THE RURAL TECHNOLOGY INITIATIVE

UNIVERSITY OF WASHINGTON
COLLEGE OF FOREST RESOURCES

SMALL FOREST LANDOWNER DATABASE

PROJECT SUMMARY

DATA COLLECTION AND CONVERSION PROCESS

A DETAILED LOOK AT THE SMALL FOREST LANDOWNER DATABASE

INTRODUCTION

The process used to create the Small Forest Landowner Database consists of essentially 6 steps; data collection, normalization, parsing, adding watersheds, integration and formatting. Essential to the project was the creation of a project web page (http://www.ruraltech.org/gis/sflodbms/) utilizing Active Server Pages (ASP) to track data collection and conversion efforts. The data collection process itself was time consuming but for the most part, the counties were eager to help with the project.

The data normalization and conversion process became easier with each successive county as better SQL and VB Scripts were written. The following is the step-by-step process that was used to collect and convert the county parcel data, as well as time estimates for collecting GIS data for counties currently without GIS.

DATA COLLECTION

Out of the 18 Western Washington counties that were contacted by the Small Forest Landowner Office and the Rural Technology Initiative, only Jefferson County refused to supply us with digital data. Each of the 18 counties was asked for tax parcel information concentrating on classified forest, designated forest, open space timber, timberlands or any forestry related activity. For those forested parcels we asked for acres, residence information, the landowners name and address, the legal description including the township, section and range, the site address and when the parcel record was last updated. In addition to tabular data we asked for any tax parcel GIS coverage layers.

Most of the counties were very helpful in getting us the data in a useable format, typically MS Excel, Access, Dbase, Comma Separated Value (CSV) and Text files. Only a few counties requested payment for the data and of those that did the prices were minimal, usually around $100. About half of the Westside counties had GIS data in addition to tabular information, sometimes supplying us with all the parcels for the county and other times only supplying us with the timbered parcels.

Data was delivered via post on CD, email, and FTP. The FTP site provided the most convenient method of data sharing among the participants in the project. In addition to the FTP site a web site was created to track the county contacts, communication with the counties, data collection and data quality status. The web site enabled many people across the state to work on the same project and pick up where others left off without duplicating efforts or going down the same path twice. The web site was essential in the completion of the data collection effort.

Of the data requested the most common missing piece needed to fulfill the requirements of the legislative report outlined in the Salmon Recovery Act was residence information. Only two counties kept track of residence status, other counties however kept track of improvements on the land. After discussions with county assessors it was recognized that improvements on the land could be a surrogate for residence information, and in the opinion of most county assessors those improvements we likely primary residences, not vacation homes. More information about how residence information was extracted from county databases is available in the accompanying detailed county reports.

NORMALIZATION

The first step in the data conversion process is normalization. Normalization is a process for converting a relation (or table) that has certain problems to two or more relations that do not have those problems.[1] All the data from the counties was normalized to the Domain/Key Normal Form ensuring no insertion or deletion anomalies. For more information about normalization, see Appendix A - Normalization Described.

Since most of the county data came in single tables, parcels were listed multiple times if there were multiple owners and owners were listed multiple times if they owned multiple pieces of property. In order to normalize the data into the Domain/Key Normal Form, four basic SQL queries (Appendix B - Normalization SQL) and a VB script were written to separate the data into four relations; LandOwner, Parcel_LandOwner, Parcel and LegalDescription. In order to ensure that the normalization was correct, a query was run to de-normalize the relations and re-assemble the original data table.

RELATION PARSING AND FORMATTING

Once the data was in four separate relations it had to be parsed into the final format. Many counties had township, range and section information in the parcel id that had to be parsed out into separate fields. City, State and zip code were often in the same field as well. To parse the data into the appropriate fields VB scripts were written. The generic version of these scripts can be found in Appendix C - Parsing Fields. These scripts had to be modified for each county to extract the appropriate data and put it into the final database format. Complete versions of these scripts for each county can be found in the accompanying detailed county reports.

After parsing and formatting the data properly for the county, the database was hand checked for redundant landowners. More often than not there were multiple entries for the same landowner with only slight differences in either their address or name. These redundant entries were combined in each county to give a better representation of the number of landowners in each county. It is interesting to note that almost all counties had this same problem. In one county, Weyerhaeuser appeared over 20 times with just slight differences in the spelling, address or zip code.

[1] Kroenke, David M., *Database Processing – Fundamentals, Design & Implementation, Seventh Edition*

ADDING WATERSHEDS AND REGIONS

The next step in the data conversion process was to identify which watersheds and DNR Regions each parcel was in. It was decided early on that in order to maintain consistency among the Westside parcels that county GIS data would not be used for this task. To identify which region and watershed particular parcels were "likely" in we used the Township, Range and Section information provided by the counties. Where available we used Quarter Section information as well. Using a GIS to analyze which Region each section had the majority of its area in we effectively determined Region information. This method produced one "likely" region candidate based on probability by area. If a section had 40% of it's area in the Central Region and 60% of its area in the Southwest Region then the section was determined to be in the Southwest Region and all parcels with legal descriptions in that section were assigned to the Southwest Region.

Determining which watersheds a parcel could be in was approached in a similar fashion. However, it was realized that unlike Regions where a landowner would only go to the local region office, parcels could be in multiple watersheds. To capture this information the database was constructed so that a parcel could belong to many watersheds and a watershed could belong to many parcels. To identify watersheds a GIS was used, determining for every section or quarter section which watersheds a parcel could be in.

To attribute the parcels with the watersheds and regions we intersected (or unioned) the statewide PLS section coverage with the WAU coverage, identifying every candidate watershed. Then, for each resulting section/WAU polygon a field (TRSQ) was added identifying the section number (ex. T08N R05W S13 Q1). For each parcel in the database the same TRSQ field was added and then joined to the section/WAU polygon attribute table. The relational join of these two tables yielded the Parcel_Watershed table (See Appendix D - Region and Watershed Determination).

There are a few issues that arise when defining watersheds in this fashion. It is certain that many of the parcels in the database are larger than one section and therefore those parcels will be represented poorly. For those parcels that are larger than one section the assessor used some criteria for deciding which section to attribute the parcel with. Until reliable GIS data can be obtained from each county, errors like this will be common.

COUNTY INTEGRATION

The final step in creating the Small Forest Landowner Database was to integrate the counties into one master database. To ensure that each parcel in the database would have a unique id, each counties parcel ids were prefixed with the statewide county code and a dash. Since we chose to use the parcel ids that the counties provided this step was necessary to guarantee a unique parcel id across the state. The same prefix process was also applied to Land Owners by county to ensure unique ONWER_ID's.

Once the unique ids had been generated, a new database was created with all the final formatting of the attributes defined. Five SQL append queries were generated (See Appendix E

- Appending Counties To Master Database) to add data from each counties database to the master database.

FINAL FORMATTING

Once all of the counties had been merged into one database, redundant landowners had to be removed. For each county, redundant landowners were removed before the final master database merger. However, since there are many landowners who own land in multiple counties, those owners could only be removed after final integration. The process of reducing redundant landowners into one was slow for the 21,000 + landowners in Washington State. A VB script was written (See) to help with this task but was far from a perfect fix for the redundancy issue in the LandOwner table.

The final step in formatting the database was to reduce the storage requirements for the data as much as possible and index the appropriate fields to speed searches and joins. Each field in the database was formatted for the least amount of storage possible based on the length of the longest member of the field and the data type. There are opportunities to further reduce the storage requirements of the database and increase performance, however, the gains would be small.

KNOWN ISSUES RELATING TO REPORTING REQUIREMENTS IN THE SALMON RECOVERY ACT

Excerpts from the Salmon Recovery Act relating to the reporting requirements of the Small Forest Landowner Office at the Department of Natural Resources followed by functionality that the SFLO Database will have to address those reporting requirements.

> *By December 1, 2000, the small forest landowner office shall provide a report to the board and the legislature containing:*
>
> *(a) Estimates of the amounts of nonindustrial forests and woodlands in holdings of twenty acres or less, twenty-one to one hundred acres, one hundred to one thousand acres, and one thousand to five thousand acres, in western Washington and eastern Washington, and the number of persons having total nonindustrial forest and woodland holdings in those size ranges;*

We have data from most every Western Washington County to provide the desired information above, but some of the county data is more accurate than others. Of the nonindustrial forests and woodlands we have “Total Acres” for all counties, but some also offer us “Total Timber Acres” or “Acres in Timberland Program” which gives us a more accurate number on land that is actually available for forest practices.

To answer this question we have available: total number of holdings in those size ranges, total number of acres in those size ranges, number of tax paying entities with total holdings in those size ranges and number of tax paying entities with holdings in each size range.

(b) Estimates of the number of parcels of nonindustrial forests and woodlands held in contiguous ownerships of twenty acres or less, and the percentages of those parcels containing improvements used:

(i) As primary residences for half or more of most years;

(ii) as vacation homes or other temporary residences for less than half of most years; and

(iii) for other uses;

"Contiguous ownerships" cannot be identified if the ownership is made up of more than one parcel unless GIS data is available. While some counties record improvements on the land, the data that is necessary to provide residency information is only available from two of the Western Washington Counties. Most counties have each parcel defined as either residential or timberland. A few counties have data that shows that a parcel has an improvement on it, but not whether it is primary or vacation home.

To answer this question we have available: It is unlikely that many "contiguous ownerships" of twenty acres of less will be made up of more than one parcel, therefore we should have a good estimate on this. If we find using GIS data that there are a significant number of holdings in contiguous ownerships that are made up of more than one parcel then we may be able to make some assumptions about counties where we do not have GIS data. In conversations with county assessors it was stated that we could assume any improvement on a timberland parcel is a home of some sort. County assessors also felt that "very few if any" of the improvements on those timberlands were vacation homes, leaving us with the assumption that any improvement is a primary home.

(c) The watershed administrative units in which significant portions of the riparian areas or total land area are nonindustrial forests and woodlands;

For counties with GIS data, riparian and total land area estimates will be fairly easy if we can get a definition of "significant". Is 30% of riparian area or 15% total land area significant? For counties without GIS data this will be a bit more of a challenge. Using Township, Range and Section information we can determine which watershed administrative unit a forested

parcel is most "likely" to belong to, meaning section xx has more area in one WAU than another. Where we have quarter and sixteenth section info this "likelihood" will be more accurate. Knowing which WAU a parcel is "likely" in will give us the ability to estimate the total land area of nonindustrial forests and woodlands. Estimates of riparian area timberlands will not be possible without some heavy-duty statistical work looking at GIS/non GIS relationships.

(d) Estimates of the number of forest practices applications and notifications filed per year for forest road construction, silvicultural activities to enhance timber growth, timber harvest not associated with conversion to nonforest land uses, with estimates of the number of acres of nonindustrial forests and woodlands on which forest practices are conducted under those applications and notifications; and

It has been determined by talking with database administrators at the DNR that linking the SFLO database with the DNR's Forest Practice Database will not be possible. The FPA database will be able to tell us the total number of applications submitted for road construction, silvicultural activities to enhance timber growth and timber harvest not associated with conversion to nonforest land uses. Using township, range and section information we may be able to estimate the number of those permits that that were conducted on nonindustrial forests since the FPA database contains TRS information for each permit. This approach will have significant error associated with it.

(e) Recommendations on ways the board and the legislature could provide more effective incentives to encourage continued management of nonindustrial forests and woodlands for forestry uses in ways that better protect salmon, other fish and wildlife, water quality, and other environmental values.

RTI will be working hard to find answers to these questions over the next few years. It is certain that the Small Forest Landowner Office will also have much to offer the legislature with respect to adaptive legislation.

5. By December 1, 2002, and every four years thereafter, the small forest landowner office shall provide to the board and the legislature an update of the report described in subsection (5) of this section, containing more recent information and describing:

(a) Trends in the items estimated under subsection (5)(a) through (d) of this section;

(b) Whether, how, and to what extent the forest practices act and rules contributed to those trends; and

(c) Whether, how, and to what extent:

(i) The board and legislature implemented recommendations made in the previous report; and

(ii) implementation of or failure to implement those recommendations affected those trends.

Trend analysis using GIS and tabular data should be straight forward if the same methods are used to collect the information in the future. However, as more counties compile GIS information, data collection and analysis techniques will change by county, and if care is not used, varied results are likely.

TIME ESTIMATES FOR GIS DATA COLLECTION

Complete GIS data was collected for only 8 of the 35 counties. A few more counties had digital parcel data but it was either incomplete or in a CAD type format. The type of work involved in generating parcel level GIS data includes locating the assessor's paper base maps, scanning the base maps, vectorization of the scanned images and then attributing and cleaning the scanned vector data. One company in particular specializes in exactly this kind of work. NOBEL Systems (http://www.nobel-systems.com) will scan and vectorize the parcel data for around $2.50 a parcel.

If it were possible to just scan and convert the forested parcels for each county the completion of the SFLO GIS would cost less than $200,000 to complete. However, NOBEL Systems and others charge based on complete map sheets. It is not possible for them to convert only the forested parcels and charge the same $2.50 rate. To pick out and vectorize only the forested parcels the costs would be something more like $20 per parcel, skyrocketing the price to at least $1,600,000. On top of the high cost of vectorizing only the forested parcels, maintenance of the GIS data would be nearly impossible.

If the Washington State Legislature is serious about gathering parcel level data from all the counties every 4 years then a statewide cadastral GIS needs to be constructed. Given the low

cost of conversion for parcels by map sheet, it is likely that the entire state could be brought up to cadastral standards for less than $10,000,000. It is appropriate to consider a statewide approach involving State, County and local funding to accomplish this task and should be looked at more closely before attempting to generate a GIS coverage for just the forested parcels.

CONCLUSIONS

Extensive consideration of the databases use must be taken into account when constructing databases of this size. Careful planning must accompany every step of the process in order to ensure that the final product will be consistent and meet the needs of the Small Forest Landowner Office. Initially, a very different product was envisioned, from data storage to the focus on GIS data everything changed.

Given the requirements spelled out in the Salmon Recovery Act it appears that GIS is a necessary tool. In the construction of the SFLO database we did not use GIS data even for the counties where we had it in order to ensure consistency across the counties and in the future. It appears to come down to a question of precision vs. accuracy. If the steps outlined here are followed in the construction of successive databases then the results should be at the very least precise, that is they will be in error towards the same side of things from year to year. In order to be accurate, a statewide parcel level GIS needs to be created. Without consistent GIS data across the state it will be difficult to gauge the validity of the results that come out of this database.

With the legislature requiring reports of the nature described within the Salmon Recovery Act it is appropriate to begin discussions about constructing a statewide GIS with some legislative funding. With initial help and investment by the state, parcel level GIS data could be created for every county, vastly simplifying the job of reporting for the Small Forest Landowner Office in the future.

APPENDIX A - NORMALIZATION DESCRIBED

Text from http://luna.pepperdine.edu/~ckettemb/class/DBNorm.html

INTRODUCTION

According to (Elmasri & Navathe, 1994), the normalization process, as first proposed by Codd (1972), takes a relation schema through a series of tests to “certify” whether or not it belongs to a certain normal form. Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF was proposed later by Boyce and Codd and is known as Boyce-Codd normal form (BCNF). All these normal forms are based on the functional dependencies among the attributes of a relation. Later, a

fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts or multi-valued dependencies and join dependencies, respectively.

Normalization of data can be looked on as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties. One objective of the original normalization process is to ensure that the update anomalies do not occur.

Normal forms provide database designers with:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.

- A series of tests that can be carried out on individual relation schema so that the relational database can be normalized to any degree. When a test fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

Normal forms, when considered *in isolation* from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. Two of these properties are:

- The *lossless* join or nonadditive join property, which guarantees that the spurious tuple problem does not occur.

- The dependency preservation property, which ensures that all *functional dependencies* are represented in some of the individual resulting relations.

DEFINITIONS

A relation is defined as a *set of tuples*. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for *all* their attributes.

Any set of attributes of a relation schema is called a superkey. Every relation has at least one superkey—the set of all its attributes. A key is a *minimal superkey*, i.e., a superkey from which we cannot remove any attribute and still have the uniqueness constraint hold.

In general, a relation schema may have more than one key. In this case, each of the keys is called a candidate key. It is common to designate one of the candidate keys as the primary key of the relation. A foreign key is a key in a relation R but it's not a key (just an attribute) in other relation R' of the same schema.

Integrity Constraints: the entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values for the primary key implies that we cannot identify some tuples. The

referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an *existing tuple* in that relation.

An attribute of a relation schema R is called a prime attribute of the relation R if it is a member of *any key* of the relation R. An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.

A functional dependency, denoted by X- > Y, between two sets of attributes X and Y that are subsets of R specifies a constraints on the possible tuples that can form a relation instance of R.

NORMAL FORMS

First Normal Form (1NF)

First normal form is now considered to be part of the formal definition of a relation; historically, it was defined to disallow *multivalued attributes, composite attributes, and their combinations.* It states that the domains of attributes must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a single *value* from the domain of that attribute.

Practical Rule[1]: "*Eliminate Repeating Groups,*" i.e., make a separate table for each set of related attributes, and give each table a primary key.

Formal Definition[2]: A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only.

Second Normal Form (2NF)

Second normal form is based on the concept of *fully functional dependency.* A functional X- > Y is a fully functional dependency is removal of any attribute A from X means that the dependency does not hold any more. A relation schema is in 2NF if every nonprime attribute in relation is *fully functionally dependent* on the primary key of the relation. It also can be restated as: a relation schema is in 2NF if every nonprime attribute in relation is not partially dependent on *any key* of the relation.

Practical Rule[1]: "*Eliminate Redundant Data,*" i.e., if an attribute depends on only part of a multivalued key, remove it to a separate table.

Formal Definition[2]: A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.

Third Normal Form (3NF)

Third normal form is based on the concept of *transitive dependency*. A functional dependency X- > Y in a relation is a transitive dependency if there is a set of attributes Z that is *not a subset* of any key of the relation, and both X- > Z and Z- > Y hold. In other words, a relation is in 3NF if, whenever a functional dependency X- > A holds in the relation, either (a) X is a superkey of the relation, or (b) A is a prime attribute of the relation.

Practical Rule[1]: "*Eliminate Columns not Dependent on Key,*" i.e., if attributes do not contribute to a description of a key, remove them to a separate table.

Formal Definition[2]: A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

Boyce-Codd Normal Form (BCNF)

Boyce-Codd normal form is stricter than 3NF, meaning that every relation in BCNF is also in 3NF; however, a relation in 3NF is *not necessarily* in BCNF. A relation schema is an BCNF if whenever a functional dependency X- > A holds in the relation, then X is a superkey of the relation. The only difference between BCNF and 3NF is that condition (b) of 3NF, which allows A to be prime if X is not a superkey, is absent from BCNF.

Formal Definition[2]: A relation is in Boyce/Codd normal form (BCNF) if and only if every determinant is a candidate key. [A determinant is any attribute on which some other attribute is (fully) functionally dependent.]

Fourth Normal Form (4NF)

Multivalued dependencies are a consequence of first normal form, which disallowed an attribute in a tuple to have a set of values. If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation instances consistent.

Fourth normal form is based on multivalued dependencies, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations. A relation scheme R is in 4NF with respect to a set of dependencies F is, for every *nontrivial* multivalued dependency X- > F, X is a superkey for R.

Practical Rule[1]: "*Isolate Independent Multiple Relationships,*" i.e., no table may contain two or more 1:n or n:m relationships that are not directly related.

Formal Definition[2]: A relation R is in fourth normal form (4NF) if and only if, whenever there exists a multivalued dependency in the R, say A- > > B, then all attributes of R are also functionally dependent on A.

Fifth Normal Form (5NF)

In some cases there may be no losses join decomposition into two relation schemas but there may be a losses join decomposition into more than two relation schemas. These cases are handled by the *join dependency* and fifth normal form, and it's important to note that these cases occur very rarely and are difficult to detect in practice.

> Practical Rule[1]: *"Isolate Semantically Related Multiple Relationships,"* i.e., there may be practical constraints on information that justify separating logically related many-to-many relationships.
>
> Formal Definition[2]: A relation R is in fifth normal form (5NF)—also called projection-join normal form (PJNF)—if and only if every join dependency in R is a consequence of the candidate keys of R.

A join dependency (JD) specified on a relations schema R, specifies a constraint on instances of R. The constraint states that *every legal instance* of R should have a losses join decomposition into sub-relations of R, that when reunited make the entire relation R. A relation schema R is in fifth normal form (5NF) (or project-join normal form (PJNF)) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(R1, R2, ..., Rn) in F (implied by F), every Ri is a superkey of R.

Domain Key Normal Form (DKNF)

We can also always define stricter forms that take into account additional types of dependencies and constraints. The idea behind domain-key normal form is to specify, (theoretically, at least) the "ultimate normal form" that takes into account all possible dependencies and constraints. A relation is said to be in DKNF if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and the key constraints specified on the relation.

For a relation in DKNF, it becomes very straightforward to enforce the constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint on the relation is enforced. However, it seems unlikely that complex constraints can be included in a DKNF relation; hence, its practical utility is limited.

NOTES:

1. Most of the modern DBMS systems offer "some kind" of DKNF normalization, by giving the designer the opportunity to assign domain and specific properties (such as key specification) to each attribute of a relation part of a schema. However, that's NOT a guarantee that the resulted relation is in DKNF.

2. The relationship between the 7 levels of normalization (1 through 5, plus BCNF and DKNF) can intuitively be represented as layered, concentric circles, with the largest circle as the 1NF, then a smaller, inside circle as the 2NF, and so on, the smallest circle being the

DKNF. This is because a relation is defined as being in a given normal form (let's say 2NF) if and only if it is already in the immediately previous normal form (i.e., 1NF) and satisfies additional requirements.

REFERENCES

Elmasri, R., & Navathe, S. (1994). *Fundamentals of Database Systems*. 2nd ed. Redwood City, CA: The Benjamin/Cummings Publishing Co. pp. 143 – 144, 401, 407 – 409, 435, 438, 440, 442 - 443.

[1]Rettig, Marc. (1995). *Database Programming & Design*. Miller Freeman, Inc.

[2]Date, C. J. (1990). *An Introduction to Database Systems*. 5th ed. Volume I. Reading, MA: Addison-Wesley Publishing Company.

APPENDIX B - NORMALIZATION SQL

All but a few counties sent their data completely de-normalized in a single table. Therefore each counties data had to be normalized into the 4 major tables: Parcel, LandOwner, Parcel_LandOwner and Legal_Description. The SQL statements for each county varied depending on column headings but for the most part were very similar.

Before these normalization SQL statements could be executed the original county table had to be checked for Null values. When attempting to construct the Parcel_LandOwner table a join must be performed between the Parcel table and the LandOwner table. If there are any blanks (or Null values) then the join will fail and you will get invalid normalization results. The function SetNull() in the VBA code module ReplaceNull.bas replaces Null values with the word “nullvalue”. After the normalization is complete the “nullvalues” can be replaced again with Null values using the ReplaceNull() function.

In addition to replacing Null values, some counties provided tables with multiple legal descriptions for the same parcel. For these parcels the legal descriptions had to be concatenated into one field for use in the SFLO database format. The VBA script CombineLegalText.bas merges multiple legal descriptions into one.

COMBINLEGALTEXT.BAS

```
Attribute VB_Name = “CombineLegalText”
Option Compare Database

Public Sub CombineLegalText()
    ' local variables
    Dim rs As New ADODB.Recordset
    Dim newRS As New ADODB.Recordset
    Dim strSql As String
    Dim newSQL As String
```

```
    ' construct SQL string
    strSQl = “SELECT * FROM TABLE ORDER BY ParcelID,
LegalLineNo;”
    newSQL = “SELECT * FROM NEW_TABLE;”

    ' open the recordset
    rs.Open strSQl, CurrentProject.Connection, adOpenDynamic,
adLockOptimistic
    newRS.Open newSQL, CurrentProject.Connection, adOpenDynamic,
adLockOptimistic

    ' Loop and concatenate
    Do While Not rs.EOF
        If rs.Fields(“LegalLineNo”) = 1 Then
            newRS.AddNew
            newRS.Fields(“ParcelID”) = rs.Fields(“ParcelID”)
            newRS.Fields(“LegalText”) = rs.Fields(“LegalText”)
            newRS.Update
        Else
            newRS.Fields(“LegalText”) = newRS.Fields(“LegalText”)
& “ “ & rs.Fields(“LegalText”)
            newRS.Update
        End If
        rs.MoveNext
    Loop

    ' close the recordsets
    rs.Close
    newRS.Close

End Sub
```

REPLACENULL.BAS

```
Attribute VB_Name = “ReplaceNull”
Option Compare Database
‘ Replace the TABLE and FIELD variables with the appropriate
table name and field name
Public Sub SetNull()
    ' local vars
    Dim rs As New ADODB.Recordset
    Dim rsSql As String

    ' generate SQL - TABLE is the Original Table Name
    rsSql = “SELECT * FROM TABLE;”

    ' open the recordset
    rs.Open rsSql, CurrentProject.Connection, adOpenDynamic,
```

```
adLockOptimistic

    ' loop and replace – FIELD is the Field Name to replace in
    theField = "FIELD"
    Do While Not rs.EOF
        If IsNull(Trim(rs.Fields(theField))) Then
            rs.Fields(theField).Value = "nullvalue"
        Else
            rs.Fields(theField) = Trim(rs.Fields(theField))
        End If
        rs.Update
        rs.MoveNext
    Loop

    ' close the recordset
    rs.Close

End Sub


Public Sub ReplaceNull()
    ' local vars
    Dim rs As New ADODB.Recordset
    Dim rsSql As String

    ' generate SQL - TABLE is the Original Table Name
    rsSql = "SELECT * FROM TABLE;"

    ' open the recordset
    rs.Open rsSql, CurrentProject.Connection, adOpenDynamic,
adLockOptimistic

    ' loop and replace – FIELD is the Field Name to replace in
    theField = "FIELD"
    Do While Not rs.EOF
        If rs.Fields(theField) = "nullvalue" Then
            rs.Fields(theField).Value = Null
        Else
            rs.Fields(theField) = Trim(rs.Fields(theField))
        End If
        rs.Update
        rs.MoveNext
    Loop

    ' close the recordset
    rs.Close

End Sub
```

USE CODE QUERY

The Use Code Query is used to filter out any parcels that are not designated as timber (Openspace, Designated, Clasified and Timberlands). This filtered table is then used for normalization into the final database format.

```
SELECT OriginalData.[Parcel Number], OriginalData.[Owners Name],
OriginalData.[Mailing Address Line 1], OriginalData.[Mailing
Address Line 2], OriginalData.[Mailing Address Line 3],
OriginalData.City, OriginalData.State, OriginalData.Zip,
OriginalData.Section, OriginalData.Township, OriginalData.Range,
OriginalData.[Brief Legal Description], OriginalData.[Use Code],
OriginalData.[Land Use Code], OriginalData.[Total Acre Per
Parcel], OriginalData.[Acres in Forestland Program] FROM
OriginalData WHERE (((OriginalData.[Use Code])="95" Or
(OriginalData.[Use Code])="88" Or (OriginalData.[Use Code])="94"
Or (OriginalData.[Use Code])="87"));
```

PARCEL QUERY

The Parcel Query is used to extract the Parcel table information out of the filtered original data table.

```
SELECT DISTINCT FilteredData.[Parcel Number] AS Expr1,
FilteredData.[Acres in Forestland Program] AS Expr2,
FilteredData.[Total Acre Per Parcel] AS Expr3, FilteredData.[Use
Code] AS Expr4, FilteredData.[Land Use Code] AS Expr5 INTO Parcel
FROM FilteredData WHERE (((FilteredData.[Use Code])="95" Or
(FilteredData.[Use Code])="94" Or (FilteredData.[Use Code])="88"
Or (FilteredData.[Use Code])="87"));
```

LAND OWNER QUERY

The Land Owner Query is used to separate out unique landowners into a separate LandOwner table. Placing landowners in a separate table helps eliminate duplicate landowners in the database. For every county there was many more parcels than landowners since many landowners owned multiple parcels. Seperating these owners into their own table revealed mistakes in the counties databases. Often, the same owner was listed in the database multiple times with only slight differences like and extra space in the address or a "." after their middle initial.

```
SELECT DISTINCT FilteredData.[Owners Name], FilteredData.[Mailing
Address Line 1], FilteredData.[Mailing Address Line 2],
FilteredData.[Mailing Address Line 3], FilteredData.City,
FilteredData.State, FilteredData.Zip INTO LandOwner FROM
FilteredData;
```

PARCEL LANDOWNER QUERY

The Parcel LandOwner Query is used to make the Parcel_LandOwner relationship table. This query joins the newly created LandOwner table with the FilteredData table and relies on the fact that there are no Null values in the joined fields. Before running the query the "OWNER_ID" AutoNumber field must be added to the LandOwner table. After the table has been saved, the AutoNumber field should be converted to an Integer field so that it does not change or generate new numbers, then the Parcel_LandOwner Query can be run.

```
SELECT LandOwner.OWNER_ID, FilteredData.[Parcel Number] INTO
Parcel_LandOwner FROM LandOwner INNER JOIN FilteredData ON
(LandOwner.City = FilteredData.City) AND (LandOwner.State =
FilteredData.State);
```

LEGAL DESCRIPTION QUERY

The Legal Description Query extracts all the Legal information out of the FilteredData table and puts it into the LegalDescription table. The CombineLegalText script will need to be run if there are multiple legal descriptions in separate records.

```
SELECT DISTINCT FilteredData.[Parcel Number], FilteredData.[Brief
Legal Description], FilteredData.Township, FilteredData.Range,
FilteredData.Section INTO LegalDescription FROM FilteredData;
```

After running the Legal Description Query an ID column must be added to the LegalDescription table as an AutoNumber. Once the table has been AutoNumbered, change the data type to Integer .

Once the counties data had been normalized into the four separate table the "nullvalues" had to be replaced with their original Null values using the ReplaceNull() function in the ReplaceNull.bas VBA module.

APPENDIX C - PARSING FIELDS

Each of the VBA code modules below will need to be modified for each county. The code below is to give a general idea of how to parse the data for each county. Since some counties have the Township, Section and Range in one field and others in many fields, the code below will have to be modified. For specific county scripts refer to the companion county-by-county manual.

PARSEPARCEL.BAS

```
Attribute VB_Name = "ParseParcel"
Public Sub Parcel()
    ' Local variables
    Dim SourceRS As ADODB.Recordset
    Dim OutputRS As ADODB.Recordset
    Dim SourceSQL As String
    Dim OutputSQL As String
    Dim varParcelField As String
```

```
    Dim varAcresField As String
    Dim varLUCField As String

    ' Set the field names
    varParcelField = “ParcelID”
    varAcresField = “Acres”
    varLUCField = “LandUseCode”

    ' Construct the SQL strings
    SourceSQL = “SELECT * FROM County_Parcel ORDER BY “ &
varParcelField & “;”
    OutputSQL = “SELECT * FROM Parcel;”

    ' Make the recordsets
    Set SourceRS = New ADODB.Recordset
    Set OutputRS = New ADODB.Recordset

    ' Open the recordsets
    SourceRS.Open SourceSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic
    OutputRS.Open OutputSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic

    ' Parse the data into the appropriate columns
    Do While Not SourceRS.EOF
        varPARCEL_ID = SourceRS.Fields(varParcelField)
        varCOUNTY_ID = ?? ' County ID
        varREGION_ID = Null ‘ Calculate this later
        varTimberAcres = Null ‘ If available replace Null
        varTotalAcres = SourceRS.Fields(varAcresField)
        varLAND_USE_CD = SourceRS.Fields(varLUCField)
        If SourceRS.Fields(varLUCField) = 11 Or
SourceRS.Fields(varLUCField) = 15 Or SourceRS.Fields(varLUCField)
= 18 Then
            varRESIDENCE_CD = 1 'primary residence
        ElseIf SourceRS.Fields(varLUCField) = 19 Then
            varRESIDENCE_CD = 2 'vacation home
        Else
            varRESIDENCE_CD = 0 'not a residence
        End If

        ' Insert records into the Parcel Table
        With OutputRS
            .AddNew
            .Fields(“PARCEL_ID”).Value = varPARCEL_ID
            .Fields(“COUNTY_ID”).Value = varCOUNTY_ID
            .Fields(“REGION_ID”).Value = varREGION_ID
            .Fields(“TimberAcres”).Value = varTimberAcres
            .Fields(“TotalAcres”).Value = varTotalAcres
            .Fields(“LAND_USE_CD”).Value = varLAND_USE_CD
```

```
            .Fields("RESIDENCE_CD").Value = varRESIDENCE_CD
            .Update
        End With

        ' Move to the next record
        SourceRS.MoveNext

    Loop

    ' Close the recordsets
    SourceRS.Close
    OutputRS.Close

End Sub
```

PARSELANDOWNER.BAS

```
Attribute VB_Name = "ParseLandOwner"
Public Sub LandOwner()
    ' Dimension the variables
    Dim OriginalRS As ADODB.Recordset
    Dim OwnerRS As ADODB.Recordset
    Dim OriginalSQL As String
    Dim OwnerSQL As String

    ' Construct the SQL strings
    OriginalSQL = "SELECT * FROM County_LandOwner ORDER BY
OWNER_ID;"
    OwnerSQL = "SELECT * FROM LandOwner;"

    ' Make the recordsets
    Set OriginalRS = New ADODB.Recordset
    Set OwnerRS = New ADODB.Recordset

    ' Open the recordsets
    OriginalRS.Open OriginalSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic
    OwnerRS.Open OwnerSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic

    ' Parse data into appropriate fields
    Do While Not OriginalRS.EOF
        varOWNER_ID = OriginalRS.Fields("OWNER_ID")
        varPrefix = Null
        varFirstName = Null
        varMiddleName = Null
        varLastName = Null
        varSuffix = Null
        varTitle = Null
        varOrg = Null
        varAddress1 = Null
```

```
        varAddress2 = Null
        varAddress3 = Null
        varCity = Null
        varState = Null
        varRegion = Null
        varPostalCode = Null
        varCountry = Null
        varTelephone = Null
        varFaxNumber = Null
        varAlternativePhone = Null
        varEmailAddress = Null
        varDateUpdated = Null
        varNotes = Null
        OriginalRS.MoveNext

        With OwnerRS
            .AddNew
            .Fields("OWNER_ID").Value = varOWNER_ID
            .Fields("Prefix").Value = varPrefix
            .Fields("FirstName").Value = varFirstName
            .Fields("MiddleName").Value = varMiddleName
            .Fields("LastName").Value = varLastName
            .Fields("Suffix").Value = varSuffix
            .Fields("Title").Value = varTitle
            .Fields("OrganizationName").Value = varOrg
            .Fields("Address1").Value = varAddress1
            .Fields("Address2").Value = varAddress2
            .Fields("Address3").Value = varAddress3
            .Fields("City").Value = varCity
            .Fields("State").Value = varState
            .Fields("PostalCode").Value = varPostalCode
            .Fields("Country").Value = varCountry
            .Fields("Telephone").Value = varTelephone
            .Fields("FaxNumber").Value = varFaxNumber
            .Fields("AlternativePhone").Value =
varAlternativePhone
            .Fields("EmailAddress").Value = varEmailAddress
            .Fields("DateUpdated").Value = varDateUpdated
            .Fields("Notes").Value = varNotes
            .Update
        End With
    Loop
    ' Close the recordset
    OriginalRS.Close
    OwnerRS.Close

End Sub
```

PARSEPARCEL_LANDOWNER.BAS

```
Attribute VB_Name = “ParseParcel_LandOwner”
Public Sub Parcel_LandOwner()
    ' Local variables
    Dim SourceRS As ADODB.Recordset
    Dim OutputRS As ADODB.Recordset
    Dim SourceSQL As String
    Dim OutputSQL As String
    Dim varParcelField As String

    ' set the parcel field name
    varParcelField = “Parcel”

    ' Construct the SQL strings
    SourceSQL = “SELECT * FROM County_Parcel_LandOwner ORDER BY “
& varParcelField & “, OWNER_ID;”
    OutputSQL = “SELECT * FROM Parcel_LandOwner;”

    ' Make the recordsets
    Set SourceRS = New ADODB.Recordset
    Set OutputRS = New ADODB.Recordset

    ' Open the recordsets
    SourceRS.Open SourceSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic
    OutputRS.Open OutputSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic

    ' Parse the data into the appropriate columns
    Do While Not SourceRS.EOF
        varOWNER_ID = SourceRS.Fields(“OWNER_ID”)
        If SourceRS.Fields(varParcelField) <> varPARCEL_ID Then
'New Parcel, grab first owner
            varPrincipalOwnerId = SourceRS.Fields(“OWNER_ID”)
            varFirstOwner = varPrincipalOwnerId
        Else 'Same Parcel, use first owner
            varPrincipalOwnerId = varFirstOwner
        End If
        varPARCEL_ID = SourceRS.Fields(varParcelField)

        ' Insert records into the Parcel Table
        With OutputRS
            .AddNew
            .Fields(“OWNER_ID”).Value = varOWNER_ID
            .Fields(“PARCEL_ID”).Value = varPARCEL_ID
            .Fields(“PrincipalOwnerId”).Value =
varPrincipalOwnerId
            .Update
        End With

        ' Move to the next record
```

```
        SourceRS.MoveNext

    Loop

    ' Close the recordsets
    SourceRS.Close
    OutputRS.Close

End Sub
```

PARSELEGALDESCRIPTION.BAS

```
Attribute VB_Name = “ParseLegalDescription”
Public Sub LegalDescription()
    ' Local variables
    Dim SourceRS As ADODB.Recordset
    Dim OutputRS As ADODB.Recordset
    Dim SourceSQL As String
    Dim OutputSQL As String
    Dim varParcelField As String
    Dim varLegalField As String

    ' Set the field names
    varParcelField = “ParcelID“
    varLegalField = “LegalText“

    ' Construct the SQL strings
    SourceSQL = “SELECT * FROM County_LegalDescription ORDER BY “
& varParcelField & “;”
    OutputSQL = “SELECT * FROM LegalDescription;”

    ' Make the recordsets
    Set SourceRS = New ADODB.Recordset
    Set OutputRS = New ADODB.Recordset

    ' Open the recordsets
    SourceRS.Open SourceSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic
    OutputRS.Open OutputSQL, CurrentProject.Connection,
adOpenDynamic, adLockOptimistic

    ' Parse the data into the appropriate columns
    Do While Not SourceRS.EOF
        varID = SourceRS.Fields(“ID”)
        varPARCEL_ID = SourceRS.Fields(varParcelField)
        varLegalDescriptionTxt = SourceRS.Fields(varLegalField)
        varLD_Shape_Id = Null
        ' Parse out the Parcel field
        varTownship = SourceRS.Fields(“TownshipNo”)
```

```
        If InStr(1, varTownship, “N”) > 0 Then
            varTownshipNo = Left(varTownship, Len(varTownship) -
1)
            varTownshipFractionCd = Null
            varTownshipDirCd = “N”
        ElseIf InStr(1, varTownship, “S”) > 0 Then
            varTownshipNo = Left(varTownship, Len(varTownship) -
1)
            varTownshipFractionCd = Null
            varTownshipDirCd = “S”
        Else
            varTownshipNo = Null
            varTownshipFractionCd = Null
            varTownshipDirCd = Null
        End If
        ' Parse out the Range field
        varRange = SourceRS.Fields(“RangeNo”)
        If InStr(1, varRange, “E”) > 0 Then
            varRangeNo = Left(varRange, Len(varRange) - 1)
            varRangeFractionCd = Null
            varRangeDirCd = “E”
        ElseIf InStr(1, varRange, “W”) > 0 Then
            varRangeNo = Left(varRange, Len(varRange) - 1)
            varRangeFractionCd = Null
            varRangeDirCd = “W”
        Else
            varRangeNo = Null
            varRangeFractionCd = Null
            varRangeDirCd = Null
        End If
        varSectionNo = Left(SourceRS.Fields(“SectionNo”), 2)
        varQuarterSectionCd = Null
        varSixteenthSectionCd = Null
        varTractName = Null
        varBlockNo = Null
        varLotNo = Null
        varAddress = Null
        varCity = Null
        varZip = Null

        With OutputRS
            .AddNew
            .Fields(“PARCEL_ID”).Value = varPARCEL_ID
            .Fields(“LegalDescriptionTxt”).Value =
varLegalDescriptionTxt
            .Fields(“LD_Shape_Id”).Value = varLD_Shape_Id
            .Fields(“TownshipNo”).Value = varTownshipNo
            .Fields(“TownshipFractionCd”).Value =
varTownshipFractionCd
            .Fields(“TownshipDirCd”).Value = varTownshipDirCd
            .Fields(“RangeNo”).Value = varRangeNo
            .Fields(“RangeFractionCd”).Value = varRangeFractionCd
```

```
            .Fields("RangeDirCd").Value = varRangeDirCd
            .Fields("SectionNo").Value = varSectionNo
            .Fields("QuarterSectionCd").Value =
varQuarterSectionCd
            .Fields("SixteenthSectionCd").Value =
varSixteenthSectionCd
            .Fields("TractName").Value = varTractName
            .Fields("BlockNo").Value = varBlockNo
            .Fields("LotNo").Value = varLotNo
            .Fields("Address").Value = varAddress
            .Fields("City").Value = varCity
            .Fields("Zip").Value = varZip
            .Update
        End With

        ' Move to the next record
        SourceRS.MoveNext

    Loop

    ' Close the recordsets
    SourceRS.Close
    OutputRS.Close

End Sub
```

APPENDIX D - REGION AND WATERSHED DETERMINATION

The first step in determining watershed and region membership was to create a table using GIS that had for each quarter section which DNR region and which watersheds a parcel in that section might belong to. To do this the statewide PLS coverage had to be divided into quarter sections to take advantage of quarter section data where we had it. The quarter sections were built using the following two ArcView Avenue scripts (View.QuarterSectionInit.ave and View.QuarterSectionRun.ave).

Once the quarter sections had been generated they were unioned with the regions and then with the watersheds. The resulting table (Table 1) showed for every ¼ section and section in the state what DNR region the section was in and if there were multiple watersheds then which watersheds the section was in.

Table 1 - Watershed and DNR Regions by section and 1/4 section

TRS	REGION_ID	WAU_ID
T01.0N R02.0E S01 Q4	11	280301
T01.0N R03.0E S01	11	280202
T01.0N R03.0E S01	11	280203
T01.0N R03.0E S01 Q1	11	280202

The Region / Watershed table was then joined to the LegalDescription table using a field that was generated with the VBA code MakeTRSQ.bas and then queried with SQL (Parcel_Wau_Query and Parcel_Region_Query) to generate the Parcel_Watershed table and fill in the Region Code in the Parcel Table.

VIEW.QUARTERSECTIONINIT.AVE

```
' View.QuarterSectionInit
'HardCore_qtr_1st.ave (old name)

'By: Tony Thatcher; tony@dtmgis.com, and a lot of help from
others

'This script and it's associated partner "HardCore_qtr_2nd.ave"
can be used to quarter
'any polygon theme.  It's intended use is for quartering PLSS
themes.

'Credit where credit is due:
'These two scripts were pieced together and modified from scripts
originally developed by
'Randy How, David Dow, and William Huber.  I've lost track of
what piece belongs to who, though
'the quartering algorhythm is entirely Randy How's with little
modification.  I upgraded
'pieces to meet my needs.
'Modifications by Phil Hurvitz <phurvitz@u.washington.edu>:
'  remove the shape field from the list of fields to select

'****Compile this script and call it HardCore_qtr_1st.
'****Compile the second script and call it HardCore_qtr_2nd.

'****See the discussion by Randy How in HardCore_Qtr_2nd for a
discussion of the
'"SplitAngleLimit" parameter that is passed from this script.

'What it does:
'This script should be attached to a button in the View document.
It grabs the selected
'shapes from the first active theme and processes them one at a
time.  It passes the shape
'to the second script which does all the hard work.  The second
script then returns a list
'of four shapes resulting from the quartering process.  The new
shapes are added to a new
'shape file and attributed with the user selected attributes from
the input theme as well as
```

```
'adding a quadrant field which is attributed with "NW", "SW",
"NE", or "SE" to identify the
'quarter.


theView = av.GetActiveDoc

' Make sure only a single theme is active  'PMH
'---------------------------------------
theThemes = theView.GetThemes
theActiveThemes = theView.GetActiveThemes
if (theActiveThemes.Count <> 1) then
   MsgBox.Error("Make a single theme active and try again.",
"Error")
   Return Nil
end
' /PMH

theTheme = theActiveThemes.Get(0)
theFTab = theTheme.GetFTab
theShapeField = theFTab.FindField("Shape")
theBitMap = theFTab.GetSelection  'PMH
theNumSelected = theBitMap.Count  'PMH

' If there is a selection, only quarter those; otherwise quarter
everything 'PMH
'---------------------------------------
if (theNumSelected = 0) then
   theSelection = theFTab
   theNumSelected = theFTab.GetNumRecords
else
   theSelection = theBitMap
   theNumSelected = theBitMap.Count
end
'/PMH

'if (theFTab.GetSelection.Count = 0) then
'  msgbox.info("Select Polygons to quarter before runing
script","No Shapes Selected!")
'  exit
'end

' The file source  'PMH
'---------------------------------------
theSN = theTheme.GetSrcName
theFN = theSN.GetFileName
theBN = theFN.GetBaseName
theDir = theFN.ReturnDir
theEntryName = theBN.Substitute(".shp", "") 'PMH
' /PMH
```

```
'Determine fields to add from base table
'---------------------------------------
'exstFields = theFTab.GetFields
exstFields = theFTab.GetFields.Clone 'PMH

' Remove the shape field (We don't want to duplicate)  'PMH
'---------------------------------------
exstFields.RemoveObj(theShapeField)
' /PMH


transferFields = MsgBox.MultiList
      (exstFields,
      "(Single click to toggle selection,"+nl+
      "click and drag for multiple selection,"+nl+
      "Press SHIFT + OK to select all fields)",
      "Fields to Add")


' Take all fields if Shift Key is down  'PMH
'---------------------------------------
if (System.IsShiftKeyDown) then
   transferFields = exstFields
end
' /PMH

'if (transferFields = nil) then
' MsgBox.info("No additional fields will be added","")
'end

' Print a Msg  'PMH
'---------------------------------------
if (transferFields = Nil) then
   av.ShowMsg("No additional fields will be added")
elseif (transferFields.Count = 0) then
   av.ShowMsg("No additional fields will be added")
end
' /PMH


'Identify Theme Name
'-------------------
'newshpname =
filedialog.Put("plss_qqtr.shp".AsFilename,"*.shp","New shape file
name")
newshapetmp = theDir.MakeTmp(theEntryName, "shp")  'PMH (auto-new
name)
newshpname = filedialog.Put(newshapetmp, "*.shp", "New shape file
name")  'PMH
```

```
' Did a new theme get selected?  ' PMH
'-------------------
if (newshpname = Nil) then
   Return Nil
end

newshpBN = newshpname.GetBaseName.Substitute(".shp", "")

' The errors go here filename  ' PMH
'--------------------------------------
errshpname = FileName.Merge(theDir.AsString, "e"+newshpBN+".shp")
' /PMH

theNewFTab = FTab.MakeNew(newshpname,POLYGON)
theNewFTab.SetEditable(true)

for each fld in transferFields
 theField = Field.Make(fld.AsString, #FIELD_CHAR,0,0)
 theField.Copy(fld)
 theNewFTab.AddFields({theField})  'PMH
end

' PMH errors FTab
'---------------------------------------
theErrFTab = FTab.MakeNew(errshpname,POLYGON)
theErrFTab.SetEditable(true)
'/ PMH

for each fld in transferFields
 theField = Field.Make(fld.AsString, #FIELD_CHAR,0,0)
 theField.Copy(fld)
 theErrFTab.AddFields({theField})  'PMH
end


' Check for existing fields  'PMH
'  Only handles up to quarter-quarter-quarter sections
'-------------------
theQField = theNewFTab.FindField("Qtr")
theQQField = theNewFTab.FindField("Qqtr")
theQQQField = theNewFTab.FindField("Qqqtr")

if ((theQField = Nil) and (theQQField = Nil) and (theQQQField =
Nil)) then
   theQFieldName = "Qtr"
elseif ((theQField <> Nil) and (theQQField = Nil) and
(theQQQField = Nil)) then
   theQFieldName = "Qqtr"
```

```
elseif ((theQField <> Nil) and (theQQField <> Nil) and
(theQQQField = Nil)) then
   theQFieldName = "Qqqtr"
end
'    /PMH

'****Note change the new field name "Qtr" in the next line to
"QtrQtr" if quartering
'****a PLSS grid that is already quartered to get attributed
quarter quarters.

theqtrField = Field.Make(theQFieldName,#FIELD_CHAR,6,0)
theNewFTab.AddFields({theqtrField})

theNewShapeField = theNewFTab.FindField("Shape")
theErrShapeField = theErrFTab.FindField("Shape")  'PMH

'''theFldsList = theErrFTab.GetFields
'''MsgBox.ListAsString(theFldsList, "", "")
'''return nil


'  Time tracking  'PMH
'--------------------
theStart = Date.Now.AsSeconds
' /PMH

' Diagnostics  'PMH
'--------------------
' PMH diagnostic
DiagBN = newshpname.GetBaseName.Substitute(".shp", "")
theDiagFN = FileName.Merge(theDir.AsString, DiagBN+".txt")
if (file.Exists(theDiagFN)) then
   File.Delete(theDiagFN)
end
theDiagLF = LineFile.Make(theDiagFN, #FILE_PERM_WRITE)
theDiagLF.WriteElt(Date.Now.SetFormat("yyyy.MM.dd
hh:m").AsString++theSN.AsString)
' /PMH

av.ShowStopButton  'PMH
aBool = false
for each rec in theSelection  'PMH (to handle selection or all
records)
'for each rec in theFTab.GetSelection
  polyShape = theFTab.ReturnValue(theShapeField,rec)
  theCurrentTime = Date.Now.AsSeconds
  theDuration = (theCurrentTime - theStart) / 60

av.ShowMsg("Processing"++rec.AsString++"of"++theNumSelected.AsStr
ing++
     theDuration.AsString++"minutes have passed.")
```

```
  ' If this is a multipart shape, skip attempting a split 'PMH
  '-------------------
  if (polyShape.AsList.Count = 1) then
     RunPoly = True
  end
  ' /PMH
  theDiagLF.WriteElt
(rec.AsString+":"+polyShape.AsList.Count.AsString)  'PMH

  ' ' Rotation angle limits when splitting the section polygons
  westAngleLimitRadians = 0.02      ' Approx 1.1 degrees
  eastAngleLimitRadians = 6.28      ' Approx 360 degrees
  '                                  '  (effectively no limit,
since there is
  '                                  '   a natural curvature in
the data which
  '                                  '   must be preserved
  '
  '
  ' ' Use some criteria to determine the rotation angle limit
(see prologue).
  ' ' The rotation angle limit is influenced by the nature of the
specific
  ' ' polygon data being processed.
  ' ' (this sample script fragment is simply using the value
"westAngleLimitRadians"
  splitAngleLimit = eastAngleLimitRadians

  'rtList = List.Make
  'Pass the polygon shape to be split and the split angle limit
value
  parmList = {polyShape, splitAngleLimit}

  'rtList = av.Run("HardCore_qtr_2nd",parmList)

  ' If this is a multipart shape, skip attempting a split 'PMH
  if (polyShape.AsList.Count = 1) then 'PMH

     rtList = av.Run("View.QuarterSectionRun",parmList)

     ' If the split resulted in 4 pieces 'PMH
     if (rtList.Count = 4) then  'PMH

        'Add the polygons to the NewFTab
        newrec = theNewFTab.AddRecord
        for each fld in transferFields
         theBaseField = theFTab.FindField(fld.AsString)
         theValue = theFTab.ReturnValue(theBasefield,rec)
         theField = theNewFTab.FindField(fld.AsString)
```

```
          theNewFTab.SetValue(theField,newrec,theValue)
         end
         theNewFTab.SetValue(theNewShapeField, newrec,
rtlist.Get(2))
         theNewFTab.SetValue(theQtrField, newrec, "SW")
         newrec = theNewFTab.AddRecord
         for each fld in transferFields
          theBaseField = theFTab.FindField(fld.AsString)
          theValue = theFTab.ReturnValue(theBasefield,rec)
          theField = theNewFTab.FindField(fld.AsString)
          theNewFTab.SetValue(theField,newrec,theValue)
         end
         theNewFTab.SetValue(theNewShapeField, newrec,
rtlist.Get(0))
         theNewFTab.SetValue(theQtrField, newrec, "NW")
         newrec = theNewFTab.AddRecord
         for each fld in transferFields
          theBaseField = theFTab.FindField(fld.AsString)
          theValue = theFTab.ReturnValue(theBasefield,rec)
          theField = theNewFTab.FindField(fld.AsString)
          theNewFTab.SetValue(theField,newrec,theValue)
         end
         theNewFTab.SetValue(theNewShapeField, newrec,
rtlist.Get(1))
         theNewFTab.SetValue(theQtrField, newrec, "NE")
         newrec = theNewFTab.AddRecord
         for each fld in transferFields
          theBaseField = theFTab.FindField(fld.AsString)
          theValue = theFTab.ReturnValue(theBasefield,rec)
          theField = theNewFTab.FindField(fld.AsString)
          theNewFTab.SetValue(theField,newrec,theValue)
         end
         theNewFTab.SetValue(theNewShapeField, newrec,
rtlist.Get(3))
         theNewFTab.SetValue(theQtrField, newrec, "SE")
      else
         'MsgBox.Error ("less than 4 shapes", "")
         continue
         'newrec = theNewFTab.AddRecord
         'theNewFTab.SetValue(theNewShapeField, newrec, polyShape)
         'theNewFTab.SetValue(theQtrField, newrec, "X")
      end

   else ' if multipart
      'continue
      theDiagLF.WriteElt("Record"++rec.AsString++"is multipart.")
      ErrRec = theErrFTab.AddRecord
      theErrFTab.SetValue(theErrShapeField, ErrRec, polyShape)
      for each fld in transferFields
         theBaseField = theFTab.FindField(fld.AsString)
         theValue = theFTab.ReturnValue(theBasefield,rec)
         theErrField = theErrFTab.FindField(fld.AsString)
```

```
          theErrFTab.SetValue(theErrField,ErrRec,theValue)
       end
       newrec = theNewFTab.AddRecord
       theNewFTab.SetValue(theNewShapeField, newrec, polyShape)
       theNewFTab.SetValue(theQtrField, newrec, "X")
    end

   '  Show progress  'PMH
   '-------------------
   progress = (rec/theNumSelected) * 100
   doMore = av.SetStatus( progress )
   if (not doMore) then
      break
   end
   ' /PMH


end
theNewFTab.SetEditable(false)
theErrFTab.SetEditable(false)

'Show the new Theme
'------------------
myTheme = ftheme.Make(theNewFTab)
myTheme.SetActive(true)
myTheme.SetVisible(true)
theView.AddTheme(mytheme)
myTheme.Invalidate(true)

' Show the error theme  'PMH
'------------------
errTheme = ftheme.Make(theErrFTab)
errTheme.SetActive(true)
errTheme.SetVisible(true)
theView.AddTheme(errTheme)
errTheme.Invalidate(true)
' /PMH

theView.GetDisplay.Flush


'  Time tracking  'PMH
'-------------------
theEnd = Date.Now.AsSeconds
theDuration = theEnd - theStart
theMinutes = (theDuration / 60).SetFormat("d.dd")
theMean = (theDuration / theNumSelected).SetFormat("d.dd")
av.ShowMsg("Process took"++theMinutes.AsString++"minutes
for"++theNumSelected.AsString++"polygons"++
   "("+theMean.AsString++"seconds per polygon).")
```

```
' /PMH
```

VIEW.QUARTERSECTIONRUN.AVE

```
' View.QuarterSectionRun
'HardCore_qtr_2nd.ave (old name)

'This script is called from HardCore_qtr_1st and returns
quartered PLSS sections.
'Original code is from Randy How with slight modifications from
Tony Thatcher
'tony@dtmgis.com.

'================================================================
======
' View.SplitPolyIntoQuarters -> SplitPolyQuarters.ave
'
' Description:  Accept an incoming polygon shape and split it
into four
'               new polygons, each "one quarter" of the original
polygon
'
'               Since the nature of the PLSS section polygons in
the
'               UTM projection is that most polygons are oriented
at a non-0
'               angle, this script attempts to estimate the angle
both in
'               the East-West direction ("horizontal") and the
North-South
'               direction ("vertical"). To avoid undue distortion
(over rotation)
'               which may be caused by some irregularly-shape
PLSS section
'               polygons, the script accepts an input parameter
designating the
'               maximum rotation angle (positive or negative)
permitted in
'               either direction.
'
' Sample script fragment for calling this script:
'
'    See sample script fragment (commented out) included at the
end of
'    this script source file.
'
'
' Globals Used:
' Parameters read:
```

```
'   1.  Self.Get(0)  polyToSplit   = A polygon shape object
representing the polygon
'                                    to be split.
'
'   2.  Self.Get(1)  angleLimit    = The maximum angle (positive
or negative), in radians,
'                                    allowed when rotating the
split lines for this
'                                    specific polygon.
'                                    The calling script varies
this maximum angle based
'                                    on the "east-ness" of the
PLSS section
'                                    polygons (the first
character in the major
'                                    map code). In general,
sections in major map codes
'                                    in the western part of the
service area (major map
'                                    code first character "N" or
lower) are held to a
'                                    fairly low maximum rotation
angle (approx 1.1 degrees,
'                                    or 0.02 radians). This limit
is to prevent distortion
'                                    with some of the ill-formed
PLSS sections resulting
'                                    from subdividing the non-
sectional areas.
'                                    In contrast, the natural
shape of the PLSS sections
'                                    in the eastern part of the
service area (major map
'                                    code first charcter "O" or
greater) in the UTM
'                                    projection is such that a
"normal" orientation angle
'                                    for these polygons is
approximately 0.06 radians
'                                    (around 3 degrees). Since
the PLSS polygons in this
'                                    part of the service area
tend to be reasonably
'                                    well-formed, this parameter
places virtually
'                                    no restrictions, being
passed in with a value of
'                                    6.28 (approximately 360
degrees). This allows the
```

```
'                                       split process to use
whatever angle is necessary
'                                       for these polygons in the
eastern half of the
'                                       utility service area.
'                                       Inspection of the initial
results of polygon splitting
'                                       identified the need for
special case handling for two
'                                       of the eastern PLSS
sections, as discussed below:
'
'                                         Special cases are needed
to limit rotation is required
'                                         for sections UJ-05 and QL-
29 due to distortions in the section
'                                         shape. These distortions
lead to exaggerated rotation angles
'                                         so this special case check
limits the rotation for these two
'                                         specific sections in the
eastern portion of the service area.
'
'
' Parameters returned:
'
'   1.  rtList(0)    NW Quad Poly    = Polygon object
representing the polygon "quarter"
'                                         for the NorthWest quadrant
of the original polygon
'
'   2.  rtList(1)    NE Quad Poly    = Polygon object
representing the polygon "quarter"
'                                         for the NorthEast quadrant
of the original polygon
'
'   3.  rtList(2)    SW Quad Poly    = Polygon object
representing the polygon "quarter"
'                                         for the SouthWest quadrant
of the original polygon
'
'   4.  rtList(3)    SE Quad Poly    = Polygon object
representing the polygon "quarter"
'                                         for the SouthEast quadrant
of the original polygon
'
'   Note:  An "empty list" is returned if an error is returned
(rtList.Count = 0)
'
' Called from:         View.QuarterSectionCompute
'
' Calls to:            None.
```

```
'================================================================
======
' Project:  Split Sections to Quarter Sections
'
'  ------------------- Change Log ------------------------------
-------
'
'  10/19/97  sak  Script originally written.
'  04/02/98  sak  Updated logic to account for the fact that the
'                 Section polygons may have a general orientation
at
'                 a non-0 degreee angle
'  04/03/98  sak  Further updated the logic to compute two
different
'                 rotation angles, one for the East-West
orientation
'                 ("horizontal") and a separate angle for the
'                 North-South orientation ("vertical"). The need
for
'                 this modification became apparent after
visually
'                 inspecting the output produced when only a
single
'                 rotation angle value was applied to both the
'                 horizontal and vertical polygon split process.
'
'================================================================
======
'
'
' Constants for use in this script

' PMH increased from 5000 to 6500 to handle tall narrow sections
lineEndOffset = 6500    ' For the UTM NAD 27 projection:
'lineEndOffset = 6500   ' Use 5000 meters for defining
sufficiently long
                        ' polylines to intersect the polygon
being split.
                        ' This is an abritraty value which must
simply be
                        ' "long enough" to make sure it
intersects the
                        ' PLSS section polygon boundaries (which
are
                        ' approximately 1609 meters (1 mile) on a
side.

' themeReport = "In SplitPolygonQuarters"+NL

' Initialize the return list
```

```
rtList = List.Make
rtList = { {}, {}, {}, {} }    ' Set up with four empty lists for
return

' Get parameters

'  Polygon object to be split
polyToSplit   = self.get(0)
angleLimit    = self.get(1)

' PMH
' lineEndOffset should equal the largest dimension of the poly
'-------------------------------
thePolyRect = polytoSplit.ReturnExtent
lineEndOffset = thePolyRect.GetHeight.max(thePolyRect.GetWidth)
'theView = av.GetActiveDoc
'theView.GetDisplay.SetExtent(thePolyRect.Scale(1.1))
'theView.Invalidate
' /PMH

angleLimitNeg = -1.0 * angleLimit

' Determine if a polygon has actually been sent
' If not, set return code and return to caller
if (polyToSplit = nil) then
  rtList.Empty           ' Empty the list to signal an error
  return rtList          ' return to the calling script
end

' ---- Calculate the rotation angle relative to "Horizontal"

   anglePoly    = polyToSplit.Clone    ' Create a working polygon

         polyCenter  = anglePoly.ReturnCenter
         polyCenterX = polyCenter.GetX
         polyCenterY = polyCenter.GetY
         pointCenter = polyCenter

         vertLineXN  = polyCenterX
         vertLineYN  = polyCenterY + lineEndOffset
         pointVertN  = Point.Make(vertLineXN,vertLineYN)

         vertLineXS  = polyCenterX
         vertLineYS  = polyCenterY - lineEndOffset
         pointVertS  = Point.Make(vertLineXS,vertLineYS)

'      "Draw" the vertical line from North-to-South so that the
'      polygons resulting from the split will be:
'       1st list element: Western-half of the angle-determination
polygon
'       2nd list element: Eastern-half of the angle-determination
polygon
```

```
        vertLine   = PolyLine.Make( { {pointVertN, pointVertS} }
)

'      The polygons resulting from this split will be:
'         1st list element: Western-half of the cloned section
polygon
'         2nd list element: Eastern-half of the cloned section
polygon
'         3rd list element: Null (not used)
'         4th list element: Null (not used)

        anglePolygonList = anglePoly.Split(vertLine)

        anglePolygonWest =  anglePolygonList.Get(0)
        anglePolygonEast =  anglePolygonList.Get(1)

        angleCenterWest  = anglePolygonWest.ReturnCenter
        angleCenterWestX = angleCenterWest.GetX
        angleCenterWestY = angleCenterWest.GetY

        angleCenterEast  = anglePolygonEast.ReturnCenter
        angleCenterEastX = angleCenterEast.GetX
        angleCenterEastY = angleCenterEast.GetY

        deltaX = angleCenterEastX - angleCenterWestX
        deltaY = angleCenterEastY - angleCenterWestY

        ' ATan: Returns the angle (in radians) for which aNumber
is the tangent.
        '       The returned angle is in the range -pi/2 .. pi/2
.
        '         Syntax
        '         aNumber.ATan

        ' For the PLSS section polygons, deltaX is a relatively
        ' large value and deltaY is a relatively small value.
        rotationAngleHorizRadians = (deltaY/deltaX).ATan

       ' msgBox.Info("rotationAngleHorizRadians = " +
rotationAngleHorizRadians.AsString,"")

        ' Limit the minimum and maximum rotation angles to avoid
        ' undue distortion in some ill-formed PLSS sections
        ' Note: The limit is passed as a calling argument.
        '       Generally, the need is to place small limits in
the
        '       western portion of the service area (where
allocating
        '       the non-PLSS sections caused several ill-formed
sections).
```

```
        '       However, that same limit must be relaxed in the
easterm
        '       portion of the service area because there is a
natural
        '       curvature to the data along the section/quarter
sections
        if (rotationAngleHorizRadians < angleLimitNeg) then
          rotationAngleHorizRadians = angleLimitNeg
        end

        if (rotationAngleHorizRadians > angleLimit) then
          rotationAngleHorizRadians = angleLimit
        end

        ' Destroy object variables
        anglePoly    = nil
        anglePolygonWest =  nil
        anglePolygonEast =  nil

' ---- Calculate the rotation angle relative to "Vertical"

        anglePoly    = polyToSplit.Clone     ' Create a working
polygon

        polyCenter  = anglePoly.ReturnCenter
        polyCenterX = polyCenter.GetX
        polyCenterY = polyCenter.GetY
        pointCenter = polyCenter

        horizLineXE  = polyCenterX + lineEndOffset
        horizLineYE  = polyCenterY
        pointHorizE  = Point.Make(horizLineXE,horizLineYE)

        horizLineXW  = polyCenterX - lineEndOffset
        horizLineYW  = polyCenterY
        pointHorizW  = Point.Make(horizLineXW,horizLineYW)

'       "Draw" the pseudo horizontal line from East-to-West so
that the
'       polygons resulting from the split will be:
'        1st list element: Northern-half of the original polygon
'        2nd list element: Southern-half of the original polygon
         horizLine   = PolyLine.Make( { {pointHorizE,
pointHorizW} } )

'       The polygons resulting from this original split will be:
'        1st list element: Northern-half of the original polygon
'        2nd list element: Southern-half of the original polygon
'        3rd list element: Null (not used)
'        4th list element: Null (not used)
        anglePolygonList = anglePoly.Split(horizLine)
```

```
        anglePolygonNorth =  anglePolygonList.Get(0)
        anglePolygonSouth =  anglePolygonList.Get(1)

        angleCenterNorth  = anglePolygonNorth.ReturnCenter
        angleCenterNorthX = angleCenterNorth.GetX
        angleCenterNorthY = angleCenterNorth.GetY

        angleCenterSouth  = anglePolygonSouth.ReturnCenter
        angleCenterSouthX = angleCenterSouth.GetX
        angleCenterSouthY = angleCenterSouth.GetY

        deltaX = angleCenterSouthX - angleCenterNorthX
        deltaY = angleCenterSouthY - angleCenterNorthY

        ' ATan: Returns the angle (in radians) for which aNumber
is the tangent.
        '       The returned angle is in the range -pi/2 .. pi/2
.
        '         Syntax
        '         aNumber.ATan

        ' For this intended adjustment, the "tangent" of the
angle in question
        ' is actually computed as deltaX/deltaY
        ' In this case, for the PLSS section polygons, deltaX is
a relatively
        ' small value and deltaY is a relatively large value.
        ' rotationAngleVertRadians = (deltaY/deltaX).ATan
        rotationAngleVertRadians = (deltaX/deltaY).ATan

       ' msgBox.Info("rotationAngleVertRadians = " +
rotationAngleVertRadians.AsString,"")

        ' Limit the minimum and maximum rotation angles to avoid
        ' undue distortion in some ill-formed PLSS sections
        ' Note: The limit is passed as a calling argument.
        '       Generally, the need is to place small limits in
the
        '       western portion of the service area (where
allocating
        '       the non-PLSS sections caused several ill-formed
sections).
        '       However, that same limit must be relaxed in the
easterm
        '       portion of the service area because there is a
natural
        '       curvature to the data along the section/quarter
sections
        if (rotationAngleVertRadians < angleLimitNeg) then
          rotationAngleVertRadians = angleLimitNeg
```

```
        end

        if (rotationAngleVertRadians > angleLimit) then
          rotationAngleVertRadians = angleLimit
        end

        ' Destroy object variables
        anglePoly    = nil
        anglePolygonNorth =  nil
        anglePolygonSouth =  nil



' --- Calculate the split lines taking into account the rotation
angles
' --- computed above
' --- In all cases, the line splitting the polygons passes
through the
' --- ArcView-derived center of the ORIGINAL polygon passed in as
the
' --- first calling parameter to this script.

   tempPoly    = polyToSplit.Clone
   polyCenter  = tempPoly.ReturnCenter
   polyCenterX = polyCenter.GetX
   polyCenterY = polyCenter.GetY
   pointCenter = polyCenter

   ' Compute rotation offset for the "Horizontal" split
   rotationOffset = (lineEndOffset) *
(rotationAngleHorizRadians.Sin)

   horizLineXE  = polyCenterX + lineEndOffset
   horizLineYE  = polyCenterY + rotationOffset  ' Add the
rotation offset
   pointHorizE  = Point.Make(horizLineXE,horizLineYE)

   horizLineXW  = polyCenterX - lineEndOffset
   horizLineYW  = polyCenterY - rotationOffset  ' Subtract the
rotation offset
   pointHorizW  = Point.Make(horizLineXW,horizLineYW)

' "Draw" the pseudo horizontal line from East-to-West so that the
' polygons resulting from the split will be:
'  1st list element: Northern-half of the original polygon
'  2nd list element: Southern-half of the original polygon
   horizLine   = PolyLine.Make( { {pointHorizE, pointHorizW} } )

' The polygons resulting from this original split will be:
'  1st list element: Northern-half of the original polygon
'  2nd list element: Southern-half of the original polygon
'  3rd list element: Null (not used)
```

```
'  4th list element: Null (not used)
   firstPolygonList = tempPoly.Split(horizLine)


'         themeReport = themeReport +
'                       " NorthHalf = " ++ NL ++
'                         firstPolygonList.Get(0).AsString ++ NL
++
'                       " SouthHalf = " ++ NL ++
'                         firstPolygonList.Get(1).AsString ++ NL


'       MsgBox.Report( themeReport, "Split Report -7- " )


' - - - - Now split these two "half polygons" to derive the
' - - - - "quarter section" polygons

   northPolygonHalf = firstPolygonList.Get(0)
   southPolygonHalf = firstPolygonList.Get(1)

' polyctr = 0  Split the northPolygonHalf polygon
' polyctr = 1  Split the southPolygonHalf polygon

   for each polyctr in 0 .. 1

     if (polyctr = 0) then
        halfPolyToSplit = northPolygonHalf
     else
        halfPolyToSplit = southPolygonHalf
     end  ' end of "if (polyctr = 0) then"


      polyList = halfPolyToSplit.AsList

      snbrParts  = halfPolyToSplit.CountParts

      spart_counter = 1

      for each sshapePart in polyList

        stempPoly      = Polygon.Make( { sshapePart } )
        tempPoly    = stempPoly

'     In all cases, the line splitting the polygons passes
through the
'     ArcView-derived center of the ORIGINAL polygon passed in as
the
'     first calling parameter to this script --> polyCenterX and
polyCenterY
```

```
      ' Compute rotation offset for the "Vertical" split
      ' Must use the "negative" value to account for the trig
function
      ' algebraic sign differences since the N-S split line is
rotated
      ' approximately 90 degrees from the E-W split line used
earlier
        rotationOffset = -1.0 * (lineEndOffset) *
(rotationAngleVertRadians.Sin)

        vertLineXN  = polyCenterX - rotationOffset  ' Subtract
the rotation offset
        vertLineYN  = polyCenterY + lineEndOffset
        pointVertN  = Point.Make(vertLineXN,vertLineYN)

        vertLineXS  = polyCenterX + rotationOffset  ' Add the
rotation offset
        vertLineYS  = polyCenterY - lineEndOffset
        pointVertS  = Point.Make(vertLineXS,vertLineYS)

'      "Draw" the vertical line from North-to-South so that the
'      polygons resulting from the split will be:
'       1st list element: Western-half of the Northern-half
polygon
'       2nd list element: Eastern-half of the Northern-half
polygon
        vertLine   = PolyLine.Make( { {pointVertN, pointVertS} }
)

'      The polygons resulting from this split will be:
'       For polyctr = 0 (Northern half of the original polygon
being split)
'         1st list element: Western-half of the Northern-half
polygon
'         2nd list element: Eastern-half of the Northern-half
polygon
'         3rd list element: Null (not used)
'         4th list element: Null (not used)
'       For polyctr = 1 (Southern half of the original polygon
being split)
'         1st list element: Western-half of the Southern-half
polygon
'         2nd list element: Eastern-half of the Southern-half
polygon
'         3rd list element: Null (not used)
'         4th list element: Null (not used)
        secondPolygonList = tempPoly.Split(vertLine)

          rtList.Set((polyctr*2    ), secondPolygonList.Get(0))
          rtList.Set(((polyctr*2)+1), secondPolygonList.Get(1))
```

```
 '        themeReport = themeReport +
 '                      (polyctr*2).AsString + " Poly1 = " ++ NL
++
 '                         secondPolygonList.Get(0).AsString ++ NL
++
 '                      ((polyctr*2)+1).AsString + " Poly2 = " ++
NL ++
 '                         secondPolygonList.Get(1).AsString ++ NL



 '       MsgBox.Report( themeReport, "Split Report -7- " )


      end  ' end of "for each sshapePart in polyList"


   end  ' end of "for each polyctr in 0 .. 1  "

      ' Destroy object variables
        tempPoly    = nil
        stempPoly   = nil

' Run garbage collection to eliminate objects marked for deletion
av.PurgeObjects

' Normal termination
  return rtList          ' return to the calling script

'  ----- end of script -----

' ===== start of sample script fragment for calling this script
=====
'
' ' Rotation angle limits when splitting the section polygons
' westAngleLimitRadians = 0.02      ' Approx 1.1 degrees
' eastAngleLimitRadians = 6.28      ' Approx 360 degrees
'                                  '  (effectively no limit,
since there is
'                                  '   a natural curvature in the
data which
'                                  '   must be preserved
'
'
' ' Use some criteria to determine the rotation angle limit (see
prologue).
' ' The rotation angle limit is influenced by the nature of the
specific
' ' polygon data being processed.
```

```
' ' (this sample script fragment is simply using the value
"westAngleLimitRadians"
' splitAngleLimit = westAngleLimitRadians
'
'    rtList = List.Make
'   ' Pass the polygon shape to be split and the split angle
limit value
'    parmList = {polyShape, splitAngleLimit}
'
'    rtList = av.Run("View.SplitPolyIntoQuarters",parmList)
'    if (rtList.Count <= 0) then
'      nbrPolySplitError = nbrPolySplitError + 1
'      continue                    ' Do not continue processing
this PLSS section;
'    end                           ' Move on to the next PLSS
section polygon
'
'    nbrNewPoly = rtList.Count
'
' '  Ensure that the split process always returned four polygons
'    if (nbrNewPoly <> 4) then
'      nbrPolySplitProb  = nbrPolySplitProb  + 1
'      continue                    ' Do not continue processing
this PLSS section;
'    end                           ' Move on to the next PLSS
section polygon
'
' '  Loop to process each of the newly-created PLSS Quarter
Section polygons
'  for each polyctr in 0 .. (nbrNewPoly-1)
'
'
'      ' Access the current quarter section polygon
'      pl = rtList.Get(polyctr)     ' Polygon is returned from
the split process
'
'      theArea = pl.ReturnArea
'      thePerimeter = pl.ReturnLength
'
'      ' Detect case of NULL Area (which indicates some type of
polygon split problem)
'      if (theArea.IsNull) then
'        nbrAreaNull = nbrAreaNull + 1
'        summaryFile.WriteElt( "  "+NL)
'        summaryFile.WriteElt
'             ( "  Polygon with NULL area (error). Polygon ID=
"+tpInPlNbrVal.AsString)
'      end
'
'     ' Convert area from square meters (UTM coordinates) to
square miles
```

```
'      areaSqMiles =
Units.ConvertArea(theArea,#UNITS_LINEAR_METERS,#UNITS_LINEAR_MILE
S)
'
'     ' Convert perimeter from meters (UTM coordinates) to miles
'      perimMiles =
Units.Convert(thePerimeter,#UNITS_LINEAR_METERS,#UNITS_LINEAR_MIL
ES)
'
'     ' Prepare to create the next record in the Quarter Section
shape file
'          rec = shpFTab.AddRecord  ' Add new record to the shape
file
'
'          shpFTab.SetValue( shpField,     rec, pl )
'
'      ' Assign values to the application-specific fields
'          shpFTab.SetValue( aaaField,    rec, aaaVal )
'          shpFTab.SetValue( bbbField,    rec, bbbVal )
'          ' ... set values for all needed fields
'
'  end  ' end of "for each polyctr in 0 .. (nbrNewPoly-1)"
' ===== end of sample script fragment for calling this script
=====
```

MAKETRSQ.BAS

```
Public Sub makeTRSQ()
    'Dimension the variables
    Dim rs As New ADODB.Recordset
    Dim cat As New ADOX.Catalog
    Dim varT, varTF, varTD As String
    Dim varR, varRF, varRD As String
    Dim varS, varQ As String

    'Open the Catalog
    Set cat.ActiveConnection = CurrentProject.Connection

    'Add the TRSQ item to the LegalDescription table
    cat.Tables("LegalDescription").Columns.Append "TRSQ",
adVarWChar, 50

    'Release the reference to the catalog
    Set cat = Nothing

    'Open the recordset
    rs.Open "SELECT * FROM LegalDescription;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic
```

```
    'Loop through and concatenate the T, R, S and Q fields to
make TRSQ
    Do While Not rs.EOF
        varT = rs.Fields("TownshipNo")
        If Len(varT) = 1 Then varT = "0" & varT
        varTF = rs.Fields("TownshipFractionCd")
        varTD = rs.Fields("TownshipDirCd")
        varR = rs.Fields("RangeNo")
        If Len(varR) = 1 Then varR = "0" & varR
        varRF = rs.Fields("RangeFractionCd")
        varRD = rs.Fields("RangeDirCd")
        varS = rs.Fields("SectionNo")
        If Len(varS) = 1 Then varS = "0" & varS
        varQ = rs.Fields("QuarterSectionCd")
        If varQ = 0 Then
            varQ = ""
        Else
            varQ = " Q" & varQ
        End If
        rs.Fields("TRSQ").Value = "T" & varT & "." & varTF &
varTD & " R" & varR & "." & varRF & varRD & " S" & varS & varQ
        rs.MoveNext
    Loop

    'Close the recordset
    rs.Close

End Sub
```

PARCEL_REGION_QUERY

```
INSERT INTO NewParcel ( PARCEL_ID, COUNTY_ID, REGION_ID,
TimberAcres, TotalAcres, LAND_USE_CD, RESIDENCE_CD )
SELECT DISTINCT Parcel.PARCEL_ID, Parcel.COUNTY_ID,
Legal_Region_Join_Query.REGION_ID, Parcel.TimberAcres,
Parcel.TotalAcres, Parcel.LAND_USE_CD, Parcel.RESIDENCE_CD
FROM Parcel INNER JOIN Legal_Region_Join_Query ON
Parcel.PARCEL_ID = Legal_Region_Join_Query.PARCEL_ID;
```

PARCEL_WAU_QUERY

```
INSERT INTO Parcel_Watershed ( PARCEL_ID, WAU_ID )
SELECT LegalDescription.PARCEL_ID, TRS_Wau_Region.WAU_ID
FROM LegalDescription, TRS_Wau_Region;
```

APPENDIX E - APPENDING COUNTIES TO MASTER DATABASE

A few steps must be taken before the individual normalized, parsed and formatted county databases can be integrated into a single database. The Master Database which contained all the final tables was constructed and then for each county the following tables were imported: Parcel, LandOwner, Parcel_LandOwner, LegalDescription and Parcel_Watershed. The VBA code Append_CountyID() function adds the CountyID and a "-" to the beginning of each OWNER_ID, PARCEL_ID and LegalDescription table ID. The reason for appending the county id to the front of each of the unique ID's in the table is to ensure that those ID's are unique across the state.

Once the ID's were appended the following five SQL append queries were run to integrate the data into one table: Append_Parcel_Query, Append_LandOwner_Query, Append_Parcel_LandOwner_Query, Append_LegalDescription_Query, Append_Parcel_Watershed_Query.

APPEND_COUNTYID.BAS

```
Public Sub AppendCountyID()
    'Dimension variables
    Dim parcelRs As New ADODB.Recordset
    Dim ownerRs As New ADODB.Recordset
    Dim parcel_ownerRs As New ADODB.Recordset
    Dim legalRs As New ADODB.Recordset
    Dim varCountyID As String
    Dim varResidenceFlag As Boolean
    Dim theCat As New ADOX.Catalog

    'Open the recordsets and the catalog
    Set theCat.ActiveConnection = CurrentProject.Connection
    'Update OWNER_ID field to be text
    If Not theCat.Tables("LandOwner").Columns("OWNER_ID").Type =
adVarWChar Then
        Err.Raise 1, "LandOwner - OWNER_ID is numeric",
"LandOwner - OWNER_ID is numeric"
    ElseIf Not
theCat.Tables("Parcel_LandOwner").Columns("OWNER_ID").Type =
adVarWChar Then
        Err.Raise 2, , "Parcel_LandOwner - OWNER_ID is numeric"
    ElseIf Not
theCat.Tables("Parcel_LandOwner").Columns("PrincipalOwnerId").Typ
e = adVarWChar Then
        Err.Raise 3, , "Parcel_LandOwner - PrincipalOwnerId is
numeric"
    ElseIf Not
theCat.Tables("LegalDescription").Columns("ID").Type = adVarWChar
Then
```

```
        Err.Raise 4, , "LegalDescription - ID is numeric"
    End If
    parcelRs.Open "SELECT * FROM Parcel ORDER BY PARCEL_ID;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic
    ownerRs.Open "SELECT * FROM LandOwner ORDER BY OWNER_ID;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic
    parcel_ownerRs.Open "SELECT * FROM Parcel_LandOwner ORDER BY
OWNER_ID;", CurrentProject.Connection, adOpenDynamic,
adLockOptimistic
    legalRs.Open "SELECT * FROM LegalDescription ORDER BY ID;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic

    'Set the countyID
    varCountyID = parcelRs.Fields("COUNTY_ID")


    'Deal with residence issues
    varResidenceFlag = False
    Do While Not parcelRs.EOF
        If parcelRs.Fields("RESIDENCE_CD") <> 0 Then
            varResidenceFlag = True
            Exit Do
        End If
        parcelRs.MoveNext
    Loop

    'Append the Parcel table
    parcelRs.MoveFirst
    Do While Not parcelRs.EOF
        parcelRs.Fields("PARCEL_ID").Value = varCountyID & "-" &
parcelRs.Fields("PARCEL_ID")
        If IsNull(parcelRs.Fields("TimberAcres")) Then
            parcelRs.Fields("TimberAcres") =
parcelRs.Fields("TotalAcres")
        ElseIf IsNull(parcelRs.Fields("TotalAcres")) Then
            parcelRs.Fields("TotalAcres") =
parcelRs.Fields("TimberAcres")
        End If
        If Not varResidenceFlag Then
            parcelRs.Fields("RESIDENCE_CD") = 9
        End If
        parcelRs.MoveNext
    Loop
    'Append the LandOwner table
    Do While Not ownerRs.EOF
        ownerRs.Fields("OWNER_ID").Value = varCountyID & "-" &
ownerRs.Fields("OWNER_ID")
        ownerRs.MoveNext
    Loop
    'Append the Parcel_LandOwner table
    Do While Not parcel_ownerRs.EOF
```

```
        parcel_ownerRs.Fields("OWNER_ID").Value = varCountyID &
"-" & parcel_ownerRs.Fields("OWNER_ID")
        parcel_ownerRs.Fields("PrincipalOwnerId").Value =
varCountyID & "-" & parcel_ownerRs.Fields("PrincipalOwnerId")
        parcel_ownerRs.Fields("PARCEL_ID").Value = varCountyID &
"-" & parcel_ownerRs.Fields("PARCEL_ID")
        parcel_ownerRs.MoveNext
    Loop
    'Append the LegalDescription table
    Do While Not legalRs.EOF
        legalRs.Fields("ID") = varCountyID & "-" &
legalRs.Fields("ID")
        legalRs.Fields("PARCEL_ID") = varCountyID & "-" &
legalRs.Fields("PARCEL_ID")
        legalRs.MoveNext
    Loop

    'Close the recordsets
    parcelRs.Close
    ownerRs.Close
    parcel_ownerRs.Close
    legalRs.Close

End Sub
```

APPEND_PARCEL_QUERY

```
INSERT INTO Parcel
SELECT Parcel1.*
FROM Parcel1;
```

APPEND_LANDOWNER_QUERY

```
INSERT INTO LandOwner
SELECT LandOwner1.*
FROM LandOwner1;
```

APPEND_PARCEL_LANDOWNER_QUERY

```
INSERT INTO Parcel_LandOwner
SELECT Parcel_LandOwner1.*
FROM Parcel_LandOwner1;
```

APPEND_LEGALDESCRIPTION_QUERY

```
INSERT INTO LegalDescription
SELECT LegalDescription1.*
FROM LegalDescription1;
```

APPEND_PARCEL_WATERSHED_QUERY

```
INSERT INTO Parcel_Watershed
SELECT Parcel_Watershed1.*
FROM Parcel_Watershed1;
```

APPENDIX F – REMOVING REDUNDANT LANDOWNERS

The final task in completing the SFLO database was to remove redundant landowners across the counties. Previously, after normalizing, parsing and formatting, the data redundant landowners were removed by hand. With over 20,000 land owners in the master database a tool was created to assist in this process. While this VBA module is far from perfect it did help to identify and remove land owners who had land across multiple counties.

MakeMailName concatenated all the land owner names into one field so that redundant owners could more easily be discovered. Since each county has different formats for storing names, getting all the name information into one field was an important step in identifying the redundant owners. RemoveDuplicates searches through the MailName field looking for a search sting. When the VBA code finds a match it eliminates that owner and assigns all of the parcels from that owner to go to the OWNER_ID that is explicitly written in the code. It is easy to remove owners that are not redundant with this tool and some modification is needed to perfect it's utility. None the less, it was helpful in reducing the number of redundant land owners.

MAKEMAILNAME

```
Sub makeMailName()
    Dim rs As New ADODB.Recordset
    rs.Open "SELECT * FROM LandOwner;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic
    Do While Not rs.EOF
        rs.Fields("MailName") = Trim(rs.Fields("LastName") & " "
& rs.Fields("FirstName") & " " & rs.Fields("MiddleName") & " " &
rs.Fields("Suffix") & " " & rs.Fields("Title") & " " &
rs.Fields("OrganizationName"))
        rs.MoveNext
    Loop
    rs.Close
End Sub
```

REMOVEDUPLICATES

```
Sub removeDuplicates()
    Dim oRs As New ADODB.Recordset
    Dim pRs As New ADODB.Recordset
    Dim varOWNER_ID As String
    Dim varSEARCH_STRING As String

    ' Set the Owner ID to KEEP
    varOWNER_ID = "29-8"
    ' Set the String to SEARCH for
    varSEARCH_STRING = "ALOHA LUMBER"

    Debug.Print ""
    Debug.Print ""
    Debug.Print "SEARCHING FOR: " & varSEARCH_STRING & " AND
REPLACING WITH OWNER ID #: " & varOWNER_ID

    oRs.Open "SELECT * FROM LandOwner;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic
    pRs.Open "SELECT * FROM Parcel_LandOwner;",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic

    Do While Not oRs.EOF
        If InStr(oRs.Fields("MailName"), varSEARCH_STRING) And
oRs.Fields("OWNER_ID") <> varOWNER_ID Then
            pRs.MoveFirst
            oPrint = False
            pPrint = False
            Do While Not pRs.EOF
                If pRs.Fields("OWNER_ID") =
oRs.Fields("OWNER_ID") And pRs.Fields("OWNER_ID") <> varOWNER_ID
Then
                    If Not oPrint Then
                        Debug.Print "OWNER_ID " &
pRs.Fields("OWNER_ID") & " changed to: " & varOWNER_ID
                        oPrint = True
                    End If
                    pRs.Fields("OWNER_ID") = varOWNER_ID
                End If
                If pRs.Fields("PrincipalOwnerId") =
oRs.Fields("OWNER_ID") And pRs.Fields("PrincipalOwnerId") <>
varOWNER_ID Then
                    If Not pPrint Then
                        Debug.Print "PrincipalOwnerId " &
pRs.Fields("PrincipalOwnerId") & " changed to: " & varOWNER_ID
                        pPrint = True
```

```
                    End If
                    pRs.Fields("PrincipalOwnerId") = varOWNER_ID
                End If
                pRs.MoveNext
            Loop
            Debug.Print oRs.Fields("OWNER_ID") & " marked for
deletion with a ##..."
            oRs.Fields("OWNER_ID") = "##" &
oRs.Fields("OWNER_ID")
        End If
        oRs.MoveNext
    Loop
    Debug.Print ""
    Debug.Print "DONE!"
    oRs.Close
    pRs.Close
End Sub
```