you need.

Sample Data The exercises in this booklet use sample data that is distributed with the TNT products. If you do not have access to a TNT products DVD, 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. Details of the process can be found in a variety of tutorial booklets, Technical Guides, and Quick Guides, which are all available from MicroImages' web site.

TNTmips® Pro and TNTmips Free TNTmips (the Map and Image Processing System) comes in three versions: the professional version of TNTmips (TNTmips Pro), the low-cost TNTmips Basic version, and the TNTmips Free version. All versions run exactly the same code from the TNT products DVD and have nearly the same features. If you did not purchase the professional version (which requires a software license key) or TNTmips Basic, then TNTmips operates in TNTmips Free mode.

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 TNTmips Free using the sample geodata provided.

Randall B. Smith, Ph.D., 4 January 2012

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 TNTmips.

http://www.microimages.com

Welcome to Building and Using Queries

TNTmips gives you great flexibility to use the database attributes of geometric (vector, CAD, shape, 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 Display you can use a query to temporarily **mark** (highlight in color) certain elements or to **select** which elements should be displayed. 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 TNT Geospatial Scripting Language (SML). You compose queries in the Script Editor window, which simplifies construction of valid queries by letting you choose fields from the available database tables and insert symbols and functions from a Script Reference window that outlines the correct syntax. The Script Editor also provides a syntax checker to help you find errors before applying the query.

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

STEPS

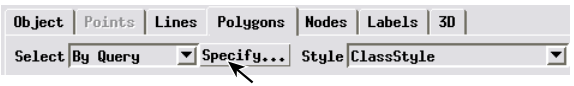
- ☑ make sure that you have a copy of the sample data in the QUERY data collection on your hard drive
- ☑ launch TNTmips
- ☑ choose Main / Display from the TNTmips menu

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 10-16 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 17-18, along with the use of conditional "if-then-else" constructions. Pages 19-22 describe marking displayed elements by query using the interactive Query Builder. Examples of queries based on the spatial and topological attributes of vector objects are found on pages 23-25. Pages 26-29 show scripts to create dynamic labels in displays and virtual fields in database tables. Pages 30-31 provide examples of queries that are useful in editing vector objects.

Select by Querying a Single Field

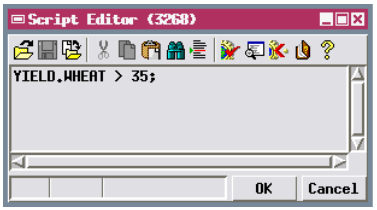
STEPS

- ☑ click on the Add Objects icon button on the Display Manager window's toolbar
- ☑ select the CBSOILS_LITE object from the CB_SOILQ Project File in the QUERY data collection
- ☑ click on the Layer Controls icon in the CBSOILS_LITE layer entry in the Display Manager
- ☑ in the Vector Layer Controls window, set the Select option on the Polygons panel to By Query and click on the adjacent Specify button



- ☑ in the Script Editor window, type the following text exactly (including capitalization):

```
YIELD.WHEAT  >  35;
```

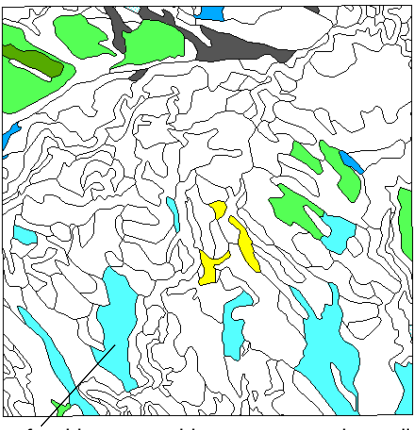


- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window to accept the display settings and display the vector object

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 (>).



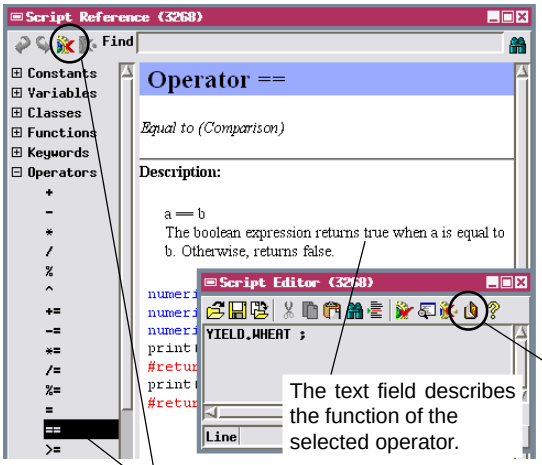
Display options for this vector object are set to draw all lines, so the outlines of all soil polygons are drawn. The polygons selected by the query are filled with colors and fill patterns that are based on the soil type and that were previously set up for display By Attribute. Several soil types meet the wheat yield selection criterion. Unselected polygons remain unfilled.

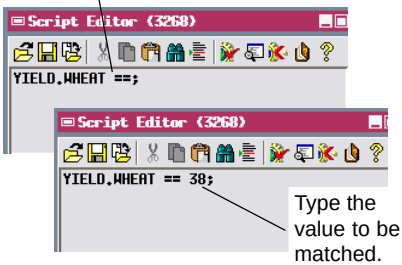
Using the Insert Operator Option

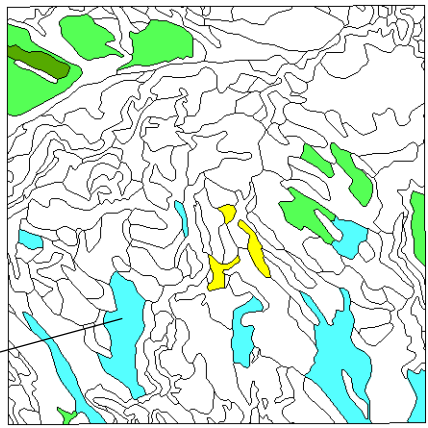
The previous query selected soil polygons belonging to several soil type classes. Let's refine the selection criterion so that the query selects only those polygons with a potential wheat yield of exactly 38 bushels per acre. To specify this selection criterion, use the "Equal to" operator (==, double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

STEPS

- ☑ reopen the Vector Layer Controls window
- ☑ click on Select: [Specify] in the Polygon Options panel
- ☑ highlight "`> 35`" in the Script Editor window and press <Delete>
- ☑ press the Script Reference icon button
- ☑ in the Script Reference window, expand the Operator list and select the "==" operator, then press the Insert icon button
- ☑ in the Script Editor window, type `38` to the right of the operator, then click [OK]
- ☑ click [OK] in the Vector Layer Controls window to accept the display settings and display the vector object







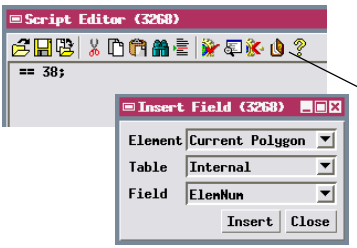
Fewer soil classes meet the more restrictive selection criterion in the revised query statement.

Using the Insert Field Option

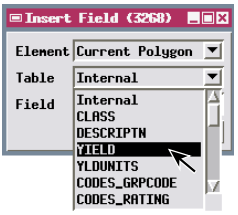
STEPS

- ☑ open the Vector Layer Controls window and the Script Editor window
- ☑ select `YIELD.WHEAT` in the existing query and press <Delete>
- ☑ press the Insert Field icon button 
- ☑ in the Insert Field window that opens, choose `YIELD` from the Table menu
- ☑ choose `OATS` from the Field menu, 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 Layer Controls window

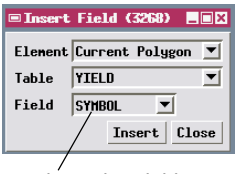
You can also use the Insert Field icon button in the Script Editor to help construct or modify queries. This button opens the Insert Field window, from which you can choose the Table and Field and automatically insert the attribute location information into your query statement in the correct form.



Press the Insert Field icon button to open the Insert Field window, which has a menu with all available database tables for the selected element type.

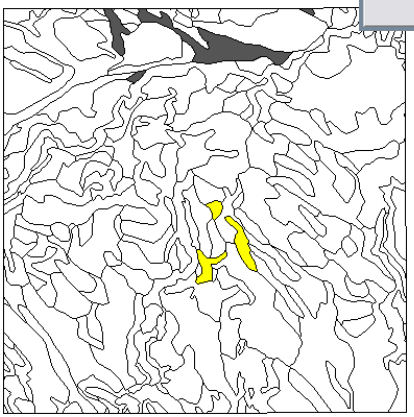


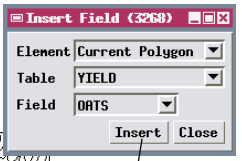
Choosing a Table...



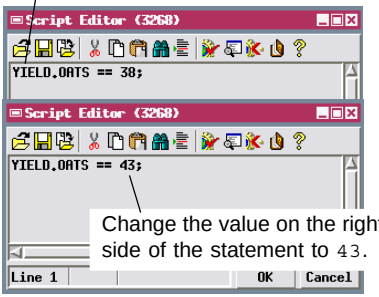
...updates the Field menu to list the fields in that table.

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





Choosing a field creates the TABLE.FIELD entry; click the Insert button to insert it into your query statement.



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

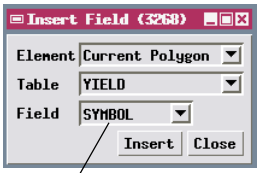
Querying a String Field

The query language used in the TNT products 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.

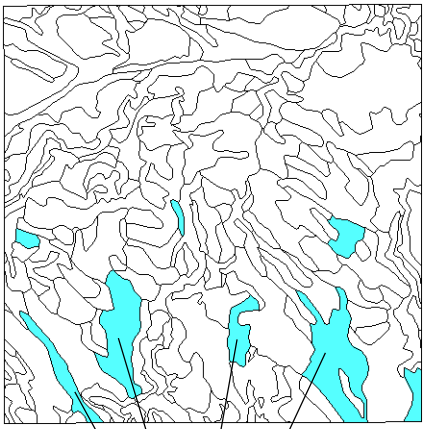
The database fields you have used in your queries so far have contained numeric data. The YIELD table for CBSOILS_LITE also contains a field named SYMBOL with soil type symbols in String format. The term "string" is short for "character string," which means that the field is not evaluated numerically, 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 Layer Controls window and the Script Editor window
- ☑ select `YIELD.OATS` 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 Layer Controls window

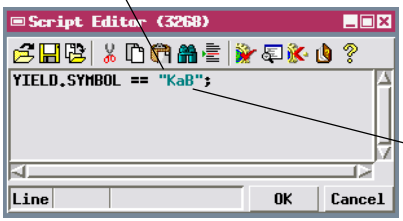


The SYMBOL field contains string values.



Selected soil polygons belonging to class KaB.

Enclose a string value in double quotes.



The syntax highlighting in the Script Editor window shows all string values in cyan color.

Checking Query Syntax

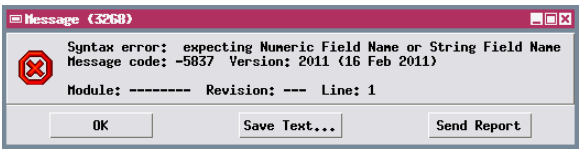
STEPS

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

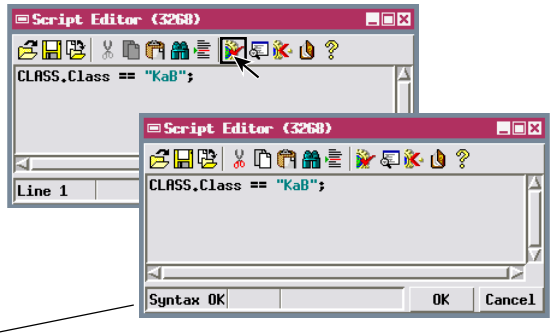
- ☑ press the Check Syntax icon button 

The rules concerning capitalization and use of quotes for string values are examples of the syntax of the TNT query language. Query syntax is checked automatically when you click [OK] to execute the query. If the query contains a syntax error, the Script Editor remains open and an error message is displayed.

You can check the syntax of a query before executing it by pressing the Check Syntax icon button in the Script Editor. The process can find misspellings, missing parentheses or other symbols, or references to nonexistent database fields. The process starts checking at the beginning of the query. If no syntax errors are encountered, the message line at the bottom of the Script Editor window reads "Syntax OK." If a syntax error is detected, a Message window opens to show an error message. In this example, the query checker detected that there is no database field named CLASS in table CLASS. After correcting a syntax error, you can use the Syntax option again to check for errors in the remainder of the query.



- ☑ note the error message, then press[OK] on the Message window
- ☑ select `CLASS.CLASS` in the query statement and press <Delete>
- ☑ press the Insert Field icon button and insert `CLASS.Class` into the query statement
- ☑ press the Check Syntax icon button 
- ☑ note the "Syntax OK" message in the status line at the bottom of the Script Editor window



The "Syntax OK" message appears if no errors are found.

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 insert the operation symbols from the Script Reference window if you wish. The Function listings in Script Reference window provide 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 icon buttons on the Script Editor window. These options allow you to save the query currently displayed in the Script Editor as a file with a .QRY file extension or as a script object in a Project File. To reopen a stored query, use the Open icon button.

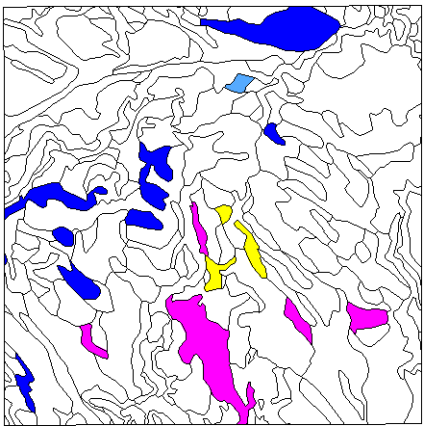
STEPS

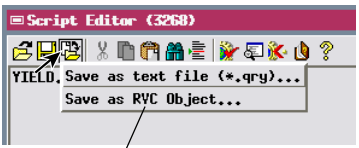
- ☑ open the Vector Layer Controls window and the Script Editor window
- ☑ select and delete the previous query
- ☑ use the Insert procedures and / or manual entry to create the following query statement:

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

- ☑ press the Save As icon button, then choose *Save as Text File (*.qry)* from the dropdown menu
- ☑ use the standard File Selection window to name a new file to contain the query
- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

Usually a query works 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 script object in a Project File.

Compound Queries

STEPS

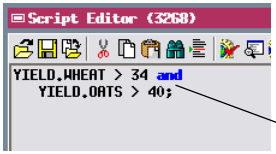
- ☑ open the Vector Layer Controls window and the Script Editor window
- ☑ delete the preceding query
- ☑ use the Insert procedures and / or manual entry to create the following query:

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

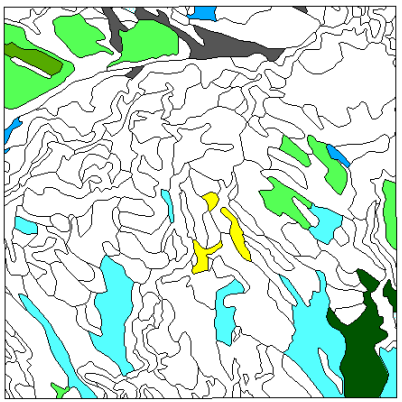
- ☑ click [OK] in the Script Editor window and in the Vector Layer Controls window
- ☑ repeat above steps but substitute "or" for "and" in the query statement

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" (||), or "not" (!). The keyword versions of these operators must be entered in all lowercase letters, or you can insert them from the Keyword list in the Script Reference window or insert symbolic versions from the Operator list.

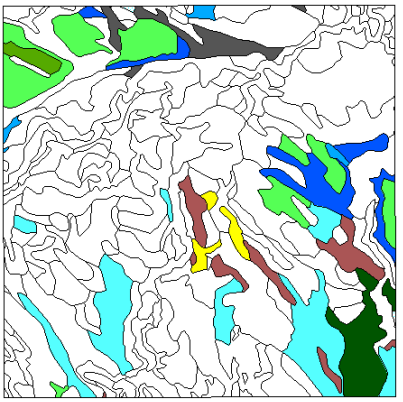
When two comparisons are linked by the logical "and" operator, both comparisons must be true in order to make the entire query statement true and select the element. When two comparisons are linked by the logical "or" operator, the query statement is true if either of the individual comparisons is true.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



A 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 sample 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.

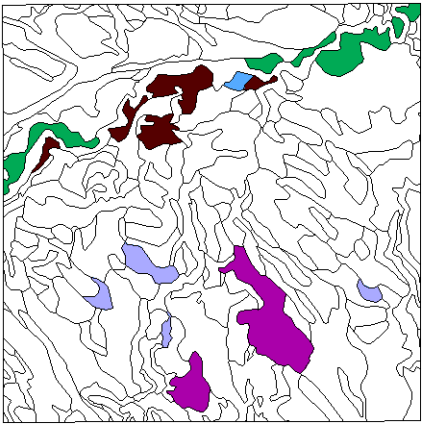
STEPS

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

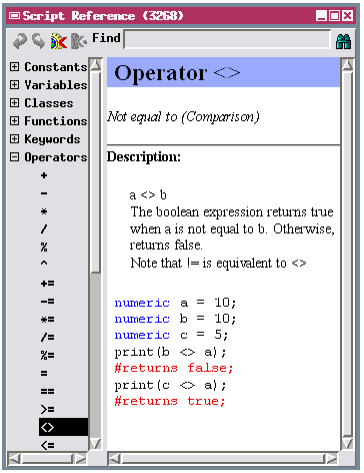
```
YIELD.WHEAT <> 0   and
YIELD.OATS * 3.25 >= YIELD.WHEAT * 4.0;
```

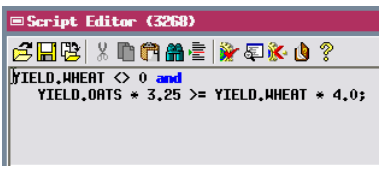
- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

This query is complicated by the fact that the potential crop 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 !=). Only the polygons for which the potential wheat yield is not equal to 0 satisfy the first part of the query, and only these polygons are subjected to the price comparison in the second line.



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

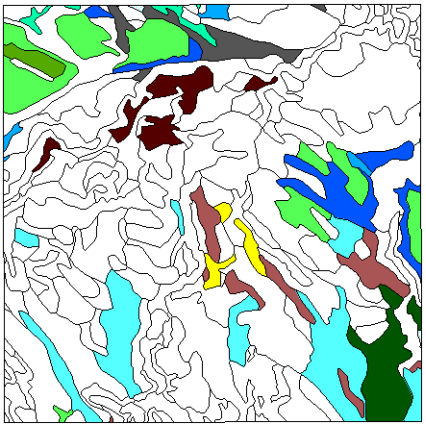
- ☑ open the Vector Layer Controls window and the Script Editor window
- ☑ choose New from the File menu
- ☑ use the Insert procedures and / or manual entry to create the following query:

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.

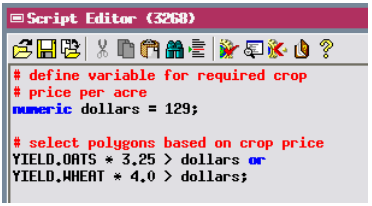
```
# define variable for required crop
# price per acre
numeric dollars = 129;

# select polygons based on crop price
YIELD.OATS * 3.25 > dollars or
YIELD.WHEAT * 4.0 > dollars;
```

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).

- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window



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 start with a lower-case letter. A variable name cannot be the same as a query keyword or a database table or field name.

Using String Variables

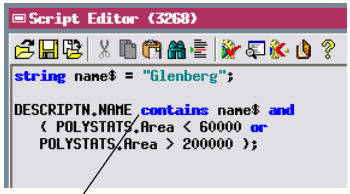
You can also define variables to contain string values. The query in this exercise defines a string variable name$, which is assigned the value “Glenberg”. (Though not required, ending a string variable name with the $ character makes it easy to differentiate numeric and string variable names.) The query 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

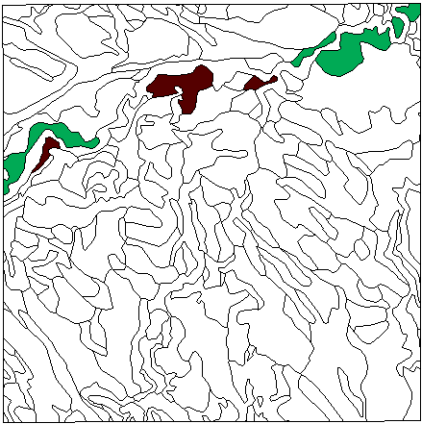
- ☑ open the Vector Layer Controls window and the Script Editor window
- ☑ choose New from the File menu
- ☑ use the Insert procedures and / or manual entry to create the following query:

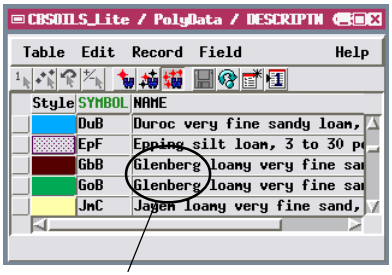
```
string name$ = "Glenberg";
DESCRIPTN.NAME contains name$ and
( POLYSTATS.Area < 60000 or
  POLYSTATS.Area > 200000 );
```

- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer 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 Layer Controls window and the Script Editor window
- ☑ choose New from the File menu
- ☑ use the Insert procedures and / or manual entry to create the following query:

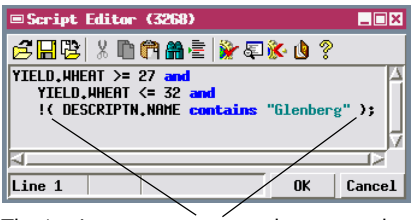
```
YIELD.WHEAT >= 27 and
   YIELD.WHEAT <= 32 and
   !( DESCRIPTN.NAME contains "Glenberg" );
```

- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

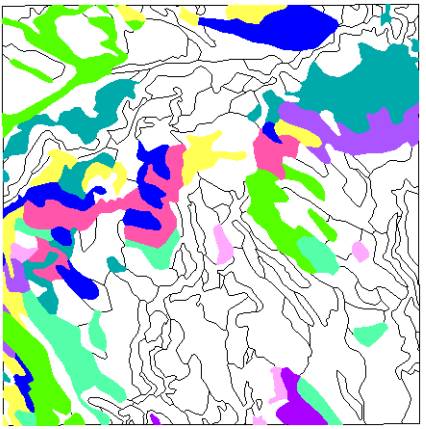
A number of soils in the Crow Butte area have potential wheat yields comparable to those of the two Glenberg soils (27 and 32 bushels per acre). The query in this exercise selects all of the soil types within this range of wheat yield values *except* the Glenberg soils.

The first two lines of the query select any polygon for which the potential wheat yields falls 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 any polygon 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 script element that follows it (including variables, operators, and comparison expressions). If you want the ‘not’ operator to apply to an entire expression (as in this example), the expression must be enclosed in parentheses.



Select Using Multiple Attached Records

The LAYER table for the soil class polygons in the CBSOILS_LITE 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 to use 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.

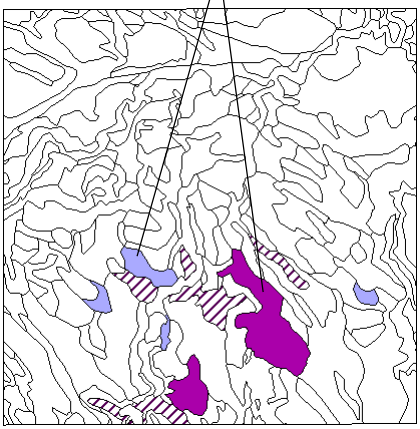
STEPS

- ☑ open the Vector Layer Controls window and the Script 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 Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

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



Find Island Polygons

STEPS

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

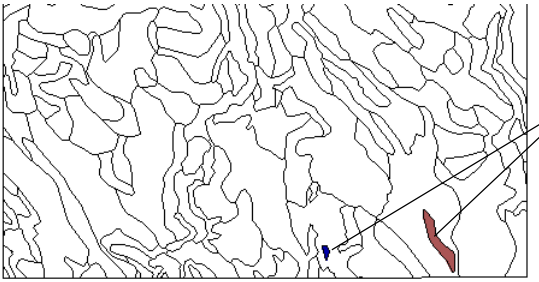
```
Internal.Inside > 0;
```

- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window
- ☑ repeat the above steps using the following query:

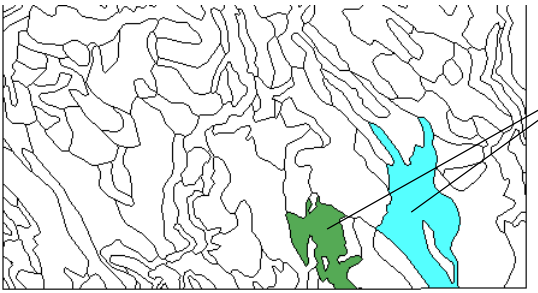
```
Internal.NumIslands > 0;
```

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 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.



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

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 Script Reference window provides access to additional variables that are used to set display characteristics. FillInside and DrawBorder are numeric variables that are assigned a value of 1 to fill selected polygons and draw a border around them. FillColor$ and DrawColor$ are string variables that are used to set the color for the polygon fill and polygon border, respectively. The value assigned to these string variables (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%).

STEPS

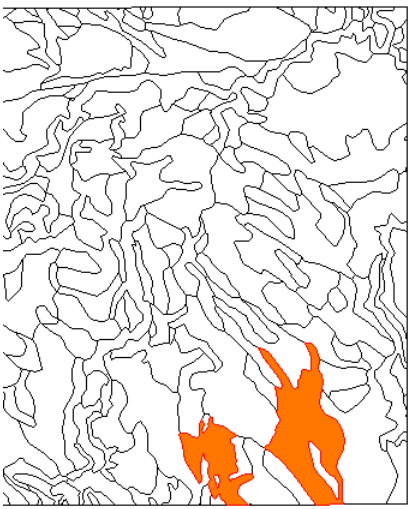
- ☑ open the Vector Layer Controls window
- ☑ in the Lines tabbed panel, turn on the Draw Lines Before Polygons toggle button
- ☑ in the Polygons tabbed panel, set the Style

option to By Script, and click the adjacent Edit button

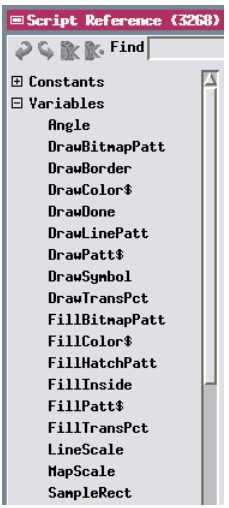
- ☑ use the Variables list in the Script Reference window to insert the variables needed to construct the following style script:

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

- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window



You can use the Script Reference window to insert predefined variables into scripts.



Compound Style Scripts

STEPS

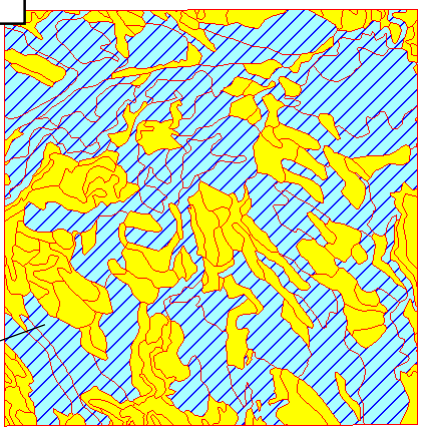
- ☑ open the Vector Layer Controls window
- ☑ in the Polygons tabbed panel, set the Select option to All, leave the Style option set to By Script, and click the adjacent Specify button
- ☑ delete the previous style script
- ☑ use the Insert procedures and / or manual entry to create the script shown below
- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

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')". (The "then" keyword is optional.) 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 (or curly brackets, { }). 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 script), overriding the style parameters defined earlier.

```
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 tutorial booklet entitled *Creating and Using Styles* for information on creating fill patterns and other styles.

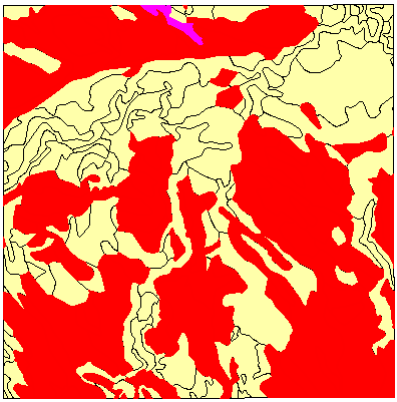


Polygons with an area of 200,000 square meters or greater are filled with the stripe pattern defined in BitmapPatt2. Smaller polygons are filled with yellow.

Mark by Query

In most of the previous exercises we have used queries to determine which vector elements are to be drawn in the view; elements not meeting the selection criteria are not drawn at all. You can also use similar queries in the Display process to temporarily "mark" displayed elements in the View. Marked elements are shown in a special color. If more than one element is marked by the query, one of the marked elements is designated as the "active" element and rendered in another designated color. In the illustration below, marked polygons are red and the active polygon in the marked set is magenta. (You can set marked and active element colors using the Options / Colors menu selection on the View window.)

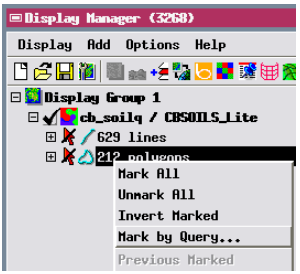
You can mark elements by query in any TNT process that shows the data in a View, including the Editor. The distinction between active and marked elements is important when you are editing geometric objects (see pages 30 and 31).



The Mark by Query window is nonmodal, which means it provides an [OK] button that accepts the query and closes the window and an [Apply] button that applies the query but leaves the window open so you can easily change the marking criteria and re-apply the new query.

STEPS

- ☑ open the Vector Layer Controls window
- ☑ on the Polygons panel set the Style menu to All Same and press [OK]
- ☑ in the Display Manager, click on the plus sign ⊞ next to the CBSOILS_LITE layer entry to expand it
- ☑ right-click on the polygon element entry in the list and choose Mark by Query from the popup menu



- ☑ in the Mark Polygons by Query window, open the Script tab
- ☑ select the template text [???] == [???] and delete it
- ☑ enter the Query shown below:

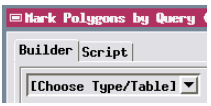
```
WLHABIT.wlopen  ==  "GOOD";
```

- ☑ press [Apply] on the Mark by Query window
- ☑ after noting the marked polygons, press the Unmark All icon button in the View window

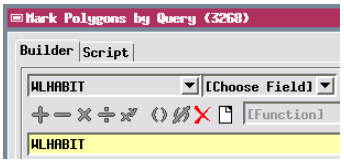
Using the Query Builder

STEPS

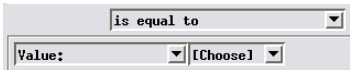
☑ switch to the Builder tab in the Mark Polygons by Query window



☑ from the Choose Type/ Table menu at the top of the panel, scroll down and choose the WLHABIT table



☑ from the Choose Field menu, scroll down and choose *wlopen*



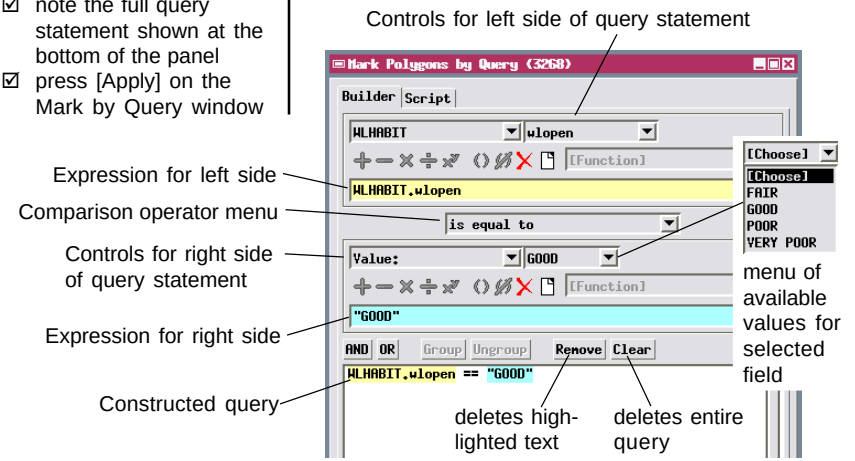
☑ from the Choose menu to the right of the Value field in the middle of the panel, choose GOOD

☑ note the full query statement shown at the bottom of the panel

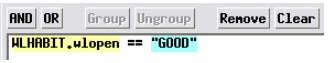
☑ press [Apply] on the Mark by Query window

The Mark by Query window for point, line, or polygon elements in a geometric layer provides an interactive Query Builder interface that simplifies constructing simple or compound queries. In this exercise we use the Query Builder to reconstruct the simple query used in the previous exercise.

Two sets of controls are provided for constructing the expressions on the left and right sides of the query statement. A menu in between allows selection of the comparison operator to use. In both expression control groups, a Choose Type/ Table menu lets you select a value type (number, constant, or text) or a table name for the expression. If you choose a table, another menu appears to the right to allow you to choose a field from the table. If you have chosen a table and field for the left side expression, the right side expression menu defaults to the *Value* setting, and an adjacent menu provides a selection of all values present in the selected field. Text fields show the left side expression (with yellow background) and right side expression (cyan background), and the text field at the bottom shows the completed query.



Compound Queries with the Builder



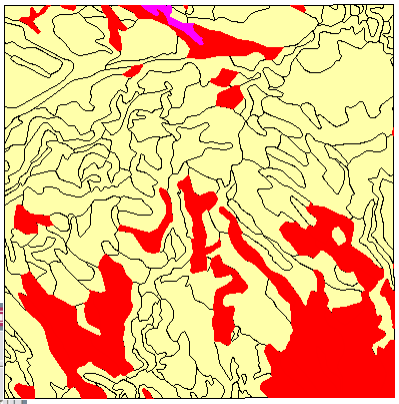
You probably noticed that the controls on the Builder panel change to reflect your selections on other controls. These changes are designed to offer appropriate choices and to guide you through construction of your query.

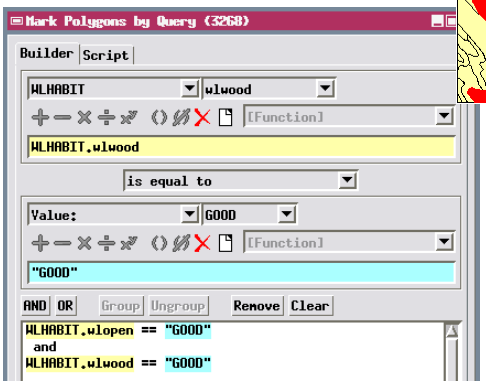
Once you have a complete simple query statement, you can go on to construct a compound query by pressing the AND or OR button above the query statement text field. This action clears the builder menus and text fields above for the left and right expressions, but retains the previous query statement in the query text field and adds the appropriate compound query keyword. You can then use the expression controls to construct the second statement for the compound query.

STEPS

- ☑ press the AND button on the Builder panel
- ☑ in the Choose Type/ Table menu for the left expression, choose the WLHABIT table
- ☑ in the Choose Field menu, choose *wlwood*
- ☑ from the Choose menu for the Value of the right-side expression, choose GOOD
- ☑ note the full compound query shown at the bottom of the panel
- ☑ press [Apply] on the Mark by Query window

The queries on this and the previous page use fields in the wildlife habitat (WLHABIT) table that list the suitability of different soil types for supporting different types of vegetation communities and associated wildlife habitat. The query on this page marks soil polygons that could support good quality open and woodland habitat.





- ☑ after noting the marked polygons, press the Unmark All icon button in the View window

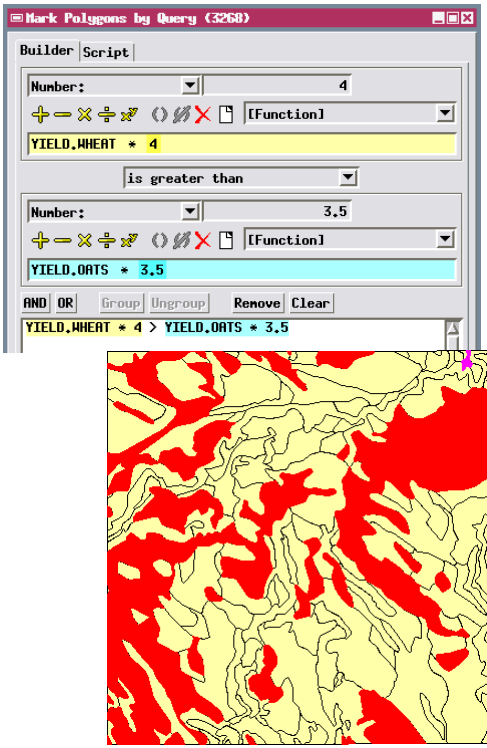
Query Calculations with the Builder

STEPS

- ☑ press the Clear button on the Builder panel
- ☑ for the left side expression, choose the YIELD table and the WHEAT field
- ☑ press the Multiply icon button in the left side expression controls
- ☑ return to the Choose Type/Table menu for this expression (which still shows YIELD) and choose Number:
- ☑ enter 4 in the field that appears to the right of the menu
- ☑ choose *is greater than* from the comparison operator menu
- ☑ for the right side expression, choose the YIELD table and the OATS field
- ☑ press the Multiply icon button in the right side expression controls
- ☑ return to the Choose Type/Table menu for this expression (which still shows YIELD) and choose Number:
- ☑ enter 3.5 in the field that appears to the right of the menu
- ☑ press [OK] on the Mark by Query window
- ☑ after noting the marked polygons, press the Unmark All icon button in the View window

The Query Builder can also be used to construct queries that incorporate numerical calculations. When you select a numeric database field for one of the expressions, a set of icon buttons become active that allow you to insert a arithmetic operator into the expression. To insert a numeric value after the operator, use the same menu you used to select the table, but this time select the value type Number, which then allows you to enter the desired numeric value. You can also use the Function menu to insert numerical functions such as round, square, and absolute value.

A query constructed using the Builder is also shown on the Script tabbed panel, where you can perform further edits if necessary to construct more complex queries.



Census Boundaries in TIGER Data

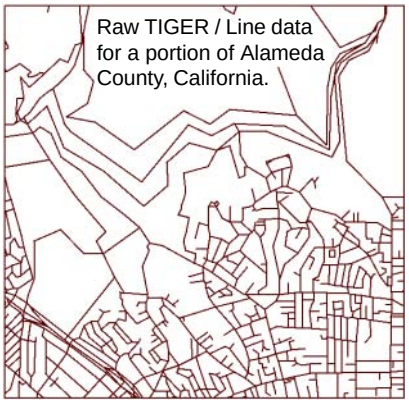
Vector objects imported from the US Census Bureau's TIGER / Line files are made up of line segments representing natural and manmade physical features as well as political boundaries. The boundaries of census tracts (and equivalent Block Numbering Areas, or BNA's) and their component census blocks usually 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. If any of these three conditions is met, the line is selected.

STEPS

- ☑ choose Display / New / 2D Display from the Display Manager
- ☑ select the ALAMEDA object from the TIGER Project File
- ☑ open the Vector Layer Controls window for the Alameda vector, set the Select option in the Lines panel to By Query, and press [Specify...]
- ☑ use the Insert procedures and / or manual entry to create the query shown below
- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window
- ☑ when you are finished with this exercise, minimize the view for Display Group 2

```
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;
```

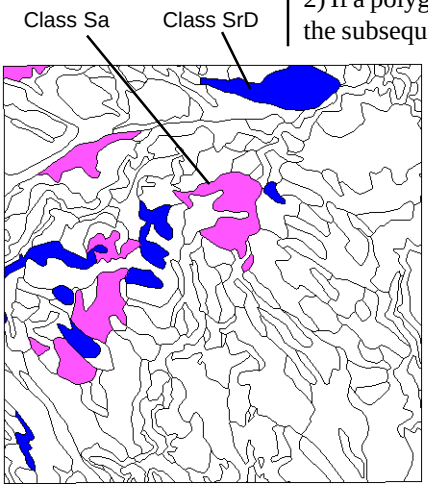


Polygon Adjacency Query: Logic

STEPS

- ☑ open the Vector Layer Controls window for the CBSOILS_LITE layer
- ☑ in the Polygons tabbed panel, set the Style option to ClassStyle
- ☑ set the Select option to By Query, and click the adjacent Specify button
- ☑ delete the previous query in the Script Editor window
- ☑ use the Insert procedures and / or manual entry to enter the query shown on the next page
- ☑ click [OK] in the Script Editor window
- ☑ click [OK] in the Vector Layer Controls window

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



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 Function list in the Script Reference 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.

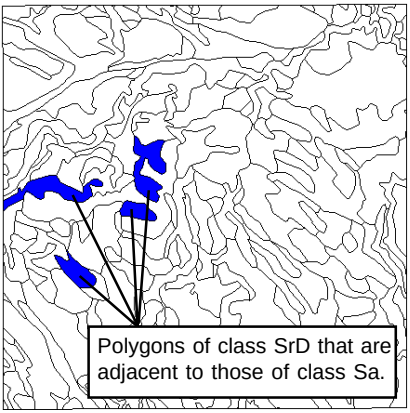
4) Check the class assignment for each adjacent polygon. If any of them match the defined adjacent class, select the current polygon for display. If none match, reject it.

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

Polygon Adjacency Query: Syntax

```
 1 if  (CLASS.Class  ==  "SrD")  then
 2   begin
 3     array  numeric  polylist  [10];  numeric  numpolys;  numeric  i;
 4     numpolys  =  GetVectorPolyAdjacentPolyList(Vect,  polylist);
 5     for  i  =  1  to  numpolys  begin
 6       numeric  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). Also declares a variable to store the number of polygons adjacent to the current polygon and a numeric counter variable for the processing loop.
4. Calls the GetVectorPolyAdjacentPolyList() function, which finds the element numbers of the adjacent polygons and stores them in the “polylist” array. It also returns the number of adjacent polygons, so the function value is assigned to “numpolys”. 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 numeric 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.



Selection Query for Dynamic Labels

STEPS

- ☑ open the Vector Layer Controls window for the CBSOILS_LITE layer
- ☑ in the Dynamic Labels section of the Polygons tabbed panel, set the Text option to By Script and press the adjacent Specify button
- ☑ enter the query shown below

```
if (POLYSTATS.Area > 50000 and
POLYSTATS.CompactRatio < 1.5)
print(CLASS.Class);
```

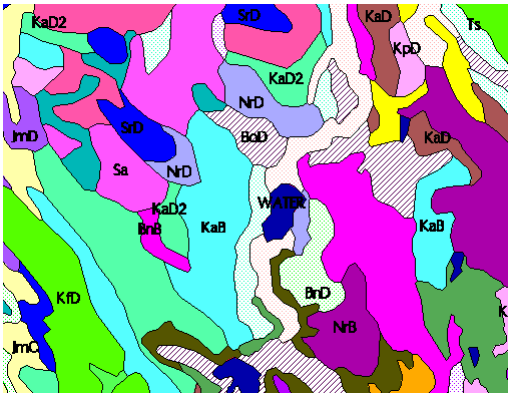
- ☑ set the Position option to Always Inside
- ☑ press [Text Style]
- ☑ press [Font] in the Style Editor window and choose mallard.of
- ☑ set the Ascender Height value to 3.00 millimeters At Scale = None
- ☑ click [OK] on the Style Editor window and on the Vector Layer Controls window

The CompactRatio (compactness ratio) field records the ratio of the boundary length of the polygon to the perimeter of a circle with equivalent area. The minimum value of 1.0 would indicate a circular polygon. The higher the compactness ratio, the more the polygon differs from a circular shape.

The Display process includes a feature to draw labels dynamically in the view for elements in a geometric object. The text for each element can be read automatically from a field in a related database table that you designate. However, you also have the option to use a query to generate the text. The query might rework the text from one or more database fields, or limit the labels to a specific set of elements.

In this exercise we use a query for dynamic polygon labels that uses two criteria to limit the label set: only larger polygons that are not too long and narrow are labeled. These selection criteria are based on the Area field and the CompactRatio field in the POLYSTATS table (see note below left).

The query for a dynamic label in a display has a special syntax. The text for the label must be specified using the SML print() function as shown.



Only larger and more compact polygons are labeled.

NOTE: The TNT Editor includes an Auto Generate Label operation for vector objects to create permanent label elements using attributes for points, lines, or polygons. The label text can be set from a single field (By Attribute) or using a query (By Script). The label text created by such a query must be assigned to a predefined string variable, Label$, in order to be returned as the label: Label$ = sprintf("%d", CLASS.ClassArea);

Computed Fields from Multiple Records

Scripts can also be used to define the values for virtual 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 numeric field in the polygon Class table for CBSOILS_LITE that shows the total area for each soil type.

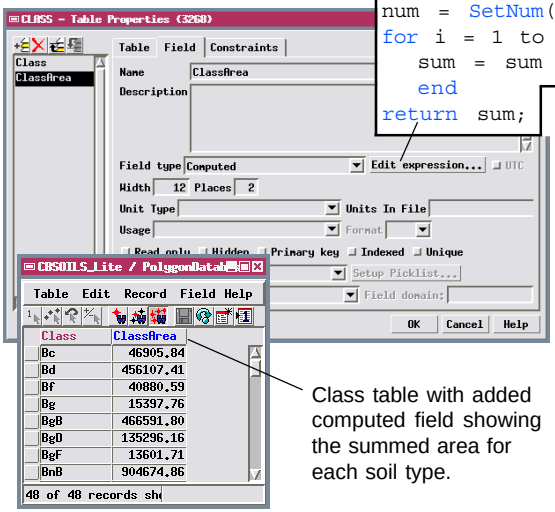
Polygon areas are stored in the POLYSTATS table for individual polygons, but we are creating the computed field in the Class table, which has one record for each soil type. The script shown here is designed to sum the polygon areas for each soil class and return that sum as the computed field value.

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.

STEPS

- ☑ right-click on the polygon element entry for the vector in the Display manager and choose Edit Relations
- ☑ in the Database Editor window, right-click on the Class table box and select Properties from the dropdown menu
- ☑ in the Table Properties window, click the Add Field icon button 
- ☑ on the Field panel, highlight the default name in the Name field and type ClassArea
- ☑ on the Field panel, select Computed from the Field Type menu and click [Edit Expression]
- ☑ enter the script shown below in the Query window

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



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

- ☑ click [OK] in the Query window
- ☑ enter 12 in the Width text box and 2 in the Places text box
- ☑ click [OK] in the Table Properties window
- ☑ double-click on the box for the Class table to open it
- ☑ choose Table / Close to close the Class table
- ☑ choose File / Close to close the Database Editor table

String Expression Fields

STEPS

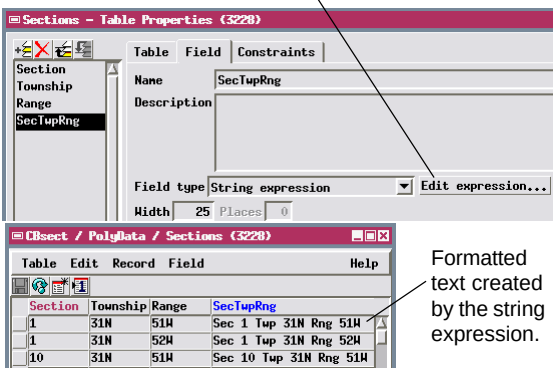
- ☑ choose Tools / Database / Edit from the TNTmips menu
- ☑ navigate to and select the CBSECT object in the CB_SECT Project File
- ☑ turn on the Polygons radio button in the Element Type window
- ☑ in the Database Editor window, right-click on the Sections table box and select Properties
- ☑ in the Table Properties window, click on the Range field in the list and click the Add Field icon button 
- ☑ on the Field panel, 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 script shown above in the Query window

A string expression field is a special type of virtual field in a database table. The simplest use of a string expression field is to link the contents of a string field in another related 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 (from one or several tables) 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 );
```

- ☑ click [OK] in the Query window
- ☑ enter 25 in the Width text box
- ☑ click [OK] in the Table Properties window
- ☑ double-click on the Sections table box to open it
- ☑ choose Table / Close to close the Sections table

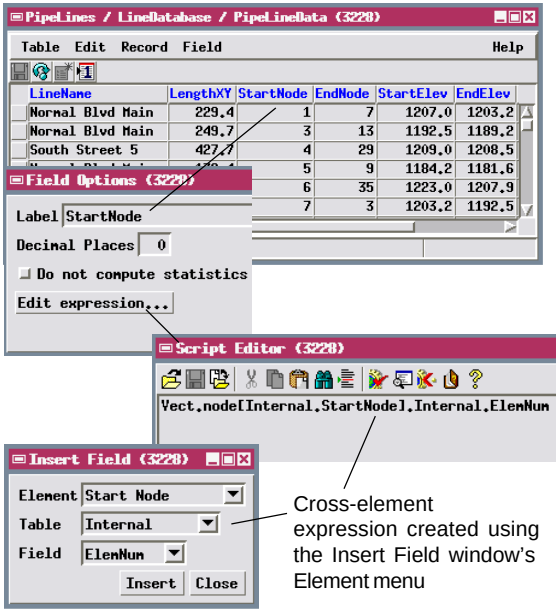


Formatted text created by the string expression.

Creating Cross-Element Expressions

Virtual fields in the element databases of a vector object can access and use attributes of other types of elements in the same vector object. For example, virtual fields for line elements can reference attributes of the each line's start and end node or the polygon to the left and right of each line. The PipeLineData table shown in this exercise includes virtual fields that show the Start Node and End Node of each line element and others that show the elevation of each of these nodes.

The expressions used to create these cross-element virtual fields require a specific structure and syntax. You can easily create the required expression using the Insert Field window. Use the Element menu on this window to select the related element from which you want to acquire the attributes, then select the table and field to automatically construct the expression, which can then be inserted into the Script Editor window.

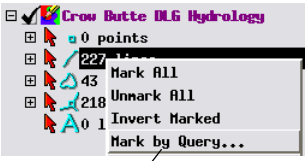


Cross-element expression created using the Insert Field window's Element menu

STEPS

- ☑ choose File / Open Database on the Database Editor window
- ☑ navigate to and select the PIPELINES object in the PIPES Project File
- ☑ turn on the Lines radio button in the Element Type window
- ☑ in the Database Editor window, right-click on the PipeLineData table box and select Open from the dropdown menu
- ☑ in the PipeLineData tabular view, right-click on the StartNode field and choose Field Options from the popup menu
- ☑ in the Field Options window, press [Edit Expression]
- ☑ in the Script Editor window, press the Insert Field icon button 
- ☑ in the Insert Field window, choose Start Node from the Element menu
- ☑ select Internal from the Table menu, then ElemNum from the Field menu
- ☑ press [Close] on the Insert Field window, [OK] on the Script Editor window, and [Cancel] on the Field Options window
- ☑ when you have completed this exercise, close the PipeLineData tabular view, then choose File / Close from all Database Editor windows

Queries to Check Digitizing Artifacts



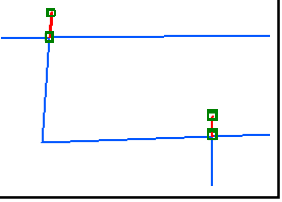
Right-click in the Editor Layer Mangager on the entry for the element type you want to edit. Choose Mark by Query to open the standard Query Editor so you can enter a selection query to mark (highlight) elements as candidates for editing.

Marking 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 queries such as the examples below. The right mouse button menu for each element type in an editable layer has a Mark by Query option that allows 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:

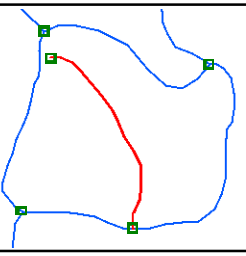
LINESTATS.Length < [your length value]



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:

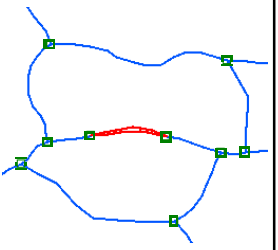
Internal.LeftPoly == Internal.RightPoly



SLIVER POLYGONS

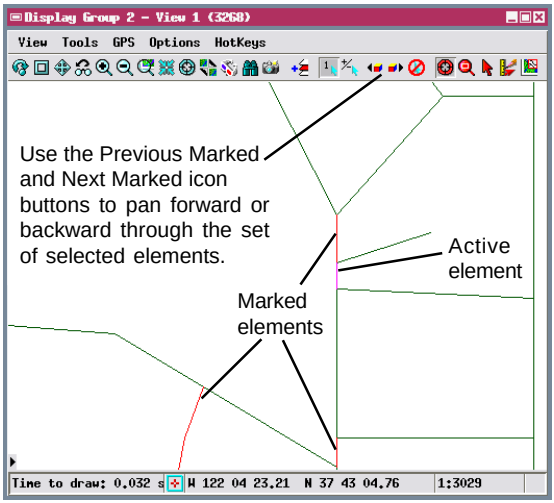
Double-tracing polygon boundaries can create extraneous sliver polygons along the boundary of two contiguous polygons. Sliver polygons usually have a much smaller area than the main polygons, and are usually highly elongate (with a high Compactness Ratio). Use a combined query on the Area and CompactRatio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

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



Pan by Query

A marking query executed in the spatial data Editor or in Display often marks more than one element. One of these marked elements is designated the "active" element; the active and marked elements are drawn in different colors in the View. Editing operations can be applied to either the active or the marked elements. You can use the Previous Marked and Next Marked icon buttons on the View window to step forward and backward through the set of marked 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 for Display Group 2
- ☑ open the Vector Layer Controls window
- ☑ set the Select option in the Lines panel to All and click [OK]
- ☑ zoom in several times until the scale shown in the view's status bar is approximately 1:3000 or 1:2500
- ☑ click the ⊞ icon for the Alameda layer entry to expand it
- ☑ click the arrow icon button for lines to enable selection
- ☑ right-click on the lines entry and choose Mark by Query from the dropdown menu
- ☑ enter the following query in the Mark by Query window:

```
LINESTATS.Length < 50
```

- ☑ click [Apply] on the Mark by Query window
- ☑ click the Next Selected icon button on the View window

The exercises in this 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 Geospatial Scripting 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 tutorial booklet *Writing Scripts with SML* and *Using CartoScripts* to expand your scripting capabilities.

Advanced Software for Geospatial Analysis

QUERIES

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

TNTmips Basic TNTmips Basic is a low-cost version of TNTmips for small projects.

TNTmips Free TNTmips Free is a free version of TNTmips for students and professionals with small projects. You can download TNTmips Free from MicroImages' web site.

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 or DVD at low cost. TNTatlas CDs/DVDs can be used on any popular computing platform.

Index

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
internet: www.microimages.com