# DEVELOPMENT OF A PROTOTYPE

# SPATIOTEMPORAL GIS WITH XML PERSISTENCE

by

WILLIAM ERIK SHEPARD

(Under the Direction of E. LYNN USERY)

## ABSTRACT

While there has been considerable research (Peuquet,1994, 2002, Langran, 1992, Worboys, 1998, Usery, 2000) dedicated to developing a complete theory and data model for spatiotemporal data, few of these initiatives have addressed the problem of persisting this data model. It has been repeatedly shown that the relational database model is insufficient for representing spatial and temporal data and is certainly insufficient for representing both types of data simultaneously. Extended relational data models for these types of data suffer from defects and particularly from the loss of clarity expressed in the data model. Object-oriented database systems are better able to capture the semantics of the spatiotemporal model, but have not gained widespread acceptance or use. The eXtensible Markup Language (XML) has achieved near ubiquity and efforts to incorporate its use for persisting semi-structured data in XML database systems have proceeded apace. This research examines the use of XML for persistence of spatiotemporal data, develops a prototype GIS application with XML persistence and compares it qualitatively to development of a GIS application with relational persistence. The development process in this work shows that implementing the spatiotemporal data model in XML is technically no more difficult than implementing it in a relational model and that software performance for this limited application is comparable. However, the XML data model is semantically much clearer than the analogous relational model and allows for better comprehension when viewed without the use of specialized software.

INDEX WORDS: Temporal GIS, Spatiotemporal, XML, Persistence

# DEVELOPMENT OF A PROTOTYPE
# SPATIOTEMPORAL GIS WITH XML PERSISTENCE

by

WILLIAM ERIK SHEPARD

B.S., The University of Georgia, 1995

M.S., The University of Georgia, 2000

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2003

DEVELOPMENT OF A PROTOTYPE

SPATIOTEMPORAL GIS WITH XML PERSISTENCE

by

WILLIAM ERIK SHEPARD

Approved:

| | |
|---|---|
| Major Professor: | E. Lynn Usery |
| Committee: | Suchendra M. Bhandarkar<br>Thomas W. Hodler<br>Vernon G. Meentemeyer<br>Clifton W. Pannell |

Electronic Version Approved:
Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2003

# DEDICATION

This dissertation is dedicated to my dad, the greatest man that I know, and to my mom for the thankless task of just being a good mom.

# ACKNOWLEDGEMENTS

I wish first to thank my major professor, Dr. E. Lynn Usery. Dr. Usery has been a tremendous mentor and encouragement to me throughout my career in GIS, beginning as an undergraduate and continuing through my Masters program and finally my Ph.D. program. Dr. Usery's belief in and respect for me has been a great motivation and the leadership that Dr. Usery has given me on scholastic issues has helped me to grow not only as a practitioner of GIScience, but as a scholar as well. I wish also to thank the other members of my committee, each of whom I also respect tremendously and who have helped me to grow professionally. Dr. Thomas Hodler has taught me that there is more to geography than GIS and more to GIS than data. Dr. Hodler has been especially helpful to me in writing both my Masters and Ph.D. theses and has provided much valuable guidance. Dr. Suchendra Bhandarkar, from the Department of Computer Science likewise has been a mentor to me on both my Masters and Ph.D. theses and is a font of knowledge on all things digital; Dr. Bhandarkar has supported and encouraged my often disconnected progression of ideas, never criticizing and always offering guidance. Dr. Vernon Meentemeyer was the first professor I had in Geography, as an undergraduate and even before I had seriously considered Geography as a profession and a way of life. I credit Dr. Meentemeyer for initially opening my eyes to the discipline and seeing it for the exciting and dynamic field that it is. Finally, but certainly not least, Dr. Clifton Pannell has helped me understand that philosophy has its rightful place in the hierarchy of things. How one thinks about something is at least as important as what one does with that thing; indeed it often is the driver that helps us appreciate how and why we do what we do.

I would be remiss in not also thanking my employer, Miner and Miner, Consulting Engineers, Inc. for supporting me financially in this endeavor. Without their support, completion of this work would not have been possible.

Finally, I must thank my wife, Julie, for her support, understanding and patience while I completed this dissertation. More than anyone else, she has taught me what it is to yearn, to strive and to achieve. Without her encouragement, this dissertation would not have been possible.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

Representation of time in information systems in general, and geographic information systems (GIS) in particular, has long been challenging. For many years, commercial GIS software has been based on the georelational model, which is a spatial extension to the relational database model. Extensive research, however, has readily demonstrated the pervasive problem of representing time in relational systems; the added complexity of the georelational model only complicates this problem. Furthermore, the Structured Query Language (SQL), with which a user queries and updates a relational database, does not easily support temporal extensions. Although there is a large body of research seeking to temporally extend SQL, virtually all of them introduce significant design tradeoffs. These factors make the relational model less than ideal for modeling spatiotemporal data.

Alternatives to the relational model have been proposed with varying degrees of success. The object-oriented database has been commercially available for more than fifteen years, yet it has seen limited market penetration compared to relational database systems such as Oracle, Microsoft SQL Server and IBM DB/2 (although all three of these vendors support object-relational extensions to some extent through Abstract Data Types, or ADTs). One of the challenges to implementing an object-oriented database system is the lack of a standard query language such as SQL for these systems. Most object-oriented database systems provide for an Application Programming Interface (API) to directly access and manipulate objects in the database as if they were instantiated objects in the calling program.

Although this has its benefits, it also requires that programmers accessing the database be familiar with an object-oriented programming language and with object-oriented principles. This is often not the case in industry; many programmers have made a career using simpler tools such as Visual Basic or web programming languages and do not have the skills required for interaction with a complex object-oriented system. Moreover, maintenance of an object-oriented database system requires a database administrator with skills in administrating such a database management system, also a difficult (and costly) proposition.

Occupying a middle ground between these extremes are programmers who have familiarity and frequently work with an object-oriented programming language, but whose organization has standardized on the better supported relational database system. This scenario is not uncommon and may be seen (for example) in a large software development environment, where the technical staff do extensive object-oriented programming, but the business has implemented a relational database management system for its capability to interface with various financial and customer information systems. In this case, the technical staff may need to work with objects in the programming environment, but may need to somehow serialize their objects to store them in the relational database. Although this is an acceptable approach, it introduces a layer of complexity between the database and the program and makes it challenging or even impossible to understand the data outside of the context of the application that uses it.

Adoption of the eXtensible Markup Language (XML), however, has been tremendously successful and XML is now largely ubiquitous throughout the Information Technology industry. XML is similar to HTML (with which many people are familiar) in that it is tag-based. However, where HTML has a relatively small set of predefined tags such as <BODY> or <TABLE> that are limited to formatting tasks, XML is open-ended and supports (in fact requires) creation of user specific tags such as <BOOK> or <TITLE> that are contextually significant to the application. In fact, while HTML is designed only to support document formatting, XML is its polar opposite - it only supports content organization and does not specify any of the particulars of how to display the content. In fact, an XML document is largely self-describing; that is, reading an XML document will provide information not only on the data that it contains, but also on the structure of the data. Figure 1.1 lists a small XML fragment describing a book catalog, which illustrates this self-describing principle.

```
<catalog>
      <book>
            <title>Huckleberry Finn</title>
            <author>Mark Twain</author>
            <timeperiod>19th Century America</timeperiod>
            <language>English</language>
      </book>
      <book>
            <title>Les Miserables</title>
            <author>Victor Hugo</title>
            <timeperiod>French Revolution</title>
            <language>French</language>
      </book>
</catalog>
```

**Figure 1.1: Book Catalog XML Fragment.**

This self-describing capability, in combination with the fact that the only tool required to create and edit XML is a text editor, has made XML tremendously useful to the commercial enterprise. In fact, XML has revolutionized the exchange of digital information. XML has replaced a myriad of proprietary and vendor-specific formats and has supplanted the aging X.400 standard as the mechanism for Electronic Data Interchange (EDI), which facilitates electronic communication between disparate systems. The Simple Object Access Protocol (SOAP), which is a standard for interoperability of web services supported by such vendors as Microsoft, Oracle, Sun and IBM, is based on XML as well.

Of course, XML is a relatively new technology and as such is not without its own problems and tradeoffs. While the XML 1.0 standard was ratified in December of 1998, the

World Wide Web Consortium (W3C) is still actively working on components of XML 2.0 and its associated technologies today. Database research based on XML is, in particular, fledgling and much of the current investigation simply involves efficient retrieval and update of XML documents. Nevertheless, research has been promising and has made significant strides in recent years. There are several available XML database systems that have overcome some of these problems; two of the most advanced are the Tamino XML database and the eXist XML database. Both are native XML databases; that is, the database represents data as XML documents and resultsets to queries are given in terms of XML rather than in terms of the tuples or relations returned by relational database systems. The major commercial relational database vendors have hedged their bets as well and have adopted XML within the database. These hybrid approaches store data in relational tables, but can transform the data into XML for output, in contrast to the native XML approach. The benefit is that the underlying data can take advantage of all of the technology present in relational systems such as indexing and transactions, but at the disadvantage of the underlying representation potentially being nothing like the resultant XML that is returned by the database.

Geographic information science has not ignored the potential of XML; indeed the OpenGIS Consortium (OGC) is currently working on its second revision of the Geography Markup Language (GML), which is a schema dialect of XML specifically targeting interchange of spatial information. The second revision of GML also includes some capability for storage of temporal and volumetric information. The key problem with GML lies in its target application. GML is intended for the interchange of spatial information, not for its dynamic storage. Although this is a subtle distinction, the difference lies in the static nature of interchange. In one scenario, an initiating system will produce a static XML (or GML) file that is then passed to a receiving computer system. The receiving system may then parse the data and act upon the information in the GML file, and may even produce a reply that is also formatted as XML (or

GML). An example of this scenario can be found in a web-based GIS where the data to populate the map are stored in a georelational database and are only converted to GML before being sent to the web browser for display. In a second scenario, a GML file may be exported from one geographic information system to be imported by a different geographic information system. An example that illustrates this scenario would be to convert between two GIS packages such as Idrisi and ERDAS Imagine. In this case, GML serves as the *lingua franca* between the two packages and the data only reside in GML during the conversion process. The GML becomes the medium of transport between the two disparate systems. In both cases, the data are static. GML is optimized for static data, which is perfectly acceptable for Online Analytic Processing (OLAP), but is entirely inappropriate for Online Transaction Processing (OLTP), which is by its very nature dynamic. In any case, the temporal extensions to GML are embryonic and are not fully developed, nor are they solidly grounded in temporal information theory. These problems must be addressed before GML native GIS may join or supplant georelational GIS.

The purpose of this dissertation is to develop an XML based model to persisting spatiotemporal data that is solidly grounded in temporal information theory and cognitive science and to apply that model to develop a prototype spatiotemporal GIS with XML persistence. This research will show that an XML schema better describes the semi-structured nature of temporal data in general, and spatiotemporal data in particular than does a relational schema. Moreover, this research will also show that implementation of XML persistence for spatiotemporal data is no more difficult than implementation of relational persistence for such data while imparting increased descriptive power. This will be demonstrated by comparing the development process for a prototype XML-based spatiotemporal GIS to the development process for an analogous relational spatiotemporal GIS and by comparing the resultant data repositories in relational and XML format. The fundamental spatiotemporal model adopted in this work is based heavily on the work of Usery *et al.* (2002) exploring temporal representation. This work, however, is in sharp contrast to Usery *et al.* (2002) in that most of that work has focused upon representation at the application level and not at the persistence level. This research will also assess the state of the art in terms of database technology present in XML databases and the implications for efficient and consistent access to spatiotemporal data.

The specific objectives of this research are:

1. Develop an XML based data structure for representing spatiotemporal data. The usefulness of XML as a mechanism for efficiently persisting temporal data is postulated based upon a variety of contributing factors. First, the most efficient database structures are hierarchical, and indeed indexing data structures such as B+-Trees, R-Trees or Quadtrees are often based upon these hierarchical structures. Second, set theory dictates the unique existence of an object. However, many versions of an object (as is the case with temporally extended features) require non-uniqueness of an object - a so-called bag. XML supports these bags better than do relational models. Third, cognitive category theory is based upon a hierarchical structure; and it has been shown in the cognitive science literature that cognitive economy is achieved by traversing multiple levels of a knowledge tree.
2. Develop a relational schema for a simple electric distribution GIS with temporal attributes and design the queries that would be required in order to access records in the database. This relational schema will contain structures such as poles, electric devices such as transformers, electric conductor (power lines) and customer data.
3. Develop a fictional spatiotemporal GIS based upon the relational schema. This GIS shall be capable of drawing the current state of the GIS at a particular time and viewing the attributes of a feature at that time. The GIS shall also be capable of querying for

spatial or thematic attributes. The GIS will support a series of queries designed to demonstrate some of the difficulties of dealing with temporal data.

4. Develop an XML based schema for the same simple electric distribution GIS with temporal attributes and design the queries that would be required to access records in the database.
5. Develop a prototype spatiotemporal GIS based on the XML schema with the same target functionality as the relational spatiotemporal GIS. Compare and contrast the complexity of the XML-based GIS software to the relational GIS software. Also compare and contrast the two development processes and the resultant data repositories.

Few commercial sectors have embraced geographic information systems to the extent of the utility industry. Geospatial information technology has enabled utilities – including electric, gas, water, wastewater, cable and fiber – to visualize the location of facilities and to manage their assets in a meaningful way. Many utilities have miles of infrastructure whose location is either unknown or is in question. As this infrastructure ages, it becomes critically more important to be able to locate and determine characteristics about this infrastructure, such as when it was constructed and what materials were used. To compensate for and overcome this difficulty, utilities have been ready adopters of commercial geographic information systems technology in the form of automated mapping and facilities management (AM/FM) solutions. However, the incredible need for capabilities to store and manipulate temporal information in a meaningful way is an as-yet unmet need.

In the electrical utility industry, the challenge of modeling and storage of asset data has been met aggressively by geographic information systems vendors. Environmental Research Institute, Inc. (ESRI), the makers of Arc/Info and the ArcGIS software system, have developed in partnership with Fort Collins, Colorado based Miner and Miner, a standard GIS model for electrical utility data. This model provides a standard starting point that is flexible enough to be tailored to a specific business. ArcGIS provides capabilities for enterprise interoperability and when paired with the Miner and Miner ArcFM software solution offers a powerful yet easy to use geographic information system that is built on this model for both electric transmission and electric distribution. GE Network Solutions, a key competitor to ESRI and Miner and Miner, offers Smallworld GIS, which can also be tailored to the electric utility market. Yet although temporal extensions to both ArcGIS and Smallworld can be found in the literature, neither has yet to offer a commercially viable solution to the temporal problem.

Data models for electrical distribution and transmission define a standard set of spatial asset features that the utility must maintain. Electric transmission consists of high voltage conduction of electricity from a generation point, such as a hydroelectric, coal or nuclear power generation plant to a substation or other distribution point on an electric utility network. Although an electric utility may generate its own power, more frequently (and particularly in the case of smaller cooperatives) power is purchased and the utility takes ownership of the power at the substation. An electric transmission GIS model defines such features as high-voltage conductor, poles and towers and electrical devices such as switches and transformers.

An electric distribution data model is suited to the utility that provides power to a service area. Electric distribution typically models such features as structures, conductors and devices. Often, electric distribution AM/FM/GIS solutions interface with customer information and financial systems and will thus contain aspects of both of these in the data model as well.

Structures are non-electric equipment that supports electrical devices and conductors. Structures may be underground or overhead, or may be located on the ground. Examples of structures are poles, pads or values. Poles support overhead equipment and may be constructed of wood, steel or concrete and are rated based upon the load that they may support. This load will consist not only of the weight of the conductor and devices on the pole, but also the forces due to wind, ice loading and angular pull at corners where the electric line changes direction. Pads protect electrical equipment located on the ground, for example transformers located near homes are typically mounted on concrete pads. Vaults may protect underground devices in locations that have underground utilities. In addition to modeling features purely for asset management, many electric distribution utilities will also apply the GIS to solve problems of construction, and often interface with third party software packages that provide engineering calculations for requirements to support loads. In these cases, the GIS may also model features such as guy wires, which play an integral role in aiding the pole to support the load that it must bear.

Conductors, put simply, conduct electricity over some distance. Although conductors are typically wire or cable, they may also be a solid piece of some conducting metal, such as copper or aluminum. Conductors may be modeled as overhead or underground features, depending of course on whether the conductor is installed overhead on a pole or is buried underground. Conductors are also often modeled in terms of primary or secondary levels of voltage. Primary conductors (both overhead and underground) distribute electricity from a source such as a substation throughout the electric network, at high voltages. Near the point of delivery to the customer, primary voltage will be converted (or transformed) to a lower voltage and carried the remainder of the way to the customer service point along a secondary conductor that is either overhead or underground. Electric distribution models will often model a conductor as a single linear feature, although in the real world it may be composed of multiple actual wires; for example, electricity is often distributed in some combination of three phases: A, B and C respectively. If a distribution conductor is carrying only a single phase of electricity, for example A phase, it will consist of only a single conductor. If, however, the conductor is carrying multiple phases of electricity – AB phase or ABC phase, it will consist of two or three conductors respectively. The GIS will model both or all three conductors as a single feature and may represent the distinction between the different actual conductors as related records in a table.

In general, electric devices either control the distribution of electricity or protect the network from faults. Transformers are the most familiar electric devices; these are the big electric boxes near houses. Transformers convert or transform electricity from one voltage to a different voltage. For example, a transformer located near a house may connect a primary and secondary conductor and transform voltage from a higher primary level to a lower secondary level before delivering it to the house. Voltage regulators may be placed throughout the network to smooth out any artifacts in the electrical signal and provide a consistent voltage. Protective network devices are not unlike the fuses or circuit breakers in the home, but on a much larger scale. A distribution network will typically make use of fuses and circuit breakers along the network to insulate and isolate the network from electrical surges due to equipment malfunction or lightning. In the event of a surge, these protective devices will trip and open the circuit so that electricity does not flow through them (much like a fuse or a circuit breaker). Other types of protective devices may have more intelligence and may attempt after an initial isolation to reestablish service once the danger has passed. Reclosers are electronic devices that initially open the circuit, but then attempt to recluse the circuit a certain number of times before failing. This assists the utility in providing better service quality to their customers while still protecting their assets.

In the GIS developed for this dissertation, a simple data model is used consisting of nine feature classes and one related table modeling structures, conductors, devices and customer information. This simple model is based on a subset of the ArcFM Electric Distribution data model developed by Miner and Miner (ESRI and Miner and Miner, 2001). These ten classes are summarized in table 1.1. The prototype GIS for this research demonstrates a typical application in a fictional set of scenarios by modeling the construction of a new subdivision beginning with the zoning of parcels from a large undeveloped tract into smaller single family residential parcels and ending with a nearly completed subdivision. In addition to storing the representation of the network at different time periods as the subdivision is constructed, the GIS also stores changes to features either through attribute update (to simulate equipment upgrade) or through spatial adjustment (perhaps due to equipment relocation or resurvey). Figure 1.2 maps the beginning and final states of the fictional prototype data.

Table 1.1: Feature Classes for Electric Distribution GIS.

| Feature Class | Description |
|---|---|
| Street Light | Street light, typically mounted on a pole |
| Meter | Customer meter location, point of service to a building |
| Transformer | Transforms primary electrical signal to secondary electrical signal by lowering voltage. Transformers and conductors may be any of A, B, C, AB, AC or ABC electrical phases. |
| Switch | Switches allow for power to be enabled or disabled to a section of the network (called a feeder). A switch may be closed (closing the circuit and allowing power to flow) or may be opened (to interrupt the flow of power, for example to do maintenance work safely). |
| Riser | Small segment of conductor that connects an above-ground conductor to an underground conductor, typically attached to the side of a pole. |
| Pole | Wood or metal structure that holds conductors, electric devices such as transformers, protective devices, or street lights |
| Secondary Conductor | Carries lower voltage power, typically to a meter location, but can also be carried to one or more additional secondary conductors. Tapped off from a primary conductor at transformer. |
| Primary Conductor | Carries high voltage power from a substation. |
| Parcel | Parcel polygons. |
| Customer | Related customer information to the parcel polygons and meter points. Customer information is stored in this related table for normalization purposes. |

# CHAPTER 2

## LITERATURE REVIEW

Geographic information science has focused upon methods for representing and analyzing spatial data. While representation originally focused upon Cartesian coordinates, in the tradition of cartography, spatial data representation has expanded in recent decades to include three-dimensional data and spherical coordinate systems. There has also been an evolution from the dualist approach to science with geography as the spatial science and history as the temporal science and increasing recognition of the interdependence between space and time. Indeed, this is precisely what General Relativity suggests (Einstein, 1920). Representation of temporal data has proved particularly challenging as its incorporation involves more than simply adding additional spatial dimensions. There have been numerous approaches proposed to modeling of spatiotemporal data, focused mostly at the conceptual level. While this research has advanced the state of knowledge, it has left a dearth of research at the physical persistence level. This problem is compounded by the fact that spatial database reseach and temporal database research have proceeded along largely independent lines.

The relational model was developed by E. F. Codd (Codd, 1970) at IBM in order to overcome the chief deficiency of the existing models of that time, the hierarchical and network models. These models, while powerful, were highly implementation dependent. Not only did it require that database users be intimately familiar with storage structures of the database, but these structures could not be generalized to other database systems. Unlike these models, the relational model exposes to the user a set of abstractions that the user can interact with and hides the implementation details. Instead of writing path access routines that required this familiarity with the underlying data structure, the user can instead access data by creating expressions in SQL. This language also has its foundations in the work by Codd (Codd, 1971b).

The relational model is based extensively on mathematical set theory (Elmasri and Navathe, 2000). Because of this basis, the operations defined in the relational model have been proven to be complete and correct. The fundamental unit in the relational model is the tuple (much like in set theory), which is simply the collection of attributes for a single record. For example, the tuple {1, 'Meadow Road', 25, 'City Road'} might be a tuple that represents a record from a transportation database where the record number is 1, the road name is Meadow Road, the road length is 25 and the road type is City Road. It should be noted that the information about the meaning of this tuple is not embedded in the tuple; only the data for the record itself is part of the tuple. In order to elucidate the meaning of this data, the database catalog must queried and paired with the tuple. Common set theoretic operations are possible on this tuple, for example the projection operation can be used to reduce the tuple to a smaller tuple (the tuple above could be projected simply to {1, 'Meadow Road'}. Likewise the Cartesian product operation can be used to join together two sets of tuples (this is the basis of table joins). In the relational model, tuples can not be nested. This leads to important limitations with the model for representing certain types of data.

Tables are created by aggregating collections of like tuples; that is, all tuples in a table must have the same set and ordering of tuples (although the tuples themselves may be in any order, as may sets). The user interacts with these tables as rows and columns where rows are tuples that are referred to as records and columns are attributes that are individual elements of the tuple. Although not strictly part of the tuple, virtually all database systems also present the schema to the user as well (as column headers) and obviate the need for the user to query the catalog directly. Collections of tuples (tables) with a common attribute can be joined together, using the Cartesian product operator, to form relations.

The SQL language is a non-procedural language that allows the user to specify queries to the database in terms of these tables and relations. SQL is based on relational algebra, which is itself and extension to set theory. Basic set theoretic operations are part of the SQL language and can be used to formulate and refine queries. Combinations of operators allow for creation of relations and returning of related tables and subsets of related tables. As with the relational model, there is no provision in the current SQL standard (SQL-92) for creating data structures beyond those provided by the model; thus, as with the model, the standard SQL language makes no provision for user defined data structures, or abstract data types; in particular, SQL does not support creation of nested tuples. Creation of such abstract data types can only be accomplished through definition of a related table that encapsulates the required nesting in the table.

The deficiencies of SQL and the relational model for persistence of spatial data have been well-noted (Egenhofer, 1992). Because SQL-92 does not support nested tuples, definition of spatial data structures has proven particularly challenging. Although such spatial data structures can be accomplished with related tables, this approach is not elegant. At a minimum, spatial data consists of a single {x,y} tuple pair to represent a point in Cartesian space. Extrapolation of this single tuple to represent more complex geometries, such as lines and polygons requires persistence of many {x, y} tuple pairs. Further extension of space into 3 dimensions requires a three-dimensional tuple, or triple {x, y ,z}. Incorporation of additional spatial dimensions, including measured shape values can further extend into multidimensional tuples such as {x, y, z, m}. In combination with such a multidimensional tuples, the subsequent addition of attributes to a point feature would require a nested tuple of the form {a1, a2, {x, y, z, m} } where a1 and a2 are attributes of the spatial feature; for linear or polygonal features the situation is even more complex, requiring even further nested tuples of the form {a1, a2, {{x1, y1, z1, m1}, {x2, y2, z2, m2}, … }}. The incorporation of time into this miasma of tuples further extends the problem, requiring tuples of the form {{tb1, td1, a1}, {tb2, td2, a2}, {{tb3, td3, {x1, y1, z1, m1}}, {tb4, td4, {x2, y2, z2, m2}}, …}} where tbx and tdx are birth and death times for the spatial or aspatial attribute.

The lack of spatial operators in the SQL language also limits the applicability of the language for performing even simple spatial queries. There are no inherent capabilities for spatial queries. For example, even common queries such as point-in-polygon or nearest neighbor searches are hard to express in the SQL language. Although these operations can be coded, they are inelegant at best.

Because of the deficiencies inherent in the core relational model, many database vendors have adopted object-relational extensions for complex data (Larson, 1995). The object-oriented database model is an evolution of the relational model that introduces the ability to define a complex object with attributes and behavior within the context of a database system, with the benefit that non-simple types of data can now be represented without the need for a complex schema with many related tables (Atkinson, 1989). Object-relational database systems are hybrid solutions that fall between relational and purely object-oriented database systems (Stonebraker *et al.*, 1990). Such hybrid systems are built fundamentally upon a relational core, but have the ability to be extended to incorporate some or all of these object-oriented features. These object-relational hybrids allow for the creation of abstract data types (ADTs) that allow the user to model the database in terms of business rules.

The chief drawback to object-oriented and object-relational database systems is the lack of a codified standard. This lack of a standard has had wide implications for the adoption of these technologies. Purely object-oriented systems have not seen wide adoption outside of academia except in a few cases where the benefit gained from their adoption exceeds the complexity of implementation and impact of abandoning of standard database approaches. Likewise, database vendors have been left with the freedom to implement their own custom relational extensions and query languages for object-relational hybrids. This has meant a wide variety of vendor specific implementations for object-oriented and object-relational systems, and hence an equally wide ranging set of approaches to technology that builds upon an object-relational core (such as spatial and temporal extensions).

There are at present two proposed standards for query and management of object-oriented and object-relational data. Although there are similiarties between these two approaches, they differ fundamentally in their goals. These two standards are the Object Query Language (OQL) and the third revision to standard SQL (SQL-3).

Stonebraker *et al.* (1990) suggested that SQL be extended to include object-oriented concepts layered on top of SQL. This approach has been followed in the present SQL-3 working revision. Work on SQL-3 began in 1992, following the release of SQL-92; SQL-3 was to be released in 1995, but to date is not complete. The SQL-3 language completely supports the relational data model and is backwards compatible as well as supports object-oriented features. Because of this, the standard at present is quite dense and numbers about 1200 pages. Because most database vendors have followed this approach anyway, but have done so with custom extensions to SQL, there is a great deal of disagreement on the syntactical structure of the language (Elmasri and Navathe, 2000).

Darwen and Date (1995) differed and asserted that the relational model itself should be extended to include object-oriented features. They also asserted that SQL was dated and should be abandoned completely in favor of a database language that supported this object-oriented extension to the relational model. They maintained that the relational model could not be extended within the constraints of the current SQL language. This difference is borne out in the divergence between the SQL and the OQL language definitions. This philosophy appears in the definition of the OQL language. OQL is much more terse and complete, but at the expense of abandoning support for the relational model in favor of a purely object-oriented approach. Defined by the ODMG standard (ODMG-93 was drafted in 1993 and revised to the ODMG 2.0 standard in 1997) for interoperability between object-oriented database managements systems, OQL is analogous to SQL in defining query capabilities.

ODMG also defines a set of language specific APIs for accessing an OODBMS from such languages as C++, Java and Eiffel. Purely object-oriented database systems are more often accessed through these APIs than through query languages. Because object-oriented database systems interact chiefly with programming languages, they suffer from differences between these programming languages. Although some facets of the language are purely syntactic, there are other aspects that are more important – for example, C++ supports multiple inheritance while Java does not. Likewise, languages such as Eiffel require an interface-based approach to methods, while other programming languages support the use of but do not require interfaces. This does complicate the use of object-oriented and object-relational systems, and requires that the database user be familiar with the target language that is accessing the database. Moreover, data access to such systems is highly interdependent upon the data model and not easily generalized to different systems.

The most common approach in the literature to incorporating spatial and temporal storage into the relational model has been to adopt such an object-relational approach and to extend the the SQL language to explicitly codify spatial data structures and operations (Huang and Lin, 1999, Herring *et al.*, 1988). A recent paper by Voisard and David (2002) has presented an excellent review of attempts to incorporate spatial data models in a relational framework. In their paper, Voisard and David investigated the representational power of extended SQL with abstract data type capabilities in comparison to an object-oriented database. The a priori decision to not investigage standard SQL as part of this work further confirms the findings of Egenhofer (1992) that SQL alone is insufficient to the task. In their work, the authors found that the relational model suffered from insufficient data structures and the necessity to implement geometric operations at a discrete lower level from the SQL abstractions (which obviates the main value of SQL to separate database implementation from interface). In comparison, a model based upon $O_2$SQL (a predecessor of the OQL language) was much more expressive.

These findings support similar earlier work examining the applicability of object-oriented models for spatial data management. Egenhofer and Frank (1989) developed concepts for such an approach and examined in particular the issues of inheritance of features. The authors found that geographic data modeling benefits greatly from the descriptive power of inheritance and propogating common properties through a model. Worboys (1994) similarly investigated object-oriented approaches to spatial data management from a higher conceptual level. His findings confirmed that the explanatory power of an object-oriented model better fit the real-world data that such a geographic information system seeks to represent. In particular, in both of these works, the importance of operations and behavior on geographic features (and the commonality between them, underscoring the value of an inheritance based system) was elucidated.

While the previous works both focused on the operational aspects of the model, particularly on map algorithms, several other authors have carried forth similar work from the perspective of pure data modeling. Hadzilacos and Tryfona (1996) present a data model for spatial features that is based on an extended-relational schema. This work is extended in Hadzilacos and Tryfona (1997) and examines the extended-relational model (which bears great similarity to object-oriented models). Their work focused upon the semantics of the data model (and how such an extended-relational model could further clarify the semantics), but still relied upon methods and operations for much of their descriptive power.

This has also been the approach of database vendors to solution of this problem. From Oracle the Oracle Spatial Data Option provides an object-relational structure in which spatial data can be stored, and an extension to the SQL language that incorporates capabilities to perform spatial queries. It should be noted that such capabilities are often handled at much lower levels in the database and not directly in the SQL language. The SQL construct that is called for the spatial analytical operation may actually be coded at a low level in the database in a native language like SQL or Java. Although its operation will be transparent to the user, such an operation is not based upon set theory. This has also been the approach of ESRI in the ArcSDE (spatial data engine) product. While such an approach is generally efficient, it is not portable and commits the user to a particular product.

Temporal database systems built upon the relational model are fraught with similar problems. Gregersen and Jensen (1999) provide a survey of attempts to extend the entity-relationship model to encompass temporality. Although the entity-relation model has been largely superseded in recent years by the Unified Modeling Language (UML) for abstraction, these attempts yield a good insight into the design principles addressed by this research. Most notable is the fact that all of the attempts required significant extensions to the underlying approach in order to explicitly include temporality. Of the ten versions reviewed, Gregersen and Jensen (1999) identify only two as being complete enough to be generally useful – the TERM model (Temporal Entity-Relationship Model), which is one of the first versions (Klopprogge and Lockeman, 1983) and the TERC+ model (Zimanyi, *et al.*, 1997). The remaining eight temporally extended entity-relational models all suffer (in the opinion of the authors) from either being too abstract or too incomplete.

The entity-relationship model is a high-level abstraction that can define the semantics of the data model, but cannot define the physical persistence model. For implementation, relational models are necessary. In fact, one of the reasons for the popularity of UML as a successor to the entity-relationship model is the semantic expressiveness of the model while still maintaining enough of the implementation details to make definition readily accomplished.

Snodgrass (1992) remains a definitive work on temporally extended relational models. Snodgrass in particular identifies the duality of time (or bitemporality) between valid time and transaction time and rightly identifies that each may have its own timescale. Each of these temporal dimensions may be represented either by timestamping attribute values or by timestamping entire tuples, with approaches to each approximately equally divided. Timestamps may be accomplished either through a single time (a chronon, the smallest unit of measure in the temporal database) or an interval with a start and end time. Also possible are representations with more than one time to represent historical occurrences.

As with SQL for spatial queries, temporal queries are characteristically hard to describe in SQL (Snodgrass, *et al.*, 1995). There have been a number of attempts to temporally extend SQL in much the same way that spatial extensions to SQL have been developed. The TQUEL (Snodgrass, 1987) and TSQL2 query languages (Snodgrass, *et al.*, 1995) were both developed with this goal in mind - as have a number of others (Navathe and Ahmed, 1989; Brusoni *et al.*, 1999), but in neither case is syntactic expression of temporal queries elegant. The concatenation of difficulties in both the spatial and temporal domains clearly demonstrates the very real difficulty in constructing an extended query language based on SQL for spatiotemporal data.

There have been some attempts to unify temporal and spatial database research in the computer science literature as well. Price *et al.* (2000) examine the constructs needed by the UML language to adequately model spatiotemporal data. They conclude that additional constructs are needed, but that the extensibility of the language can support this need. Yazici *et al.* (2001) examined the problem from the perspective of semantic modeling and also focused heavily on conceptual modeling in UML. However, where Price *et al.* focused on the mechanics of the model, Yazici *et al.* concentrated on high-level constructions. Such approaches are typical of the attempts to unify the literature as it has been generally recognized that the difficulties inherent in modeling spatial and temporal data discretely multiply in combination and an object-oriented model is necessary in order to overcome these obstacles.

Database systems based upon XML have provided a recent alternative to relational database management systems that overcome some of the deficiencies of the model. XML was originally conceived in 1998 as a markup language similar to HTML. Unlike HTML, which is based upon a predefined and limited set of tags, the structure and tags of XML are completely customizable. Moreover, HTML is designed as a markup language to control presentation, while XML says nothing about the presentation of the data it contains, but only formats the data in a meaningful and consistent way. The tag structure of XML facilitates management of metadata, since the tags themselves become metadata about the data contained in the tag. Because XML is plain text, it is also a lightweight standard that is easily transporatable between heterogeneous systems, without binary encoding difficulties or characterset conversions that are often necessary between ASCII based systems such as UNIX or Windows and EBCDIC based systems such as VM or MVS. Furthermore, the protocols for parsing and accessing XML documents, particularly the Document Object Model (DOM) and the Simple API for XML (SAX) are standardized and have many implementations on virtually every operating system platform. The extensibility of the language combined with its portability have made the use of XML ubiquitous for formatting messages to integrate disparate systems; hence XML has realized wide acceptance by the commercial sector. Many modern software packages also support conversion from their native document format to an XML formatted file. XML has become a digital Rosetta Stone, linking together disparate processes and systems. XML underpins the modern internet economy and Web Services, the current architecture for distributed computing, relies heavily on XML through standards such as the SOAP.

Like object-oriented and object-relational database systems, XML databases can persist semistructured data. Relational database systems excel at efficient storage and management of structured data where all of the data conform rigorously to a specified schema. Because these data are structured – the same attributes occur on each record – the table is a good abstraction; tables present data to the user in terms of the same set of rows and columns or records and attributes. In contrast, formats such as plain text store unstructured data. In unstructured data, data may occur in any order, or no order, and may have any set (or no set) of attributes. Typically, unstructured data cannot be easily queried. Between these two extremes, semi-structured data usually has an overriding schema governing the macro organization of the document, but in specific records of the document the degree to which the schema is implemented may vary widely. While relational data models cannot efficiently work with such semi-structured data, XML handles them quite well. Inherent functionality in the XML language does facilitate querying of the semistructured data.

The ready adoption of XML as a industry standard language has engendered a whole research agenda for database systems based on XML. XML database research seeks to extend XML beyond its file-based roots to incorporate database technologies such as transaction management and query optimization. Chaudhri *et al.* (2003) provide an excellent introduction to the various current XML technologies and their application for database management. XML is comprised of a number of subset standards defined by the World Wide Web Consortium (W3C). Because many of these standards are still undergoing defition, aspects of the language that are necessary for database implementation are not yet finished. Nevertheless, XML database research has an active and aggressive agenda, and XML database systems have shown promise to provide the ubiquity of SQL with the power of object-oriented approaches.

Of the varied XML language definitions, several are of key interest for representing and persisting data in a semi-structured XML format. The XML language itself, as already described, consists of document structure and constituent data. The structure is comprised of tags and attributes. A tag may be simple or may be complex; simple tags contain only data while complex tags may contain nested tags or both tags and data. Attributes are attached to the tag to provide additional metadata about the data stored inside the tag. Figure 2.1 presents a deliberately frivolous XML fragment designed to illustrate the structure of the language with respect to tags, attributes and data.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml_document>
      <simple_tag>simpledata</simple_tag>
      <simple_tag2 attribute1="value">simpledata2</simple_tag2>
      <complex_tag attribute2="value2">
            <complex_tag2 attribute3="value3">
                  <simple_tag3>simpledata2</simple_tag3>
                  <simple_tag4 attribute4="value4">
                        simpledata3
                  </simple_tag4>
            </complex_tag2>
      </complex_tag>
</xml_document>
```

**Figure 2.1: XML Fragment.**

The Data Type Definition (DTD) language was developed to facilitate creation of schema documents for XML files. While the DTD was successful in providing these capabilities, it suffered from several particular drawbacks. Among these, one of the most important was the lack of capabilities to identify the data type. This is a critical need for database systems, which have requirements to efficiently store and access data. Without a priori knowledge of the data type of a particular attribute, it is difficult to develop efficient query capabilities. The XML

Schema Definition language (XSD) superseded the DTD language and incorporates many of the missing components from DTD that are necessary for XML based persistence. The XSD fragment listing in Figure 2.2 provides the schema necessary for the XML fragment in Figure 2.1 Figure 2.3 lists the XML fragment in Figure 2.1 with the necessary information to reference the associated XSD file.

The eXtensible Stylesheet Language (XSL) enables formatting of XML documents that XML itself does not. Like the HTML stylesheet language CSS (Cascading Stylesheets), XSL applies rules to parts of an XML document to format it; as a consequence of these capabilities, XSL can also transform a document from one form to another. XSL is made up of three subset languages, two of which are described here.

The first subset language of XSL is the the XPath language, which has similar capabilities to the UNIX regular expression syntax on which such tools as awk, sed and grep are based. While not a true query language, XPath has the capability to perform complex pattern matching in XML documents relatively quickly. XPath is a key component of XML data storage as many other constituent languages in the XML family are based on it. Using XPath, a single node (which is delimited by a tag) or a collection of nodes (each with their own tags) may be retrieved from the document. XPath expressions may search for tag names or values (data) or by attribute names and values. Table 2.1 lists several XPath expressions against the sample XML fragment from figures 2.1 and 2.3 and the result sets that are returned by the expressions. The XPath expressions listed in table 2.1 are not comprehensive, but do give an idea of the syntax of the language.

The second subset language of XSL is the XSL Transformation language (XSLT). XSLT uses XPath to query parts of the XML document and to transform them into a different form. The most common example is the XHTML language , which is nothing more than an XSL stylesheet applied to an XML file to produce an HTML web document suitable for publication.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
      <xs:element name="complex_tag">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="complex_tag2"/>
                  </xs:sequence>
                  <xs:attribute name="attribute2" type="xs:string"
                  use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="complex_tag2">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="simple_tag3"/>
                        <xs:element ref="simple_tag4"/>
                  </xs:sequence>
                  <xs:attribute name="attribute3" type="xs:string"
                  use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="simple_tag" type="xs:string"/>
      <xs:element name="simple_tag2">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="attribute1"
                              type="xs:string" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="simple_tag3" type="xs:string"/>
      <xs:element name="simple_tag4">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="attribute4"
                              type="xs:string" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="xml_document">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="simple_tag"/>
                        <xs:element ref="simple_tag2"/>
                        <xs:element ref="complex_tag"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

**Figure 2.2: XSD to Describe XML Fragment.**

```
<?xml version="1.0" encoding="UTF-8"?>
<xml_document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XMLFragment.xsd">
      <simple_tag>simpledata</simple_tag>
      <simple_tag2 attribute1="value">simpledata2</simple_tag2>
      <complex_tag attribute2="value2">
            <complex_tag2 attribute3="value3">
                  <simple_tag3>simpledata2</simple_tag3>
                  <simple_tag4 attribute4="value4">
                        simpledata3
                  </simple_tag4>
            </complex_tag2>
      </complex_tag>
</xml_document>
```

**Figure 2.3: XML Fragment with XSD Schema.**

However, XSLT is much more powerful and can be used for a variety of other uses as well. XSLT includes mathematical, string, conditional and loop operators so that transformations may be quite complex. Another common use of XSLT might be to integrate two disparate systems; unlike many earlier standards such as Electronic Interchange (EDI), XML and XSL Transformations are quite facile at moving from one schema to another that is completely different. Using mathematical operators, units of measure can even be converted between XML documents (for example, miles might be converted to meters). All of this happens transparently to and indepently of the user if such a stylesheet is applied to the XML file. Figure 2.4 illustrates an XSL stylesheet and the XML fragment in figures 2.1 and 2.3 transformed using this stylesheet.

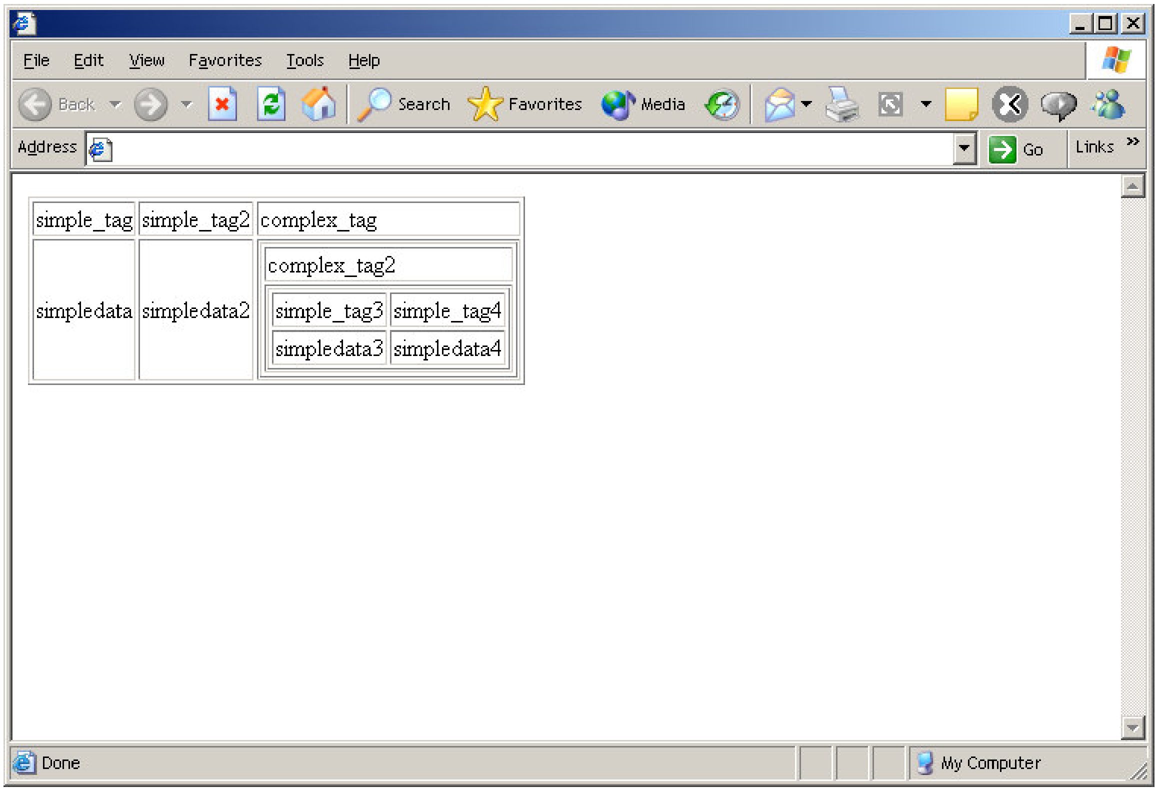
Several new standards and query languages are currently under finalization (Bonifati and Lee, 2001). Query languages can be divided broadedly into two camps; the SQL/XML standard is under development by the SQLX group, comprised primarily of relational database vendors such as Oracle. The SQL/XML standard facilitates interoperability between a relational database and XML; particularly for returning relational data from a SQL query as XML. The XML-SQL languge (Pankowski, 2002) follows a similar approach.

Table 2.1: XPath Expressions and Result Sets.

| Expression | Result Set |
|---|---|
| xml_document | Returns the complete XML document from the root node. |
| xml_document/* | Returns each of the nodes under the xml_document node: simple_tag, simple_tag2 and complex_tag2 |
| xml_document/simple_tag | Returns the simple_tag node |
| xml_document/simple_tag2[@attribute1=”value”] | Returns all simple_tag2 nodes (if there are more than one) with an attribute1 that has a value of “value”. |
| xml_document[simple_tag2/@attribute1=”value”] | Returns an xml_document node that has a simple_tag2 node with an attribute named attribute1 that has a value of “value”. |

In contrast, the XQuery language is designed specifically for use with XML documents. XQuery is based on XPath expressions, but extends the language by adding “FLOWR” constructs (pronounced flower). In addition to the capabilities already present in the XPath language, XQuery adds constructs to iterate over a set (for), assign variables (let), specify search criteria (where), set the result order (order by) and return a specific result set (return). Using these constructs, XQuery can be as terse as SQL. Figure 2.5 illustrates a SQL clause and a comparable XQuery clause for the book catalog listed in the previous chapter. Shanmugasundaram *et al.* (2001a) and

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <xsl:template match="/">
            <html>
                  <head />
                  <body>
                        <xsl:for-each select="xml_document">
                              <xsl:if test="position()=1">
                                    <table border="1">
                                          <thead><tr>
                                                <td>simple_tag</td>
                                                <td>simple_tag2</td>
                                                <td>complex_tag</td>
                                          </tr></thead>
                                          <tbody>
                                                .
                                                .
                                                .
                                          </tbody>
                                    </table>
                              </xsl:if>
                        </xsl:for-each>
                  </body>
            </html>
      </xsl:template>
</xsl:stylesheet>
```



<table>
<tr><td>simple_tag</td><td>simple_tag2</td><td>complex_tag</td></tr>
<tr><td>simpledata</td><td>simpledata2</td><td><table><tr><td>complex_tag2</td></tr><tr><td><table><tr><td>simple_tag3</td><td>simple_tag4</td></tr><tr><td>simpledata3</td><td>simpledata4</td></tr></table></td></tr></table></td></tr>
</table>

Figure 2.4: XSL Stylesheet Fragment and Resulting Web Page.

SQL:

```
select author
from catalog
where title = “Huckleberry Finn”
```

XQuery:

```
for $c in catalog
where $c.title = “Huckleberry Finn”
return $c.author
```

**Figure 2.5: Comparable SQL and XQuery Queries.**

Shanmugasundaram *et al.* (2001b) provide a good summary of the XQuery language. Using XQuery, discrete XML documents can even be joined together in the same way that relational tables can be joined with SQL (based on a primary key – foreign key equality). This enables full relational capabilities within the XML store.

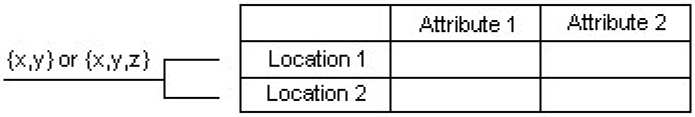
The first XML database capabilities simply allowed for retrieval of standard relational data as XML documents. Because many websites are driven by backend database systems, this was a huge benefit for being able to retrieve data from the database in a language of the internet. Following this capability for production as XML, database vendors began looking at ways to incorporate XML into the fundamental layers of the database. These capabilities were generally realized either by decomposing the tag-value pairs in the XML into relational data or by storing the complete XML in a character large object (CLOB), which is analogous to the binary large object (BLOB). Storing the XML wholesale, of course, meant that query of the XML data was next to impossible. The very nature of XML as semistructured also precludes simple decomposition to structured relational tables (Florescu and Kossman, 1999).

XML database systems come in many flavors, ranging from systems that can only store XML files in BLOB form and thus do not support querying, to database systems that map XML data to relational or object-relational structures (many database vendors currently offer this level of support, including Oracle, Microsoft SQLServer, and IBM DB/2) to pure XML databases such as Tamino (Schoning, 2001) , Lore (McHugh *et al.*, 1997) and TIMBER (Jagadish *et al.*, 2002).

XML has previously been used as the basis for representing spatial data (Smith *et al.*, 2001), temporal data (Marian *et al.*, 2001) and spatiotemporal data (Brinkhoff and Weitkämper, 2001), although none have considered XML for these domains specifically from the aspect of persistent semantic representation. XML also is the basis for the dialect GML, which has found its way into a number of commercial geographic information systems. The newest revision of GML (version 3.0) has some capabilities for assigning temporal attributes, and other approaches have been devised as well (Zipf and Krüger, 2001). However, these capabilities are still rudiumentary and specify only reference systems. Moreover, GML is designed for interchange (as are many dialects of XML) , not for persistence of large volumes of data.

The representation of time in geographic information systems has long been a challenge for geographic information science. The study of the representation of phenomena that change with time has a long history of research from authors such as Peuquet (1994, 2002), Langran (1992), Worboys (1998) and others. A recent proposal by Usery *et al.* (forthcoming), described initially in Usery (2000) seeks to unify much of the disparate work in representing temporal geographic data in a holistic framework that incorporates raster, vector and mathematical model forms.

Berry (1964) is a foundational work familiar to every student of geographic information science. Although it does not treat temporal data, the geographic matrix that it describes has been the basis for the georelational model that has completely dominated the industry. The georelational model has been so pervasive in part because it dovetails so precisely into the standard relational model. In the geographic matrix, a feature is composed of its attributes and its location. The intersection of these attributes and locations forms the geographic matrix (Figure 2.6).



| | Attribute 1 | Attribute 2 |
|---|---|---|
| Location 1 | | |
| Location 2 | | |

Figure 2.6: Geographic Matrix.

As with temporal database literature, early models for temporally extended spatial data focused on extending the georelational model. The difficulty with this extension to the georelational model (and theoretically to Berry's matrix) is that a geographic phenomenon may be variable in both attributes and location, but may still be the same feature (for example, a glacier is continuously moving slightly and may have a variable depth depending on snow melt). Berry's matrix does not account for this condition, precisely because it was not designed to deal with temporal conditions. In making space the fundamental identifier of a geographic feature, Berry's matrix ignores the fact that a geographic feature can change and evolve over time, and that in fact location may even be a parametric function of time.

More recent approaches have looked beyond the georelational model and its limitations. Wachowicz (1999) reviews object-oriented methods for spatiotemporal data modeling and presents the Spatiotemporal Data Model (STDM). The STDM draws upon the seminal work of Hägerstrand (1975) which developed the Time Geography framework. In Time Geography (and by extension in the STDM), time and space are integrated into a single holistic framework. Spatial and temporal dimensions are represented in terms of space-time paths in which an entity follows a path as it evolves in its spatial position in time. While conceptually powerful, Time Geography has been difficult to represent in traditional relational models. Wachowicz argues persuasively that an object-oriented design better captures the semantics of Time Geography and demonstrates its application in a SmallWorld GIS implementation.

A number of methods have been suggested to model time in spatial systems. These can be divided into absolute and relative views in which time and space are either explicitly or implicitly modeled and into space-dominant, time dominant or integrated space-time views, for a total of six possible models. Langran (1989, 1992) terms these varied methods dimensional dominance and presents a review of these. One of the salient conclusions that Langran reaches is that for GIS a spatial dominant approach is beneficial, particularly when temporal extensions are accomplished through attribute versioning (as opposed to tuple versioning).

The absolute-space viewpoint corresponds to the mechanism by which time is most readily implemented in current geographic information systems. In the space-dominant view, the distribution of space is principal, and the arrangement of that space over time is secondary. The model then becomes a series of partial or complete snapshots for each of the different times, arranged like layers in the GIS. This approach is most commonly taken in attempts to apply commercial GIS to the problem of temporality, because commercial GIS software supports these layers readily.

In contrast, the absolute-time viewpoint explicitly adds a dimension to represent time. While this approach has been successful in carrying forth two-dimensional GIS past the so-called 2 ½ dimensional approach and into the third dimension, it has severe limitations in modeling time. This is because time cannot be considered to be another spatial dimension, because time has its own set of constraints.

The absolute space-time view converges the absolute-space and absolute-time views into a holistic viewpoint. In the absolute space-time view, the mode of interest is not simply in a geographic phenomenon's positing or temporal existence, but rather the phenomenon's location in time. Change is shown along a timeline where location is an indexed function of time; the important feature then is not location in time or space, but rather the process that defines position in time and space. The TEMPEST system (Peuquet, 1994) was the first geographic information system to merge the absolute-space and absolute-time views.

Each of these absolute models has a relative analogue. The relative-space viewpoint de-emphasizes the absolute fixed position in space of a phenomenon and instead focuses upon the relative positions of features in space. This is an important viewpoint for spatial topological relationships. Likewise, the relative-time view de-emphasizes the absolute fixed position of time of a phenomenon and instead focuses upon the relative positions of features in time. In many cases, this relativity is that time is measured relative to some temporal order; for example of days of the week, which are relative to the week as a whole. Relative space-time emphasizes both the relationships of space and of time. An important fact to note about relative space-time is that it may not – and probably will not – be definable in a Euclidean geometry. Relative space-time reflects the relative position of objects in space and time, and because of the interplay between the two aspects, the phenomenon of interest may be grossly distorted. An overriding assumption of the relative space-time is that time and space cannot exist independently; indeed, they are interdependent.

Tversky and Taylor (1998) and Block (1998) both examine the cognitive aspects of temporality. Block describes a model of cognitive perception of temporality in which the importance of the directionality of the temporal arrow is stressed. Block argues that our own perception of the progress of time has dictated the direction that time takes. Block (1998) further argues for a hierarchical structure in which newer memories are superimposed on older memories also defines the directionality of the temporal arrow. Tversy and Taylor (1998) differ and suggest that language is a fundamental descriptor of both spatial and temporal thinking. According to the authors, these concepts are intrinsically cultural and thus attached to language; only through language are such concepts expressed.

Frank (1998) presents a taxonomy of spatiotemporal representation divided into linear and cyclical types; linear time is time that continues identifinitely while cyclical time is based on recurring temporal structures such as days, weeks or months (for example, Friday at 5 PM is payday is a cyclical temporal structure). Of these two types, each can be further subdivided into discrete and continuous structures. Discrete structures typically serve as events (something happened at a certain time) while continuous structures demarcate durations (e.g. between two times). Based on this taxonomy of time, time can be linearly evolving, as is the case with Time Geography and space/time lines, but this taxonomy also adequately describes branching time in which multiple possible outcomes may be possible from a single event. Branching time is a theoretical possibility, although not directly perceivable by us since the directionality of the time arrow dictates that we can only witness a single possible outcome to an event. Figure 2.7 illustrates this concept of branching time. Hazleton (1998) further describes branching time but with the additional complexity of bi-directional branching time. Whereas branching time implicitly asserts that multiple possible outcomes are possible from a single event, bi-directional branching further supposes that multiple events may lead to the same outcome. Figure 2.8 shows this bi-directional branching model.

A slightly different approach to understanding temporality is described in Christakos (2000). In this work on spatiotemporal geostatistics, Christakos seeks to understand not the modeling of spatiotemporal data, but the analysis. Christakos presents an excellent review of spatiotemporal geometry and argues that instead of mapping space and time to a 4-dimensional spatial structure (as is often done), that instead a metric should be derived in similar fashion to the Minkowski p-metric where dimensionality is mapped to an alternative structure. Christakos further suggests that accomplishing this will create a map between physical space/time and spatiotemporal events in the map; from these spatiotemporal events relationships can be constructed between events, as can more advanced constructs such as

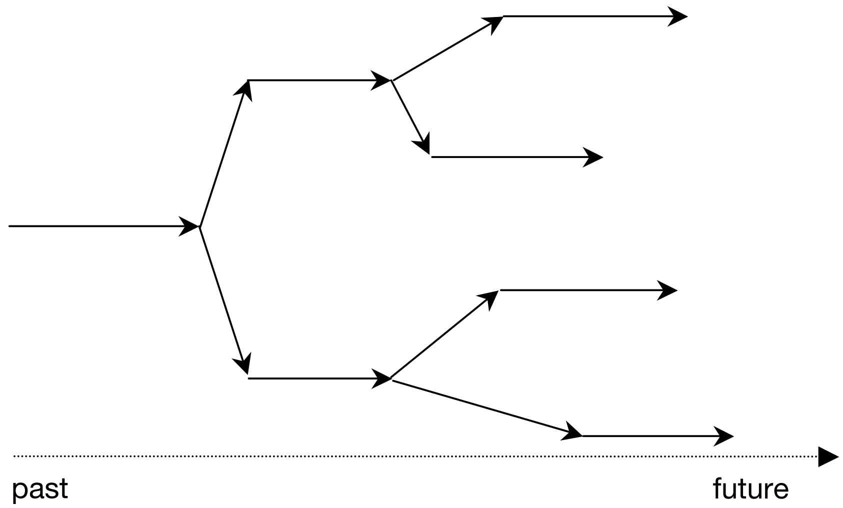


Figure 2.7: Branching time.

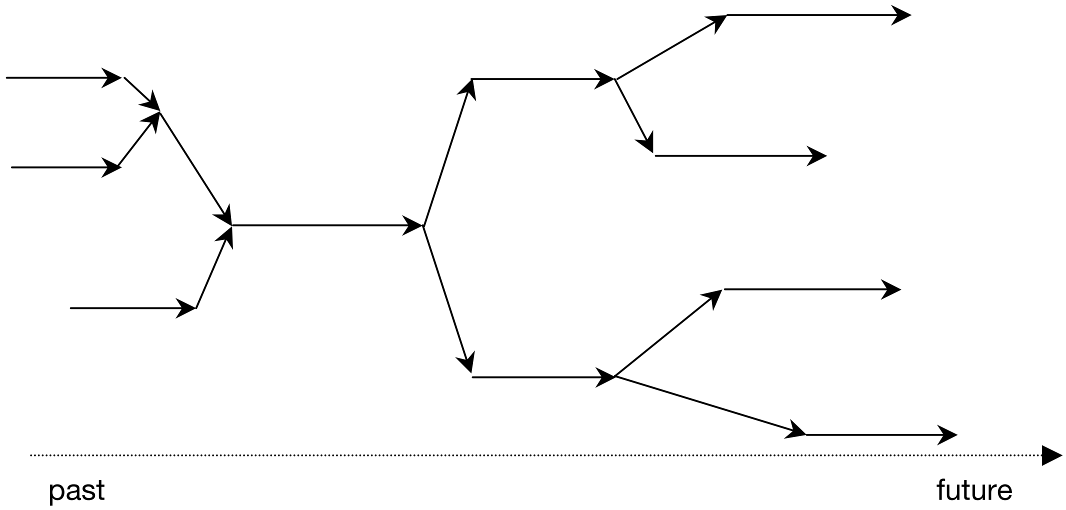


Figure 2.8: Bi-Directional Branching time.

spatiotemporal lines and areas. A spatiotemporal line corresponds roughly to that of the Time Geography of Hägerstrand, while spatiotemporal areas may serve as boundaries between epochs. Christakos also suggests that such spatiotemporal maps may be based on non-Euclidean geometries, an idea that occurs fairly often in the spatiotemporal literature.

Christakos *et al.* (2001) present an applied approach to modeling temporality based on raster data. In this work, Christakos again focuses on analytical capabilities but with particular applications to imagery and geostatistical surfaces such as those produced by Kriging. Yuan (2001) also examines an application of spatiotemporal modeling, in this case to weather analysis. Unlike Christakos, the model presented by Yuan is an integrated one with both raster and vector properties.

Usery (forthcoming) presents a unified model that models not only temporality but also multidimensional spatial features. In this model, a feature is modeled in terms of attributes and relationships between attributes. Using an object-oriented approach, an attribute may be of any of a number of complex types, all of which inherit from the attribute base class. Examples of attributes may be spatial types such as points, lines, polygons, rasters or geospatial types which are these types extended with their appropriate georeferencing information. Note that both vector and raster data are represented. Higher dimensional types may also be attributes (such as three-dimensional constructions) as may be both event and duration temporal values. Mathematical models may also be attributes that attach to a particular feature to describe complex processes such as a dispersing pollutant in water, which is characteristically difficult to describe simply in terms of coordinates but is better described by a mathematical model that predicts its dispersive pattern in time and space. Attributes may also be aggregations of multiple attributes (for example, an attribute may have a spatial component and a temporal component as aggregated attributes into a single, complex attribute). Relationships are similarly described as the relation between attributes and may be 1-to-1, 1-to-many or many-to-many. The advantage to this model is that it equally facilly describes both transaction and valid times and offers methods for either discrete or continuous times. The model may be defined cleanly using an object-oriented notation such as the Unified Modeling Language. However, the inheritance scheme is inherently hierarchical and will be difficult to persist in a relational database.

Geographic cognition has provided an important component to geographic representation. Cognitive geography is a heavily researched area as well with a growing body of literature that strives to understand ways in which the human mind deals with spatial data, temporalities and relativities. The recognition of the role that cognition plays in basic geographic understanding is driving research in a variety of disciplines and illustrating clearly that geography is a human science as well as a physical science.

The University Consortium of Geographic Information Science has identified cognition and geographic representation as long-term research challenges and there has been a commensurate history of geographic information science research in these areas (Goodchild, 1992; Couclelis and Gale, 1996; Nyerges, 1991; Molenaar, 1991; Frank, 1998b; McMaster, 1991; Egenhofer *et al.*, 1999; Mark, 1993). Peuquet in particular has been active in spatial data models and the role that geographic cognition plays in their development (Peuquet, 1984; Peuquet, 1988). The importance of image schematas and the interrelationship of geographic information science to cognitive theory has been explored as well (Kuhn and Frank, 1991; Frank and Raubal, 1998). Finally, the role of qualitative reasoning has been examined (Freska, 1991; Hernandez, 1993). The interplay of each of these apparently disparate research questions has directed a general expansion in fundamental understanding of geographic cognition and its role in representation.

An understanding of spatial and temporal cognition has also previously driven the development of data structures based on such cognitive processes. Habel and Eschenbach

(1997) provides an excellent interdisciplinary review of spatial and temporal cognition from  a cognitive science perspective integrating the literatures of both cognitive psychology and artificial intelligence.  The authors make and substantiate the argument that the current approach of cognitive semantics is one of metaphorical assignment.  Temporality is often thought of in terms of spatial properties; however, the authors argue that this approach is only partly correct and leaves glaring holes in the application of certain types of linguistic operations for one or the other of these domains.  Instead there is an underlying structure from which spatial and temporal structures are derived (analogously to inheritance in the object-oriented literature) and that shared behavior is inherited from this shared source.  Thus, although there is similarity, spatial data structures and temporal data structures should be handled (and modeled) fundamentally differently.

Feature-based approaches to GIS are semantically rich data structures that are generally now regarded as the current state-of-the-art.  Geographic data models in the past have been based largely on associating attributes with geometry, while neglecting other properties of the data.  In particular, these previous approaches have neglected the semantics of spatial data.  Tang *et al.* (1996) presents a feature-based model derived from object-oriented methodologies for vector data while Usery (1996) presents a similar model for raster data.  In such feature-based approaches to modeling geographic features, the representation in the database corresponds to real-world features rather than simply to stored geometry.  This model is more difficult to represent in relational database systems and may require multiple tuples for the same feature (for multipart or multitemporal features), but is semantically superior to previous approaches.  Using extended relational models or object-oriented models, composition may be applied to organize the feature as one complete unit in the database. Virtually all modern geographic information systems represent data in this way.

Freska (1997) also addresses this issue of the structure of representation and argues that while physics, mathematics and other “hard” sciences represent space and time as fundamentally continuous and mapped over a four-dimensional space ($\Re^4$), spatial and temporal data structures fundamentally decompose into qualitative structures. It is, Freska asserts, these qualitative structures that allow fundamental comparison operations to occur naturally. Thus the similarities between the spatial and temporal domains are qualitative rather than quantitative and semantic mappings between the two should be accomplished at this qualitative level.

There is a great deal of research in the cognitive geography and cognitive psychology literatures supporting hierarchical spatial reasoning. Hierarchical organization of the data facilitates cognitive economy and minimization of a search space in order to retrieve data. With hierarchical spatial reasoning, levels of detail are swapped in and out of working memory in order to deal with the most appropriate amount of information. Bisseret and Montarnal (1996) examines the dichotomy between network and hierarchy-based viewpoints. This work found that in fact hierarchical reasoning was the predominant mechanism in wayfinding tasks. The authors suggested that in cases where a starting point for a wayfinding task was imposed a network viewpoint predominated but was still superseded at various points by a hierarchical perspective. They concluded that network (or tour) viewpoints only prevail in cases where a specific task structures the cognitive map and that the hierarchical view is the fundamental spatial representation.

Rinck, *et al.* (1997) examined response times for subjects in retrieval of spatial information from memory and observed that retrieval was categorical rather than geometric. Accessibility of information depended upon the number of intervening objects between a start and end point rather than purely the distance between the objects. In some cases, Euclidean distance was observed to be used secondarily when additional information was necessary.

This work further substantiates the hierarchical organization of data; if only geometric information were relevant than accessibility times should linearly increase with distance along the path, which was demonstrated not to be the case. Newcombe, *et al.* (1999), however, disputes the nonmetric nature of spatial representation by arguing that systematic distortions in representations can be explained through hierarchical categorization whereby there may be uncertainty at each category but not a distortion and that distortion is only introduced at the superset of hierarchies when differences in uncertainties are combined. While the operational mechanism is unclear, it is clear that the brain most likely does not store spatial representations in terms of physical Euclidean geometry.

Hierarchical spatial reasoning theories suggest that humans partition space into a number of subspaces of varying resolution (Car and Frank, 1994, Papadias *et al.*, 1996, Car, 1998). Such subspaces collapse and expand as the level of detail required changes. Such hierarchical patterns are evident throughout the current theories of human cognition (Anderson, 2000) and suggest that there is some merit to this. Hierarchical spatial reasoning also matches well with the findings of Freundschuh and Egenhofer (1997) that there are six categories of space – manipulable object space, non-manipulable object space, environmental space, geographic space, panoramic space and map space – in their theory of naïve geography, which describes how people spontaneously think about space. Hierarchical spatial reasoning does present some computational challenges (Papadias and Egenhofer, 1997), but also has a number of benefits for organization of data and filtering appropriately to the level of detail required for the particular task. Hierarchical spatial reasoning also offers a mechanism for cognitive organization of spatial databases (Rajabifard *et al.*, 2000) that expands well to fit with both earlier (Usery, 1996) and contemporary (Mennis *et al.*, 2000) research into incorporating such cognitive principles into spatial databases. Given the observation by Hirtle (1998) that GIS could in fact serve as an analog for memory, it becomes clear that incorporating cognitive principles – and in particular, hierarchical spatial reasoning which maps well to the human methodology for navigation – is key to being able to replicate such cognitive navigation capabilities in a machine system.

In Car *et al.* (2001), the authors examine organization of spatial data in terms of a hierarchical graph structure with graphs and subgraphs. The authors presented tasks to both humans and machines to assess the ability to perform wayfinding, but only allowed a subset of spatial information to be accessed at any single time.  Car *et al.* (2001) reported that performance was better using this hierarchical organization than using non-hierarchical structures and suggest that benefits increase as the number of nodes in the network increase, particularly with regards to problem-solving.

Graham, *et al.* (2000) further substantiates the hierarchical nature of spatial representation and its particular applications to problem solving.  In this work, subjects were presented with a traveling salesman problem (a Hamiltonian cycle in which each node in a network must be visited once and in which the cycle must start and end on the same node). The authors found that comparisons of typical artificial intelligence techniques did not suitably model response times.  A solution for the problem based upon a hierarchical structure instead modeled well the performance times of the subjects, indicating the hierarchical nature of problem solving as well as representation.

An interesting examination of the relationship between space and time was reported by Boroditsky (2000).  In this study, structuring of conceptualizations of time and space were compared with the consideration that space was concrete while time was abstract.  There appears to be a mapping between space and time in effect, and the author reported that spatial structures could in fact be used to think about temporal information.  A remarkable finding in this study, however, was that given enough consideration of a spatial representation, it could actually be stored as a temporal representation, obviating the need for access to spatial schemas. This finding could suggest a mapping from three dimensional space to a single dimensional timeline as a method for cognitive efficiency and may explain why wayfinding tasks and distances are sometimes structured in terms of time along a path.

Category theory relates to hierarchical spatial reasoning and has as one of its products in geographic information science the feature-based model. Like hierarchical spatial reasoning, category theory also has concerns with issues of cognitive economy. However, where hierarchical spatial reasoning is more concerned with the implicit nesting and ordering of spatial phenomena (such as nested political units like counties, states and countries), category theory applies the hierarchical approach to the attribute aspect of the geographic feature. In the cognitive literature, the notion of a prototype has been well studied (Rosch, 1978). A prototype is the most general instance of a thing; for example, identification of a feature as a table has a semantic value that exceeds identification of the feature as furniture, while identification of that feature of a coffee table adds considerably less semantic value. In category theory, the prototype level of representation is also the most cognitively efficient level and is thus the starting point for a cognitive process. As more detailed knowledge is required, the cognitive process can access continually expanding (or deeper) levels of representation from the original, shallow representation. Thus, as in hierarchical spatial reasoning, category theory is concerned with the cognitive economy of a particular representation. Usery (1996) has explored the application of category theory to geographic information science.

The value of hierarchical approaches have long been appreciated by database researchers. Such hierarchical methods are the foundation of database indexing techniques that allow for efficient access to stored data. In the traditional database literature, hierarchical tree database structures such as B-Trees and $B^{+}$-Trees have been used to implement such indexing (Elmasri and Navathe, 2000). There are a wide variety of tree structures, each with different purposes and design goals; however, there are commonalities between them.

Generically speaking, every tree structure is a hierarchical representation that is a specialization of a directed graph (Lewis and Denenberg, 1991). As with a directed graph, a tree is comprised of nodes and edges, where each pair of nodes is connected to an edge and the tail is at the node from which the edge originates while the head is at the node to which the edge terminates. Each node has exactly one parent (ancestor), but may have many children (descendants). Also, each tree has a single node as its root, or highest level. A node that has no children is called a leaf node, and the distribution of leaf nodes is one of the defining characteristics of the type of tree.

Two properties of trees that are of particular importance to the design of efficient algorithms are the height of the tree and the depth of its nodes. The height of a node is defined as the number of edges between the node and its farthest descendant; the depth of the node describes the number of edges from the node to the root of the tree. The height of the tree is the largest depth of the nodes of the tree. The height of the tree dictates the maximum time required to search through the tree for a particular node; the greater the height of the tree the longer the search that is required. Because of this, minimizing the height of the tree is of primary interest in the design of tree structures. Figure 2.9 illustrates these concepts.

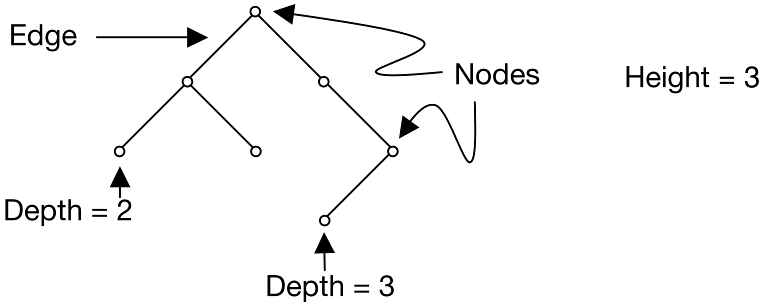


**Figure 2.9: Tree Structure Terminology.**

Spatial data structures are also based on tree structures and hierarchically organize space. Application of spatial data structures allows for efficient searching of space by recursively decomposing the space to be searched. Examples of spatial data structures may be found in Lewis and Denenberg (1991) and Samet (1990a and 1990b); two examples are the k-d tree which multidimensionally decomposes the dimensional domains and the Quadtree which recursively tessellates space. The principal difference between the two is that in the Quadtree, space is uniformly distributed while in the k-d tree, the dimensional domain is uniformly distributed. Figures 2.10 and 2.11 illustrate these two fundamental spatial data structures.

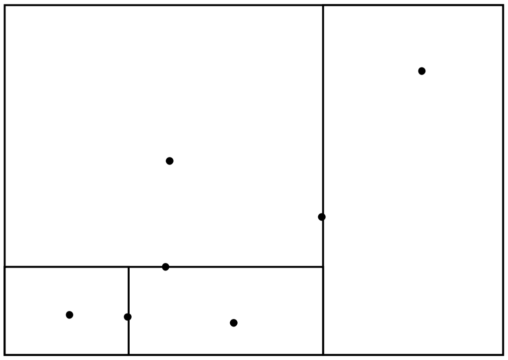


Figure 2.10: k-d Search Tree (k=2).

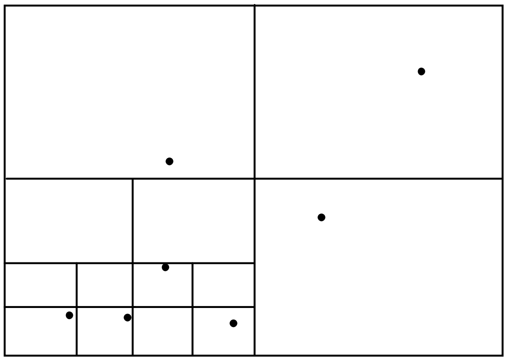


Figure 2.11: Quadtree.

Other examples of spatial data structures can be found in a long history of work. Of these, the most widely applied are the R-Tree family of data structures. The R-Tree has been in the literature for nearly twenty years (Guttman, 1984), while the $R^+$-Tree (Sellis *et al.*, 1987) and $R^*$-Tree (Beckmann *et al.*, 1990) have been more recently developed. At with the k-d tree structures, the R-Tree structures are designed analogously to tessellate the frequency distribution of features in geometric space. The R-Tree class spatial data structures are the most frequently spatial data structures in commercially available spatial database systems.

As with thematic and spatial indexing, various temporal indices have been proposed that are based on hierarchical structures. It should be noted that these indices typically rely on a database schema where versioning is at the tuple level and not at the attribute level – that is, a change in a record requires a whole new row and not just an updated attribute. This is in keeping with standard approaches to temporal relational models. Elmasri *et al.* (1993) proposed one of the earliest of these, the Time Index, in which, indexing is done over the single dimension of valid time ranges. The Time Index has the drawback that it is space inefficient, so various other approaches have been proposed as well. The TP-Index (Shen, *et al.*, 1994) is another early method for temporal indexing that instead maps the temporal range into two-dimensional space and uses spatial indexing methods to accomplish searches. The authors, however, note that while this method is more space efficient than the Time Index, it has the drawback that it is biased toward certain types of queries; for non-preferred queries the TP-Index provides near-worst case performance. An improvement over the TP-Index is the $B^+$-Tree based TP-Index, which performs ordering over the two-dimensional space and maps the results into a $B^+$-Tree (Goh, *et al.*, 1996). These temporal indices all have the property that records in the database do not need to be in sorted temporal order, a requirement for indexing valid time, which may have updates that are out of temporal sequence. There are several proposed temporal indices that do require sorted order and thus are only useful for indexing transaction time in the database; an example of these is the Time-Split B-Tree (Lomet and Salzberg, 1993). Nascimento and Dunham offer one of the most recent approaches to indexing valid time using the standard $B^+$-Tree data structure (Nascimento and Dunham, 1999). As with spatial data structures, the various proposed temporal data structures tessellate time into evenly distributed temporal units (such as does the Quadtree for spatial indexing) or tessellate time into even distributions in the frequency domain of the indexed valid time values (analogously to the R-Tree data structure).

The corollary to considering spatial and temporal indices separately, of course, is to consider a combined spatiotemporal index. A number of proposals for spatiotemporal indices have been made to date (Theodoris *et al.*, 1998, Nascimento and Silva, 1998) that add time as an additional dimension to an existing spatial index. Saltenis and Jensen (2002) argue that such approaches are not ideal. According to the authors, the addition of time as a spatial dimension does not take advantage of the distinct properties of time as they differ from space and the combination of purely temporal and purely spatial indices is simply not effective. The authors propose an extended $R^*$-Tree, which is a spatial index, with two temporal dimensions for valid time and transaction time (with the distinction that these new dimensions are temporal in nature and not spatial as many multidimensional index structures are). This $R^{ST}$-Tree spatiotemporal index is one of the only such to support indexing over both valid time and transaction time. As with the purely temporal indices, these spatiotemporal indices are also all hierarchically based (and often based on extensions to existing spatial indices).

Persistence of spatiotemporal data requires a confluence of academic disciplines. Geographic information science has expanded understanding of spatiotemporal representation, particularly from a cognitive viewpoint, and has generated a wide range of approaches to such representation. Database research in spatial, temporal and semistructured data representation have also offered new and exciting ways to persist such data in a database.  The possibility for management of spatiotemporal data incommercial and corporate geographic information systems has never been closer to realization.

# CHAPTER 3

## METHODOLOGICAL DEVELOPMENT

We begin by constructing a non-extended relational model for representing spatiotemporal vector data and by identifying the desirable features for this model. The reason that we chose a non-extended relational model over an extended model with all of its desirable characteristics is to enable a like-as-like comparison between the relational model thus constructed here and the XML-based model derived as part of this work. Extended relational models with spatial constructs have a number of advantages that overcome the limitations of the standard relational model including mechanisms for better descriptions of spatial data and operations. However, as has been shown, similar attempts to engender the same benefits for temporal data and for spatiotemporal data have not been as successful. This work will show that not only can spatial constructs be represented in an XML model, but also temporal and spatiotemporal constructs.

Based on the standard relational model, all data must be represented in terms of tuples and relationships between tuples. Considering only spatial characteristics, we turn our attention first to representing the simplest spatial primitive – the point – in terms of the relational model. The point is the easiest to represent in the relational model because there is a 1-to-1 mapping between it and the geographic feature. A generalized relational model for point data is presented in Figure 3.1. This model is similar to the georelational model commonly used in geographic information systems but without the use of specialized spatial structures for comparative purposes.

```
Point Feature
ATTRIBUTE 1
ATTRIBUTE 2
.
.
.
ATTRIBUTE n
X (or POINT_X)
Y (or POINT_Y)
```

**Figure 3.1: Point Feature Schema.**

For the point feature, each of the attributes is listed as attribute 1-n and may correspond to any data about the point location depending on the domain of the representation. By way of an example, the point feature might represent a sample location for a soil analysis. Each of the attributes may in turn represent data about that location such as the soil pH, potassium content, etc. Note that even at this simple representation, some semantic difficulties are already present. The actual point location is divided into two distinct attributes called x and y for the Cartesian coordinates of the point feature. There is semantically no connection between these attributes (save for the understanding that the professional geographer brings to the problem). An alternative that does more explicitly represent the connection is to label these attributes as point_x and point_y to better indicate that these two attributes are part of the greater whole. We grant that extended relational models, particularly those based on an object-relational framework, have already accomplished this connection through the use of abstract data types where the attribute may be called point or shape and have two corresponding nested attributes x and y. However, in a non-extended relational model such as we are considering here, this is not the case.

Representation of the next spatial primitive – the line – adds a layer of complexity to our model. The line is composed of many points arranged in a linear fashion. Because there as a

1-to-1 correspondence between the line feature (for example a river) and its attributes (flow rate, temperature, etc.) we wish to model the feature in the relational model in such a fashion. However, the 1-to-many relationship between the feature and the vertices that comprise the line renders this difficult to accomplish in a single table. If the maximum possible number of vertices were known a priori, then attributes numbering that number of vertices for each of the x and y coordinates could be accomplished. Alternatively, a single line attribute could be defined with the contents of the line field containing a delimited list of attributes, for example of the form $x_1,y_1;\ldots;x_n,y_n$. The drawback to the first approach is the wasted space if the number of actual vertices does not equal the projected number of vertices. The drawback to the second approach of course is that the data itself now describes the structure of the data rather than the schema of the model.

A third alternative to modeling such a 1-to-many relationship is through the use of a related table to maintain the vertex data for the line. This semantically the most clear and is the most space efficient of the designs presented thus far. In this approach, any summary data about the line as a whole is maintained in a line feature relational table while individual vertex information is stored in a related line vertex table. The relationship is maintained on the basis of a unique identifier such as the feature id presented earlier. Note the lack of semantic clarity imposed by storing summary shape features such as line length in the line feature table while storing individual vertex information in the related table. Also required for such a related schema is the use of a sequence attribute in the vertices table to indicate the relative ordering of vertices since the relational model does not guarantee tuples are stored or returned in a particular order. Figure 3.2 illustrates these three alternatives.

Modeling polygon features is similar to modeling linear features in that they are collections of vertices as well. The principal difference between polygonal and linear features is that polygonal features are closed, so that the last vertex is equal to the first vertex and the

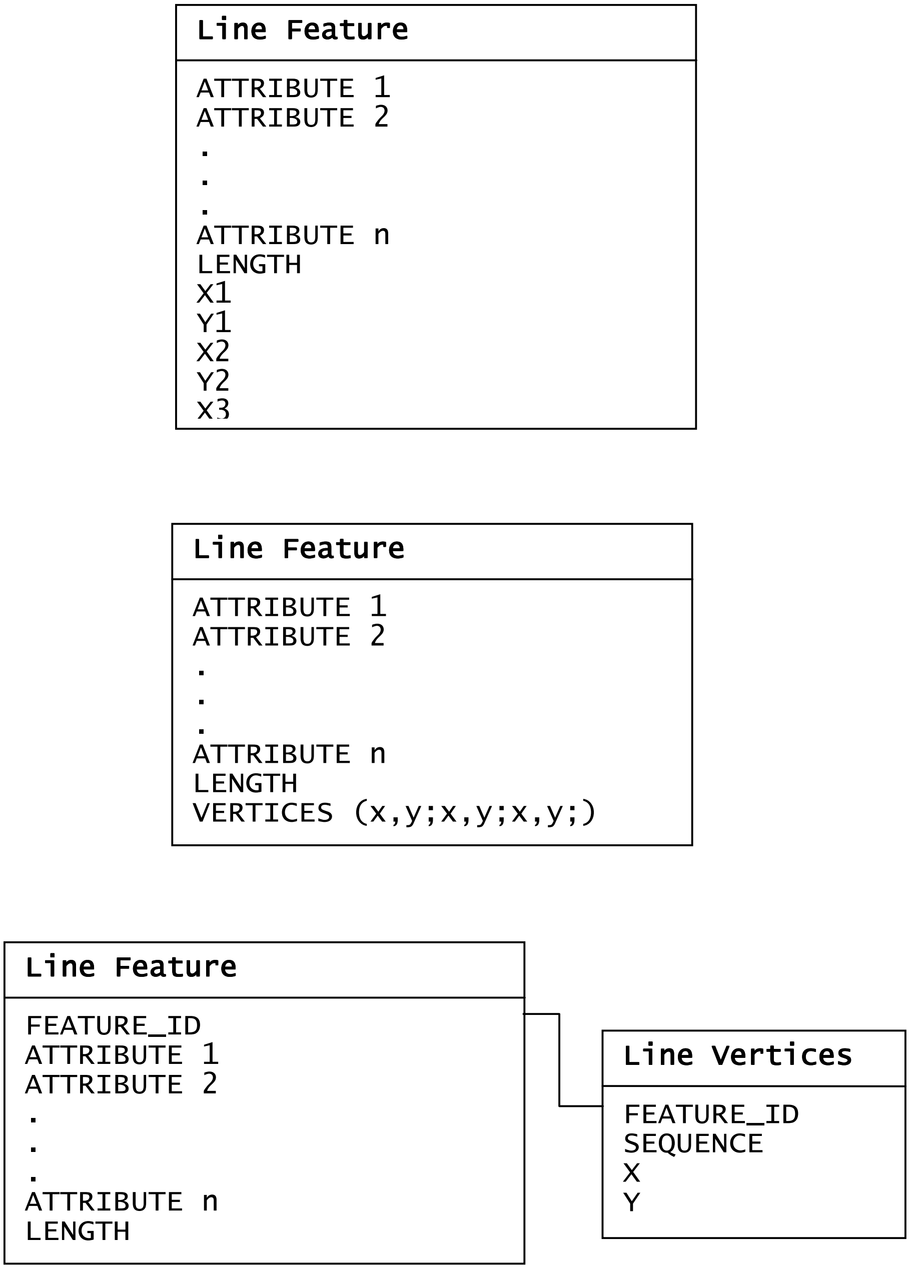


Figure 3.2: Three Alternative Line Feature Schemas.

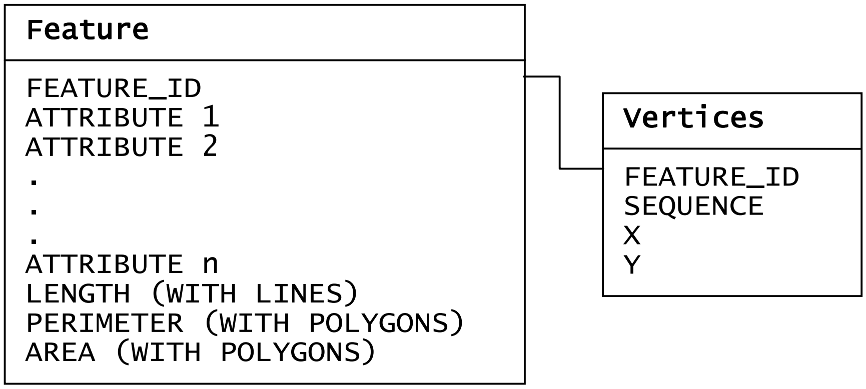


**Figure 3.3: Generalized Feature Schema.**

different summary shape variables (perimeter is equivalent to length but applies to areal features and such areal features also have a computed area associated with them). The choice of whether or not to explicitly include the final coordinate is a choice left to the data modeler. Summary shape attributes are stored in the feature table much like the length attribute in the line feature table.

More complex geometric shapes are possible by aggregating these primitives; examples of these might include networked features as collections of linear features, regions as collections of areal features or higher dimensional (three dimensional) features. These complex features may be constructed through further related detail tables to the summary tables and are not presented here in detail.

We maintain that of the three options in Figure 3.2, the third is the semantically clearest, with the related vertex table. However, this creates a departure from the scheme devised to maintain the point feature and prevents generalization. An alternative point structure might be through the use also of a related table that contains its single vertex as well. This would allow us to devise a more general relational structure for all spatial features whether 0, 1 or 2 dimensional. Figure 3.3 presents this general spatial representation. Note that the presence of length, perimeter or area attributes is dependent upon the geometric type. We maintain that this does not affect the clarity of the model as these can be treated as all other attributes associated with the feature. In the case of modeling of points, there would be a 1-to-1 correspondence between the feature and its associated vertex.

Construction of relationships is straightforward here as it is in the general relational model. Maintenance of primary key – foreign key pairs in the source and related table will adequately maintain the relationship between features and additional data. The feature – vertex relationship is one example where the feature identifier in the feature table serves as the primary key while the feature identifier in the vertex table is the foreign key.

It can be argued that this exercise is not necessary; the current existence of extended relational models that are more efficient and more semantically elegant renders it extraneous. However, we have gone through this exercise because it does not require the use of specialized software such as Oracle Spatial or ArcSDE and would be accessible to any enterprise wishing to model such data. It also provides a better foundation for illustrating the benefits of XML based persistence of the same data.

In this research we have taken liberties to ignore some facets of temporal data for the sake of simplicity. For example, we treat only uni-temporal time – either valid or transaction time, but in this work valid time. We will discuss implications for bi-temporality in the results section of this work. We also do not consider the distinction between the various types of time (linear, cyclical, discrete or continuous) and address only linear time with durations. We will further address these distinctions as well in the results.

To extend our simple generalized spatial schema to incorporate valid time as durations, we first add two fields to the model for a birth date and a retirement date, which we will refer to as BORN and RETIRED, respectively. This corresponds to tuple-level versioning and is the simplest to accomplish in a relational model. Attribute versioning cannot be easily accomplished in a relational database design without the use of complex structures. The disadvantage to tuple-level versioning is immediately obvious: every time an attribute is updated, a new tuple with new BORN and RETIRED attributes must be generated. This is very space-inefficient.

In order to properly model also the related vertices, each vertex must also have an associated BORN and RETIRED attribute. This is necessary since geometric updates are possible in our model, and if we did not attach temporal attributes we would not know which vertex was associated with which time. This is a kind of attribute-level versioning since we can update vertex values without needing additional tuples in the feature table (with the exception of the fact that changes to vertices would change summary shape attributes such as length or area). Figure 3.4 illustrates this revised schema incorporating temporality into our generalized spatial model.

Figure 3.5 shows a simple polygon feature such as a parcel for purposes of dissecting this model. This polygon feature has four vertices and an additional attribute in addition to FEATURE_ID, BORN and RETIRED attributes. To this polygon feature are made two changes;

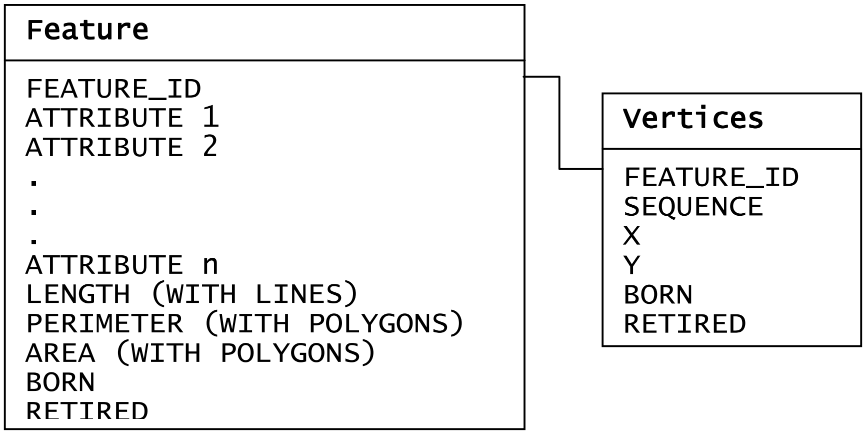
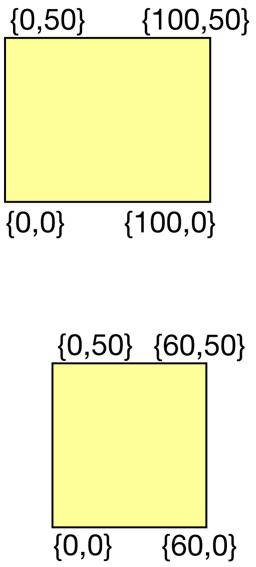


**Figure 3.4: Generalized Spatiotemporal Schema.**

the first is a spatial coordinate update while the second updates the aspatial attribute. We examine the storage requirements in our relational model for persisting this feature and its versions.

Note first that maintaining the versions of this feature requires three tuples; one for each of the original state and two successive changes. Because each of these tuples has the same the feature identifier (because they are the same feature), we cannot use simply the feature identifier attribute as the primary key for the table. Instead, we have a complex primary key consisting of FEATURE_ID, BORN and RETIRED attributes. Note also the space that is wasted by duplicating attributes in successive tuples where only a subset of attributes changed. This is the value of attribute-level versioning, but as stated previously, it is difficult to accomplish in relational models without complex relationships. Because multiple tuples are required for this single feature, semantic clarity is also sacrificed; requiring the user to make the association that these are different versions of the same feature and to scrutinize each of the tuples to discern the differences.

Note also that we have duplication in the vertex list as well. Even though we can accomplish attribute-level versioning with individual vertices, the necessity to maintain the sequence number requires that we regenerate the list for each iteration. Thus, although there are only four vertices – and only two changed – we must regenerate the entire list. We cannot be assured that a vertex has not been inserted into or removed from the list, changing the relative placement of the vertex in the list. Also note that the valid times for the vertices are not exactly the same as for the feature itself. In this case, the attribute update in the third state updates the tuple but does not require a commensurate change in the vertex list. Therefore, the relationship cannot be based solely upon the primary key established in the feature table of FEATURE_ID, BORN and RETIRED because the BORN and RETIRED values may be different in the related table. This problem only pertains to temporal relationships; simple relationships



STATE 1:
FEATURE_ID: 1
BORN: 1/1/1999
RETIRED: 1/1/2000
VALUE: $30000
AREA: 5000
PERIMETER: 300

STATE 2:
FEATURE_ID: 1
BORN: 1/1/2000
RETIRED: 2/1/2000
VALUE: $30000
AREA: 3000
PERIMETER: 220

STATE 3:
FEATURE_ID: 1
BORN: 2/1/2000
RETIRED: 1/1/2001
VALUE: $20000
AREA: 3000
PERIMETER: 220

Feature Table

| FEATURE_ID | BORN | RETIRED | VALUE | AREA | PERIMETER |
|---|---|---|---|---|---|
| 1 | 1/1/1999 | 1/1/2000 | 30000 | 5000 | 300 |
| 1 | 1/1/2000 | 2/1/2000 | 30000 | 3000 | 220 |
| 1 | 2/1/2000 | 1/1/2001 | 20000 | 3000 | 220 |

Vertex Table

| FEATURE_ID | BORN | RETIRED | X | Y | SEQUENCE |
|---|---|---|---|---|---|
| 1 | 1/1/1999 | 1/1/2000 | 0 | 0 | 1 |
| 1 | 1/1/1999 | 1/1/2000 | 100 | 0 | 2 |
| 1 | 1/1/1999 | 1/1/2000 | 100 | 50 | 3 |
| 1 | 1/1/1999 | 1/1/2000 | 0 | 50 | 4 |
| 1 | 1/1/2000 | 1/1/2001 | 0 | 0 | 1 |
| 1 | 1/1/2000 | 1/1/2001 | 60 | 0 | 2 |
| 1 | 1/1/2000 | 1/1/2001 | 60 | 50 | 3 |
| 1 | 1/1/2000 | 1/1/2001 | 0 | 50 | 4 |

**Figure 3.5: Temporal Evolution of Simple Parcel Polygon.**

based on a simple primary – foreign key relationship are not affected by this. The net result, however, is that relationships with temporal attributes are complicated and cannot be accomplished simply through a join operation. Instead, this relationship must be accomplished by first selecting the appropriate tuple from the source table and building a query from the BORN and RETIRED attributes to query against the related table. We clearly have sacrificed considerable semantic clarity here.

We have approached our development of the XML-based persistence model by considering the multidimensional model of Usery *et al.* (forthcoming). In this model, features are comprised of attributes and relationships. Attributes may be simple or complex and composed of primitives or other attributes; relationships may be between features or between parts of features. We have not, however, wholesale adopted this model. Whereas the model of Usery *et al.* is space-time dominant, so that space and time have equal dimensional dominance, our persistence model is space-dominant with attribute-level versioning (the approach recommended by Langran, 1992 for GIS applications). Temporality is attached to model attributes as components of the attribute. In terms of the language of XML, an attribute corresponds to an element delimited by a pair of tags. Temporality is modeled by attributes on the tag. Figure 3.6 depicts an example model attribute and the location of attached temporality.

```
<cost born=”1/1/1999” retired=”1/1/2000”>
      30000
</cost>
```

**Figure 3.6: XML Attribute with Temporality.**

XML also supports the creation of nested attributes as we were unable to do with a non-extended relational model, so that we can construct a shape feature that supports the full semantics of the shape. Using the same mechanism for temporality, we can alternatively either attach temporality to the entire shape feature or to individual components of the shape. We have chosen to attach temporality to the entire shape since changes to components of the shape will change the shape as a whole. We have modeled shape components as being comprised of as 1 or more coordinates, each of which in turn has x and y values. Line shapes have an additional length component while polygon shapes have additional perimeter and area components. Although envelope values have not been discussed yet (or considered) in the relational model, the XML shape also contains an envelope with minimum and maximum x and y values. Figure 3.7 illustrates this complex shape structure by way of an example.

Features are constructed analogously to the multidimensional model of Usery *et al.* by aggregating collections of attributes (or tags) – be they simple attributes or complex attributes such as shapes. XML does not prohibit multiple occurrences of a tag (or an attribute in our language), so multiple tags with the same name but differing BORN and RETIRED dates allow for attribute versioning. The XSD schema language allows the designer of the XML document to manage whether single or multiple instances of a tag are allowed and how many are allowed. Thus, although there is flexibility in multiple versions of a tag, the document designer can still maintain control. The feature identifier is also treated as an attribute of the feature tag, because it is not truly an attribute of the feature but rather is a meta-attribute. This attribute is not part of the real-world feature, but is an implicit part of the database. For semantic reasons, it is therefore not treated as an equal to the rest of the feature attributes.

Figure 3.8 illustrates the XML document to persist the parcel polygon of Figure 3.5 with all of its versions. Figure 3.9 shows this XML fragment as viewed through a popular XML editor, XMLSpy. XML editors further improve the clarity of the document by arranging it in a familiar tabular format. Fragments of the XML document can be accessed by further opening hierarchical links in the document. Even in text form, the relationship of various aspects of the versions of the feature are clearer than in the relational design. In tabular form, the real advantage is understood.

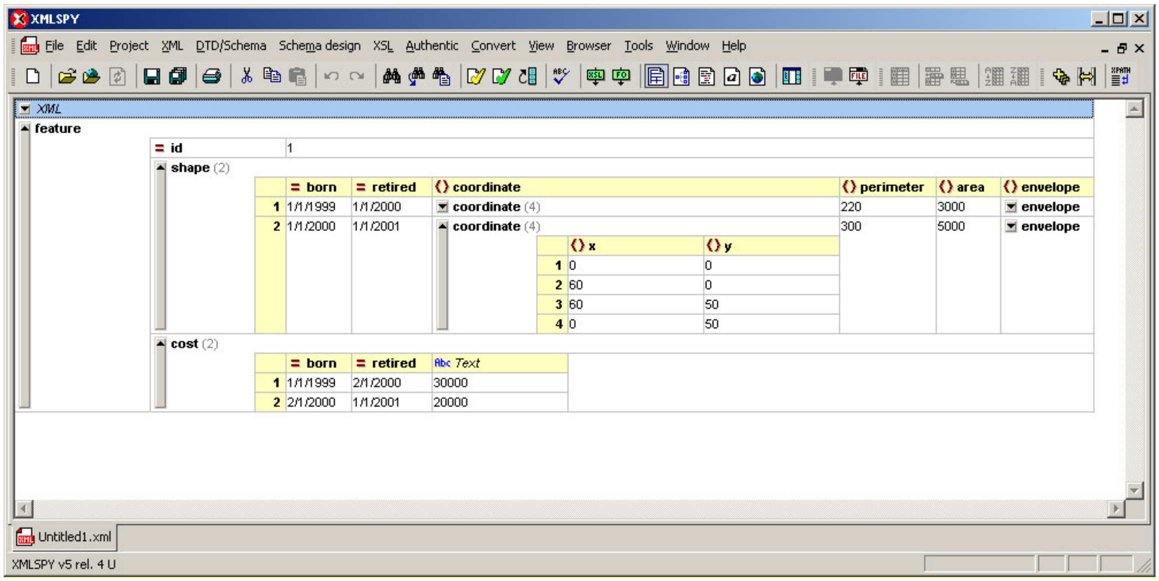
```
<shape born=”1/1/1999” retired=”1/1/2000”>
      <coordinate>
            <x>0</x>
            <y>0</y>
      </coordinate>
      <coordinate>
            <x>100</x>
            <y>0</y>
      </coordinate>
      <coordinate>
            <x>100</x>
            <y>50</y>
      </coordinate>
      <coordinate>
            <x>0</x>
            <y>50</y>
      </coordinate>
      <perimeter>300</perimeter>
      <area>5000</area>
      <envelope>
            <xmin>0</xmin>
            <ymin>0</ymin>
            <xmax>100</xmax>
            <ymax>50</ymax>
      </envelope>
</shape>
```

**Figure 3.7: XML Shape Structure.**

At the present, relating to other tables is as difficult in XML as it is in relational design. This is due to the immaturity of a standard that will ultimately simplify this process – the still-incomplete XQuery specification for querying XML documents. The XQuery specification will further advance the capabilities of the language for persistence in data stores and will allow for query in a similar way to the SQL standard for relational databases. Moreover, XQuery provides a way to link or relate together two XML fragments, a process that is at present still difficult.

```
<feature id=”1”>
      <shape born=”1/1/1999” retired=”1/1/2000”>
            <coordinate>
                  <x>0</x>
                  <y>0</y>
            </coordinate>
            <coordinate>
                  <x>100</x>
                  <y>0</y>
            </coordinate>
            <coordinate>
                  <x>100</x>
                  <y>50</y>
            </coordinate>
            <coordinate>
                  <x>0</x>
                  <y>50</y>
            </coordinate>
            <perimeter>220</perimeter>
            <area>3000</area>
            <envelope>
                  <xmin>0</xmin>
                  <ymin>0</ymin>
                  <xmax>100</xmax>
                  <ymax>50</ymax>
            </envelope>
      </shape>
      <shape born=”1/1/2000” retired=”1/1/2001”>
            <coordinate>
                  <x>0</x>
                  <y>0</y>
            </coordinate>
            <coordinate>
                  <x>60</x>
                  <y>0</y>
            </coordinate>
            <coordinate>
                  <x>60</x>
                  <y>50</y>
            </coordinate>
            <coordinate>
                  <x>0</x>
                  <y>50</y>
            </coordinate>
            <perimeter>300</perimeter>
            <area>5000</area>
            <envelope>
                  <xmin>0</xmin>
                  <ymin>0</ymin>
                  <xmax>60</xmax>
                  <ymax>50</ymax>
            </envelope>
      </shape>
      <cost born=”1/1/1999” retired=”2/1/2000”>30000</cost>
      <cost born=”2/1/2000” retired=”1/1/2001”>20000</cost>
</feature>
```

**Figure 3.8: XML Representation of Simple Parcel Polygon.**



feature

| id | 1 |
|---|---|

shape (2)

| | born | retired | coordinate | perimeter | area | envelope |
|---|---|---|---|---|---|---|
| 1 | 1/1/1999 | 1/1/2000 | coordinate (4) | 220 | 3000 | envelope |
| 2 | 1/1/2000 | 1/1/2001 | coordinate (4) | 300 | 5000 | envelope |

| | x | y |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 60 | 0 |
| 3 | 60 | 50 |
| 4 | 0 | 50 |

cost (2)

| | born | retired | Text |
|---|---|---|---|
| 1 | 1/1/1999 | 2/1/2000 | 30000 |
| 2 | 2/1/2000 | 1/1/2001 | 20000 |

Figure 3.9: XMLSpy View of Simple Parcel Polygon.

One other aspect of the XML language that is complete, but has not as yet found its way into current Document Object Model implementations is the ability to compare dates using the XPath language. At present, date comparisons are compared as strings – thus “12/1/2000” is less than “2/1/2000”, even though it obviously it is not. The XPath version 2.0 language has addressed this, but at present most implementations are still based on XPath 1.0. In this research, this was overcome by instead representing the date in decreasing hierarchy – year, month, date – so that instead, 20001201 is greater than 20000201 as it should be. This is a temporary drawback that does indeed affect the clarity of the representation, but one that is in the process of being corrected.

Because this persistence model does not use Time Geography as its basis, the question of spatiotemporal geometry has not been addressed. In particular, the question of analytical capabilities is not dealt with here. Examination the implication of XML based persistence and development of analytical capabilities would require a more explicit treatment of spatiotemporal geometry.

Even with the relative immaturity of the XML language and standards for persistence, this persistence model provides a basic foundation for storage of spatiotemporal data in a semantically meaningful way.  As research and development improves, XML will continue to advance as a viable platform for long-term storage of such spatiotemporal data without sacrificing the clarity of the representation.

# CHAPTER 4

# PROTOTYPE DEVELOPMENT

The development of theory is fundamental to research and to the advancement of knowledge. To place theory in a societal context, however, that theory must have some application to that society. In this research, a prototype was developed to actualize the theory that has been developed thus far and to demonstrate one potential application of this theory.

In the course of this work, two versions of a spatiotemporal GIS were designed and implemented with an application to electric utility AM/FM. These GIS are functionally equivalent and provide the same capabilities. The only difference between them is that the first GIS is a prototype spatiotemporal GIS which relies on XML for persistence while the second GIS is based on the industry standard relational model, constructed for purposes of comparison. Both of these GIS were implemented using Visual Basic 6.0, a popular tool for client/server software development.

While Visual Basic 6.0 does not have some of the more advanced language features that are found in other industry programming languages such as C++ and Java, it does have excellent support for interface with standard relational database systems ranging from Microsoft Access on the low end to Oracle, SQL Server and DB/2 on the high end. In fact, the most common use for Visual Basic is to apply rapid application development (RAD) techniques to design user interface applications to database systems at a variety of scales ranging from small workgroups to the enterprise level.

Microsoft has also recognized the value of the XML language to the developer and to the enterprise, and has provided support for emerging XML standards. It is important to note, however, that these standards are still emerging and are subsequent to revision. The Microsoft XML (MSXML) extensions support version 1.0 of the XML language and XPATH specification. At the time of this writing, development of the XPATH 2.0 specification together with the XQuery query language are under development and will likely provide more streamlined ways to accomplish the desired goal than are at present possible with the tools currently available. For this reason, the methods presented here should be taken as a proof-of-concept and not as an absolute technical recommendation.

The most important point to be taken from these two GIS is the construction of the prototype spatiotemporal GIS with XML persistence and its comparison to a similarly functioned GIS based on relational persistence in terms of development effort and syntactic expressiveness. It should be pointed out also that this research was concerned with the representation of these spatiotemporal data and not with the specifics of database management. As such these GIS are only capable of displaying created data and do not provide capabilities for updating the data, nor do they provide support for indexing, multiple concurrent users or transactions. Such future directions for these items will be discussed in the conclusion of this work.

## Data Development

The first step to implementing these GIS was to develop the data that would be used to populate them. For this work, data were developed in Arc/Info coverage format using Arc/Info Workstation 8.3. An examination of the methodology used to develop these data clearly elucidates the necessity for development of a syntactically elegant spatiotemporal data model.

Attempting to represent and manage fundamentally temporal data in the non-temporal coverage model was difficult and confusing at best.

The AM/FM data model used in the course of this prototype has been greatly simplified from what would be required by an actual utility implementing such a model. This was done principally for the purpose of managing the requirements of this prototype. A typical electric distribution data model defines approximately 25 – 40 feature classes to represent the utility's assets. For this prototype, nine feature classes and one object class were chosen. In the language of ESRI ArcGIS, the distinction between a feature class and an object class is that a feature class is composed of spatial features that can be mapped in the GIS while an object class has only aspatial attributes that relate to features in a feature class. These ten classes consist of three conductors (primary and secondary conductors and risers), one structure (poles), four electric devices (switches, transformers, lights and meters) and two customer information layers (spatial parcel polygons and aspatial customer information). Many of these feature classes require further defined subtypes to more specifically describe them. For example the primary conductor layer consists of overhead and underground primary features. These data layers were described in the introduction to this work.

These data describe a simple model of electric distribution for a fictional area of approximately one square mile. In this scenario, prototype data are first developed on January 1, 1994 and features that existed at that time are attributed with a birth date accordingly. One of the features that existed at that time was a large undeveloped parcel of land. On January 1, 2002, development was hypothetically begun on that parcel to subdivide it into residential parcels and to provide electric service to those parcels. As development and subdivision proceeded in stages, the GIS was updated to reflect the changing current status until finally completed. Figure 4.1 shows an animation with the initial and final states of the prototype data. In addition, as certain maintenance activities were hypothetically performed or as customer information changed, the GIS was updated to reflect these changes. The result is a temporally attributed GIS for a period from January 1, 1994 to February 1, 2006 (the last date of update in this hypothetical scenario).

These data were developed only as a fictional example and are not intended to represent a real place on the Earth, nor should they be taken as such. As described subsequently, these data (in particular the customer information) were populated randomly and are not real data. The prototype data is not georeferenced and is not placed in a real-world context; coordinate boundaries for the prototype data range from ± 2400 feet along the x-axis and ± 2250 feet along the y-axis in the Cartesian plane. The extension of the dates in the prototype data nearly three-years in the future from the time of writing should further illustrate the fictional nature of these data.

As previously stated, data development was managed in Arc/Info coverages. For each feature class and object class developed in this prototype, two date attributes were added to the appropriate coverage or INFO table. These attributes, BORN and RETIRED described the date that a particular record was created or deprecated. Although it might be desirable in a true GIS to tag time as well as date on a feature, in this work only a date was tagged for simplicity. In addition, each coverage or INFO table also was populated with a FEATURE_ID attribute that uniquely tagged the feature across all temporal incarnations. Because a feature could change with time, there might be multiple features in a single coverage with the same FEATURE_ID but with different BORN and RETIRED attributes. These features would be considered to be the same feature at different times. This will be described better subsequently with an example.

For this prototype, eight scenarios were enumerated to be accomplished in the sample data. These scenarios are represented in the source coverage data and as such had to be identified before the actual data production began.  These eight scenarios are identified as follows:

1. Staged subdivision of a single large parcel over a 2 ½ year period.  This would require multiple revisions of the original parcel polygon during that time.
2. Staged construction of electric utility features over the same 2 ½ year period.  This would require only single revisions of these electric features, but with temporal attributing for birth and (if appropriate) retirement dates.
3. Reconfiguration of a transformer that was installed with the wrong phase to the correct phase required by the services drawing power from it.  In this case, the feature is the same feature, but with a revision to one of its aspatial attributes.
4. Movement of a light and pole per a customer request.  In this case, the spatial location of the light and pole features would be revised, as would the secondary conductor providing power to the light feature.  All three of these features would remain the same feature but would have their spatial attributes revised.
5. Maintenance on a cut section of primary conductor feeding the entire subdivision. Electric utilities often abandon underground features and install new ones when replacement is necessary because of the cost involved in excavating the original feature.  In this example, a section of primary conductor is cut and is isolated from the network and abandoned in place.  A second section of conductor is installed offset from the original section by a foot and is attached to the network to restore power to the subdivision.  This requires modifying the spatial attributes of the original section of conductor (to shorten it and isolate it from the network) and installing a new conductor feature with a birth date representing when it was installed.

6. Customer information change. In this scenario, five customers move over a two- year period, requiring that the customer information table be updated. This illustrates how related records would be updated in such a temporally aware system.
7. A faulty meter is replaced. This requires retirement of the original meter feature and installation of a new meter feature and demonstrates how the temporally aware relationship to the customer table behaves when the underlying feature related to the table changes.
8. A customer gets married and changes her name. This requires an update to the customer record but does not generate a new customer record.

Creation of the switch feature class was the simplest to accomplish; this sample data consisted of only a single switch point feature that existed without revision. The schema for the switch feature class is described in table 4.1. The evolution of the switch feature class is animated in Figure 4.2 with the parcel polygons for reference. Also relatively simple to create was the riser feature class. As with the switch, there were no revisions to these features and were relatively few of them (only 2 riser features). The riser schema is described in Table 4.2 and evolution of the riser feature class is animated in Figure 4.3. All of the remaining features classes required temporal revision and were more difficult to create.

The primary conductor coverage consisted of 170 features, with one feature having two revisions and being replaced by a different feature, for a total of 171 features in the coverage. A look at the specifics of the replaced conductor will illustrate how the temporal revision attributes and feature identifier manage the lifetime of the feature. Figure 4.4 illustrates the primary conductor feature class zoomed in around one end of the replaced conductor. In the

Table 4.1: Switch Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of switch, typically identifies whether the switch is overhead or underground and the design of the switch such a load-break switch. |
| PHASE | CHARACTER | The operating phase of the switch. Phase may be any of A, B, or C phases for a single- phase switch, AB, AC or BC for a two-phase switch or ABC for a three-phase switch. |
| OPVOLTAGE | CHARACTER | The current operating voltage of the switch. |
| NOMVOLTAGE | CHARACTER | Nominal voltage is the rated voltage of the switch, operating voltage may be less than or equal to the nominal voltage. |
| BORN | DATE | Birth date of the specified revision of the feature. |
| RETIRED | DATE | Retirement date of the specified revision of the feature. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier that tags a feature across all revisions. |
| [SHAPE] | SHAPE | Not an explicit attribute of the coverage, but integrated into the coverage, represents the geometry of the feature. |

Table 4.2: Riser Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of riser. |
| PHASE | CHARACTER | Operating phase of the riser. |
| MATERIAL | CHARACTER | Construction material of the riser. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

scenario, the conductor with the feature identifier 36 was cut and abandoned in place, then replaced with the conductor with feature identifier 170. Feature 36 was born (in this database) on January 1, 1994 and remained unaltered until it was cut on October 18, 2002. When it was cut, maintenance crews disconnected it from the network and updated the GIS to reflect that the feature had been changed. In the Arc/Info coverage that was used to populate the source data for the GIS, this was represented as two distinct coverage features with different geometry, but with the same FEATURE_ID value of 36 to identify them as the same feature. Therefore, the original feature 36 had a BORN value of 1/1/1994 and a RETIRED value of 10/18/2002 to reflect the lifetime of that revision. The second feature 36 had a BORN value of 10/18/2002 and no RETIRED value to represent the lifetime of this second revision. In the source data in Arc/Info, then, these are two distinct features. In the migrated data in both XML and the relational database, these are two revisions to a single feature. The new conductor with FEATURE_ID 170 was also installed and also has a BORN value of 10/18/2002 and no RETIRED value.

Table 4.3 describes the schema for the primary conductor feature class. There are six types of primary conductor features, for one, two and three phase overhead and underground primary, respectively. Figure 4.5 animates the evolution of this feature class.

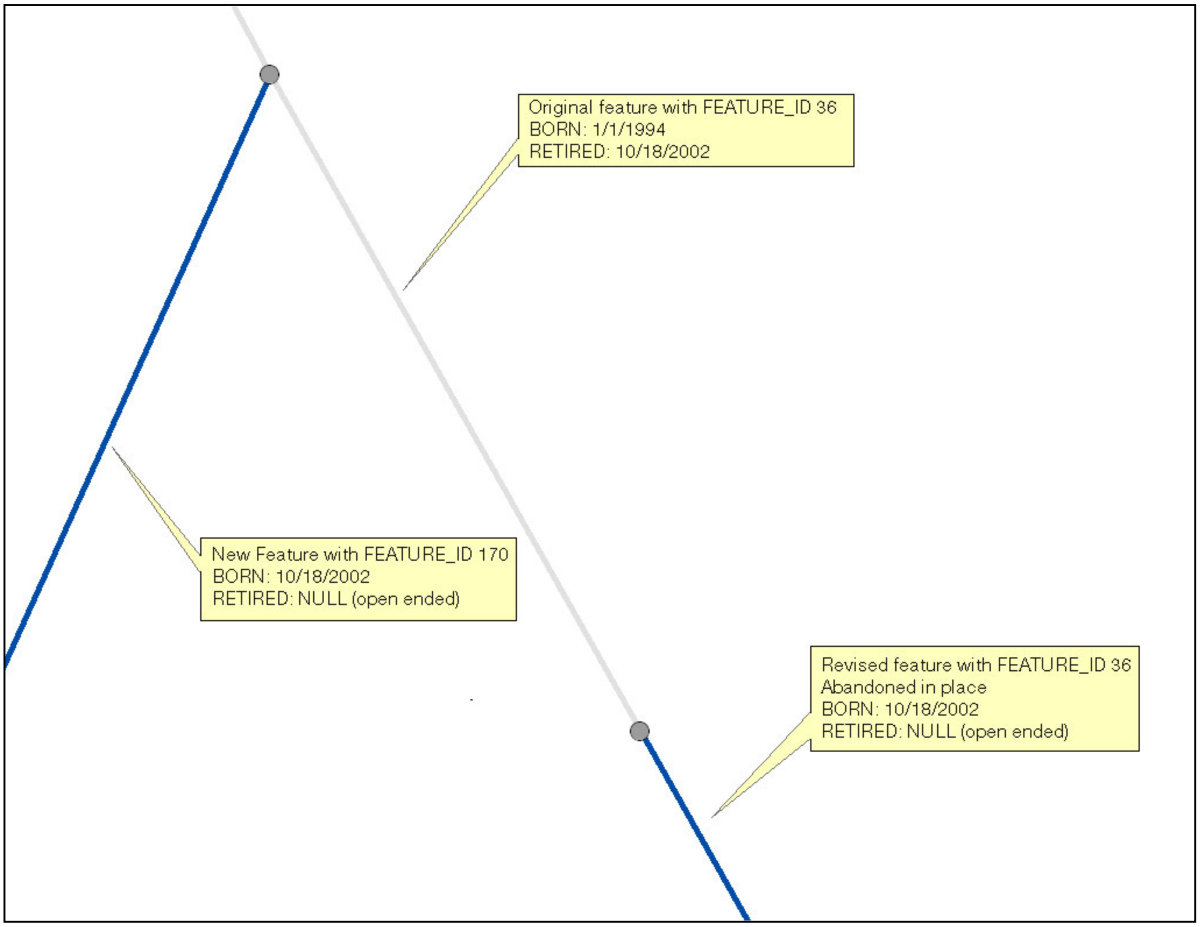

Original feature with FEATURE_ID 36
BORN: 1/1/1994
RETIRED: 10/18/2002
New Feature with FEATURE_ID 170
BORN: 10/18/2002
RETIRED: NULL (open ended)
Revised feature with FEATURE_ID 36
Abandoned in place
BORN: 10/18/2002
RETIRED: NULL (open ended)

**Figure 4.4: Revision to Replace a Cut Primary Conductor.**

Like the primary conductor features, the secondary conductor features were installed at intervals, and like the primary conductor there is one instance of a multiple revision secondary conductor. There are 508 secondary conductor features, each of which attach to the primary conductor at a transformer feature and provide power to either a meter or a light feature (509 features in the coverage including the two revisions of one feature). The schema for the secondary conductor is identical to the schema for the primary conductor, albeit with different values. There are two types of secondary conductors, for secondary underground and service underground; the distinction is that service underground provides service to a meter while secondary underground supplies power to a light or other device. This schema is reviewed in Table 4.4. Figure 4.6 animates the evolution of the secondary conductor feature class.

**Table 4.3: Primary Conductor Feature Class Schema.**

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of conductor. |
| PHASE | CHARACTER | Operating phase of the conductor. |
| OPVOLTAGE | CHARACTER | Operating voltage of the conductor. |
| NOMVOLTAGE | CHARACTER | Nominal voltage of the conductor. |
| CONDMATERIAL | CHARACTER | The material used by the conductor; utilities typically have stock materials that are used for conductors. |
| NEUTMATERIAL | CHARACTER | Material used by the conductor neutral wire; conductors must have a neutral wire that carries current back to the source to close the circuit. |
| CONDSIZE | CHARACTER | Thickness of the conductor. |
| NEUTSIZE | CHARACTER | Thickness of the neutral wire. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

**Table 4.4: Secondary Conductor Feature Class Schema.**

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of conductor. |
| PHASE | CHARACTER | Operating phase of the conductor. |
| OPVOLTAGE | CHARACTER | Operating voltage of the conductor. |
| NOMVOLTAGE | CHARACTER | Nominal voltage of the conductor. |
| CONDMATERIAL | CHARACTER | The material used by the conductor. |
| NEUTMATERIAL | CHARACTER | Material used by the conductor neutral wire. |
| CONDSIZE | CHARACTER | Thickness of the conductor. |
| NEUTSIZE | CHARACTER | Thickness of the neutral wire. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

The feature in the secondary conductor feature class with multiple revisions represents a customer request for a streetlight movement (presumably, paid for by the customer). Figure 4.7 illustrates the revisions to this secondary conductor feature along with the revisions of the associated pole and light features. In this case, the streetlight, the pole and the secondary conductor were all installed – the streetlight (feature 62) and pole (feature 69) on 1/21/2003 and the secondary conductor (feature 361) to provide power to it later on 3/27/2003. When the customer requested that the light be moved, revisions were created for each of these three features, with the result that each of the secondary conductor, light and meter classes each contain two features with the same feature identifier, but with different BORN and RETIRED attributes to represent the different revisions of the features. Again, while the coverage will contain two features, the resultant XML and relational models will contain a single feature with multiple revisions.

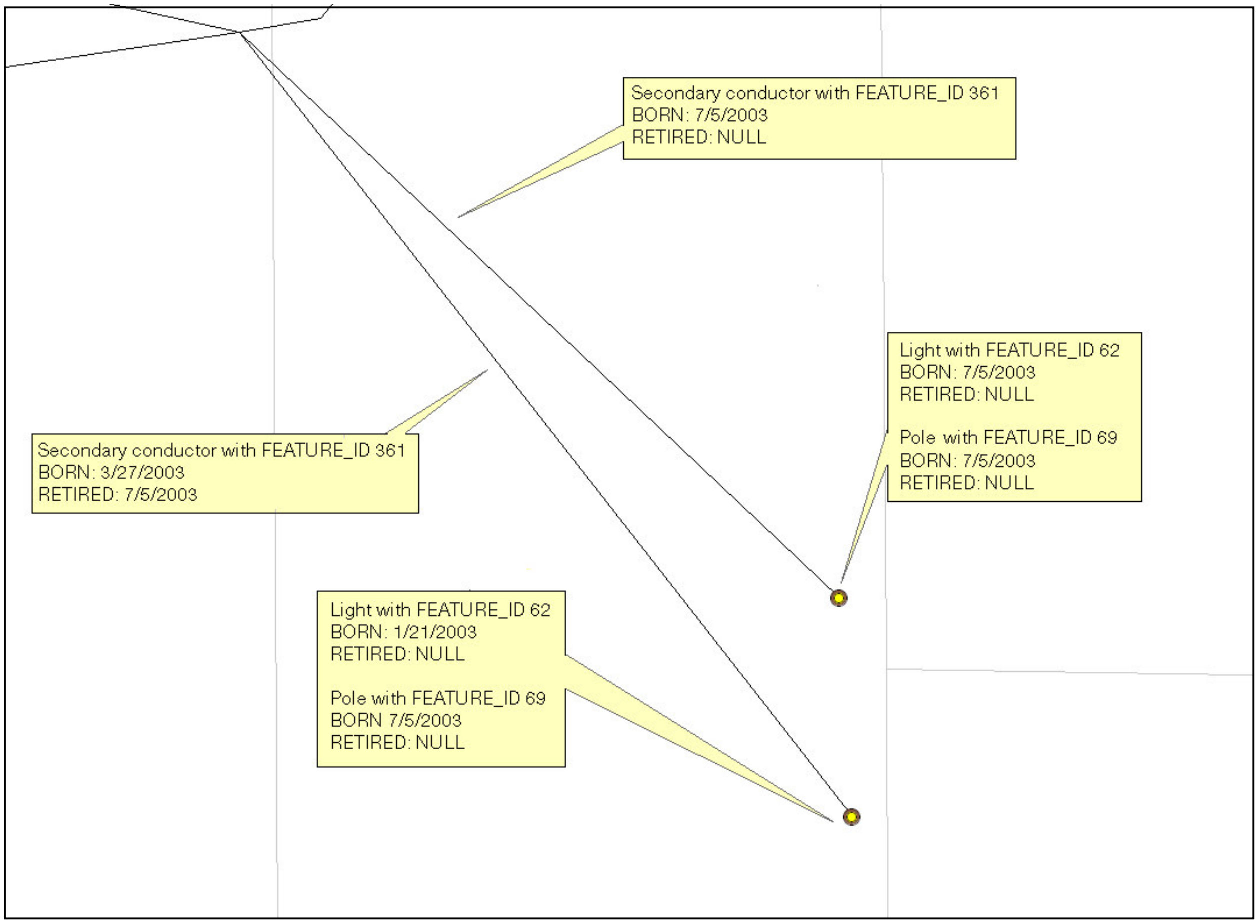


**Figure 4.7: Streetlight and Associated Features Movement.**

The transformer feature class consists of 57 features, and like the conductor feature classes contains a single feature that has multiple revisions (for a total of 58 features in the coverage). Also like the conductor feature classes, transformer features were installed progressively. In this case, the transformer with FEATURE_ID 18 was incorrectly installed A phase when it should have been installed as B phase. When the meters and secondary conductor were installed, the problem was uncovered. Figure 4.8 animates the evolution of the transformer feature class. Table 4.5 describes the schema for the transformer feature class. All 57 transformer features are of type single phase underground.

The pole feature class consists of 79 features installed progressively, one with multiple revisions (for a total of 80 features in the coverage). The multiple revision instance was described earlier in with the secondary conductor feature class and is illustrated in Figure 4.7. Table 4.6 describes the schema for the pole feature class. In this data, poles are either of type wood power pole or non-wood street light pole. Figure 4.9 animates the evolution of the pole feature class.

The light feature class consists of 63 features installed progressively, one with multiple revisions (for a total of 64 features in the coverage). The multiple revision instance was described earlier with the secondary conductor feature class and is illustrated in Figure 4.7. Table 4.7 describes the schema for the light feature class. Figure 4.10 animates the evolution of the light feature class.

The last two feature classes, meter and parcel, and the customer object class were the most difficult to create. Because they are interrelated, they will be described together. The basic configuration of the two feature classes and the object class is that, as with the other feature classes, each contains a BORN and RETIRED attribute. The meter and parcel feature classes each also have a defined FEATURE_ID attribute, while the customer class has a defined CUSTOMER_ID attribute. The meter and parcel feature classes also contain the CUSTOMER_ID attribute. Relationships between the meter feature class and customer object class and between the parcel feature class and customer object class are maintained using a standard primary key / foreign key relationship on the CUSTOMER_ID attribute.

Figure 4.11 animates the evolution of the meter feature class. Figure 4.12 animates the evolution of the parcel layer through subdivision. Table 4.8 describes the schema for the meter feature class. Table 4.9 describes the schema for the parcel feature class.

Table 4.5: Transformer Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of transformer. |
| PHASE | CHARACTER | Operating phase of the transformer. |
| RATEDKVA | FLOAT | The kilovolt load rating of the transformer. |
| HSCONFIG | CHARACTER | Configuration on the high side of the transformer (where the primary conductor enters the transformer). |
| LSCONFIG | CHARACTER | Configuration on the low side of the transformer (where the secondary conductor leaves the transformer). |
| OPVOLTAGE | CHARACTER | Operating voltage of the conductor. |
| NOMVOLTAGE | CHARACTER | Nominal voltage of the conductor. |
| LSVOLTAGE | CHARACTER | Low side voltage of the transformer, indicates the transformed voltage on the low side of the transformer. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

Table 4.6: Pole Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| TYPE | CHARACTER | The type of pole. |
| GROUND | CHARACTER | Type of ground on the pole, if any. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

Table 4.7: Light Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| LIGHTFEED | CHARACTER | Source of light electricity, either overhead or underground. |
| STREETTYPE | CHARACTER | Type of streetlight, for example city streetlight. |
| LAMPTYPE | CHARACTER | The type of lamp, describes the chemical reaction that produces the light, for example sodium. |
| LIGHTSTYLE | CHARACTER | Light style, for example open or closed lamp. |
| PHOTOCELL | CHARACTER | Type of photocell used by the lamp, specific to the type of installation. |
| LIGHTROLE | CHARACTER | Specifies where the lamp is located, for example inside city street. |
| WATTAGE | INTEGER | Wattage rating of the light. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| [SHAPE] | SHAPE | Geometry of the feature. |

Table 4.8: Meter Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| PHASE | CHARACTER | Phase of the meter. |
| SVC_CURR_RATING | CHARACTER | Current rating in amperes for the meter. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| CUSTOMER_ID | LONG INTEGER | Unique customer identifier; foreign key relating to the customer table. |
| [SHAPE] | SHAPE | Geometry of the feature. |

Table 4.9: Parcel Feature Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |
| FEATURE_ID | LONG INTEGER | Unique feature identifier. |
| CUSTOMER_ID | LONG INTEGER | Unique customer identifier; foreign key that relates to the customer table. |
| [SHAPE] | SHAPE | Geometry of the feature. |

The customer objects correspond to meter and parcel features, thus as parcels and meters were created and updated, appropriate customer records were created and updated as well. Table 4.10 describes the schema for the customer object class, which was stored in an INFO database table.

Customer data were fabricated using random data. Because there were 331 customer records, 20 first names and 20 last names were randomly paired in such a way as to insure uniqueness of the name combinations (for example John and Joe first names with Smith and Black last names would generate John Smith, Joe Black, Joe Smith and John Black). Not all name combinations were used, but all names generated were unique. Street addresses were generated by assigning groups of parcels to street names and then numbering the parcels successively in increments of 4 (for example, 101, 105, 109, etc.). BORN dates for purchase of properties and creation of customer records were generated by adding a random duration from 1 to 90 days after the meter was installed to simulate staggered purchases. In this way, the customer data was similar to actual data that would be found in a corporate utility AM/FM/GIS.

Table 4.10: Customer Object Class Schema.

| ATTRIBUTE | TYPE | DESCRIPTION |
|---|---|---|
| CUSTOMER_ID | LONG INTEGER | Unique customer identifier; primary key that relates to the meter and parcel tables. |
| NAME | CHARACTER | Customer name. |
| HOUSE_NUMBER | INTEGER | Customer house number. |
| STREET | CHARACTER | Customer street name. |
| CITY | CHARACTER | Customer city. |
| STATE | CHARACTER | Customer state. |
| ZIP_CODE | INTEGER | Customer zip code. |
| BORN | DATE | Birth date of the feature revision. |
| RETIRED | DATE | Retirement date of the revision. |

To understand the steps required to create these feature classes, a review of the temporal changes to the classes is necessary. In this scenario, the parcel features were originally created on January 1, 1994 and first subdivided on January 7, 2002. At that time, the geometry of the original large parcel (FEATURE_ID 1) was altered, thus generating an additional record in the coverage (although still the same feature). On October 28, 2002, the parcel polygon with FEATURE_ID 1 was again subdivided, again altering the geometry and requiring an additional coverage feature. It should be noted that in practice these additional parcel polygon coverage features were actually maintained in distinct polygon coverages due to the complications that stored topology would introduce with overlapping or quasi-overlapping features. Conceptually, however, these features can be considered as if in a single polygon coverage. On September 15, 2003, the parcel polygon with FEATURE_ID 1 was again subdivided, this time retiring the feature as it had been fully subdivided. There are 536 distinct polygon features by the time that the large polygon is subdivided, with 3 additional representations of the parcel polygon with FEATURE_ID 1 for a total 539 polygon coverage records thus far in 4 distinct polygon coverages.

At this point, the parcels have been subdivided, but construction has not yet been completed. There are, therefore, no associated customer records. Thus all parcel polygons have a CUSTOMER_ID value of 0. Meters are installed on the parcels in stages and assigned a BORN date as appropriate; however, the properties have not yet been purchased and so here as well there are no associated customer records. All meter records thus far have a CUSTOMER_ID value of 0. There are, at this point, a total of 331 distinct meter records.

As properties are purchased, customer records are generated and related to the appropriate meter and parcel features. Because updating the CUSTOMER_ID value in the meter or parcel feature is an attribute change, it requires that the current incarnation of the feature be retired and a new feature generated with the update CUSTOMER_ID. In this scenario, all properties constructed are sold eventually, generating 331 additional meter records and 331 additional parcel records (presumably the remaining polygons either have no data created for them or have not been sold). The additional parcel features are in a separate coverage to avoid topology problems. There are now a total 662 meter records and 870 parcel records in 5 distinct polygon coverages. This completes construction and initial purchase of the properties.

In this sample data, three additional scenarios affect these three classes. In the first scenario, a faulty meter (FEATURE_ID 172) must be replaced, generating a new meter record in the coverage with FEATURE_ID 332. Because the customer did not change, in both cases the CUSTOMER_ID remains the same – 490. In the second scenario, five homes are sold (including the faulty meter with CUSTOMER_ID 490). For each sale, new CUSTOMER_ID records must be created, and CUSTOMER_ID values must be updated in the parcel and meter coverages. In the final scenario, a customer gets married and changes her name. For this, a new customer record must be created with the same CUSTOMER_ID, while the old customer record is retired. Meter and parcel records do not have to be retired, however, since the CUSTOMER_ID attribute did not change.

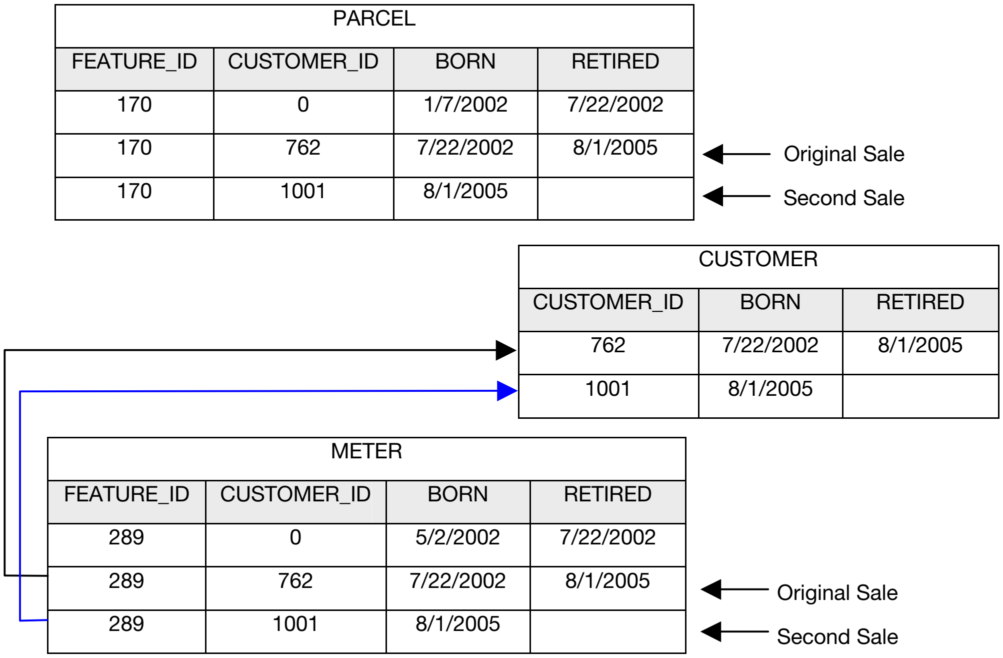
For the first sale, the property with CUSTOMER_ID 762 is sold on August 1, 2005. This causes customer record 762 to be retired on August 1, 2005 and a new record with CUSTOMER_ID 1001 to be created on the same date. This requires the associated meter and parcel records to be retired on this date and new records to be created with the updated CUSTOMER_ID value. It should be remembered that this is only for purposes of creating and staging data in the non-temporally aware Arc/Info. In the temporal GIS, these would all be single features with multiple revisions. At the conclusion of this sale, there are 2 customer records and 3 meter and parcel records respectively for this single property (one record for before the first purchase, one record for the first purchase and one record for the second purchase).

In the second sale, the property with the meter that had been replaced is sold (CUSTOMER_ID 490) on November 15, 2004. Customer record 490 is retired on this date and a new record with CUSTOMER_ID 1002 is born on this date. Meter and parcel records are retired and born with new CUSTOMER_ID values as well. At the end of this sale, there are 2 customer records, 4 meter records and 3 parcel records for this property. The 3 parcel records are for the same events as the first sale. The 4 meter records are for before the first sale, for the first sale, for the replaced meter and for the second sale.

The third sale is like the first sale; the property with CUSTOMER_ID 217 is sold on February 1, 2006. The original customer record is retired on August 1, 2004 and replaced with CUSTOMER_ID 1003 and meter and parcel features are retired and replaced with updated CUSTOMER_ID values as well. At the conclusion of this sale, there are again 2 customer records and 3 meter and parcel records respectively. The fifth sale follows the same pattern as the first and third sales; the customer record with CUSTOMER_ID 475 is retired and replaced with a new customer record with CUSTOMER_ID 1005, again requiring retired and new meter and parcel records respectively and again with a total of 2 customer records and 3 meter and parcel records.

In the fourth sale (CUSTOMER_ID 342), suppose that the seller could not sell immediately and so disconnected power from the property while trying to sell. In this case, the customer record is retired on April 30, 2005, and the associated meter and parcel records are retired as well. New meter and parcel records are born with a CUSTOMER_ID value of 0, since there is at this point no associated customer record. When the property is finally sold on July 10, 2005, these parcel and meter records are then retired and replaced with new records with an updated CUSTOMER_ID value of 1004 and a new customer record with the same CUSTOMER_ID is generated. At the end of this financial transaction, there are 2 customer records and 4 meter and parcel records, respectively (one record for before the first sale, one record for the first sale, one record for transferal out of the original owner's name and one record for the second sale). Recall that parcel polygons at different times are maintained in separate coverages. For those still counting, the result is a total of 669 meter records at various times to represent 332 distinct meter features, a total of 336 customer records and a total of 876 parcel polygon records in 7 parcel polygon coverages to represent 537 distinct parcel features.

While the creation, retirement and management of these features may seem complicated, these are real-world issues that would affect any utility implementing a spatiotemporal AM/FM/GIS, and may be on the simple side of modeling problems. To further clarify the creation and retirement of features in this scenario, Figures 4.13 – 4.18 illustrate the processes for each of the five sales, the meter replacement and the customer name change.



PARCEL

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 170 | 0 | 1/7/2002 | 7/22/2002 |
| 170 | 762 | 7/22/2002 | 8/1/2005 |
| 170 | 1001 | 8/1/2005 | |

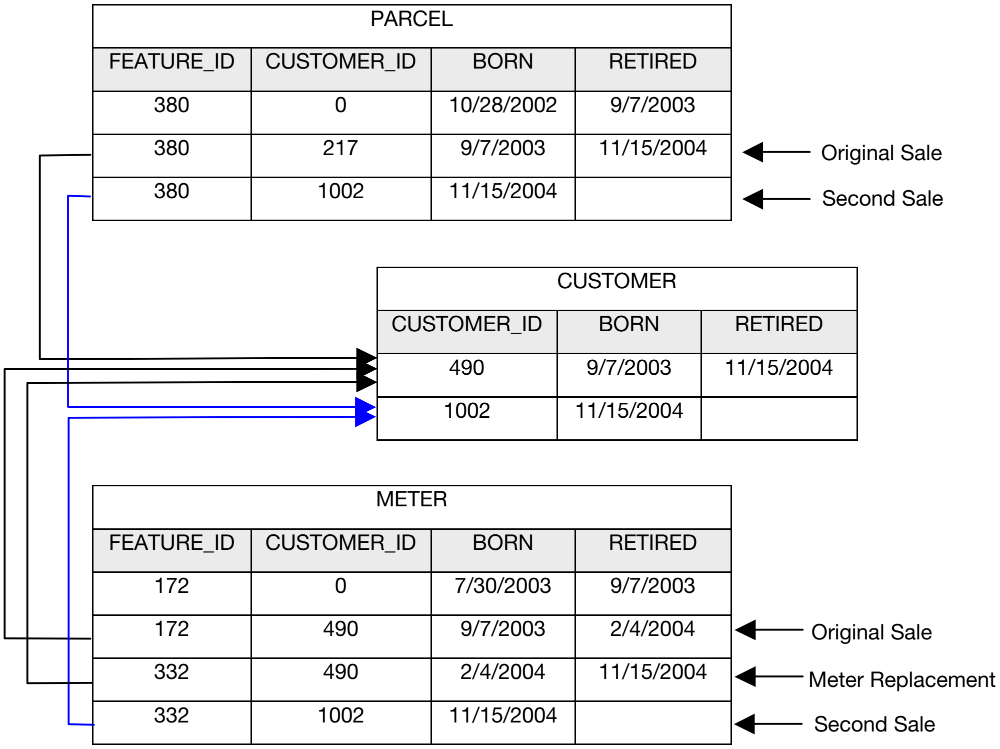
CUSTOMER

| CUSTOMER_ID | BORN | RETIRED |
|---|---|---|
| 762 | 7/22/2002 | 8/1/2005 |
| 1001 | 8/1/2005 | |

METER

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 289 | 0 | 5/2/2002 | 7/22/2002 |
| 289 | 762 | 7/22/2002 | 8/1/2005 |
| 289 | 1001 | 8/1/2005 | |

Figure 4.13: Records for First Property Sale.



PARCEL

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 380 | 0 | 10/28/2002 | 9/7/2003 |
| 380 | 217 | 9/7/2003 | 11/15/2004 |
| 380 | 1002 | 11/15/2004 | |

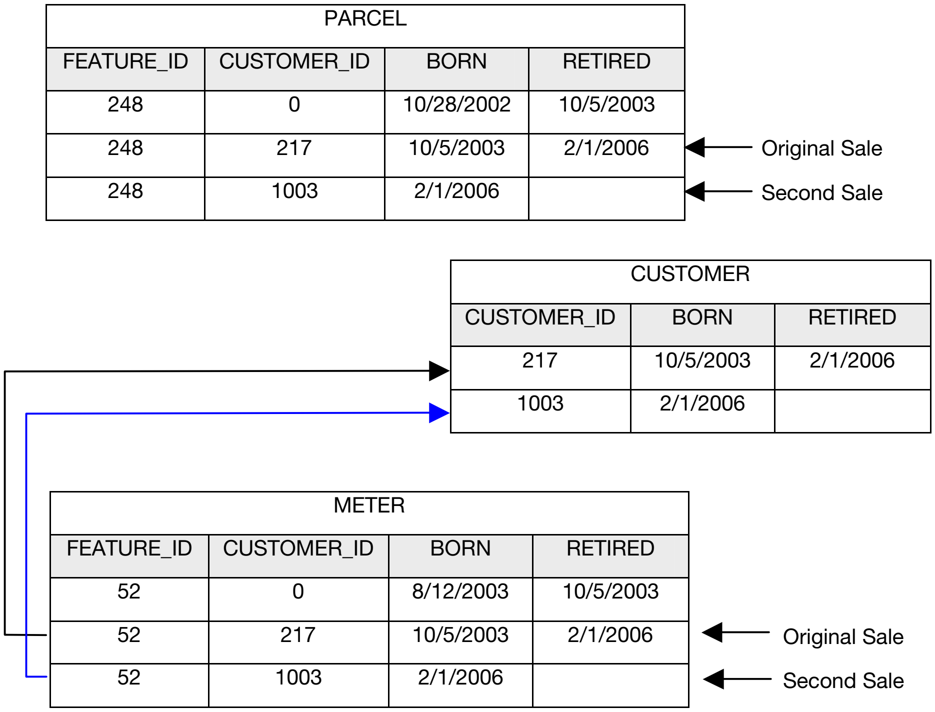
CUSTOMER

| CUSTOMER_ID | BORN | RETIRED |
|---|---|---|
| 490 | 9/7/2003 | 11/15/2004 |
| 1002 | 11/15/2004 | |

METER

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 172 | 0 | 7/30/2003 | 9/7/2003 |
| 172 | 490 | 9/7/2003 | 2/4/2004 |
| 332 | 490 | 2/4/2004 | 11/15/2004 |
| 332 | 1002 | 11/15/2004 | |

Figure 4.14: Records for Second Property Sale and Meter Replacement.

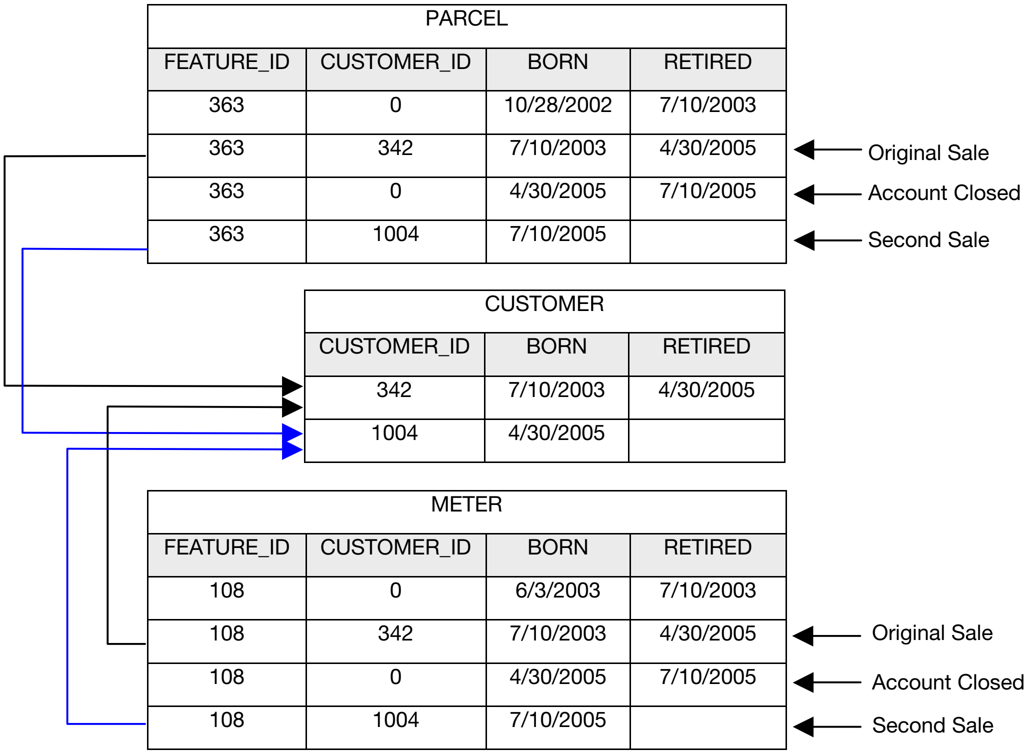


| PARCEL | | | |
|---|---|---|---|
| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
| 248 | 0 | 10/28/2002 | 10/5/2003 |
| 248 | 217 | 10/5/2003 | 2/1/2006 |
| 248 | 1003 | 2/1/2006 | |

| CUSTOMER | | |
|---|---|---|
| CUSTOMER_ID | BORN | RETIRED |
| 217 | 10/5/2003 | 2/1/2006 |
| 1003 | 2/1/2006 | |

| METER | | | |
|---|---|---|---|
| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
| 52 | 0 | 8/12/2003 | 10/5/2003 |
| 52 | 217 | 10/5/2003 | 2/1/2006 |
| 52 | 1003 | 2/1/2006 | |

Figure 4.15: Records for Third Property Sale.

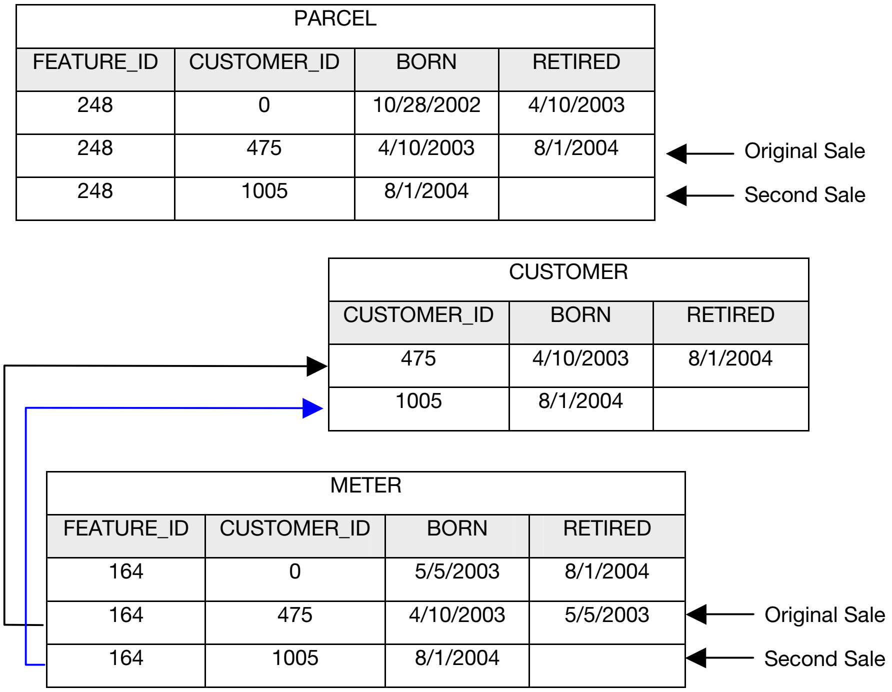


| PARCEL | | | |
|---|---|---|---|
| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
| 363 | 0 | 10/28/2002 | 7/10/2003 |
| 363 | 342 | 7/10/2003 | 4/30/2005 |
| 363 | 0 | 4/30/2005 | 7/10/2005 |
| 363 | 1004 | 7/10/2005 | |

| CUSTOMER | | |
|---|---|---|
| CUSTOMER_ID | BORN | RETIRED |
| 342 | 7/10/2003 | 4/30/2005 |
| 1004 | 4/30/2005 | |

| METER | | | |
|---|---|---|---|
| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
| 108 | 0 | 6/3/2003 | 7/10/2003 |
| 108 | 342 | 7/10/2003 | 4/30/2005 |
| 108 | 0 | 4/30/2005 | 7/10/2005 |
| 108 | 1004 | 7/10/2005 | |

Figure 4.16: Records for Fourth Property Sale.



PARCEL

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 248 | 0 | 10/28/2002 | 4/10/2003 |
| 248 | 475 | 4/10/2003 | 8/1/2004 |
| 248 | 1005 | 8/1/2004 | |

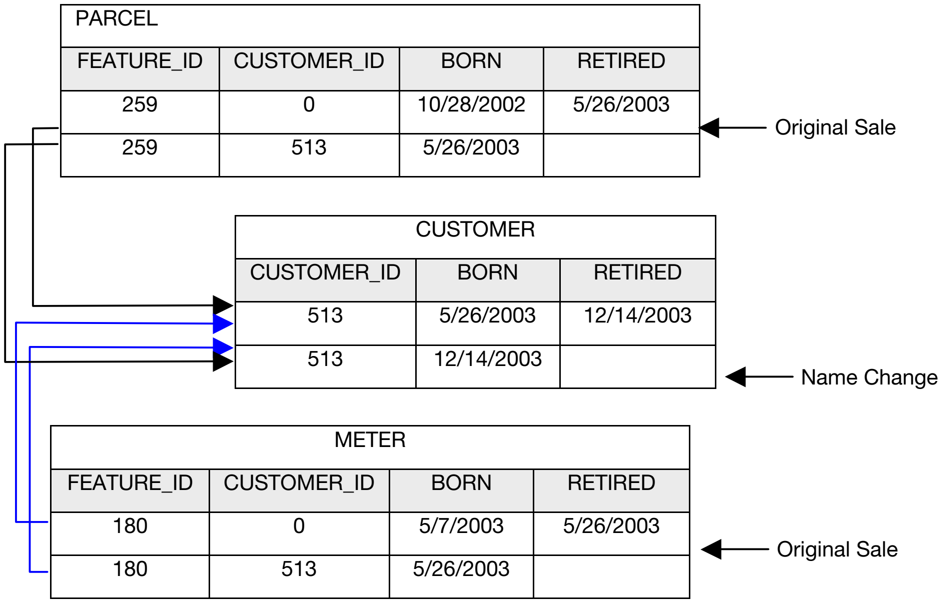
CUSTOMER

| CUSTOMER_ID | BORN | RETIRED |
|---|---|---|
| 475 | 4/10/2003 | 8/1/2004 |
| 1005 | 8/1/2004 | |

METER

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 164 | 0 | 5/5/2003 | 8/1/2004 |
| 164 | 475 | 4/10/2003 | 5/5/2003 |
| 164 | 1005 | 8/1/2004 | |

Figure 4.17: Records for Fifth Property Sale.



PARCEL

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 259 | 0 | 10/28/2002 | 5/26/2003 |
| 259 | 513 | 5/26/2003 | |

CUSTOMER

| CUSTOMER_ID | BORN | RETIRED |
|---|---|---|
| 513 | 5/26/2003 | 12/14/2003 |
| 513 | 12/14/2003 | |

METER

| FEATURE_ID | CUSTOMER_ID | BORN | RETIRED |
|---|---|---|---|
| 180 | 0 | 5/7/2003 | 5/26/2003 |
| 180 | 513 | 5/26/2003 | |

Figure 4.18: Records for Customer Name Change.

These figures clearly show the complicated relationships that would be required to support this relational model.

Conversion to the relational data model was accomplished in two steps. First, the coverages and customer INFO table developed thus far were imported into a personal geodatabase format. The geodatabase format is the ESRI solution for storing spatial data in standard relational database systems. The personal geodatabase is based upon the Microsoft Access database, while enterprise geodatabases may be built in Oracle, SQL Server, DB/2 or other commercial relational database systems. Attribute data are stored directly in appropriate columns. However, shape data are stored in a proprietary binary ESRI format and cannot be directly read by applications that do not make use of the ESRI data structures.

Once the coverages and INFO table had been imported to personal geodatabase format, all of the data were then located in appropriate Microsoft Access tables, and in fact in ESRI geodatabases tables in the Microsoft Access database. Creation of the geodatabase also requires a large number of metadata tables that are not directly usable by the end user. Finally, the shape data were stored in a binary column.

In order to move the shape data from the binary ESRI format into a readable format for non-ESRI client applications (such as the GIS developed in course of this work), a Visual Basic for Applications (VBA) macro was created in ESRI ArcMap.

VBA is integrated into ArcMap and is a subset of the Visual Basic language. ArcMap VBA macros may be developed to interact with GIS data through the object-oriented ArcObjects library. This library allows (among other things) for a feature class to be iterated over with attributes extracted. This VBA macro looped through each of the feature classes in the geodatabase to extract the vertices, area, length, perimeter and bounding box (as appropriate for line and polygon feature classes) for each feature in the feature class. Bounding boxes were not extracted for point features as points are 0-dimensional and hence maximum and minimum x, y coordinates are exactly equal to the x, y coordinates of the point itself. Also extracted were the feature id and the BORN and RETIRED attributes. The VBA macro used to extract the shape data is listed in Appendix A.

Because there is a one-to-many relationship between features and their vertices for linear and areal feature classes, vertices were stored in a related table name as *featureclass*_Shape (for example, the vertex data for the Light table would be stored in the related Light_Shape table. Each vertex record stores the feature id, BORN and RETIRED attributes (since there may be more than one shape revision for a given feature), the vertex order number (that is, the first vertex has a vertex order of 1, the second has a vertex order of 2 and so forth). Although this is a somewhat complicated scheme, with a multipart key based upon feature id, BORN and RETIRED values, the alternative to store these vertices directly in the appropriate table would have required storage in a concatenated string formatted such as "$x_1,y_1;x_2,y_2;x_3,y_3;\ldots;x_n,y_n;$". This would have required more resources from the GIS, as it had to iteratively extract the vertices from the string. Bounding box vertices are also stored in this Shape table, with vertex order values of –2 for bounding box minimum coordinates and –1 for bounding box maximum coordinates. Although this obfuscates the model by storing two semantically different data cases in the same table, this was done to avoid the need (and overhead) of constructing a third table to hold bounding box vertices. An alternative could also have been to store these bounding box coordinates in four columns in the feature class table (as are the area, light and perimeter attributes), but this was decided against because the bounding box coordinates are vertices and this model retained more (albeit imperfect) semantic consistency.

The final steps involved in conversion to the relational database were to combine discreet parcel tables and cleanup of the personal geodatabase to effectively reduce it to a simple Access database. Because the parcels had to be represented in distinct coverages for distinct times, the seven tables imported from the seven parcel coverages had to be combined into a single parcel table. To clean up the geodatabase and revert it to a simple database, extra metadata tables were dropped since the GIS is not ESRI based. Any columns brought over as legacy from the original coverage were also dropped as well (for example original object id values from the coverage). Finally, the binary shape field was dropped.

To store metadata about the model, such as feature class, object and relationship definitions, and symbology, a series of catalog tables were also created and populated. These were Catalog_FeatureClass, Catalog_ObjectClass, Catalog_Relationship, Catalog_BoundBox and Catalog_Symbology; their applications can be readily discerned from their names. The final relational schema is diagrammed using the Entity-Relational model in appendix B.

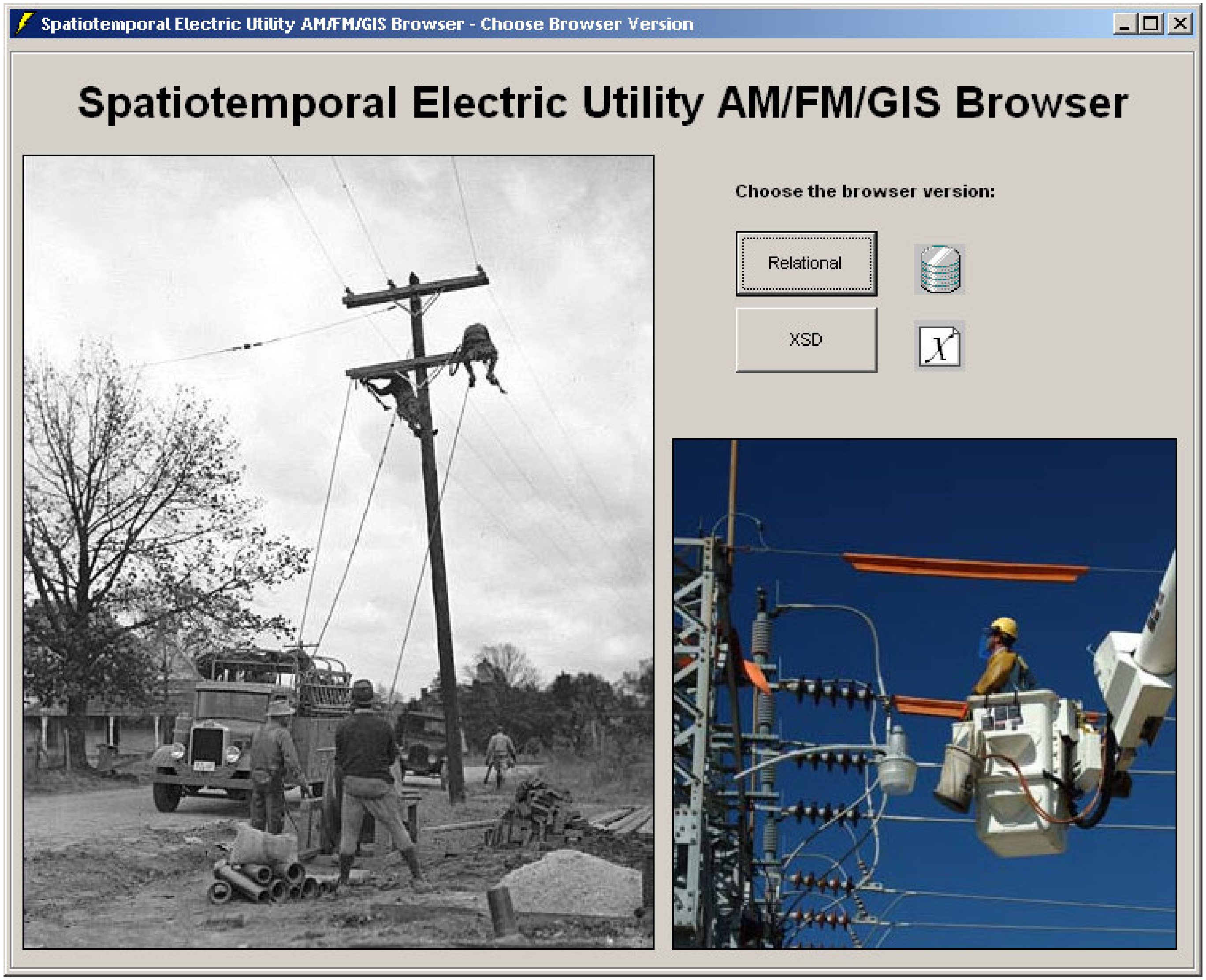
Conversion of the XML data was more straightforward. Because the extraction of data from the proprietary ESRI coverage had already been undertaken in the conversion to the relational database, this database was used as the basis for populating the XML schema. The only task required to accomplish this then was to map from the relational schema to the XML schema. Although this was conceptually simple, in practice it illustrated well the potential pitfalls of such relational –to-XML mappings and the challenges faced by those who persist XML data in relational databases. The most straightforward way to accomplish this mapping was through the use of a Visual Basic script, which essentially opened a cursor into each of the feature classes and object class (and related shape tables) and populated an instance of the Microsoft Document Object Model (DOM) for XML; this DOM object in turn was used to persist the final XML. The Visual Basic script used to accomplish this mapping is listed in Appendix C. While this was satisfactory for converting the feature and object classes, it proved unwieldy for converting the catalog tables. For this reason, and because there was little data in the catalog tables, the catalog was converted manually.

Once the XML data had been populated into one of eleven XML files (one for each of the feature classes and one for the object class and catalog, respectively), XMLSpy 5.4 was used to attempt to extract the schema definitions from the XML files. XML stores schema information separately from the data using one of a number of mechanisms. For this work, the schema definition language XSD was used. Although XMLSpy can generate XSD schema definitions automatically from a specified XML file, the resulting XSD files thus generated were too literal in representing the existing data in the system rather than the generalities of the data as a whole. For example, the FEATURE_ID field might have a domain list that constrains its values to those in the XML file (for example, 1 to 100), whereas it would be more desirable to simply constrain this data to a particular data type (such as an integer). Because the XSD generation was less than optimal, XSD files were created for each of the XML files manually. These XSD files are used to validate the input XML file to insure that the data in it meets the desired schema criteria. There is no universally accepted notation for diagramming XSD schemas. Although the Unified Modeling Language (UML) is often adopted for modeling both relational and XML schemas, it is not completely suited to describing the semistructured XML file. The XSD schema file is self-describing and easily understood and often serves as its own diagram. The XSD schema files are listed in Appendices D-N.

## Software Development

The GIS software was developed once data development was completed. Although there were two distinct GIS developed in the course of this work, both were developed in a single Visual Basic project. Upon starting the GIS software, the user selects from a menu to execute either the XML based version or the relational version of the GIS. Figure 4.19 illustrates this selection menu. Program listings for the selection menu and the startup basic module governing the Visual Basic project are listed in appendices O and P respectively.

The GIS were developed cooperatively and are functionally identical, differing only in their persistence model. The relational GIS uses a Microsoft Access database while the XML



Spatiotemporal Electric Utility AM/FM/GIS Browser - Choose Browser Version

Spatiotemporal Electric Utility AM/FM/GIS Browser

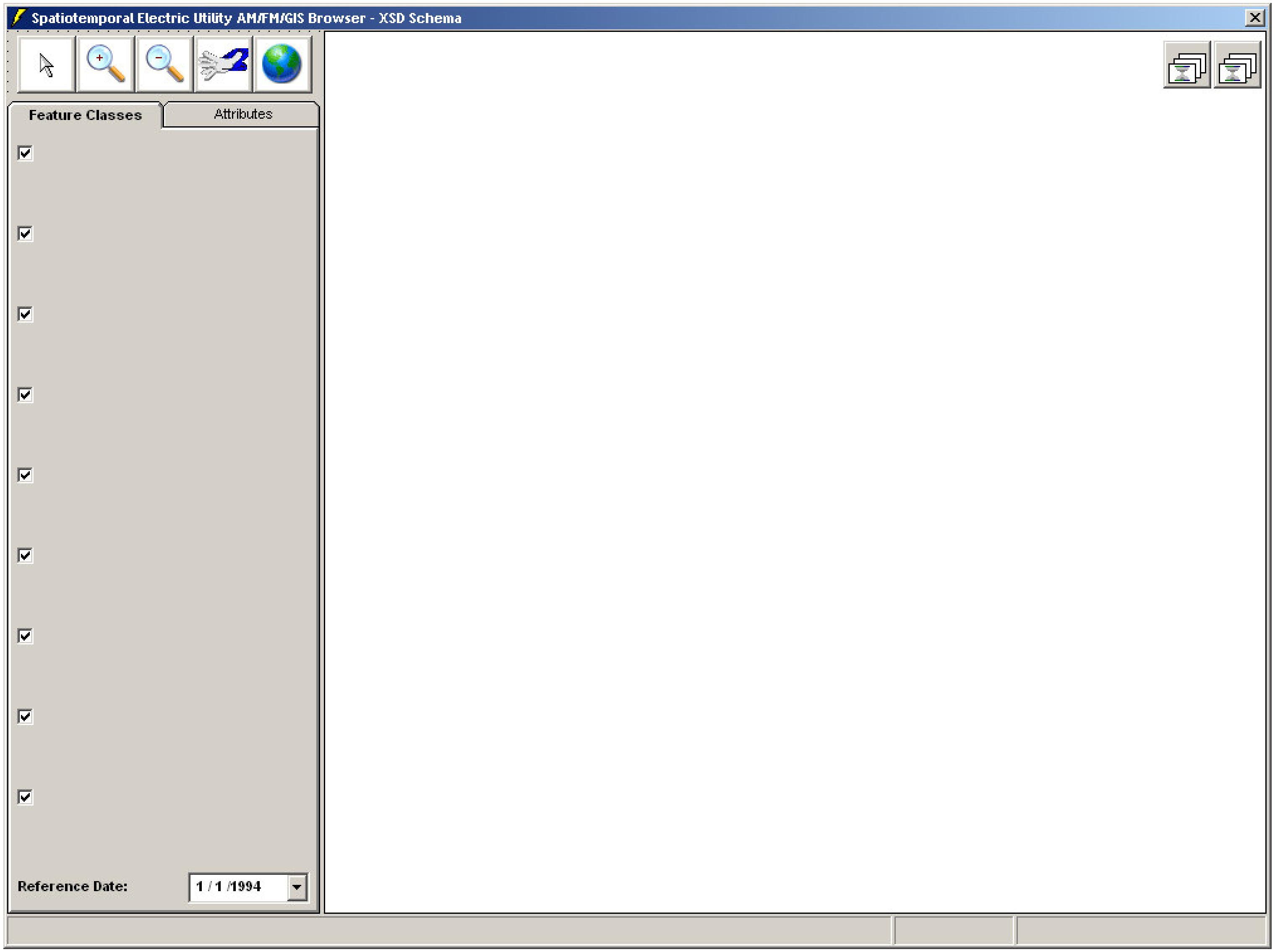
Choose the browser version:

Figure 4.19: Selection menu to choose the GIS version.

version of the GIS uses XML files for persistence. Because both GIS are functionally identical, they will be described together at an architectural level. Internal design differences between the GIS to accommodate one or the other of the persistence models will be deferred to the results and discussion section of this dissertation as these differences belie the fundamental research question of this work. The relational version of the GIS is listed in appendix Q. The XML version of the GIS is listed in appendix R.

Each GIS consists of a single form that is divided into a map area, a toolbar and two tabs. Figure 4.20 illustrates the form layout of the GIS with the first tab active. Figure 4.21 illustrates the form layout of the GIS with the second tab active. As can be seen from Figure 4.21, the design version of the form does not have feature classes hard coded, but has allocated up to nine feature class control locations where the drawing of the feature class can be enabled or disabled. At the top of the form is the toolbar with five tools. The first of these tools is the selection tool, which allows for selection of a single feature from the map. The remaining tools are map extent tools to allow zooming in and out of the map, panning the map, and resetting the map extent to the full extent of the data. Both the feature classes to be loaded and the full extent of the data are stored in the catalog tables. At the bottom of this tab is a calendar control that allows the user to set the current reference date for the system. Using this date, the system can be set to have a different operational date to allow for viewing and querying of data at any time in the life of the database. By default the reference date is set to the current date.

Dominating most of the GIS is the map drawing area; in this area the map is drawn with respect to the current map extent (set using the zoom and map extent tools). Finally, at the bottom of the form is a status bar that provides system messages to the user as well as the current map position and map scale.



Spatiotemporal Electric Utility AM/FM/GIS Browser - XSD Schema
Feature Classes
Attributes
☑
☑
☑
☑
☑
☑
☑
☑
☑
Reference Date:
1 / 1 /1994

Figure 4.20: Form Design for GIS and GIS Table of Contents.

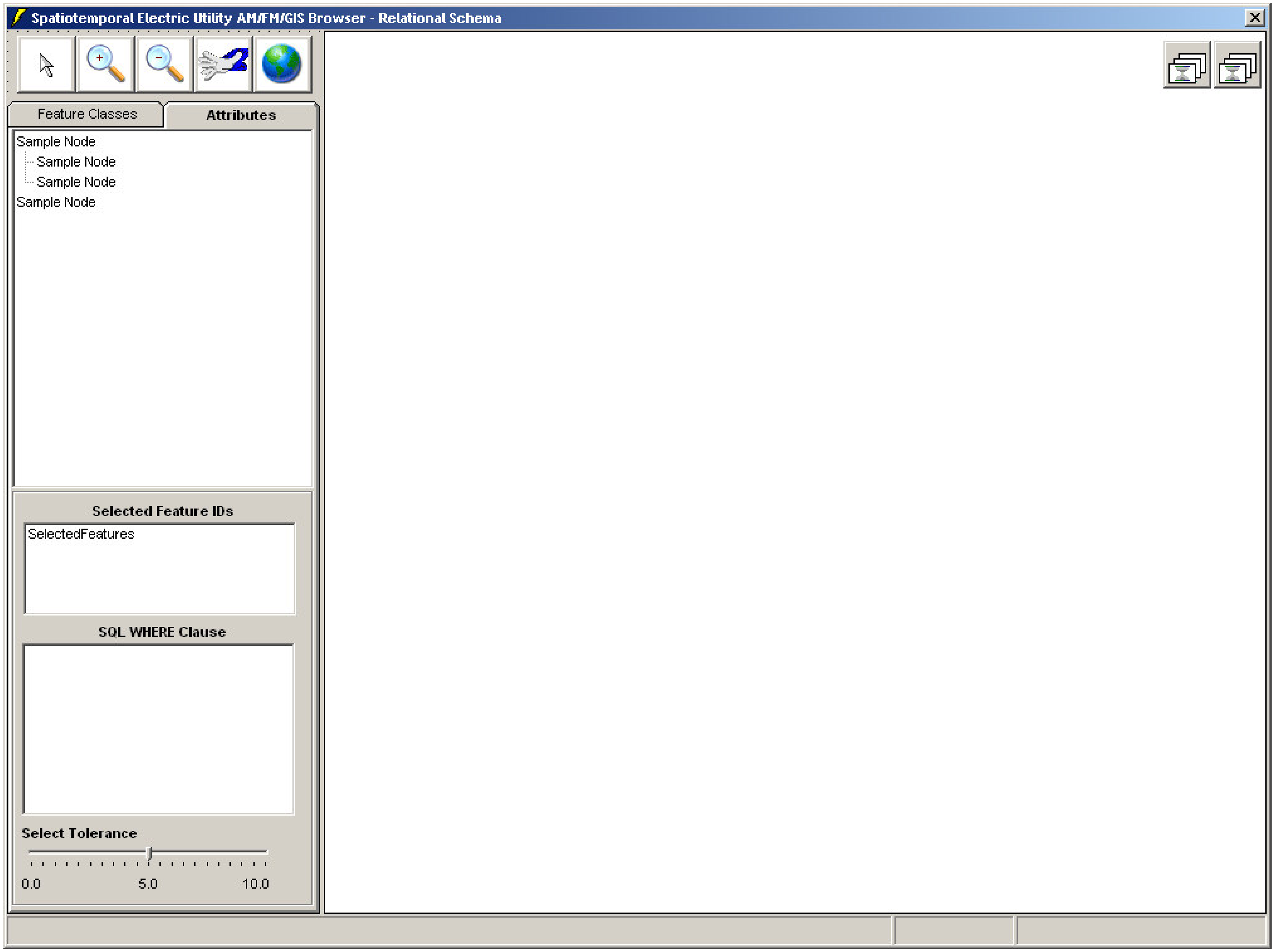
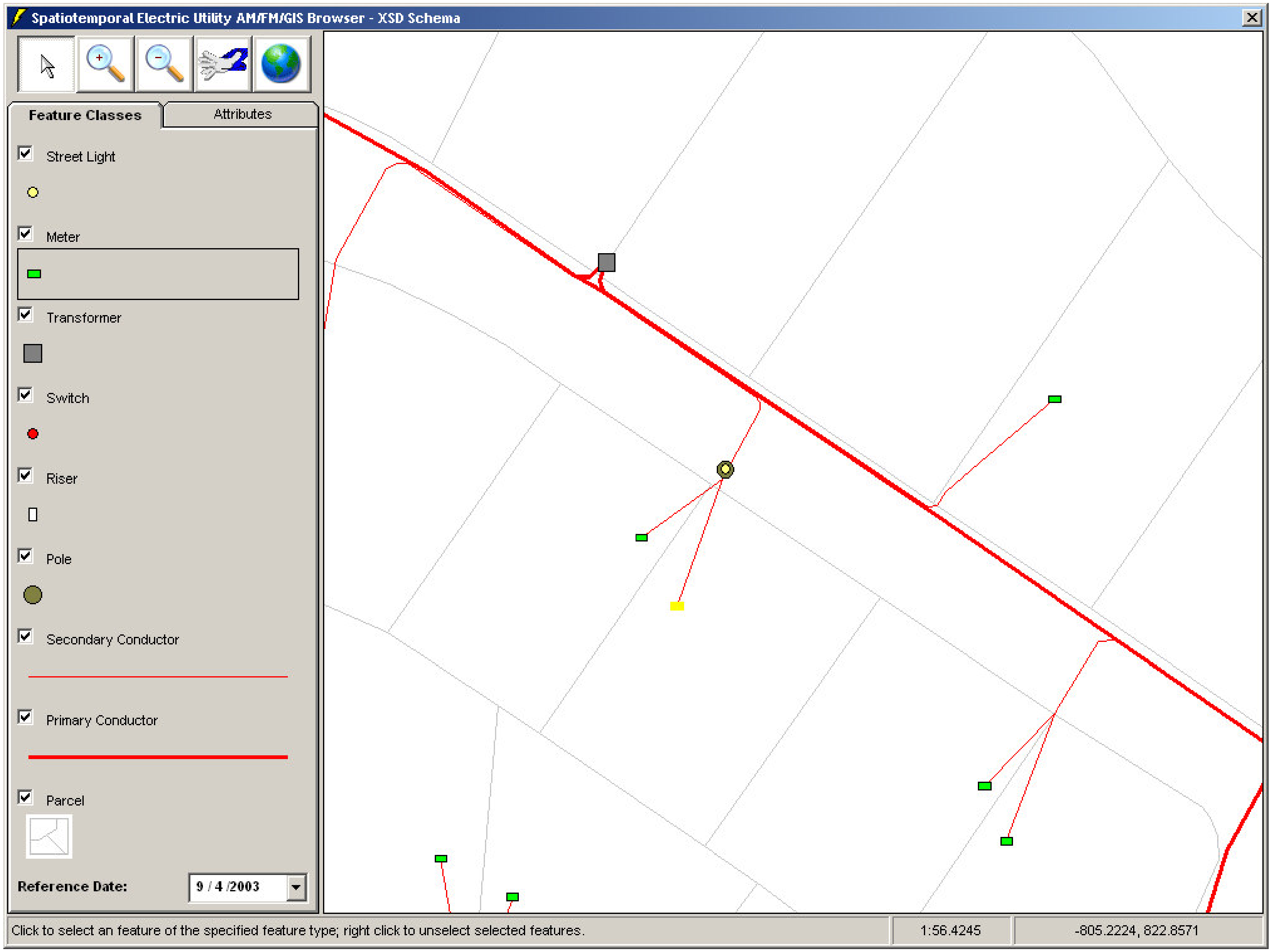


Figure 4.21: Form Design for GIS and GIS Attributes and Query.

The remaining functionality is provided on the second tab, which is divided into four sets of controls. The first control is a tree list that lists the attributes for the currently selected feature, which may be chosen either by clicking on the map to select a single feature or through a user-defined query. The second control is a listbox that lists all of the features selected through either a spatial or attribute query; although multiple features may be listed in this box only one set of attributes may be viewed at a time. The current set of attributes can be changed by selecting a different feature id from this list. The third control allows a user to specify a query in the supported query language of the system; for the relational version of the GIS this is a SQL query and for the XML version this is an XPath query. In keeping the design as simple as possible, this query definition box only allows the user to define the predicate of the query and not the subject. For the SQL query, this corresponds to the WHERE clause, while for the XPath query this corresponds to parameters between a pair of [ ] braces. The final control is a slider that allows the user to adjust the tolerance of the spatial selection.

Functionally these GIS are quite simple and support only a limited range of tasks that are typical of those that might be found in an online kiosk-style application. However, internally a significant amount of design was required to seamlessly support the same capabilities in both GIS. Figures 4.22 and 4.23 illustrate the running GIS with data loaded; note that Figure 4.22 illustrates the table of contents for the XML version of the GIS with a feature selected while Figure 4.23 illustrates the attributes of the same feature drawn from the relational database. This shows clearly that the XML schema can support the same range of functionality as the relational schema.The most involved aspect of the GIS development was the accessing of related data. Because the relationship between the source table and the related table could be many-to-many, depending upon the particular arrangement of records, a SQL join was impractical. For this reason, the key value was extracted from the source table and used to query the related table directly. These related records were then added to the


Spatiotemporal Electric Utility AM/FM/GIS Browser - XSD Schema
Feature Classes
Attributes

☑ Street Light ○

☑ Meter 🟩

☑ Transformer ■

☑ Switch ●

☑ Riser ▯

☑ Pole ●

☑ Secondary Conductor

☑ Primary Conductor

☑ Parcel

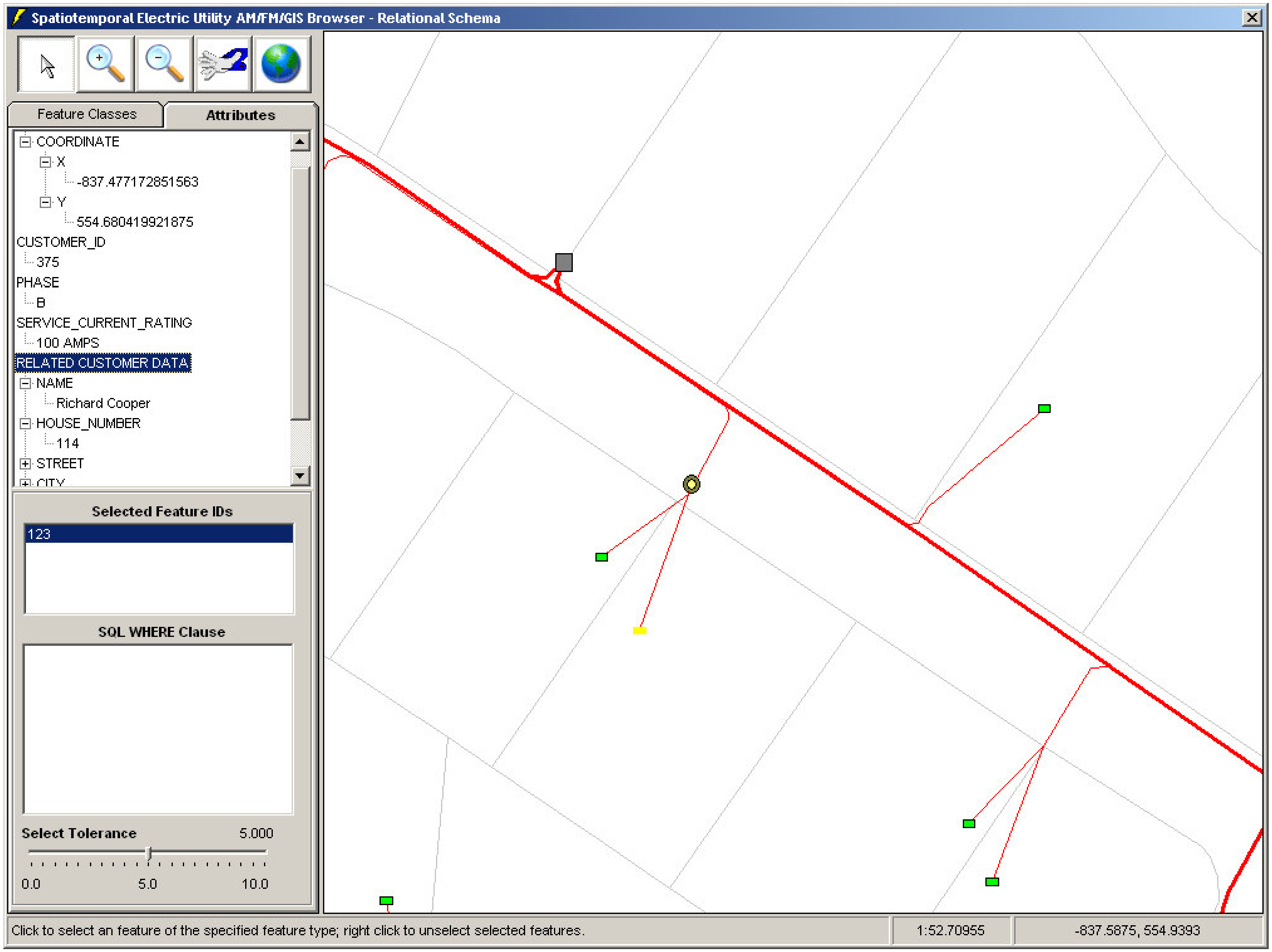
Reference Date: 9 / 4 /2003

Click to select an feature of the specified feature type; right click to unselect selected features.

1:56.4245

-805.2224, 822.8571

Figure 4.22: Running GIS – XML Version.



Spatiotemporal Electric Utility AM/FM/GIS Browser - Relational Schema

Feature Classes | Attributes

```
⊟ COORDINATE
   ⊟ X
      -837.477172851563
   ⊟ Y
      554.680419921875
CUSTOMER_ID
   375
PHASE
   B
SERVICE_CURRENT_RATING
   100 AMPS
RELATED CUSTOMER DATA
⊟ NAME
   Richard Cooper
⊟ HOUSE_NUMBER
   114
⊞ STREET
⊞ CITY
```

**Selected Feature IDs**

123

**SQL WHERE Clause**

**Select Tolerance** 5.000

0.0 5.0 10.0

Click to select an feature of the specified feature type; right click to unselect selected features. | 1:52.70955 | -837.5875, 554.9393

Figure 4.23: Running GIS – Relational Version.

attribute list along with the records from the source table.  A similar method was used with the related XML table due to some incomplete language features for relating XML tables.  These design considerations will be more thoroughly reviewed in the results and discussion.

# CHAPTER 5

# RESULTS AND DISCUSSION

There are several criteria that are explored here in order to qualitatively compare development of the XML-based prototype spatiotemporal GIS to the relational spatiotemporal GIS. These are the semantic clarity of the data store, ability to accurately represent spatiotemporal data and ease of development in a standard development environment such as Visual Basic. Also evaluated are the space requirements for persistence of the data and ubiquity in industry.

## Ubiquity

The question of ubiquity is by far the easiest to answer. The relational database model has been in existence for more than thirty years and has certainly achieved ubiquity throughout the information technology sector. However, research has shown that the relational model is insufficient semantically and analytically for representation of spatial or temporal data. A number of extended-relational models have been presented; of these, virtually none have been readily accepted in the marketplace. The object-oriented database model clearly has both the semantic and analytical capabilities to persist spatial, temporal and spatiotemporal data. However, acceptance of the object-oriented database outside of a few specialized areas has been limited, thus limiting the applicability for an object-oriented persistence scheme in general industry.

By far the most commercially successful method for persisting spatial data has been the object-relational hybrid, particularly from vendors such as ESRI and Oracle with the ArcSDE and Oracle Spatial products. The object-relational hybrid has sufficient semantic expressiveness to represent spatial data, and specialized spatial operators have been built into these products to enable efficient and reliable access. There are no similar products for temporal databases and certainly none for spatiotemporal databases. Thus, for current needs, the object-relational approach is insufficient for persistence of this data and we must therefore choose between object-oriented or relational models with the characteristics previously discussed.

XML has achieved ubiquity in the marketplace as an efficient transport mechanism, however it has not yet gained widespread acceptance for persistence. This is in the process of changing, however, as the database industry embraces XML in the database – either through the relational model as expressed by SQL/XML or through the native XML database as expressed by XQuery. Although both standards are still incomplete, the appearance of XML in the offerings of database vendors has already signified acceptance of this mechanism as an important future development in the industry. The flexibility of XML to model virtually any domain – including spatiotemporal data, as demonstrated in this research – means that XML has great promise for representing spatial data, and advances in XML databases further presuppose the same promise for persisting data.

## Space Requirements

The question of space requirements for persisting XML data is dependent on the data modeled. However, in this research, it was found that space requirements were less for for an equivalent non-extended relational model. The relational data was stored in 24 tables in a Microsoft Access database (which is comprised of a single disk MDB file). The space requirements for the Microsoft Access database were 2.97 MB for this particular prototype. In comparison, the XML data was stored in 11 XML disk files and 11 XSD disk files (for the schema definition) that required 1.85 MB. The space requirements for persistence in either an extended-relational model or an object-relational model were not explored.

## Accurate Spatiotemporal Representation

In both the relational and the XML data models, the spatiotemporal data were accurately represented and there was no loss of information resulting from either data model. In the relational model, multiple tuples stored each version of the feature in the database. As each tuple was written, the previous tuple was accurately marked as retired so that there was no confusion as to which tuple existed at which time. In the XML model, versioned attributes were used to achieve the same effect. Each attribute was marked with an appropriate retirement date as a new attribute was added, so that existence of attributes was also unambiguous. Thus, the accuracy of both representations was equal.

## Semantic Clarity

Semantic clarity is harder to assess, and is admittedly subjective. However, in viewing the final data in the relational database and the XML files (and particularly with the use of a tool such as XMLSpy for tabular viewing of the XML files), we felt that semantic clarity was much higher in the XML store than in the relational store.

The persistence of this spatiotemporal data in the relational schema required two tables to the single table required by XML. This is addressed in an extended-relational model (at the expense of ubiquity) and so cannot be held as a serious impediment. Nevertheless, in our non-extended schema, we required two tables while in XML (without extensions), we were able to model the spatial data as a subset part of the feature as it rightly is.

Also, in our non-extended relational model, characteristics of spatial data were separated between the two tables, with summary information in the feature table and vertices in the vertex table. This again is handled by extended-relational models at the expense of ubiquity. It is also adequately handled by the XML model where spatial characteristics such as area or length can be stored with the vertices.

The main area where the XML model proved to be much clearer semantically was in the issue of temporality. In the relational model (and this would also be the case in an extended-relational model for spatial data), storage of versions of a feature required multiple tuples and required the user to make the logical association between these multiple versions and the feature. Moreover, multiple versions of a feature are not multiple features, and yet the semantics of the relational model treat equally different versions of features and different features – both are represented by discrete tuples in the database. The relational model also treats the temporal data implicitly as having the same structure as spatial and attribute data, which is clearly counterindicated in the literature, by placing these attributes at the same location in the schema as the rest of the attributes.

The XML model much clearly separates the features from their constituent versions by storing features at the highest level in the hierarchy and versioning attributes within the context of the hierarchy instead of at an equal level. A review of the attributes that changed at a particular time is also clearer, as only the changed attribute has a new time assigned to it. In the relational model, the entire tuple is versioned, which requires the user to scan the complete tuple to determine the change made to it.

Finally, the feature identifier itself is more appropriately represented as a meta-attribute of a feature by not placing it on equal footing with the actual feature's attributes. The combination of these factors makes the XML document much easier to view and understand without the use of specialized software.

## Ease of Software Development

Because a complete development process was undertaken for both database models, this researcher is well able to make assertions regarding the facility with which client software was developed for each of these data models. Most commercial software developers are familiar with data access routines for relational database systems; one of the most popular of these is the Open Database Connectivity standard (ODBC) found in Microsoft software and operating systems. This standard is not unlike similar standards found in other languages, such as JDBC for Java, but there are subtle differences.

In comparison, the Document Object Model (DOM) has emerged as one standard for reading and writing XML files. Other similar standards are under development by the World Wide Web Consortium (W3C). Because DOM and similar standards are maintained by an independent party, the data access routines for XML documents is actually much simpler from platform to platform; the DOM is similar in Visual Basic, Java and C++ and much less dependent on vendor implementations. This is one immediate benefit of XML based persistence.

The other question that must be addressed then is the comparison between a database access methodology such as ODBC and XML access through DOM. We address this question by comparing actual snippets of code from both versions of the GIS developed in this work.

The code snippet required to open a connection to an ODBC database, in this case Access, is listed in Figure 5.1. The commensurate code snippet for XML access is listed in Figure 5.2. The relational version requires that a SQL string be created, a connection object to the database and a recordset object to hold the data. The SQL string is populated and supplied to the recordset, which is opened. The XML code snippet instantiates an XML document object and opens it. While this may seem like less lines, in reality the XQuery specification will likely provide new access routines that are more similar to SQL access routines than to current methods to open XML documents.

```
Dim conn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim sqlString As String
Conn.open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=DB.mdb"
SqlString = “SELECT * FROM TABLE”
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
```

**Figure 5.1: Code Snippet to Open an ODBC Connection.**

```
Dim XMLDoc As New DOMDocument40
XMLDoc.Load “table.xml”
```

**Figure 5.2: Code Snippet to Open an XML Document.**

The data access routines are slightly different between SQL and XPath, but XPath is not more complicated. Opening and querying data with SQL requires the use of a recordset object, which can then be accessed through use of a cursor. Values can be accessed through the cursor. Querying an XML document with XPath requires the use of an XML node object, which also can be looped through. Thus the mechanism for both SQL and XPath, while technically different are conceptually similar. Figures 5.3 and 5.4 illustrate the SQL and XPath access routines respectively.

```
Dim rs As New ADODB.Recordset
rs.open “select * from table”, Conn, adOpenKeyset, adLockOptimistic
rs.MoveFirst
Do While Not rs.EOF
    Debug.Print rs("VAL").Value
    rs.MoveNext
Loop
rs.Close
```

**Figure 5.3: Code Snippet to Query a Relational Database.**

```
Dim XMLDoc As New DOMDocument40
Dim XMLNode As IXMLDOMNode
XMLDoc.Load “table.xml”
For Each XMLNode In XMLDoc.selectNodes(“*/val”)
      Debug.Print XMLNode.Text
      Debug.Print XMLNode.Attributes.getNamedItem(“attribute”)
Next XMLNode
```

**Figure 5.4: Code Snippet to Query an XML Document.**

The remainder of the software functionality is virtually identical in both the SQL and XPath versions; only the data access routines are different. The net result is that software development, while not easier with XPath, is no more difficult. Software development is comparable in both models.

# CHAPTER 6

# CONCLUSIONS, ISSUES AND FUTURE DIRECTIONS

The prototype XML-based spatiotemporal GIS developed in this work has shown that development of an XML based spatiotemporal data store for is feasible and achievable using current technology. Although there are still incomplete parts of the XML standard languages, particularly the XQuery standard, development of an XML based store is still possible simply using the built-in facilities of the language (particularly XPath). While some database operations are convoluted (particularly with regards to linking tables), the semantics of the store are certainly clearer when stored in XML than when stored in an equivalent relational database. The sheer volume of work that has been dedicated to the various data models in relational and extended-relational form belies the difficulty in conceptually clearly and semantically accurately modeling such spatiotemporal data in such a structured data store. The semi-structured and customizable nature of XML is better suited to the semi-structured and gradually evolving data of the real-world as expressed in time and space.

To review the objectives of this dissertation, they are restated here:

1. Develop an XML based data structure for representing spatiotemporal data. The usefulness of XML as a mechanism for efficiently persisting temporal data is postulated based upon a variety of contributing factors. First, the most efficient database structures are hierarchical, and indeed indexing data structures such as B+-Trees, R-Trees or Quadtrees are often based upon these hierarchical structures. Second, set

theory dictates the unique existence of an object. However, many versions of an object (as is the case with temporally extended features) require non-uniqueness of an object - a so-called bag. XML supports these bags better than do relational models. Third, cognitive category theory is based upon a hierarchical structure; and it has been shown in the cognitive science literature that cognitive economy is achieved by traversing multiple levels of a knowledge tree.

2. Develop a relational schema for a simple electric distribution GIS with temporal attributes and design the queries that would be required in order to access records in the database. This relational schema will contain structures such as poles, electric devices such as transformers, electric conductor (power lines) and customer data.
3. Develop a fictional spatiotemporal GIS based upon the relational schema. This GIS shall be capable of drawing the current state of the GIS at a particular time and viewing the attributes of a feature at that time. The GIS shall also be capable of querying for spatial or thematic attributes. The GIS will support a series of queries designed to demonstrate some of the difficulties of dealing with temporal data.
4. Develop an XML based schema for the same simple electric distribution GIS with temporal attributes and design the queries that would be required to access records in the database.
5. Develop a prototype spatiotemporal GIS based on the XML schema with the same target functionality as the relational spatiotemporal GIS. Compare and contrast the complexity of the XML-based GIS software to the relational GIS software. Also compare and contrast the two development processes and the resultant data repositories.

With regards to the first objective, a sufficiently generic XML based data structure was constructed to model several different data types including points, lines, polygons and non-spatial object classes. Temporal attributes were organized as a sub-hierarchical level of each object such that changes in state for any attribute (whether spatial or aspatial) generated a new node in the hierarchy. Thus objects, attributes and temporality were chosen as the fundamental hierarchical organization, with space treated as an attribute on equal par with aspatial attributes.

The capability to store bags (for non-unique or repeating attributes) was particularly important. Because of the selection of time as the fundamental node in the hierarchy, it was guaranteed that attributes would be non-unique in each collection. Each attribute may occur multiple times at discrete temporal locations, so for example a SHAPE attribute (to store the spatial location) may exist multiple times on the same object node, with individual temporal times. The capability to store not only tags (columns in the relational model) and data (rows in the relational model), but attributes or metadata about the specific instance of that tag and data provides the capability for better data definition that ultimately supports this property of non-uniqueness. In actuality, the combination of all three – tags, data and attributes – are themselves unique, but because elaboration is possible over the relational model of simple rows and columns, data that are really unique – but cannot sufficiently be represented as unique in the relational model – may be adequately modeled in XML.

Hierarchical spatial reasoning and category theory contributed to the substantive development of this XML data structure. Many of the same factors that make efficient the cognitive process also drive efficiency in terms of searching a hierarchically organized set of data. Category theory shows that people fundamentally organize their world into a hierarchy of objects. Although not explicit as part of the XML data structure, this organization is implicitly present in the separation of object classes into discrete documents or tables. Because XML is semi-structured, objects of different types may be stored as nodes of the same root – with different schema. Initial prototyping of the data structure produced a model where all objects were stored together and it was quickly obvious that each data type should be stored on a separate node separately for semantic clarity and for efficient search access to the data, as shown in Figure 6.1. Data types were divided into different documents because it limited the amount of data that needed to be loaded in order to complete a search (only the poles, for example). This is in reality the same strategy as dictated by cognitive economy – only data needed for a particular cognitive process are loaded.

In terms of modeling temporality, two designs were considered. In the first design, birth and retirement metadata are stored as attributes of the associated attribute tag. In the second design, each attribute is modeled singularly as a node in the hierarchy with discrete revision nodes under each attribute node. For this application, the first of these designs was selected principally because it simplified the overall model and the path access statements to query the model. However, the second design is equally valid and has the additional advantage that temporal indexing would be more straightforward (although this was not an issue in this prototype as indexing was not implemented). The benefit of XML is that either of these designs is equally expressable in a schema. Figures 6.2 and 6.3 illustrate these two design strategies for comparison.

```
<data>
      <pole/>
      <conductor/>
      <switch/>
</data>
```

**Figure 6.1: Node Organization of Data Types.**

```
<feature id=”1”>
      <attribute>
            <revision born=”031001” retired=”031002”>value</time>
            <revision born=”031002” retired=”031003”>value</time>
      </attribute>
</feature>
```

**Figure 6.2: Temporal Attributes as Nodes.**

```
<feature id=”1”>
      <attribute born=”031001” retired=”031002”>value</attribute>
      <attribute born=”031002” retired=”031003”>value</attribute>
</feature>
```

**Figure 6.3: Temporal Attributes as Tags.**

The second through fifth objectives were all accomplished cooperatively because of the interdependence among them. In order to address the second objective, the ArcGIS Electric Distribution data model (ESRI and Miner and Miner, 2001) was used as the basis for developing a simple schema for electric distribution. Although the ArcGIS Electric Distribution data model contains approximately 40 feature and object classes, a subset of only 10 were selected and individual attributes were chosen from within each of the selected feature and object classes to limit the amount of data to be generated. Selection of attributes was limited to the most important attributes for each type of feature, such as phase designation for electric features or material for structure features. Addition of temporal attributes required extending the table structure to include birth and retirement. However, the structure of the relational schema made it necessary to utilize a complex key of FEATURE_ID, BORN and RETIRED and required one-to-many type relationships to completely model the temporality. The queries required to access this data were standard SQL queries and were defined when the spatiotemporal GIS software was developed. The third objective was completed by constructing the spatiotemporal GIS software in Visual Basic, using Microsoft Access as the database server. Construction of the spatiotemporal GIS was straightforward and although the queries were complex and would be semantically unclear to someone not familiar with the schema, they were straightforward to define.

The fourth objective was completed by mapping the relational schema into the XML model developed in the first objective. This was straightforward and done by exporting the data from the relational schema in an XML form that fit the schema developed to match each of the feature classes. As described previously, each feature class was stored in a separate document, and for each a separate XSD schema was defined to allow for validation of the data in the document as complete and correct and also to allow for use of the schema by the software in a manner not unlike the schema catalog maintained by relational systems. The resultant database of XML documents clearly illustrated the benefit of XML over relational models for representing complex temporal data. Viewing of the XML database as compared to viewing of the relational database was more natural and required much less interpretation on the part of the observer because of the hierarchical organization of the data and elimination of the need for one-to-many relationships. Development of the queries for this XML schema was carried out for completion of the fifth and final objective and was found to not be simpler than developing the SQL queries, but was not more difficult either. The semantic clarity of the queries themselves was not greatly increased. However, development of the current generation of XML query languages may improve this situation. Development of the prototype XML-based spatiotemporal GIS required replacement of data access routines from the existing relational spatiotemporal GIS and development of both versions proved to have about the same level of complexity. Thus, development effort stayed relatively constant for both the relational and the XML-based spatiotemporal GIS. However, viewing of the native form of the data, without the GIS, clearly indicated the advantages of XML over relational models.

There are, of course, a number of issues that this work has not explored. The immaturity of the XQuery language presents problems for complex relational models.

However, this standard is near final revision and this problem should be mitigated soon. Understanding the implications of the XQuery language as it pertains to spatiotemporal data will be key and will likely require as much investigation as has been devoted to this issue in the relational database literature.

This work also has only explored valid time. Any actual XML-based spatiotemporal GIS implementation would require capabilities to store both transaction time and valid time. Extension into two discrete temporal axes would have complicated the design beyond the desired ends for this research. Exploration of inclusion of these temporal domains is necessary for a real-world implementation.

This research also did not explore the implications of a writable database. This research was limited only to read-only access to the database. Of course, at the current time, the XQuery language also does not define definition or manipulation capabilities and is itself read-only; the current XML standard to write XML databases requires typically interaction through an API such as DOM (Document Object Model) or SAX (Simple API for XML). This is a deficiency in the XML language that has not yet matured.

This research also considered only a limited scope of spatial data. The work by Usery *et al.* (forthcoming) has a much wider set of options for data representation. In particular, their work also explores modeling of raster data, geospatial data (as distinct from spatial data by incorporating projection information) and true mathematical formulae for better representation. This work focused entirely on simple vector features. Persistence of true spatiotemporal data will need to include these types of features. The purpose of this work was simply to show proof of concept in development of an XML-based prototype.

The same problems that currently plague the relational model in terms of spatial (and temporal) operators are still present in the current XML implementation. Standards such as GML do mitigate some of these problems, but further operators will need to be defined. Unlike the SQL language, however, there are more options for accessing XML data including XPath, XQuery and XSLT. A combination of these will allow for more integrated development of operators than is possible in SQL.

Finally, this research has not considered physical database issues. There is a whole body of research aimed at the physical level of the database. Transaction management is necessary in order to insure ACID properties in the database (atomicity, consistency, isolation and durability) so that changes made – particularly in multi-user situations – are not dependent upon the order of operations made by the user. Query optimization and indexing provide improved performance over full scan of data and must also be considered for aspatial, spatial, temporal and spatiotemporal indices. Memory management – both dynamic memory and disk space – are also key. In this research, the entire table was loaded into memory and accessed through the Document Object Model. In a real implementation, it would be more likely that some part of the document would be dynamically loaded – this is key for very large data sets – and dynamic memory would be paged in and out as required. Memory management would allocate the paging space required in the database to load the data and would manage whether such pages were clean or dirty (i.e. whether the data block had been written to since it was loaded) and manage the commensurate commitment of dirty pages back to the database. Current native XML database offerings have already explored some of these issues.

This research did successfully demonstrate that current capabilities in the XML language would support persistence of spatiotemporal data for read-only access and would support creation of a prototype spatiotemporal GIS. It has been shown that an apples-to-apples comparison where the underlying relational database also did not have specialized data structures or indexing provided comparable performance to an underlying XML data store (in actuality, the XML was slightly – but inconsequentially – faster). Drawing was readily accomplished, as was querying, and table joins were also accomplished – albeit in a less semantically clear way than the rest of the operations. XML holds great promise for persisting complex data such as that in the spatiotemporal model and is already ubiquitous in the industry, unlike pure object-oriented databases.

There are a number of potential future directions for this work. First, each of the issues expressed above must be explored before successful adoption of the model. Second, as the GML (Geography Markup Language) matures, it is likely that the GIS community will see better adoption of and support for XML in spatial databases. The inclusion of temporal units of measure thus far forbears better support in later versions. It is thus conceivable that native XML databases may support spatiotemporal data. Of course, GML is simply itself a dialect of XML and thus the above issues equally apply to it as well.

The development of XML originally as an internet language has also endowed it with certain capabilities because of that fact. The existence of the internet as a low-bandwidth communication medium (with most people still connecting at or below 56 KBPS) has stimulated investigation of efficient mediums of transport. Geospatial and spatiotemporal data in particular have benefited from this because they are already large volume data sets. The Standardized Vector Graphics (SVG) language is based on XML and SVG browsers that extend current internet browsers are already available. SVG can be transmitted as plain-text and the browser can handle the work of rendering the data. Using existing XML techniques such as XSLT, an XML or GML document can be transformed to SVG and sent to the browser for rendering. This holds great promise for internet-connected and mobile devices. It also takes the burden of graphic implementation off the database and puts it in the browser. Current research in the direction of SVG will also benefit such spatial and temporal data.

Of course, the benefits of spatiotemporal representation has implications far beyond the limited application illustrated here for AM/FM/GIS. In the first case, AM/FM/GIS itself is broader than presented here and includes additional types of utilities as well as other model domains where management of facilities from  a spatial perspective is beneficial.  Obviously, there are applications here.  However, any aspect of geographic representation would benefit from an ability to manage data not only spatially but also temporally.  Geographic data change with time, and as Einstein demonstrated space and time are inextricably intertwined. Additional domains include disparate areas from demographics to emergency management; from storage and analysis of satellite imagery (with an appropriately expanded study of such persistence) to forestry.  In short, any storage of spatial data would benefit from an ability to store representations of that data as it changes with time for benefits from historical analysis to predictive modeling.  While the ability to model such data is key, the ability to store that data in a long-term stable way is just as important in order to obviate the need to remodel data each time.  Relational data stores simply are not up to the task of modeling the semantic complexity of spatiotemporal data.

The industry as a whole has seen a shift from modeling structured data in relational systems to modeling unstructured or semistructured data.  The adoption of XML has already begun both in information science in general and in geographic information science in particular.  XML has moved into the database and the benefits of XML for modeling spatiotemporal data in a semantically clear way far supersede any potential that the relational model offers.

# REFERENCES

Anderson, J. 2000. *Cognitive Psychology and Its Implications, 5th Edition*. Worth Publishers, New York.

Atkinson, M., F. Banchilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik. 1989. The Object-Oriented Database System Manifesto. *ALTAIR Technical Report.* 30(89).

Beckmann, N., H. Kriegel, R. Schneider and B. Seeger. 1990. The $R^*$-Tree: An Efficient and Robust Access Method for Points and Rectangles. *Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data*. pp. 322-331.

Bernhardsen, T. 1992. *Geographic Information Systems*. Arendal, Norway: Viak IT.

Berry, B. 1964. Approaches To Regional Analysis: A Synthesis. *Annals, Association of American Geographers*. 54:2-11.

Block, R. 1998. Psychological Time and the Processing of Spatial Information. In M. Egenhofer and R. Golledge (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*, Oxford University Press, New York, pp. 119-130.

Bisseret, A. and C. Montarnal. 1996. Linearization in Spatial Descriptions: Tour or Hierarchical Structures? *Current Psychology of Cognition.* 15(5): 487-512.

Brinkhoff, T. and J. Weitkämper. 2001. Continuous Queries within an Architecture for Querying XML-Represented Moving Objects. Lecture Notes in Computer Science.

A. Bonifati and D. Lee. 2001. Technical Survey of XML Schema and Query Languages. *Technical report, UCLA Computer Science Department.*

Boroditsky, L. 2000. Metaphoric Structuring: Understanding Time Through Spatial Metaphors. *Cognition.* 75(1): 1-28.

Brusoni, V., L. Console, P. Terenziani and B. Pernici. 1999. Qualitative and Quantitative Temporal Constraints and Relational Databases: Theory, Architecture and Applications. *IEEE Transactions in Knowledge and Data Engineering.* 11(6): 948-968.

Burrough, P. 1986. *Principles of Geographical Information Systems for Land Resources Assessment*. New York, NY: Oxford University Press.

Darwen, H. and C. Date. 1995. The Third Manifesto. *SIGMOD Record*. 24(1):39-49.

Car, A. and A. Frank. 1994. General Principles of Hierarchical Spatial Reasoning - The Case of Wayfinding. In: *Advances in GIS Research, Proceedings of the 6th Symposium*. 2: 646-664.

Car, A. (1998). Hierarchical Spatial Reasoning: a GeoComputation Method. The 3rd *International Conference on GeoComputation*, University of Bristol, United Kingdom. GeoComputation CD-ROM.

Car, A., G. Taylor and C. Brunsdon. 2001. An Analysis of the Performance of a Hierarchical Wayfinding Computational Model Using Synthetic Graphs. 25(1): 69-88.

Chaudhri, A., A. Rashid and R. Zicari (eds.). 2003. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley, New York, New York.

Christakos, G. 2000. *Modern Spatiotemporal Geostatistics.* Oxford University Press, New York, New York.

Christakos, G., P. Bogaert and M. Serre. 2001. *Temporal GIS: Advanced Functions for Field-Based Applications*. Springer-Verlag, New York, New York.

Codd, E. 1970. A Relational Model for Large Shared Databanks. *Communications of the ACM*. 13(6): 377-390.

Codd, E. 1971a. Normalized Data Base Structure: A Brief Tutorial. *IBM Research Report RJ935*.

Codd, E. 1971b. Data Base Sublanguage Founded on the Relational Calculus. *IBM Research Report RJ909*.

Coppock, J. and D. Rhind. 1995. The History of GIS in Geographical Information Systems – Principles and Applications, in D. Maguire, M. Goodchild and D. Rhind (ed.). New York: 21-43.

Couclelis, H. and N. Gale. 1986. Space and Spaces. *Geografiska Annaler*. 68B-1, pp. 1-12.

Egenhofer, M. and A. Frank. 1989. Object-Oriented Modeling in GIS: Inheritance and Propogration. *Proceedings of Auto-Carto 9*, pp. 588-598.

Egenhofer, M. 1992. Why Not SQL! *International Journal of Geographic Information Systems.* 6(2): 71-85.

Egenhofer, M., D. Peuqeut, J. Glasgow, O. Gunther and J. Herring. 1999. Progress in Computational Methods for Representing Geographical Concepts. *International Journal of Geographical Information Science*. 13(8): 775-796.

Einstein, A. 1920. Relativity: The Special and General Theory. Henry Holt, New York, New York.

Elmasri, R., G. Wuu and V. Kouramajian. 1993. The Time Index and the Monotonic B$^+$-Tree. In: *Temporal Databases (*Tansel, Clifford, Gadia, Jojodia, Segev and Snodgrass (eds), pp. 433-456. The Benjamin Cummings Publishing, New York.

Elmasri, R. and B. Navathe. 2000. *Fundamentals of Database Systems, 3rd Edition*. Addison-Wesley, Reading, Massachusetts.

ESRI and Miner and Miner. 2001. *Electric Distribution: ArcGIS Data Models*. ESRI, Redlands, California.

Florescu, D. and D. Kossman. 1999. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. *Rapport de Recherche, Techreport 3680, INRIA.*

Frank, A. 1998a. Different Types of “Times” in GIS. In M. Egenhofer and R. Golledge (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*, Oxford University Press, New York, pp. 40-62.

Frank, A. 1998b. Formal Models for Cognition – Taxonomy of Spatial Location Description and Frames of Reference. In K. Wender (ed.) *Spatial Cognition*. Springer-Verlag, Berlin, pp. 293-312.

Frank, A. and M. Raubal. 1998. Formal Specification of Image Schemata - A Step Towards Interoperability in Geographic Information Systems. *Spatial Cognition and Computation*. 1(1): 67-101.

Freska, C. 1991. Qualitative Spatial Reasoning. In D. Mark and A. Frank (eds.), *Cognitive and Linguistic Aspects of Geographic Space*. Kluwer Academic Publishers, Netherlands, pp. 361-372.

Freska, C. 1997. Spatial and Temporal Structures in Cognitive Processes. In C. Freska, M. Jantzen and R. Valk (eds). *Foundations of Computer Science. Potential – Theory – Cognition*. Berlin: Springer-Verlag, pp. 379-387.

Freundschuh, S. and M. Egenhofer. 1997. Human Conceptions of Space: Implications for GIS. *Transactions in GIS*. 2(4):361-375.

Friedman, J., J. Bentley, and R. Finkel. 1977. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software.* 3(3): 209-226.

Goh, C., H. Lu, B. Ooi and K. Tan. 1996. Indexing Temporal Data Using Existing B$^+$-Trees. *Data and Knowledge Engineering*. 18: 147-165.

Goodchild, M. 1992. Geographical Information Science. *International Journal of Geographical Information Systems*. 6(1): 31-45.

Graham, S., A. Joshi and Z. Pizlo. 2000. The Traveling Salesman Problem: A Hierarchical Model. *Memory and Cognition.* 28(7): 1191-1204.

Gregerson, H. and C. Jensen. 1999. Temporal Entity-Relationship Models – A Survey. *IEEE Transactions in Knowledge and Data Engineering.* 11(3): 464-497.

Guttman, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. *Transactions of the ACM*. pp. 47-57.

Habel, C. and C. Eschenbach. 1997. Abstract Structures in Spatial Cognition. In C. Freska, M. Jantzen and R. Valk (eds). *Foundations of Computer Science. Potential – Theory – Cognition*. Berlin: Springer-Verlag, pp. 369-378.

Hadzilacos, T. and N. Tryfona. 1996. Logical Data Modeling for Geographic Applications. *International Journal of Geographical Information Systems*. 10(2): 179-203.

Hadzilacos, T. and N. Tryfona. 1997. An Extended Entity-Relationship Model for Geographic Applications. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 26(3): 24-29.

Hägerstrand, T. 1975. Space, Time and Human Conditions. In Karlqvist, A., Lunqvist, L. and Snickars, F. (eds.) *Dynamic Allocation of Urban Space*. Farnborough,: Saxon House, pp. 3 – 14.

Hazelton, N. 1998. Some Operational Requirements for a Multi-Temporal 4-D GIS. In M. Egenhofer and R. Golledge (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*, Oxford University Press, New York, pp. 63-73.

Hernandez, D. 1993. Maintaining Qualitative Spatial Knowledge. In A. Frank and I Compari (eds.) *Spatial Information Theory: A Theoretical Basis for GIS*, European Conference, COSIT '93, Mariana Marina, Elba Island, Italy. Springer-Verlag, Berlin, pp. 36-53.

Herring, J., R. Larsen and J. Shivakumar. 1988. Extensions to the SQL Query Language to Support Spatial Analysis in a Topological Database. *GIS/LIS '88, Proceedings of the Third International Conference.* pp. 741-750.

Hirtle, S. 1998. The Cognitive Atlas: Using GIS as a Metaphor for Memory. In: *Spatial and Temporal Reasoning in Geographic Information Systems* (M. Egenhofer and R. Golledge, eds.), pp.263-272. Oxford University Press, New York.

Huang, B. and H. Lin. 1999. Design of a Query Language for Accessing Spatial Analysis in the Web Environment. *GeoInformatica* 3(2): 165-183.

Jagadish, H., S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C.Yu. 2002. TIMBER: A native XML database. VLDB Journal. 11(4): 274-291.

Klopprogge, M. and P. Lockeman. 1983. Modeling Information Preserving Databases: Consequences of the Concept of Time. In *Proceedings of the 9th International Conference on Very Large Databases,* pp. 399-416.

Kuhn, W. and A. Frank. 1991. A Formalization of Metaphors and Image-Schemas in User Interfaces. In D. Mark and A. Frank (eds.) *Cognitive and Linguistic Aspects of Geographic Space*, Kluwer Academic Publishers, Netherlands, pp. 419-434.

Langran, G. 1989. A Review of Temporal Database Research and its use in GIS Applications. *International Journal of Geographic Information Systems*. 3(3): 215-232.

Langran, G. 1992. *Time in Geographic Information Systems.* Taylor and Francis, New York, New York.

Larson, J. 1995. *Database Directions: From Relational to Distributed, Multimedia and Object-Oriented Database Systems*. Prentice Hall, Upper Saddle River, NJ. 261 pp.

Lewis, H. and L. Denenberg. 1991. *Data Structures and Their Algorithms*. Harper Collins Publishers. New York, NY. 509 pp.

Lomet, D. and B. Salzberg. 1993. Transaction Time Databases. *Temporal Databases: Theory, Design and Implementation.* pp. 388-417.

Marian, A., Abiteboul, S., Cobena, G., Mignet, L. 2001. Change-Centric Management of Versions in an XML Warehouse. In: Proceedings of 27th International Conference on Very Large Data Bases, pp. 581—590.

Mark, D. 1993. Toward a Theoretical Framework for Geographic Entity Types. In A. Frank and I Compari (eds.) *Spatial Information Theory: A Theoretical Basis for GIS*, European Conference, COSIT '93, Mariana Marina, Elba Island, Italy. Springer-Verlag, Berlin, pp. 270-283.

McHugh, J., S. Abiteboul, R. Goldman, D. Quass, and J. Widom. 1997. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54-66.

McMaster, R. 1991. Conceptual Frameworks for Geographical Knowledge. In B. Buttenfield and R. McMaster (eds.) *Map Generalization: Making Decisions for Knowledge Representation* (London: Longman Scientific Publications), pp. 21-39.

Mennis, J., D. Peuquet and L. Qian. 2000. A Conceptual Framework for Incorporating Cognitive Principles into Geographical Database Presentation. *International Journal of Geographical Information Science.* 14(6):501-520.

Molenaar, M. 1991. Status and Problems of Geographical Information Systems: The Necessity of a Geoinformation theory. *ISPRS Journal of Photogrammetry and Remote Sensing.* 46: 85-103.

Nascimento, M. and M. Dunham. 1999. Indexing Valid Time Databases Via $B^+$-Trees. *IEEE Transactions on Knowledge and Data Engineering.* 11(6): 929-947.

Navathe, S. and R. Ahmed. 1989. A Temporal Relational Model and Query Language. *Information Sciences.* 49(1-3): 147-175.

Newcombe, N., J. Huttenlocher, E. Sandberg, E. Lie, and S. Johnson. 1999. What Do Misestimations and Asymmetries in Spatial Judgement Indicate About Spatial Representation? *Journal of Experimental Psychology: Learning, Memory, & Cognition*. 25(4): 986-996.

Nyerges, T. 1991. Representing Geographical Meaning. In B. Buttenfield and R. McMaster (eds). *Map Generalization: Making Decisions for Knowledge Representation* (London: Longman Scientific Publications), pp. 59-85.

Peuquet, D. 1984. A Conceptual Framework and Comparison of Spatial Data Models. *Cartographica.* 21(4): 66-113.

Peuquet, D. 1988. Representations of Geographic Space: Toward a Conceptual Synthesis. *Annals of the Association of American Geographers.* 78(3): 375-394.

Peuquet, D. 1994. It's About Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. *Annals of the Association of American Geographers.* 84(3): 441-461.

Peuquet. 2002. *Representations of Space and Time*. Guilford Press, New York, New York.

Rajabifard, A., F. Escobar and I. Escobar. 2000. Hierarchical Spatial Reasoning Applied to Spatial Data Infrastructures. *Cartography*. 29(2): 41-50.

Rinck, M., A. Haehnel, G. Bower, Glowalla and Ulrich. 1997. The Metrics of Spatial Situation Models. *Journal of Experimental Psychology: Learning, Memory, & Cognition*. 23(3): 622-637.

Pankowski, T. 2002. XML-SQL: An XML Query Language Based on SQL and Path Tables. *Lecture Notes in Computer Science.* 2490: 184-209.

Papadias, D., M. Egenhofer and J. Sharma. 1996. Hierarchical Reasoning About Direction Relations. In: *Fourth ACM Workshop on Advances in Geographic Information Systems*. 105-112.

Papidas, D. and M. Egenhofer. 1997. Algorithms for Hierarchical Spatial Reasoning. *GeoInformatica*. 1(3): 251-273.

Price, R., N. Tryfona and C. Jensen. 2000. Extended Spatiotemporal UML: Motivations, Requirements, and Constructs. *Journal of Database Management.* 11(4): 13-27.

Rosch, E. 1978. Principles of Categorization. In E. Rosch and B. Lloyd (eds), *Cognition and Categorization* (New York: Halstead Press) 27-48.

Saltenis, S. and C. Jensen 2002. R-tree Based Indexing of General Spatio-Temporal Data. *The VLDB Journal.* 11(1): 1-16.

Samet, H. 1990a. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA. 491 pp.

Samet, H. 1990b. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, MA. 499 pp.

Schoning, H. 2001. Tamino - A DBMS designed for XML. In Proceedings of the 17th International Conference on Data Engineering, pp. 149-154.

Sellis, T., N. Roussopoulus and C. Faloutsos. 1987. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. *Proceedings of the 13th VLDB Conference.* pp. 507- 518.

Shanmugasundaram, J., J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton, and I. Tatarinov. 2001a. A General Technique for Querying XML Documents Using a Relational Database System. *SIGMOD RECORD*, 30(3): 20-26.

Shanmugasundaram, J., J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. 2001b. Querying XML Views of Relational Data. In Proceedings of VLDB, pp. 261-270, Rome, Italy.

Shen, H., B. Ooi, and H. Lu. 1994. The TP-index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases. *Proceedings of the 10th IEEE International Conference on Data Engineering.* pp. 274-281.

Smith, T. G. Janee, J. Frew and A. Coleman. 2001. The Alexandria Digital Earth Prototype System. *In Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries, Roanoke, VA.* pp. 118-119.

Snodgrass, R. 1987. The Temporal Query Language Tquel. *ACM Transactions on Database Systems*. 12(2): 247-298.

Snodgrass, R. 1992. Temporal Databases. *Theories and Methods of Spatiotemporal Reasoning in Geographic Space (A. Frank, I Campari and U. Formanti, editors)*. pp. 22-64.

Snodgrass, R., I. Ahn, G. Ariav, D. Batory, J. Clifford, C. Dyreson, R. Elmasri, F. Grandi, C. Jensen, W. Käfer, N. Kline, K. Kulkarni, T. Leung, N. Lorentzos, J. Roddick, A. Segev, M. Soo and S. Sripada, 1995. The TSQL2 Temporal Query Language. Kluwer Academic Publishers, 1995, 674 pp.

Stonebraker, M., L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein and D. Beech. 1990. Third-Generation Database System Manifesto. *SIGMOD Record*. 19(3).

Tang, A., T, Adams, and E. Usery. 1996. A Spatial Data Model Design for Feature-Based Geographical Information Systems. *International Journal of Geographical Information Systems*. 10(5): 643-659.

Theodoris, Y., T. Sellis, A. Papdopoulos and Y. Manolopoulos. 1998. Specifications for Efficient Indexing in Spatiotemporal Databases. Proceedings 10th International Conference on Scientific and Statistical Database Management (SSDBM 98). pp. 123-132.

Tversky, B. and H. Taylor. 1998. Acquiring Spatial and Temporal Knowledge from Language. In M. Egenhofer and R. Golledge (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*, Oxford University Press, New York, pp. 155-166.

Usery, E. 1993. Category Theory and the Structure of Features in Geographic Information Systems. *Cartography and Geographic Information Systems*. 20(1):5-12.

Usery, E. 1996. A Feature-Based Geographic Information System Model, *Photogrammetric Engineering and Remote Sensing*. 62(7): 833-838.

Usery, E. 2000. Multidimensional Representation of Geographic Features. *Proceedings, XIXth International Society for Photogrammetry and Remote Sensing Congress, Amsterdam, International Archives of Photogrammetry and Remote Sensing, Volume XXXIII, Part B4/3, Commission 4,* pp. 240-247*.*

Usery, E., G. Timson and M. Coletti. (forthcoming). *Multidimensional Representation of Geographic Features*.

Voisard, A. and D. Benoit. 2002. A Database Perspective on Geospatial Data Modeling. *IEEE Transactions on Knowledge and Data Engineering.* 14(2): 226-243.

Wachowicz, M. 1999. *Object-Oriented Design for Temporal GIS.* Taylor and Francis, New York, New York.

Worboys, M. 1994. Object-Oriented Approaches to Geo-Referenced Information. *International Journal of Geographic Information Systems.* 8(4): 385-399.

Worboys, M. 1998. A Generic Model for Spatio-Bitemporal Geographic Information. In M. Egenhofer and R. Golledge (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*, Oxford University Press, New York, pp. 25-39.

Yazici, A. Q. Zhu, and N. Sun. 2001. Semantic Data Modeling of Spatiotemporal Database Applications. *International Journal of Intelligent Systems.* 16(7): 881-904.

Yuan, M. 2001. Representing Complex Geographic Phenomena with Both Object and Field-Like Properties. *Cartography and Geographic Information Science.* 28(2):83-96.

Zimanyi, E., C. Parent, S. Spaccapietra, and A. Pirotte. 1997. TERC+ : A Temporal Conceptual Model. In *Proceedings of the International Symposium on Digital Media Information Base, DMIB'97,*Nara, Japan.

Zipf, A. and S. Krüger. 2001. TGML: Extending GML by Temporal Constructs.  A Proposal  for a Spatiotemporal Framework in XML. *ACM-GIS.*

# APPENDIX A

## PROGRAM LISTING FOR GETGEOMETRY.VBA

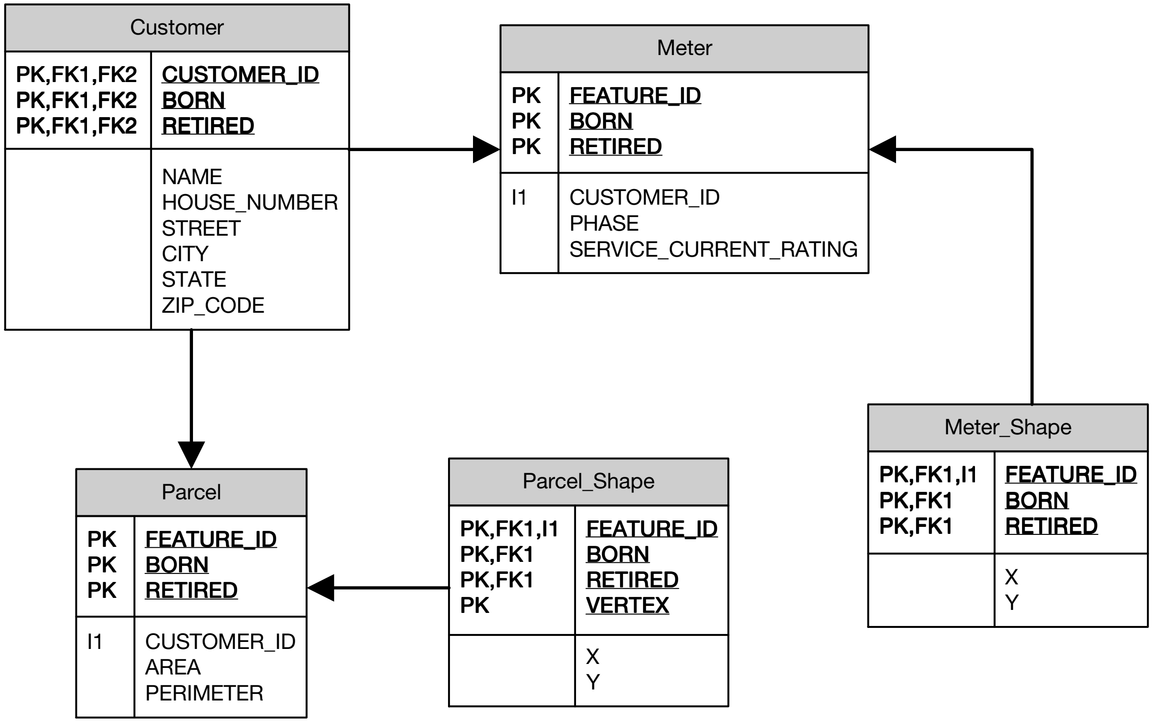
```
'====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'====================================================================
' Script:       GetGeometry
' Description:  Extracts the shape and other geometric properties
'               from the coverages and updates an
'               already-generated Microsoft Access database.
'====================================================================

Option Explicit                     ' Require declaration of variables

'====================================================================
' Variable declarations
'====================================================================
Dim mxDoc As IMxDocument
Dim Map As IMap
Dim pFeatureClass As IFeatureClass
Dim lNum As Integer
Dim coverName As String
Dim coverType As String
Dim Layer As IFeatureLayer
Dim FCursor As IFeatureCursor
Dim Feature As IFeature
Dim FeaturePoint As IPoint
Dim FeaturePoints As IPointCollection
Dim p As Integer
Dim fid As Long
Dim born As Date
Dim retired As Date
Dim conn As ADODB.Connection
Dim expr As String
Dim pArea As IArea
Dim pLength As Double
Dim pCurve As ICurve
Dim pEnv As IEnvelope

' Open the map document

Set mxDoc = Application.Document
Set Map = mxDoc.FocusMap

' Open a connection to the database

Set conn = New ADODB.Connection
conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & " data.mdb"

' Loop through each of the coverages in the map

For lNum = 0 To (Map.LayerCount - 1)

    Debug.Print coverName

    ' Get the coverage name

    coverName = Map.Layer(lNum).Name

    ' Get the coverage type
```

```
Select Case coverName
    Case "priconductor", "secconductor"
        coverType = "line"
    Case "parcel1", "parcel2", "parcel3", "parcel4", "parcel3a", "parcel4a", "parcel4b"
        coverType = "polygon"
    Case Else
        coverType = "point"
End Select

' Extract the vertices

Set Layer = Map.Layer(lNum)
Set FCursor = Layer.Search(Nothing, False)
Set Feature = FCursor.NextFeature
While Not (Feature Is Nothing)

    ' Get the feature ID, born and retired values

    fid = Feature.Value(Feature.Fields.FindField("FEATURE_ID"))
    born = Feature.Value(Feature.Fields.FindField("BORN"))
    retired = Feature.Value(Feature.Fields.FindField("RETIRED"))

    If (fid <> 0) Then

        Select Case coverType

            Case "polygon"

                 ' Get the geometry

                Set FeaturePoints = Feature.Shape
                For p = 0 To (FeaturePoints.PointCount - 1)
                    expr = "INSERT INTO parcel_shape VALUES (" & fid & ",#" & born & "#,#" & _
                           retired & "#," & CStr(p + 1) & "," & FeaturePoints.Point(p).X & "," & _
                           FeaturePoints.Point(p).Y & ")"
                    conn.Execute expr
                Next p
                Set pArea = Feature.Shape
                Set pCurve = Feature.Shape
                pLength = pCurve.Length

                ' Get the envelope

                Set pEnv = Feature.Extent
                expr = "INSERT INTO parcel_shape VALUES (" & fid & ",#" & born & "#,#" & retired & _
                       "#,-1," & pEnv.XMin & "," & pEnv.YMin & ")"
    conn.Execute expr
    expr = "INSERT INTO parcel_shape VALUES (" & fid & ",#" & born & "#,#" & retired & _
           "#,-2," & pEnv.XMax & "," & pEnv.YMax & ")"
    conn.Execute expr

    ' Store in the access database

    expr = "UPDATE parcel SET AREA = " & CStr(pArea.Area) & " WHERE FEATURE_ID = " & _
           fid & " AND BORN = #" & born & "# AND RETIRED = #" & retired & "#"
    conn.Execute expr
    expr = "UPDATE parcel SET PERIMETER = " & CStr(pLength) & " WHERE FEATURE_ID = " & _
           fid & " AND BORN = #" & born & "# AND RETIRED = #" & retired & "#"
    conn.Execute expr

Case "line"

    ' Get the geometry

    Set FeaturePoints = Feature.Shape
    For p = 0 To (FeaturePoints.PointCount - 1)
        expr = "INSERT INTO " & coverName & "_shape VALUES (" & fid & ",#" & born & "#,#" _
               & retired & "#," & CStr(p + 1) & "," & FeaturePoints.Point(p).X & "," & _
               FeaturePoints.Point(p).Y & ")"
        conn.Execute expr
    Next p
    Set pCurve = Feature.Shape
    pLength = pCurve.Length

    ' Get the envelope

    Set pEnv = Feature.Extent
    expr = "INSERT INTO " & coverName & "_shape VALUES (" & fid & ",#" & born & "#,#" _
           & retired & "#,-1," & pEnv.XMin & "," & pEnv.YMin & ")"
    conn.Execute expr
    expr = "INSERT INTO " & coverName & "_shape VALUES (" & fid & ",#" & born & "#,#" _
           & retired & "#,-2," & pEnv.XMax & "," & pEnv.YMax & ")"
    conn.Execute expr

    ' Store in the access database

    expr = "UPDATE " & coverName & " SET LENGTH = " & CStr(pLength) & _
           " WHERE FEATURE_ID = " & fid & " AND BORN = #" & born & "# AND RETIRED = #" _
           & retired & "#"
    conn.Execute expr

Case "point"
```

```
                    ' Get the geometry and store it in the database

                    Set FeaturePoint = Feature.Shape
                    expr = "INSERT INTO " & coverName & "_shape VALUES (" & fid & ",#" & born & "#,#" _
                           & retired & "#,1," & FeaturePoint.X & "," & FeaturePoint.Y & ")"
                    conn.Execute expr

            End Select

        End If

        ' Iterate to the next feature in the cursor

        Set Feature = FCursor.NextFeature

    Wend

Next lNum

' Close the access connection

conn.Close
```

# APPENDIX B

## RELATIONAL SCHEMA



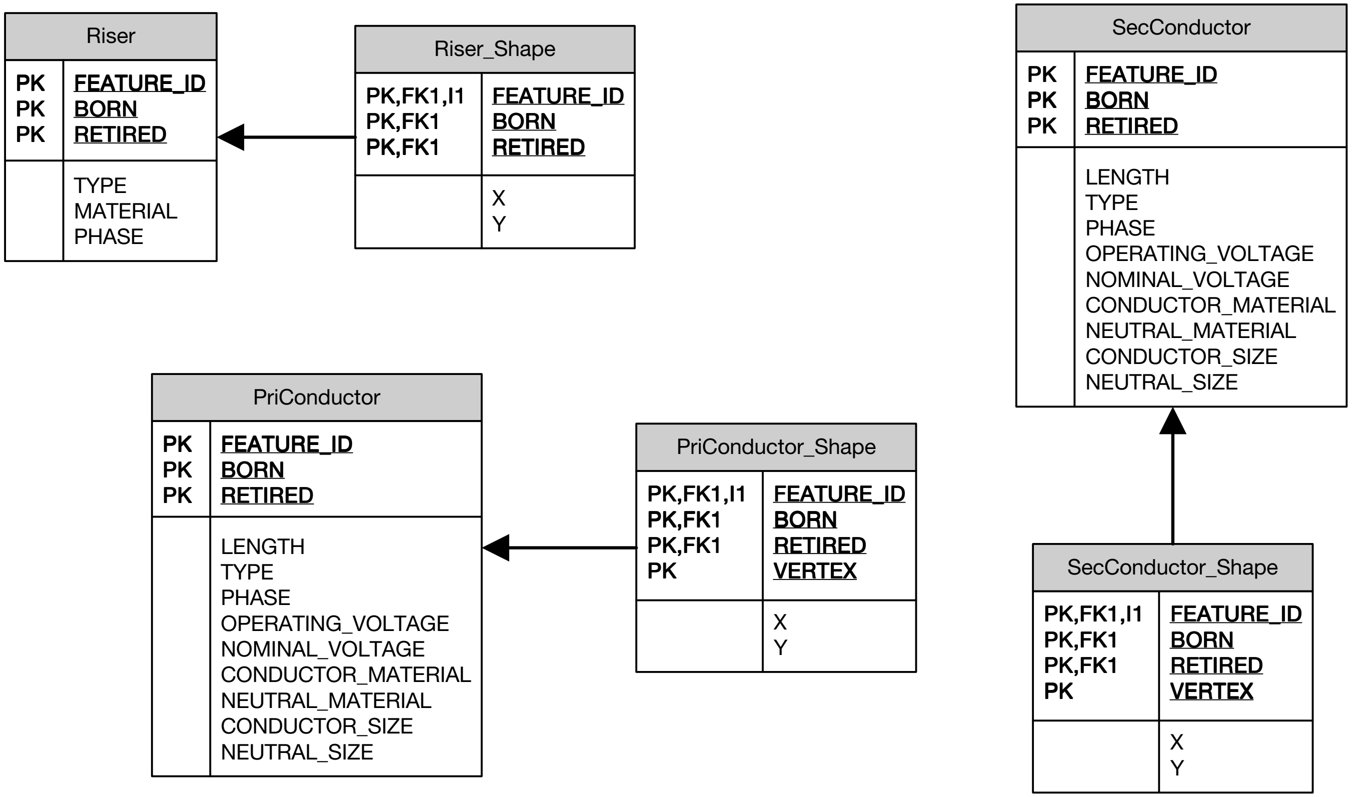
| Customer | |
|---|---|
| **PK,FK1,FK2**<br>**PK,FK1,FK2**<br>**PK,FK1,FK2** | **CUSTOMER_ID**<br>**BORN**<br>**RETIRED** |
| | NAME<br>HOUSE_NUMBER<br>STREET<br>CITY<br>STATE<br>ZIP_CODE |

| Meter | |
|---|---|
| **PK**<br>**PK**<br>**PK** | **FEATURE_ID**<br>**BORN**<br>**RETIRED** |
| I1 | CUSTOMER_ID<br>PHASE<br>SERVICE_CURRENT_RATING |

| Parcel | |
|---|---|
| **PK**<br>**PK**<br>**PK** | **FEATURE_ID**<br>**BORN**<br>**RETIRED** |
| I1 | CUSTOMER_ID<br>AREA<br>PERIMETER |

| Parcel_Shape | |
|---|---|
| **PK,FK1,I1**<br>**PK,FK1**<br>**PK,FK1**<br>**PK** | **FEATURE_ID**<br>**BORN**<br>**RETIRED**<br>**VERTEX** |
| | X<br>Y |

| Meter_Shape | |
|---|---|
| **PK,FK1,I1**<br>**PK,FK1**<br>**PK,FK1** | **FEATURE_ID**<br>**BORN**<br>**RETIRED** |
| | X<br>Y |



| Riser | |
|---|---|
| **PK** | **FEATURE_ID** |
| **PK** | **BORN** |
| **PK** | **RETIRED** |
| | TYPE |
| | MATERIAL |
| | PHASE |

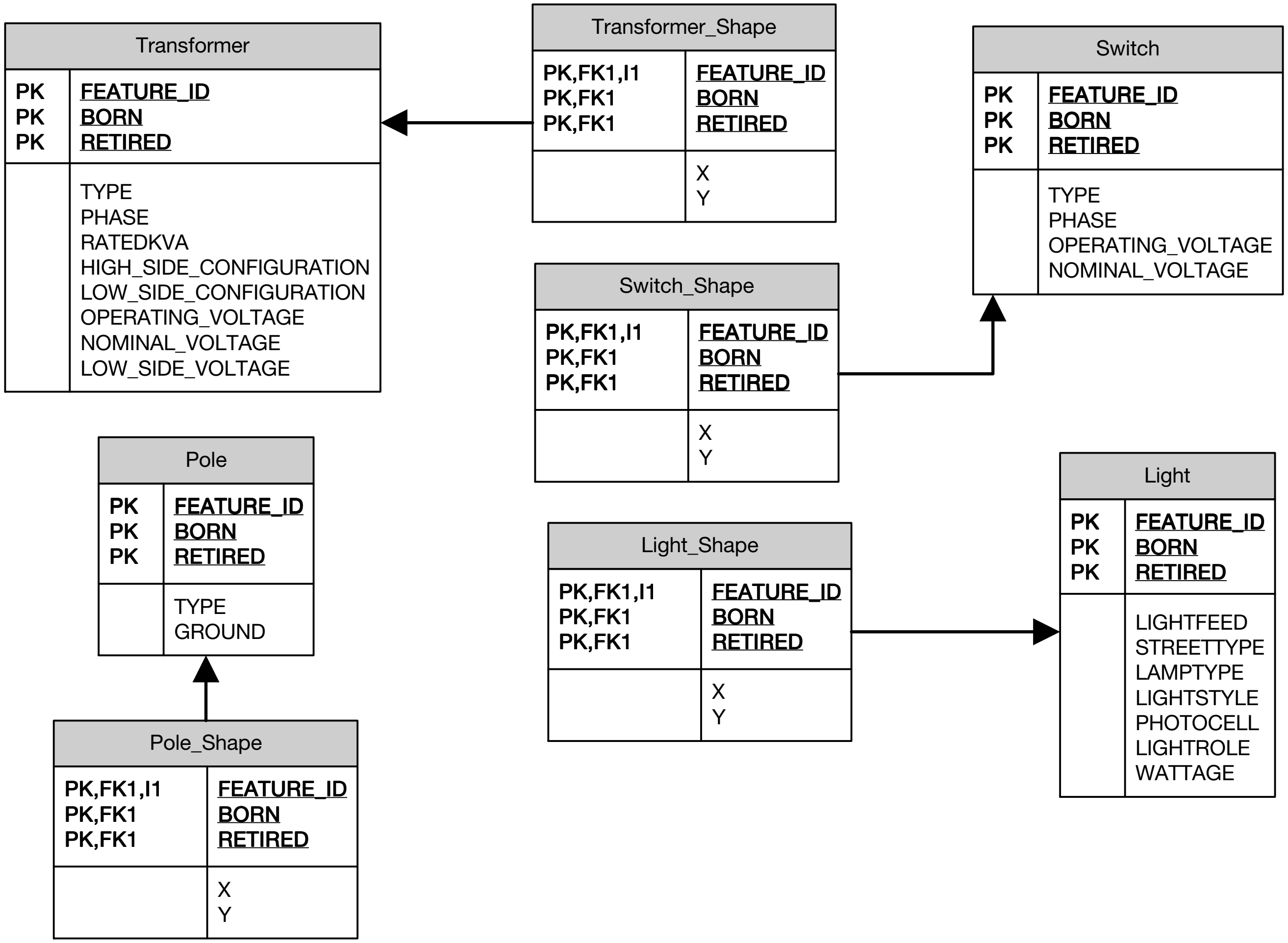
| Riser_Shape | |
|---|---|
| **PK,FK1,I1** | **FEATURE_ID** |
| **PK,FK1** | **BORN** |
| **PK,FK1** | **RETIRED** |
| | X |
| | Y |

| SecConductor | |
|---|---|
| **PK** | **FEATURE_ID** |
| **PK** | **BORN** |
| **PK** | **RETIRED** |
| | LENGTH |
| | TYPE |
| | PHASE |
| | OPERATING_VOLTAGE |
| | NOMINAL_VOLTAGE |
| | CONDUCTOR_MATERIAL |
| | NEUTRAL_MATERIAL |
| | CONDUCTOR_SIZE |
| | NEUTRAL_SIZE |

| PriConductor | |
|---|---|
| **PK** | **FEATURE_ID** |
| **PK** | **BORN** |
| **PK** | **RETIRED** |
| | LENGTH |
| | TYPE |
| | PHASE |
| | OPERATING_VOLTAGE |
| | NOMINAL_VOLTAGE |
| | CONDUCTOR_MATERIAL |
| | NEUTRAL_MATERIAL |
| | CONDUCTOR_SIZE |
| | NEUTRAL_SIZE |

| PriConductor_Shape | |
|---|---|
| **PK,FK1,I1** | **FEATURE_ID** |
| **PK,FK1** | **BORN** |
| **PK,FK1** | **RETIRED** |
| **PK** | **VERTEX** |
| | X |
| | Y |

| SecConductor_Shape | |
|---|---|
| **PK,FK1,I1** | **FEATURE_ID** |
| **PK,FK1** | **BORN** |
| **PK,FK1** | **RETIRED** |
| **PK** | **VERTEX** |
| | X |
| | Y |



| Transformer | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| | TYPE<br>PHASE<br>RATEDKVA<br>HIGH_SIDE_CONFIGURATION<br>LOW_SIDE_CONFIGURATION<br>OPERATING_VOLTAGE<br>NOMINAL_VOLTAGE<br>LOW_SIDE_VOLTAGE |

| Transformer_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1 | FEATURE_ID<br>BORN<br>RETIRED |
| | X<br>Y |

| Switch | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| | TYPE<br>PHASE<br>OPERATING_VOLTAGE<br>NOMINAL_VOLTAGE |

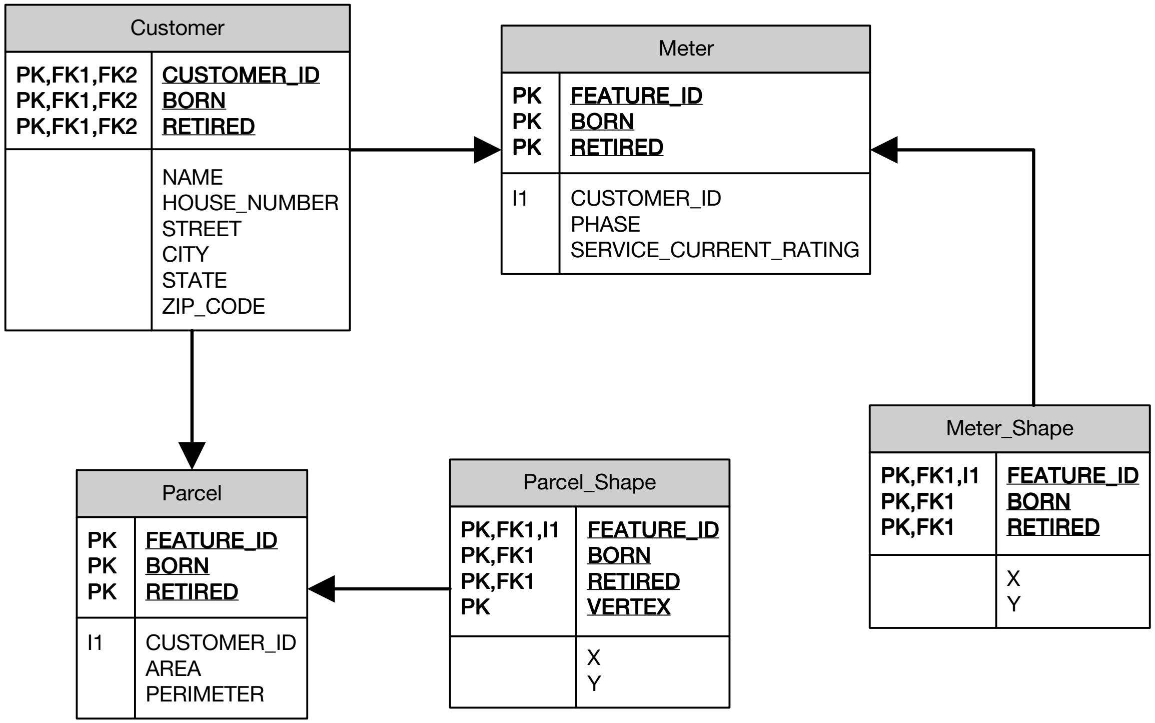
| Switch_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1 | FEATURE_ID<br>BORN<br>RETIRED |
| | X<br>Y |

| Pole | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| | TYPE<br>GROUND |

| Light_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1 | FEATURE_ID<br>BORN<br>RETIRED |
| | X<br>Y |

| Light | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| | LIGHTFEED<br>STREETTYPE<br>LAMPTYPE<br>LIGHTSTYLE<br>PHOTOCELL<br>LIGHTROLE<br>WATTAGE |

| Pole_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1 | FEATURE_ID<br>BORN<br>RETIRED |
| | X<br>Y |



| Customer | |
|---|---|
| PK,FK1,FK2<br>PK,FK1,FK2<br>PK,FK1,FK2 | CUSTOMER_ID<br>BORN<br>RETIRED |
| | NAME<br>HOUSE_NUMBER<br>STREET<br>CITY<br>STATE<br>ZIP_CODE |

| Meter | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| I1 | CUSTOMER_ID<br>PHASE<br>SERVICE_CURRENT_RATING |

| Parcel | |
|---|---|
| PK<br>PK<br>PK | FEATURE_ID<br>BORN<br>RETIRED |
| I1 | CUSTOMER_ID<br>AREA<br>PERIMETER |

| Parcel_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1<br>PK | FEATURE_ID<br>BORN<br>RETIRED<br>VERTEX |
| | X<br>Y |

| Meter_Shape | |
|---|---|
| PK,FK1,I1<br>PK,FK1<br>PK,FK1 | FEATURE_ID<br>BORN<br>RETIRED |
| | X<br>Y |

# APPENDIX C

## PROGRAM LISTING FOR MDB2XML.VBA

```
'=====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'=====================================================================
' Script:       MDB2XML
' Description:  Converts the MS Access database into XML format.
'=====================================================================

Option Explicit                      ' Require declaration of variables
```

```
'====================================================================
' Variable declarations
'====================================================================
Dim rs As ADODB.Recordset
Dim rs2 As ADODB.Recordset
Dim sqlString As String
Dim Tables As New Collection
Dim ccls As Integer
Dim f As Integer
Dim CurrentRecord As Long
Dim xTable As New DOMDocument40
Dim cNode As IXMLDOMNode
Dim fNode As IXMLDOMNode
Dim lNode As IXMLDOMNode
Dim cAtt As IXMLDOMAttribute
Dim DateUpdated As Boolean
Dim xmin As Double
Dim xmax As Double
Dim ymin As Double
Dim ymax As Double
Dim area As Double
Dim perimeter As Double

' Establish the connection to the access database

Set Conn = New ADODB.Connection
Conn.open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & App.Path & "\data\DB.mdb"

' Get the list of feature classes and object classes to convert

sqlString = "SELECT * FROM CATALOG_OBJECTCLASS"
Set rs = New ADODB.Recordset
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
rs.MoveFirst
Do While Not rs.EOF
    Tables.Add rs("TABLE").Value
    rs.MoveNext
Loop
rs.Close
sqlString = "SELECT * FROM CATALOG_FEATURECLASS"
Set rs = New ADODB.Recordset
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
rs.MoveFirst
Do While Not rs.EOF
    Tables.Add rs("TABLE").Value
    rs.MoveNext
```

```
Loop
rs.Close

' Loop through each of the classes and create new XML and export

For ccls = 1 To Tables.Count

' Create a new XML document

    If (Tables.Item(ccls) <> “Customer”) Then
        Set xTable = New DOMDocument40
        Set cNode = xTable.createElement("featureclass")
        xTable.appendChild cNode
    Else
        Set xTable = New DOMDocument40
        Set cNode = xTable.createElement("objectclass")
        xTable.appendChild cNode
    End If

    ' Populate depending on the type of class

    Select Case Tables.Item(ccls)
        Case "Customer"
            sqlString = "SELECT * FROM CUSTOMER ORDER BY CUSTOMER_ID, BORN, RETIRED"
            Set rs = New ADODB.Recordset
            rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
            rs.MoveFirst
            CurrentRecord = 0
            Do While Not rs.EOF
                If (rs("CUSTOMER_ID").Value > CurrentRecord) Then
                    CurrentRecord = rs("CUSTOMER_ID").Value
                    Set fNode = xTable.createElement("object")
                    xTable.lastChild.appendChild fNode
                    Set cAtt = xTable.createAttribute("id")
                    cAtt.Value = CurrentRecord
                    fNode.Attributes.setNamedItem cAtt
                End If
                For f = 0 To (rs.fields.Count - 1)
                    If (rs.fields.Item(f).Name <> "CUSTOMER_ID") And (rs.fields.Item(f).Name <> "BORN") _
                    And (rs.fields.Item(f).Name <> "RETIRED") Then
                        DateUpdated = False
                        For Each lNode In fNode.selectNodes(LCase(rs.fields.Item(f).Name))
                            If (lNode.Text = rs(rs.fields.Item(f).Name).Value) Then
                                Set cAtt = lNode.Attributes.getNamedItem("retired")
                                cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                                DateUpdated = True
                        End If
                    Next lNode
                    If (Not DateUpdated) Then
                        Set cNode = xTable.createElement(LCase(rs.fields.Item(f).Name))
                        cNode.Text = rs(rs.fields.Item(f).Name).Value
                        Set cAtt = xTable.createAttribute("born")
                        cAtt.Value = Format(rs("born").Value, "yyyymmdd")
                        cNode.Attributes.setNamedItem cAtt
                        Set cAtt = xTable.createAttribute("retired")
                        cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                        cNode.Attributes.setNamedItem cAtt
                        fNode.appendChild cNode
                    End If
                End If
            Next f
            rs.MoveNext
        Loop
        xTable.save App.Path & "\data\" & Tables.Item(ccls) & ".xml"
    Case "Meter"
        sqlString = "SELECT * FROM " & Tables.Item(ccls) & _
                    " ORDER BY FEATURE_ID, BORN, RETIRED, CUSTOMER_ID"
        Set rs = New ADODB.Recordset
        rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
        rs.MoveFirst
        CurrentRecord = 0
        Do While Not rs.EOF
            If (rs("FEATURE_ID").Value > CurrentRecord) Then
                CurrentRecord = rs("FEATURE_ID").Value
                Set fNode = xTable.createElement("feature")
                xTable.lastChild.appendChild fNode
                Set cAtt = xTable.createAttribute("id")
                cAtt.Value = CurrentRecord
                fNode.Attributes.setNamedItem cAtt
                sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " _
                            & CurrentRecord & " ORDER BY BORN, RETIRED"
                Set rs2 = New ADODB.Recordset
                rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
                rs2.MoveFirst
                DateUpdated = False
                Do While Not rs2.EOF
                    For Each lNode In fNode.selectNodes("shape")
                        If (lNode.lastChild.firstChild.Text = rs2("X").Value) And _
                        (lNode.lastChild.lastChild.Text = rs2("Y").Value) Then
                            Set cAtt = lNode.Attributes.getNamedItem("retired")
                            cAtt.Value = Format(rs2("retired").Value, "yyyymmdd")
                            DateUpdated = True
            End If
        Next lNode
        If (Not DateUpdated) Then
            Set cNode = xTable.createElement("shape")
            Set cAtt = xTable.createAttribute("born")
            cAtt.Value = Format(rs2("born").Value, "yyyymmdd")
            cNode.Attributes.setNamedItem cAtt
            Set cAtt = xTable.createAttribute("retired")
            cAtt.Value = Format(rs2("retired").Value, "yyyymmdd")
            cNode.Attributes.setNamedItem cAtt
            Set lNode = xTable.createElement("coordinate")
            cNode.appendChild lNode
            Set lNode = xTable.createElement("x")
            lNode.Text = rs2("X").Value
            cNode.lastChild.appendChild lNode
            Set lNode = xTable.createElement("y")
            lNode.Text = rs2("Y").Value
            cNode.lastChild.appendChild lNode
            fNode.appendChild cNode
        End If
        rs2.MoveNext
    Loop
    rs2.Close
End If
For f = 0 To (rs.fields.Count - 1)
    If (rs.fields.Item(f).Name <> "FEATURE_ID") And (rs.fields.Item(f).Name <> "BORN") _
    And (rs.fields.Item(f).Name <> "RETIRED") Then
        DateUpdated = False
        For Each lNode In fNode.selectNodes(LCase(rs.fields.Item(f).Name))
            If (lNode.Text = rs(rs.fields.Item(f).Name).Value) Then
                Set cAtt = lNode.Attributes.getNamedItem("retired")
                cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                DateUpdated = True
            End If
        Next lNode
        If (Not DateUpdated) Then
            Set cNode = xTable.createElement(LCase(rs.fields.Item(f).Name))
            cNode.Text = rs(rs.fields.Item(f).Name).Value
            Set cAtt = xTable.createAttribute("born")
            cAtt.Value = Format(rs("born").Value, "yyyymmdd")
            cNode.Attributes.setNamedItem cAtt
            Set cAtt = xTable.createAttribute("retired")
            cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
            cNode.Attributes.setNamedItem cAtt
            fNode.appendChild cNode
        End If
```

```
            End If
        Next f
        rs.MoveNext
    Loop
    xTable.save App.Path & "\data\" & Tables.Item(ccls) & ".xml"
Case "Switch", "Riser", "Light", "Pole", "Transformer"
    sqlString = "SELECT * FROM " & Tables.Item(ccls) & " ORDER BY FEATURE_ID, BORN, RETIRED"
    Set rs = New ADODB.Recordset
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    CurrentRecord = 0
    Do While Not rs.EOF
        If (rs("FEATURE_ID").Value > CurrentRecord) Then
            CurrentRecord = rs("FEATURE_ID").Value
            Set fNode = xTable.createElement("feature")
            xTable.lastChild.appendChild fNode
            Set cAtt = xTable.createAttribute("id")
            cAtt.Value = CurrentRecord
            fNode.Attributes.setNamedItem cAtt
            sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " _
                        & CurrentRecord & " ORDER BY BORN, RETIRED"
            Set rs2 = New ADODB.Recordset
            rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
            rs2.MoveFirst
            DateUpdated = False
            Do While Not rs2.EOF
                For Each lNode In fNode.selectNodes("shape")
                    If (lNode.lastChild.firstChild.Text = rs2("X").Value) And _
                    (lNode.lastChild.lastChild.Text = rs2("Y").Value) Then
                        Set cAtt = lNode.Attributes.getNamedItem("retired")
                        cAtt.Value = Format(rs2("retired").Value, "yyyymmdd")
                        DateUpdated = True
                    End If
                Next lNode
                If (Not DateUpdated) Then
                    Set cNode = xTable.createElement("shape")
                    Set cAtt = xTable.createAttribute("born")
                    cAtt.Value = Format(rs2("born").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    Set cAtt = xTable.createAttribute("retired")
                    cAtt.Value = Format(rs2("retired").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    Set lNode = xTable.createElement("coordinate")
                    cNode.appendChild lNode
                    Set lNode = xTable.createElement("x")
                    lNode.Text = rs2("X").Value
                    cNode.lastChild.appendChild lNode
                    Set lNode = xTable.createElement("y")
                    lNode.Text = rs2("Y").Value
                    cNode.lastChild.appendChild lNode
                    fNode.appendChild cNode
                End If
                rs2.MoveNext
            Loop
            rs2.Close
        End If
        For f = 0 To (rs.fields.Count - 1)
            If (rs.fields.Item(f).Name <> "FEATURE_ID") And (rs.fields.Item(f).Name <> "BORN") _
            And (rs.fields.Item(f).Name <> "RETIRED") Then
                DateUpdated = False
                For Each lNode In fNode.selectNodes(LCase(rs.fields.Item(f).Name))
                    If (lNode.Text = rs(rs.fields.Item(f).Name).Value) Then
                        Set cAtt = lNode.Attributes.getNamedItem("retired")
                        cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                        DateUpdated = True
                    End If
                Next lNode
                If (Not DateUpdated) Then
                    Set cNode = xTable.createElement(LCase(rs.fields.Item(f).Name))
                    cNode.Text = rs(rs.fields.Item(f).Name).Value
                    Set cAtt = xTable.createAttribute("born")
                    cAtt.Value = Format(rs("born").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    Set cAtt = xTable.createAttribute("retired")
                    cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    fNode.appendChild cNode
                End If
            End If
        Next f
        rs.MoveNext
    Loop
    xTable.save App.Path & "\data\" & Tables.Item(ccls) & ".xml"
Case "PriConductor", "SecConductor"
    sqlString = "SELECT * FROM " & Tables.Item(ccls) & " ORDER BY FEATURE_ID, BORN, RETIRED"
    Set rs = New ADODB.Recordset
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    CurrentRecord = 0
    Do While Not rs.EOF
        If (rs("FEATURE_ID").Value > CurrentRecord) Then
            CurrentRecord = rs("FEATURE_ID").Value
    Set fNode = xTable.createElement("feature")
    xTable.lastChild.appendChild fNode
    Set cAtt = xTable.createAttribute("id")
    cAtt.Value = CurrentRecord
    fNode.Attributes.setNamedItem cAtt
End If
sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " & _
            CurrentRecord & " AND BORN = #" & rs("BORN").Value & "# AND RETIRED = #" & _
            rs("RETIRED").Value & _
            "# AND VERTEX < 0 ORDER BY FEATURE_ID, BORN, RETIRED, VERTEX"
Set rs2 = New ADODB.Recordset
rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
rs2.MoveFirst
DateUpdated = False
Do While Not rs2.EOF
    If (rs2("VERTEX").Value = -2) Then
        xmin = rs2("X").Value
        ymin = rs2("Y").Value
    ElseIf (rs2("VERTEX").Value = -1) Then
        xmax = rs2("X").Value
        ymax = rs2("Y").Value
    End If
    rs2.MoveNext
Loop
rs2.Close
For Each lNode In fNode.selectNodes("shape")
    If (lNode.selectSingleNode("boundbox/xmin").Text = xmin) And _
    (lNode.selectSingleNode("boundbox/ymin").Text = ymin) And _
    (lNode.selectSingleNode("boundbox/xmax").Text = xmax) And _
    (lNode.selectSingleNode("boundbox/ymax").Text = ymax) Then
        Set cAtt = lNode.Attributes.getNamedItem("retired")
        cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
        DateUpdated = True
    End If
Next lNode
If (Not DateUpdated) Then
    Set cNode = xTable.createElement("shape")
    Set cAtt = xTable.createAttribute("born")
    cAtt.Value = Format(rs("born").Value, "yyyymmdd")
    cNode.Attributes.setNamedItem cAtt
    Set cAtt = xTable.createAttribute("retired")
    cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
    cNode.Attributes.setNamedItem cAtt
    Set lNode = xTable.createElement("boundbox")
    cNode.appendChild lNode
    Set lNode = xTable.createElement("xmin")
    lNode.Text = xmin
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("ymin")
    lNode.Text = ymin
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("xmax")
    lNode.Text = xmax
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("ymax")
    lNode.Text = ymax
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("length")
    lNode.Text = rs("LENGTH").Value
    cNode.appendChild lNode
    sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " _
                & CurrentRecord & " AND BORN = #" & rs("BORN").Value & _
                "# AND RETIRED = #" & rs("RETIRED").Value & _
                "# AND VERTEX > 0 ORDER BY FEATURE_ID, BORN, RETIRED, VERTEX"
    Set rs2 = New ADODB.Recordset
    rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs2.MoveFirst
    DateUpdated = False
    Do While Not rs2.EOF
        Set lNode = xTable.createElement("coordinate")
        cNode.appendChild lNode
        Set lNode = xTable.createElement("x")
        lNode.Text = rs2("X").Value
        cNode.lastChild.appendChild lNode
        Set lNode = xTable.createElement("y")
        lNode.Text = rs2("Y").Value
        cNode.lastChild.appendChild lNode
        rs2.MoveNext
    Loop
    rs2.Close
    fNode.appendChild cNode
End If
For f = 0 To (rs.fields.Count - 1)
    If (rs.fields.Item(f).Name <> "FEATURE_ID") And (rs.fields.Item(f).Name <> "BORN") _
    And (rs.fields.Item(f).Name <> "RETIRED") And (rs.fields.Item(f).Name <> "LENGTH") Then
        DateUpdated = False
        For Each lNode In fNode.selectNodes(LCase(rs.fields.Item(f).Name))
            If (lNode.Text = rs(rs.fields.Item(f).Name).Value) Then
                Set cAtt = lNode.Attributes.getNamedItem("retired")
                cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                DateUpdated = True
            End If
```

```
                Next lNode
                If (Not DateUpdated) Then
                    Set cNode = xTable.createElement(LCase(rs.fields.Item(f).Name))
                    cNode.Text = rs(rs.fields.Item(f).Name).Value
                    Set cAtt = xTable.createAttribute("born")
                    cAtt.Value = Format(rs("born").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    Set cAtt = xTable.createAttribute("retired")
                    cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    fNode.appendChild cNode
                End If
            End If
        Next f
        rs.MoveNext
    Loop
    xTable.save App.Path & "\data\" & Tables.Item(ccls) & ".xml"
Case "Parcel"
    sqlString = "SELECT * FROM " & Tables.Item(ccls) & _
                " ORDER BY FEATURE_ID, BORN, RETIRED, CUSTOMER_ID"
    Set rs = New ADODB.Recordset
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    CurrentRecord = 0
    Do While Not rs.EOF
        If (rs("FEATURE_ID").Value > CurrentRecord) Then
            CurrentRecord = rs("FEATURE_ID").Value
            Set fNode = xTable.createElement("feature")
            xTable.lastChild.appendChild fNode
            Set cAtt = xTable.createAttribute("id")
            cAtt.Value = CurrentRecord
            fNode.Attributes.setNamedItem cAtt
        End If
        perimeter = rs("PERIMETER").Value
        area = rs("AREA").Value
        sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " _
                    & CurrentRecord & " AND BORN = #" & rs("BORN").Value & "# AND RETIRED = #" _
                    & rs("RETIRED").Value & _
                    "# AND VERTEX < 0 ORDER BY FEATURE_ID, BORN, RETIRED, VERTEX"
        Set rs2 = New ADODB.Recordset
        rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
        rs2.MoveFirst
        DateUpdated = False
        Do While Not rs2.EOF
            If (rs2("VERTEX").Value = -2) Then
                xmin = rs2("X").Value
        ymin = rs2("Y").Value
    ElseIf (rs2("VERTEX").Value = -1) Then
        xmax = rs2("X").Value
        ymax = rs2("Y").Value
    End If
    rs2.MoveNext
Loop
rs2.Close
For Each lNode In fNode.selectNodes("shape")
    If (lNode.selectSingleNode("boundbox/xmin").Text = xmin) And _
    (lNode.selectSingleNode("boundbox/ymin").Text = ymin) And _
    (lNode.selectSingleNode("boundbox/xmax").Text = xmax) And _
    (lNode.selectSingleNode("boundbox/ymax").Text = ymax) And _
    (lNode.selectSingleNode("perimeter").Text = perimeter) And _
    (lNode.selectSingleNode("area").Text = area) Then
        Set cAtt = lNode.Attributes.getNamedItem("retired")
        cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
        DateUpdated = True
    End If
Next lNode
If (Not DateUpdated) Then
    Set cNode = xTable.createElement("shape")
    Set cAtt = xTable.createAttribute("born")
    cAtt.Value = Format(rs("born").Value, "yyyymmdd")
    cNode.Attributes.setNamedItem cAtt
    Set cAtt = xTable.createAttribute("retired")
    cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
    cNode.Attributes.setNamedItem cAtt
    Set lNode = xTable.createElement("boundbox")
    cNode.appendChild lNode
    Set lNode = xTable.createElement("xmin")
    lNode.Text = xmin
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("ymin")
    lNode.Text = ymin
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("xmax")
    lNode.Text = xmax
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("ymax")
    lNode.Text = ymax
    cNode.lastChild.appendChild lNode
    Set lNode = xTable.createElement("area")
    lNode.Text = area
    cNode.appendChild lNode
    Set lNode = xTable.createElement("perimeter")
```

```
            lNode.Text = perimeter
            cNode.appendChild lNode
            sqlString = "SELECT * FROM " & Tables.Item(ccls) & "_Shape WHERE FEATURE_ID = " _
                        & CurrentRecord & " AND BORN = #" & rs("BORN").Value & _
                        "# AND RETIRED = #" & rs("RETIRED").Value & _
                        "# AND VERTEX > 0 ORDER BY FEATURE_ID, BORN, RETIRED, VERTEX"
            Set rs2 = New ADODB.Recordset
            rs2.open sqlString, Conn, adOpenKeyset, adLockOptimistic
            rs2.MoveFirst
            DateUpdated = False
            Do While Not rs2.EOF
                Set lNode = xTable.createElement("coordinate")
                cNode.appendChild lNode
                Set lNode = xTable.createElement("x")
                lNode.Text = rs2("X").Value
                cNode.lastChild.appendChild lNode
                Set lNode = xTable.createElement("y")
                lNode.Text = rs2("Y").Value
                cNode.lastChild.appendChild lNode
                rs2.MoveNext
            Loop
            rs2.Close
            fNode.appendChild cNode
        End If
        For f = 0 To (rs.fields.Count - 1)
            If (rs.fields.Item(f).Name <> "FEATURE_ID") And (rs.fields.Item(f).Name <> "BORN") _
            And (rs.fields.Item(f).Name <> "RETIRED") And (rs.fields.Item(f).Name <> "AREA") _
            And (rs.fields.Item(f).Name <> "PERIMETER") Then
                DateUpdated = False
                For Each lNode In fNode.selectNodes(LCase(rs.fields.Item(f).Name))
                    If (lNode.Text = rs(rs.fields.Item(f).Name).Value) Then
                        Set cAtt = lNode.Attributes.getNamedItem("retired")
                        cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                        DateUpdated = True
                    End If
                Next lNode
                If (Not DateUpdated) Then
                    Set cNode = xTable.createElement(LCase(rs.fields.Item(f).Name))
                    cNode.Text = rs(rs.fields.Item(f).Name).Value
                    Set cAtt = xTable.createAttribute("born")
                    cAtt.Value = Format(rs("born").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    Set cAtt = xTable.createAttribute("retired")
                    cAtt.Value = Format(rs("retired").Value, "yyyymmdd")
                    cNode.Attributes.setNamedItem cAtt
                    fNode.appendChild cNode
```

```
                        End If
                    End If
                Next f
                rs.MoveNext
            Loop
            xTable.save App.Path & "\data\" & Tables.Item(ccls) & ".xml"
    End Select
Next ccls

Conn.Close
```

# APPENDIX D

## SCHEMA LISTING FOR CATALOG.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="catalog">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="boundbox" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="objectclass" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="featureclass" minOccurs="1" maxOccurs="unbounded"/>
                        <xs:element ref="relationship" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
```

```
</xs:element>
<xs:element name="boundbox">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="xmin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="xmax" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymax" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="xmin" type="xs:short"/>
<xs:element name="ymin" type="xs:short"/>
<xs:element name="xmax" type="xs:short"/>
<xs:element name="ymax" type="xs:short"/>
<xs:element name="objectclass">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="table" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="table" type="xs:string"/>
<xs:element name="featureclass">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="type" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="table" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="symbology" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="tocimage" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="type">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="line"/>
                  <xs:enumeration value="point"/>
                  <xs:enumeration value="polygon"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
<xs:element name="symbology">
```

```
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="symbol" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="size" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="color" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="symbol">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="circle"/>
                  <xs:enumeration value="rectangle"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
<xs:element name="size">
      <xs:complexType mixed="true">
            <xs:sequence>
                  <xs:element ref="height" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="width" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="height" type="xs:double"/>
<xs:element name="width" type="xs:double"/>
<xs:element name="color">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="red" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="green" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="blue" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="red" type="xs:short"/>
<xs:element name="green" type="xs:short"/>
<xs:element name="blue" type="xs:short"/>
<xs:element name="tocimage" type="xs:string"/>
<xs:element name="relationship">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="source_featureclass" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="destination_objectclass" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="key_field" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="source_featureclass" type="xs:string"/>
      <xs:element name="destination_objectclass" type="xs:string"/>
      <xs:element name="key_field" type="xs:string"/>
</xs:schema>
```

# APPENDIX E

## SCHEMA LISTING FOR CUSTOMER.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="objectclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="object" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="object">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="house_number" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="street" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="city" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="state" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="zip_code" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="name">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="house_number">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:short">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="street">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="city">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
```

```
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="state">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="zip_code">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:short">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX F

## SCHEMA LISTING FOR LIGHT.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
```

```
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="lightfeed" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="streettype" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="lamptype" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="lightstyle" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="photocell" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="lightrole" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="wattage" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:short" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="shape">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
                  </xs:sequence>
                  <xs:attribute name="born" type="xs:integer" use="required"/>
                  <xs:attribute name="retired" type="xs:integer" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="coordinate">
            <xs:complexType>
                  <xs:all>
                        <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
                  </xs:all>
            </xs:complexType>
      </xs:element>
      <xs:element name="x" type="xs:double"/>
      <xs:element name="y" type="xs:double"/>
      <xs:element name="lightfeed">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="streettype">
            <xs:complexType>
                  <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="lamptype">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="lightstyle">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="photocell">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="lightrole">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
      </xs:element>
      <xs:element name="wattage">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:short">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX G

## SCHEMA LISTING FOR METER.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="customer_id" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="service_current_rating" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="customer_id">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:integer">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="phase">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="service_current_rating">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX H

## SCHEMA LISTING FOR PARCEL.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="customer_id" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="boundbox" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="area" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="perimeter" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="boundbox">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="xmin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="xmax" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymax" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="xmin" type="xs:double"/>
<xs:element name="ymin" type="xs:double"/>
<xs:element name="xmax" type="xs:double"/>
<xs:element name="ymax" type="xs:double"/>
<xs:element name="area" type="xs:double"/>
<xs:element name="perimeter" type="xs:double"/>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
```

```
      <xs:element name="customer_id">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:integer">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX I

## SCHEMA LISTING FOR POLE.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="ground" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="ground">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
            </xs:complexType>
        </xs:element>
    </xs:schema>
```

# APPENDIX J

## SCHEMA LISTING FOR PRICONDUCTOR.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="operating_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="nominal_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="conductor_material" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="neutral_material" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="conductor_size" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="neutral_size" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="boundbox" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="length" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="boundbox">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="xmin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="xmax" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymax" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="xmin" type="xs:double"/>
<xs:element name="ymin" type="xs:double"/>
<xs:element name="xmax" type="xs:double"/>
<xs:element name="ymax" type="xs:double"/>
<xs:element name="length" type="xs:double"/>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="phase">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="operating_voltage">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="nominal_voltage">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="conductor_material">
```

```
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="neutral_material">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="conductor_size">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="neutral_size">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX K

## SCHEMA LISTING FOR RISER.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="material" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="material">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="phase">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX L

## SCHEMA LISTING FOR SECCONDUCTOR.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="operating_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="nominal_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="conductor_material" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="neutral_material" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="conductor_size" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="neutral_size" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="boundbox" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="length" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="boundbox">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="xmin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymin" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="xmax" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="ymax" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="xmin" type="xs:double"/>
<xs:element name="ymin" type="xs:double"/>
<xs:element name="xmax" type="xs:double"/>
<xs:element name="ymax" type="xs:double"/>
<xs:element name="length" type="xs:double"/>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="phase">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="operating_voltage">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="nominal_voltage">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="conductor_material">
```

```
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="neutral_material">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="conductor_size">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="neutral_size">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX M

## SCHEMA LISTING FOR SWITCH.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="operating_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="nominal_voltage" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="phase">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="operating_voltage">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="nominal_voltage">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="born" type="xs:integer" use="required"/>
                              <xs:attribute name="retired" type="xs:integer" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

# APPENDIX N

## SCHEMA LISTING FOR TRANSFORMER.XSD

```
<!-- =================================================================
 Copyright (c) 2003 Erik Shepard

 Permission to use, copy, modify, distribute, and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that (i) the above copyright notices and this permission
 notice appear in all copies of the software and related
 documentation, and (ii) the name of Erik Shepard may not be used in
 any advertising or publicity relating to the software without the
 specific, prior written permission of Erik Shepard.

 THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

 IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
 INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
 WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
 NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
 LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 PERFORMANCE OF THIS SOFTWARE.
================================================================= -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="featureclass">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="feature" minOccurs="1" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="feature">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="shape" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="phase" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="ratedkva" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="high_side_configuration" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="low_side_configuration" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="operating_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="nominal_voltage" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="low_side_voltage" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:short" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="shape">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="coordinate" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="born" type="xs:integer" use="required"/>
            <xs:attribute name="retired" type="xs:integer" use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="coordinate">
      <xs:complexType>
            <xs:all>
                  <xs:element ref="x" minOccurs="1" maxOccurs="1"/>
                  <xs:element ref="y" minOccurs="1" maxOccurs="1"/>
            </xs:all>
      </xs:complexType>
</xs:element>
<xs:element name="x" type="xs:double"/>
<xs:element name="y" type="xs:double"/>
<xs:element name="type">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="phase">
      <xs:complexType>
```

```
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="ratedkva">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:integer">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="high_side_configuration">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="low_side_configuration">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
</xs:element>
<xs:element name="operating_voltage">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="born" type="xs:integer" use="required"/>
                        <xs:attribute name="retired" type="xs:integer" use="required"/>
                  </xs:extension>
            </xs:simpleContent>
```

```
                  </xs:complexType>
            </xs:element>
            <xs:element name="nominal_voltage">
                  <xs:complexType>
                        <xs:simpleContent>
                              <xs:extension base="xs:string">
                                    <xs:attribute name="born" type="xs:integer" use="required"/>
                                    <xs:attribute name="retired" type="xs:integer" use="required"/>
                              </xs:extension>
                        </xs:simpleContent>
                  </xs:complexType>
            </xs:element>
            <xs:element name="low_side_voltage">
                  <xs:complexType>
                        <xs:simpleContent>
                              <xs:extension base="xs:string">
                                    <xs:attribute name="born" type="xs:integer" use="required"/>
                                    <xs:attribute name="retired" type="xs:integer" use="required"/>
                              </xs:extension>
                        </xs:simpleContent>
                  </xs:complexType>
            </xs:element>
      </xs:schema>
```

# APPENDIX O

## PROGRAM LISTING FOR CHOOSER.FRM

```
'====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'====================================================================
' Form:         Chooser
' Description:  Allows the user to choose the browser version and
'               sets and returns a status variable used by the main
'               subroutine.
'====================================================================

Option Explicit                      ' Require declaration of variables
```

```
'====================================================================
' Variable declarations
'====================================================================
Public ChosenBrowser As String      ' Public variable for the chosen
                                    ' browser.

'====================================================================
' Subroutine:   ChooseRelational_Click
' Description:  Responds to the ChooseRelational button click event
'               and sets the chosen browser to relational.
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub ChooseRelational_Click()
    ChosenBrowser = "Relational"
    Unload Me
End Sub

'====================================================================
' Subroutine:   ChooseXML_Click
' Description:  Responds to the ChooseXML button click event
'               and sets the chosen browser to XML.
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub ChooseXML_Click()
    ChosenBrowser = "XML"
    Unload Me
End Sub
```

# APPENDIX P

## PROGRAM LISTING FOR STARTUP.BAS

```
'====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'====================================================================
' Module:       StartUp
' Description:  Queries the user as to whether to run the XML based
'               version of the browser or the relational version of
'               the browser and runs the appropriate version.
'====================================================================

Option Explicit                     ' Require declaration of variables

'====================================================================
' Subroutine:   Main
' Description:  Initialize interface at application start.
' Arguments:    None
' Return Value: None
'====================================================================
Sub Main()

    Chooser.Show vbModal           ' Display the chooser menu

    If Chooser.ChosenBrowser = "Relational" Then
        rBrowser.Show vbModal      ' Run the relational browser
    ElseIf Chooser.ChosenBrowser = "XML" Then
        xBrowser.Show vbModal      ' Run the XML browser
    End If

End Sub
```

# APPENDIX Q

## PROGRAM LISTING FOR RBROWSER.FRM

```
'====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'====================================================================
' Form:         rBrowser
' Description:  The main interface for the application, which
'               supports the following:
'
'               1.  Drawing features from feature classes
'               2.  Dynamically enabling / disabling drawing feature
'                   classes
'               3.  Set a reference date to be the "current
'                   operational date" for operations in the system
'               4.  Map extent controls (zoom in and out, pan, full
'                   extent)
'               5.  Select features graphically or through XPath query
'               6.  View and edit attributes on a point or line
'
'               The key feature of this application is that there are
'               no user defined data structures; relational tables
'               are the dynamic data structures for all operations.
'====================================================================

Option Explicit                     ' Require declaration of variables

'====================================================================
' Import GDI function for double buffered drawing
'====================================================================
Private Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal x As Long, ByVal y As Long, ByVal
nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal xSrc As Long, ByVal ySrc As Long, ByVal
dwRop As Long) As Long

'====================================================================
' Variable declarations
'====================================================================
Private ButtonTool As String        ' Current tool from the button bar
Private Bound(4) As Single          ' Map xmin,ymin,xmax,ymax
Private ScaleFactor As Single       ' Current map scale (1:SF)
Private SelectTolerance As Single   ' Selection tolerance from 0-2'
Private Click(2) As Single          ' Last click x and y coordinates
Private BufferedMap As PictureBox   ' Temporary picturebox to hold
                                    ' picture for double buffering
Private Conn As ADODB.Connection    ' Persistent connection to Access
Private CurrFC As Integer           ' Current feature class index
Private Record As Long              ' Current feature record number

'====================================================================
' Subroutine:   Form_Load
' Description:  Initialize interface at application start
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub Form_Load()

    Dim Name As String
    Dim sqlString As String
    Dim rs As New ADODB.Recordset
    Dim i As Integer
```

```
' Set a reference to the PictureBox Maps for feature drawing

If Not (rBrowser Is Nothing) Then Set BufferedMap = Map

' Open a connection to the Access database

Set Conn = New ADODB.Connection
Conn.open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & App.Path & "\data\DB.mdb"

' Load feature class names into the table of contents

sqlString = "select * from Catalog_FeatureClass order by draworder"
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
For i = 1 To rs.RecordCount
    DrawClass(i - 1).Tag = rs("NAME").Value
    DrawClass(i - 1).Visible = True
    ClassName(i - 1).Caption = rs("NAME").Value
    ClassName(i - 1).Tag = rs("TABLE").Value
    ClassName(i - 1).Visible = True
    ClassImage(i - 1).Picture = LoadPicture("bitmaps\" & rs("TOCIMAGE").Value)
    ClassImage(i - 1).Visible = True
    rs.MoveNext
Next i
ClassImage(0).BorderStyle = vbFixedSingle
rs.Close

' Set map extent

GetBounds Bound(1), Bound(2), Bound(3), Bound(4)
ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
DisplayScale

' Set the initial tolerances and status variables

SelectTolerance = 5
SelectToleranceLabel.Caption = "5.000"
ButtonTool = "Select"

' Set the classes tab to be active

Tabs.Tab = 0

' Set the reference date to today's date

ReferenceDate.Value = Now
```

```
    ' Set the feature class and draw the map

    ChangeFeatureClass 0
    DrawFeatures
    RefreshForm

End Sub

'====================================================================
' Subroutine:   Form_Unload
' Description:  Shutdown interface at application end
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub Form_Unload(Cancel As Integer)
    Conn.Close
End Sub

'====================================================================
' Subroutine:   Toolbar_ButtonClick
' Description:  Event handler for button clicks on the toolbar
' Arguments:    1. Toolbar button
' Return Value: None
'====================================================================
Private Sub Toolbar_ButtonClick(ByVal Button As MSComctlLib.Button)

    ' Set the tool to be equal to the button's key (if appropriate)
    ' and operate on the button

    Select Case Button.Key

        Case "Select"

            ButtonTool = Button.Key

        Case "ZoomIn" ' Toggle zooming in

            If ButtonTool = Button.Key Then
                ButtonTool = "None"
            Else
                ButtonTool = Button.Key
            End If

        Case "ZoomOut" ' Toggle zooming out

            If ButtonTool = Button.Key Then
```

```
                ButtonTool = "None"
            Else
                ButtonTool = Button.Key
            End If

        Case "Pan" ' Toggle panning

            If ButtonTool = Button.Key Then
                ButtonTool = "None"
            Else
                ButtonTool = Button.Key
            End If

        Case "FullExtent" ' Redraw the full map

            GetBounds Bound(1), Bound(2), Bound(3), Bound(4)
            ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
            DisplayScale
            DrawFeatures
            DrawSelected

    End Select

    ' Update the form

    RefreshForm

End Sub

'====================================================================
' Subroutine:   DrawClass_Click
' Description:  Event handler for checkbox indicating whether or not
'               to draw the associated feature class, redraws the
'               screen after enabling or disabling
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub DrawClass_Click(index As Integer)
    DrawFeatures
    DrawSelected
    RefreshForm
End Sub

'====================================================================
' Subroutine:   ClassImage_Click
' Description:  Event handler for changing feature classes.  Feature
'               classes can be changed either by clicking the image
'               or the name of the feature class.  Redraws the screen
'               after changing feature classes (if a feature was
'               selected).
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub ClassImage_Click(index As Integer)
    ChangeFeatureClass index
    If (SelectedFeatures.ListCount > 0) Then
        ClearSelection
    Else
        RefreshForm
    End If
End Sub

'====================================================================
' Subroutine:   ClassName_Click
' Description:  Event handler for changing feature classes.  Feature
'               classes can be changed either by clicking the image
'               or the name of the feature class.  Redraws the screen
'               after changing feature classes (if a feature was
'               selected).
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub ClassName_Click(index As Integer)
    ChangeFeatureClass index
    If (SelectedFeatures.ListCount > 0) Then
        ClearSelection
    Else
        RefreshForm
    End If
End Sub

'====================================================================
' Subroutine:   ReferenceDate_CloseUp
' Description:  Event handler for changing the reference date, redraws
'               the map and refreshes the form after setting the
'               current record to nothing
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub ReferenceDate_CloseUp()
    ClearSelection
End Sub
```

```
'====================================================================
' Subroutine:   SelectedFeatures_Click
' Description:  Event handler for click on the selected feature IDs
'               list box, sets the current record, updates the
'               form with attributes and redraws the map
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub SelectedFeatures_Click()

    Dim i As Integer
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    Dim fieldName As String
    Dim KeyValue As Long
    Dim TableName As String
    Dim KeyName As String

    ' Check to see if a feature was selected before continuing

    If (SelectedFeatures.Text <> "") Then
        Record = CLng(SelectedFeatures.Text)
    Else
        If (Record <> 0) Then
            DrawFeatures
        End If
        Record = 0
        Attributes.Nodes.Clear
        DrawSelected
        RefreshForm
        Exit Sub
    End If

    ' Determine if a relationship exists and get the appropriate table and key fields

    KeyName = ""
    sqlString = "SELECT * FROM Catalog_Relationships WHERE source_featureclass = """ & _
       ClassName(CurrFC).Caption & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    If (rs.RecordCount = 1) Then
        rs.MoveFirst
        TableName = rs("DESTINATION_OBJECTCLASS").Value
        KeyName = rs("KEY_FIELD").Value
    End If
    rs.Close
```

```
sqlString = "SELECT * FROM Catalog_ObjectClass WHERE name = """ & ClassName(CurrFC).Caption & """"
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
If (rs.RecordCount = 1) Then
    rs.MoveFirst
    TableName = rs("TABLE").Value
End If
rs.Close

' Loop through and place the attributes in the attribute tree list
' Note that shapes must be handled separately from other
' attributes because are complex typed.  Points have a single
' coordinate while lines have many coordinates and a bounding
' box and length attribute

' Place the coordinates in the attributes list

Attributes.Nodes.Clear
Attributes.Nodes.Add , , "SHAPE", "SHAPE"
If (GetClassType(ClassName(CurrFC).Caption) = "point") Then
    sqlString = "SELECT * FROM " & ClassName(CurrFC).Tag & "_Shape WHERE feature_id = " & _
    Record & " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
    "# order by feature_id"
Else
    sqlString = "SELECT * FROM " & ClassName(CurrFC).Tag & "_Shape WHERE feature_id = " & Record _
    & " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
    "# and vertex > 0 order by feature_id, vertex"
End If
rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
i = 0
rs.MoveFirst
Do While Not rs.EOF
    i = i + 1
    Attributes.Nodes.Add "SHAPE", tvwChild, "COORDINATE " & CStr(i), "COORDINATE"
    Attributes.Nodes.Add "COORDINATE " & CStr(i), tvwChild, "X " & CStr(i), "X"
    Attributes.Nodes.Add "X " & CStr(i), tvwChild, , rs("X").Value
    Attributes.Nodes.Add "COORDINATE " & CStr(i), tvwChild, "Y " & CStr(i), "Y"
    Attributes.Nodes.Add "Y " & CStr(i), tvwChild, , rs("Y").Value
    rs.MoveNext
Loop
rs.Close

' Place the bounding box coordinates in the attributes list

If (GetClassType(ClassName(CurrFC).Caption) <> "point") Then
    sqlString = "SELECT * FROM " & ClassName(CurrFC).Tag & "_Shape WHERE feature_id = " & Record & _
    " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
    "# and vertex < 0 order by feature_id, vertex"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    Do While Not rs.EOF
        If (rs("VERTEX").Value = -2) Then
            Attributes.Nodes.Add "SHAPE", tvwChild, "BOUNDBOX", "BOUNDBOX"
            Attributes.Nodes.Add "BOUNDBOX", tvwChild, "XMIN", "XMIN"
            Attributes.Nodes.Add "XMIN", tvwChild, , rs("X").Value
            Attributes.Nodes.Add "BOUNDBOX", tvwChild, "YMIN", "YMIN"
            Attributes.Nodes.Add "YMIN", tvwChild, , rs("Y").Value
        Else
            Attributes.Nodes.Add "BOUNDBOX", tvwChild, "XMAX", "XMAX"
            Attributes.Nodes.Add "XMAX", tvwChild, , rs("X").Value
            Attributes.Nodes.Add "BOUNDBOX", tvwChild, "YMAX", "YMAX"
            Attributes.Nodes.Add "YMAX", tvwChild, , rs("Y").Value
        End If
        rs.MoveNext
    Loop
    rs.Close
End If

' Place other spatial attributes in the attributes list

KeyValue = 0
If (GetClassType(ClassName(CurrFC).Caption) <> "point") Then
    sqlString = "SELECT * FROM " & ClassName(CurrFC).Tag & " WHERE feature_id = " & Record & _
    " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
    "# order by feature_id"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    If (GetClassType(ClassName(CurrFC).Caption) = "line") Then
        Attributes.Nodes.Add "SHAPE", tvwChild, "LENGTH", "LENGTH"
        Attributes.Nodes.Add "LENGTH", tvwChild, , rs("LENGTH").Value
    ElseIf (GetClassType(ClassName(CurrFC).Caption) = "polygon") Then
        Attributes.Nodes.Add "SHAPE", tvwChild, "AREA", "AREA"
        Attributes.Nodes.Add "AREA", tvwChild, , rs("AREA").Value
        Attributes.Nodes.Add "SHAPE", tvwChild, "PERIMETER", "PERIMETER"
        Attributes.Nodes.Add "PERIMETER", tvwChild, , rs("PERIMETER").Value
    End If
    rs.Close
End If

' Place the rest of the attributes in the attributes list

sqlString = "SELECT * FROM " & ClassName(CurrFC).Tag & " WHERE feature_id = " & Record &
    " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
```

```
        "# order by feature_id"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    For i = 0 To (rs.fields.Count - 1)
        fieldName = rs.fields.Item(i).Name
        If (fieldName <> "FEATURE_ID") And (fieldName <> "BORN") And (fieldName <> "RETIRED") And _
        (fieldName <> "AREA") And (fieldName <> "PERIMETER") And (fieldName <> "LENGTH") Then
            Attributes.Nodes.Add , , UCase(fieldName), UCase(fieldName)
            Attributes.Nodes.Add UCase(fieldName), tvwChild, , rs(fieldName).Value
        End If
        If (fieldName = KeyName) Then
            KeyValue = rs(fieldName).Value
        End If
    Next i
    rs.Close

    ' Load any related data

    If (KeyValue <> 0) Then
        Attributes.Nodes.Add , , "RELATED " & UCase(TableName) & " DATA", "RELATED " & _
               UCase(TableName) & " DATA"
        sqlString = "SELECT * FROM " & TableName & " WHERE " & KeyName & " = " & KeyValue & _
               " and born <= #" & ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
               "# order by " & KeyName
        rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
        rs.MoveFirst
        For i = 0 To (rs.fields.Count - 1)
            fieldName = rs.fields.Item(i).Name
            If (fieldName <> KeyName) And (fieldName <> "BORN") And (fieldName <> "RETIRED") Then
                Attributes.Nodes.Add "RELATED " & UCase(TableName) & " DATA", tvwChild, _
                       UCase(fieldName), UCase(fieldName)
                Attributes.Nodes.Add UCase(fieldName), tvwChild, , rs(fieldName).Value
            End If
        Next i
        rs.Close
    End If

    ' Redraw the map and the form

    DrawFeatures
    DrawSelected
    RefreshForm

End Sub

'=====================================================================
' Subroutine:   SQLQuery_KeyPress
' Description:  Event handler for the query string text box.  Pressing
'               enter will call select features with the specified
'               query string.
' Arguments:    None
' Return Value: None
'======================================================================
Private Sub SQLQuery_KeyPress(KeyAscii As Integer)
    If (KeyAscii = 13) Then
      SelectFeatures SQLQuery.Text
    End If
End Sub

'======================================================================
' Subroutine:   SelectToleranceSlider_Scroll
' Description:  Responds to select tolerance slide event, sets the
'               selection tolerance and updates the form
' Arguments:    None
' Return Value: None
'======================================================================
Private Sub SelectToleranceSlider_Scroll()
    SelectTolerance = SelectToleranceSlider.Value / 1000
    SelectToleranceLabel.Caption = Format(SelectTolerance, "#0.000")
End Sub

'======================================================================
' Subroutine:   Map_MouseDown
' Description:  Handle the mouse down click event for selection and
'               map extent tools
' Arguments:    1. Button (left or right mouse)
'               2. Shift (true or false)
'               3. x click on map
'               4. y click on map
' Return Value: None
'======================================================================
Private Sub Map_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
    Dim y1 As Single
    Dim i As Integer

    ' Set the coordinates and operate on the selected button.  Also
    ' create a new point out of the coordinates to work with later.

    x1 = TrX(x)
    y1 = TrY(y)
    DisplayPosition x1, y1

    ' Act on the appropriate button tool

    Select Case ButtonTool

        Case "Select"

            ' Search for features at the click point

            If (Button = vbLeftButton) Then
                SelectFeatures "", x1, y1
            Else
                ClearSelection
            End If

        Case "ZoomIn", "ZoomOut"

            ' Handle map extent tools.  For moving points use the
            ' window location rather than the map location for
            ' smore accurate movement tracking

            If Button = vbLeftButton Then
                ButtonTool = ButtonTool & "Active"
                Click(1) = x
                Click(2) = y
            End If
            Exit Sub

    End Select

End Sub

'====================================================================
' Subroutine:   Map_MouseUp
' Description:  Handle mouse up event for selection and map extent
'               tools on the map
' Arguments:    1. Button (left or right mouse)
'               2. Shift (true or false)
'               3. x click on map
'               4. y click on map
' Return Value: None
'====================================================================
Private Sub Map_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
```

```
 Dim y1 As Single
 Dim cx As Single
 Dim cy As Single
 Dim dx As Single
 Dim dy As Single

 ' Set the coordinates and operate on the selected button.  Also
 ' create a new point out of the coordinates to work with later.

 x1 = TrX(x)
 y1 = TrY(y)
 DisplayPosition x1, y1

' Act on the selected button tool

 Select Case ButtonTool

     Case "ZoomInActive"

         If Button = vbLeftButton Then

             If Abs(Click(1) - x) > Abs(Click(2) - y) Then
                 If (Click(1) < x) Then
                     Bound(1) = TrX(Click(1))
                     Bound(3) = x1
                 Else
                     Bound(1) = x1
                     Bound(3) = TrX(Click(1))
                 End If
                 If (Click(2) > y) Then
                     cy = TrY(Click(2) - (Abs(Click(2) - y) / 2))
                 Else
                     cy = TrY(y - (Abs(y - Click(2)) / 2))
                 End If
                 ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
                 Bound(2) = -cy - (Map.ScaleHeight * ScaleFactor / 2)
                 Bound(4) = -cy + (Map.ScaleHeight * ScaleFactor / 2)
             Else
                 If (Click(2) < y) Then
                     Bound(2) = -TrY(Click(2))
                     Bound(4) = -y1
                 Else
                     Bound(2) = -y1
                     Bound(4) = -TrY(Click(2))
                 End If
                 If (Click(1) > x) Then
```

```
                cx = TrX(Click(1) - (Abs(Click(1) - x) / 2))
            Else
                cx = TrX(x - (Abs(x - Click(1)) / 2))
            End If
            ScaleFactor = Abs(Bound(2) - Bound(4)) / Map.ScaleHeight
            Bound(1) = cx - (Map.ScaleWidth * ScaleFactor / 2)
            Bound(3) = cx + (Map.ScaleWidth * ScaleFactor / 2)
        End If
        ButtonTool = "ZoomIn"
        DisplayScale
        DrawFeatures
        DrawSelected
        RefreshForm
        Exit Sub

    End If

Case "ZoomOutActive"

    If Button = vbLeftButton Then

        If Abs(Click(1) - x) > Abs(Click(2) - y) Then
            ScaleFactor = ScaleFactor * Map.ScaleWidth / Abs(Click(1) - x)
        Else
        End If
        If (Click(2) > y) Then
            cy = TrY(Click(2) - (Abs(Click(2) - y) / 2))
        Else
            cy = TrY(y - (Abs(y - Click(2)) / 2))
        End If
        If (Click(1) > x) Then
            cx = TrX(Click(1) - (Abs(Click(1) - x) / 2))
        Else
            cx = TrX(y - (Abs(x - Click(1)) / 2))
        End If
        Bound(1) = cx - (Map.ScaleWidth * ScaleFactor / 2)
        Bound(3) = cx + (Map.ScaleWidth * ScaleFactor / 2)
        Bound(2) = -cy - (Map.ScaleHeight * ScaleFactor / 2)
        Bound(4) = -cy + (Map.ScaleHeight * ScaleFactor / 2)
        ButtonTool = "ZoomOut"
        DisplayScale
        DrawFeatures
        DrawSelected
        RefreshForm
        Exit Sub
```

```
            End If

        Case "Pan"

            If Button = vbLeftButton Then

                dx = x1 - ((Bound(1) + Bound(3)) / 2)
                dy = y1 + ((Bound(2) + Bound(4)) / 2)

                Bound(1) = Bound(1) + dx
                Bound(2) = Bound(2) - dy
                Bound(3) = Bound(3) + dx
                Bound(4) = Bound(4) - dy
                DisplayScale
                DrawFeatures
                DrawSelected
                RefreshForm

            End If

    End Select

End Sub

'=====================================================================
' Subroutine:   Map_MouseMove
' Description:  Handles mouse move click event for map extent tools
' Arguments:    1. Button (right or left mouse button)
'               2. Shift (true or false)
'               3. x click on map
'               4. y click on map
' Return Value: None
'=====================================================================
Private Sub Map_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
    Dim y1 As Single

    ' Set the coordinates

    x1 = TrX(x)
    y1 = TrY(y)
    DisplayPosition x1, y1

    ' Operate on the appropriate button tool
    If (ButtonTool = "ZoomInActive") Or (ButtonTool = "ZoomOutActive") Then
        Map.Cls
        Map.ForeColor = RGB(50, 50, 50)
        Map.FillStyle = vbFSTransparent
        Map.Line (Click(1), Click(2))-(x, y), , B
    End If

End Sub

'====================================================================
' Subroutine:   ChangeFeatureClass
' Description:  Sets the feature class, resets the record
'               and redraws the screen.
' Arguments:    1. Table of contents index of feature class
' Return Value: None
'====================================================================
Public Sub ChangeFeatureClass(index As Integer)
    CurrFC = index
    Record = 0
    RefreshForm
End Sub

'====================================================================
' Subroutine:   RefreshForm
' Description:  Updates the buttons, mouse pointers and table of
'               contents and refreshes the screen
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub RefreshForm()

    Dim Btn As Button
    Dim i As Integer

    ' Set all buttons unselected

    For Each Btn In Toolbar.Buttons
        Btn.Value = tbrUnpressed
    Next Btn

    ' Process the tools selection

    Select Case ButtonTool
        Case "Select"
            Toolbar.Buttons("Select").Value = tbrPressed
            DisplayMessage "Click to select an feature of the " & _
```

```
               "specified feature type; right click to " & "unselect selected features."
        Case "ZoomIn", "ZoomInActive"
            Toolbar.Buttons("ZoomIn").Value = tbrPressed
            DisplayMessage "Click to zoom in around the click rectangle."
        Case "ZoomOut", "ZoomOutActive"
            Toolbar.Buttons("ZoomOut").Value = tbrPressed
            DisplayMessage "Click to zoom out around the click rectangle."
        Case "Pan", "PanActive"
            Toolbar.Buttons("Pan").Value = tbrPressed
            DisplayMessage "Drag the map to pan."
        End Select

    ' Process the current feature class

    For i = 0 To 8
        If (CurrFC = i) Then
            ClassImage(i).BorderStyle = 1
        Else
            ClassImage(i).BorderStyle = 0
        End If
    Next i

    ' Update the mouse pointer

    Select Case ButtonTool
        Case "Select"
            Map.MousePointer = vbDefault
        Case "ZoomIn", "ZoomInActive"
            Map.MouseIcon = imlPointers.ListImages(1).Picture
            Map.MousePointer = vbCustom
        Case "ZoomOut", "ZoomOutActive"
            Map.MouseIcon = imlPointers.ListImages(2).Picture
            Map.MousePointer = vbCustom
        Case "Pan", "PanActive"
            Map.MouseIcon = imlPointers.ListImages(3).Picture
            Map.MousePointer = vbCustom
    End Select

    ' Refresh the form

    Me.Refresh

End Sub

'====================================================================
' Subroutine:   DisplayMessage
' Description:  Writes a specified message to the status bar
' Arguments:    1. Message string
' Return Value: None
'====================================================================
Public Sub DisplayMessage(Message As String)
    UpdateStatusBar Message, 1
End Sub

'====================================================================
' Subroutine:   DisplayScale
' Description:  Writes the current scale factor to the status bar
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DisplayScale()
    UpdateStatusBar "1:" & CStr(ScaleFactor), 2
End Sub

'====================================================================
' Subroutine:   DisplayPosition
' Description:  Writes the current position to the status bar
' Arguments:    1. x coordinate in map units
'               2. y coordinate in map units
' Return Value: None
'====================================================================
Public Sub DisplayPosition(x As Single, y As Single)
    UpdateStatusBar x & ", " & y, 3
End Sub

'====================================================================
' Subroutine:   UpdateStatusBar
' Description:  Displays a user specified message in the specified
'               status bar panel (1 - 3)
' Arguments:    1. User specified message
'               2. Status bar panel number
' Return Value: None
'====================================================================
Public Sub UpdateStatusBar(Message As String, Panel As Integer)
    StatusBar.Panels.Item(Panel).Text = Message
End Sub

'====================================================================
' Subroutine:   ShowError
' Description:  Displays a dialog with an error message
' Arguments:    1. Error string
' Return Value: None
```

```
'====================================================================
Public Sub ShowError(Error As String)
    MsgBox "Error : " & Error, vbOKOnly + vbExclamation, "Error Message"
End Sub

'====================================================================
' Subroutine:   DrawFeatures
' Description:  Draws features from the XML tables
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DrawFeatures()

    Dim j As Integer
    Dim x0 As Single
    Dim y0 As Single
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim sqlString As String
    Dim rs As New ADODB.Recordset
    Dim Color As Long
    Dim MarkerWidth As Single
    Dim MarkerHeight As Single
    Dim LineWidth As Integer
    Dim Symbol As String
    Dim ClassType As String
    Dim CurrentFeature As Long

    ' Set the hourglass since drawing takes a little bit of time
    ' We set both the map and the form since we don't which input came
    ' from.  The mousepointer of the input handle will remain as
    ' drawing commences

    Map.MousePointer = vbHourglass
    Me.MousePointer = vbHourglass

    ' Store the image in memory for more efficient drawing

    Map.AutoRedraw = True

    ' Check to see if all of the map has been initialized;
    ' if not then exit

    If (ScaleFactor = 0) Then Exit Sub
```

```
' Clear the device

Map.Cls

' Loop through each of the feature classes

For j = 8 To 0 Step -1

    If (DrawClass(j).Value = 1) Then

        ' Get the symbology

        ClassType = GetClassType(ClassName(j).Caption)
        If (ClassType = "point") Then
            Color = GetSymbolColor(ClassName(j).Caption, ClassType)
            Symbol = GetSymbolType(ClassName(j).Caption)
            If (Symbol = "circle") Then
                MarkerWidth = GetSymbolWidth(ClassName(j).Caption)
                MarkerHeight = GetSymbolHeight(ClassName(j).Caption)
            Else
                MarkerWidth = GetSymbolWidth(ClassName(j).Caption)
                MarkerHeight = GetSymbolHeight(ClassName(j).Caption)
            End If
        Else
            Color = GetSymbolColor(ClassName(j).Caption, ClassType)
            LineWidth = GetSymbolLineWidth(ClassName(j).Caption)
        End If

        ' Redraw the features

        If (ClassType = "point") Then
            sqlString = "select * from " & ClassName(j).Tag & "_Shape where born <= #" & _
                   ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
                   "# order by feature_id"
        Else
            sqlString = "select * from " & ClassName(j).Tag & "_Shape where born <= #" & _
                   ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
                   "# and vertex > 0 order by feature_id, vertex"
        End If
        rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
        If (rs.RecordCount > 0) Then
            rs.MoveFirst
            If (ClassType <> "point") Then
                CurrentFeature = rs("FEATURE_ID").Value
                x1 = rs("X").Value
                y1 = rs("Y").Value
                x0 = x1
                y0 = y1
                Do While Not rs.EOF
                    If (rs("FEATURE_ID").Value <> CurrentFeature) Then
                        If (ClassType = "polygon") Then
                            x2 = x0
                            y2 = y0
                            DrawLine ClassName(j).Caption, x1, y1, x2, y2, Color, LineWidth
                        End If
                        CurrentFeature = rs("FEATURE_ID").Value
                        x1 = rs("X").Value
                        y1 = rs("Y").Value
                        x0 = x1
                        y0 = y1
                    Else
                        x2 = rs("X").Value
                        y2 = rs("Y").Value
                        DrawLine ClassName(j).Caption, x1, y1, x2, y2, Color, LineWidth
                        x1 = x2
                        y1 = y2
                    End If
                    rs.MoveNext
                Loop
            Else
                Do While Not rs.EOF
                    DrawPoint ClassName(j).Caption, rs("X").Value, rs("Y").Value, Color, _
                          MarkerWidth, MarkerHeight, Symbol
                    rs.MoveNext
                Loop
            End If
        End If
        rs.Close

    End If

Next j

' BitBlt the Map contents to Map and refresh the screen

BitBlt Map.hDC, 0, 0, Map.ScaleWidth, Map.ScaleHeight, BufferedMap, 0, 0, &HCC0020
Map.Refresh
Map.AutoRedraw = False

' Reset the mousepointer for the form
```

```
    Me.MousePointer = vbDefault

End Sub

'====================================================================
' Subroutine:   DrawSelected
' Description:  Draws the selected feature
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DrawSelected()

    Dim x0 As Single
    Dim y0 As Single
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim Color As Long
    Dim MarkerWidth As Single
    Dim MarkerHeight As Single
    Dim LineWidth As Integer
    Dim Symbol As String
    Dim ClassType As String
    Dim sqlString As String
    Dim rs As New ADODB.Recordset

    ' Get the symbology

    ClassType = GetClassType(ClassName(CurrFC).Caption)
    If (ClassType = "point") Then
        Color = GetSymbolColor(ClassName(CurrFC).Caption, ClassType)
        Symbol = GetSymbolType(ClassName(CurrFC).Caption)
        If (Symbol = "circle") Then
            MarkerWidth = GetSymbolWidth(ClassName(CurrFC).Caption)
            MarkerHeight = GetSymbolHeight(ClassName(CurrFC).Caption)
        Else
            MarkerWidth = GetSymbolWidth(ClassName(CurrFC).Caption)
            MarkerHeight = GetSymbolHeight(ClassName(CurrFC).Caption)
        End If
    Else
        Color = GetSymbolColor(ClassName(CurrFC).Caption, ClassType)
        LineWidth = GetSymbolLineWidth(ClassName(CurrFC).Caption)
    End If

    ' Draw the currently selected (or edited) feature
    If (Record > 0) Then
        If (ClassType = "point") Then
            sqlString = "select * from " & ClassName(CurrFC).Tag & "_Shape where feature_id = " & _
               Record & " and born <= #" & ReferenceDate.Value & "# and retired > #" & _
               ReferenceDate.Value & "# order by feature_id"
        Else
            sqlString = "select * from " & ClassName(CurrFC).Tag & "_Shape where feature_id = " & _
               Record & " and born <= #" & ReferenceDate.Value & "# and retired > #" & _
               ReferenceDate.Value & "# and vertex > 0 order by feature_id, vertex"
        End If
        rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
        rs.MoveFirst
        If (ClassType <> "point") Then
            x1 = rs("X").Value
            y1 = rs("Y").Value
            x0 = x1
            y0 = y1
            Do While Not rs.EOF
                x2 = rs("X").Value
                y2 = rs("Y").Value
                DrawLine ClassName(CurrFC).Caption, x1, y1, x2, y2, Color, LineWidth, True
                x1 = x2
                y1 = y2
                rs.MoveNext
            Loop
            If (ClassType = "polygon") Then
                x2 = x0
                y2 = y0
                DrawLine ClassName(CurrFC).Caption, x1, y1, x2, y2, Color, LineWidth, True
            End If
        Else
            Do While Not rs.EOF
                DrawPoint ClassName(CurrFC).Caption, rs("X").Value, rs("Y").Value, Color, _
                       MarkerWidth, MarkerHeight, Symbol, True
                rs.MoveNext
            Loop
        End If
        rs.Close
    End If

End Sub

'=====================================================================
' Subroutine:   DrawPoint
' Description:  Draws a feature point with the appropriate symbology.
'               Draws a selected feature point in cyan.
' Arguments:    1. Feature class name
'               2. x coordinate of the point
'               3. y coordinate of the point
'               4. Optionally whether the point is currently selected
' Return Value: None
'====================================================================
Public Sub DrawPoint(ClassName As String, x As Single, y As Single, Color As Long, Width As Single, _
Height As Single, Symbol As String, Optional Selected As Boolean = False)

    ' Set the color

    If Selected Then
        BufferedMap.ForeColor = vbYellow
        BufferedMap.FillColor = vbYellow
    Else
        BufferedMap.ForeColor = vbBlack
        BufferedMap.FillColor = Color
    End If

    ' Draw the appropriate symbol

    BufferedMap.FillStyle = vbFSSolid
    If (GetSymbolType(ClassName) = "circle") Then
        BufferedMap.Circle (UTrX(x), UTrY(y)), Width
    Else
        BufferedMap.Line (UTrX(x) - Width, UTrY(y) - Height)-(UTrX(x) + Width, UTrY(y) + Height), , B
    End If

End Sub

'====================================================================
' Subroutine:   DrawLine
' Description:  Draws a feature line segment with the appropriate
'               symbology. Draws a selected feature line in cyan.
' Arguments:    1. Feature class name
'               2. first x coordinate of the line segment
'               3. first y coordinate of the line segment
'               4. last x coordinate of the line segment
'               5. last y coordinate of the line segment
'               6. Optionally whether the feature that the line
'                  segment is part of is currently selected
' Return Value: None
'====================================================================
Public Sub DrawLine(ClassName As String, x1 As Single, y1 As Single, x2 As Single, y2 As Single, Color As
Long, Width As Integer, Optional Selected As Boolean = False)
```

```
    ' Set the color

    If Selected Then
        BufferedMap.ForeColor = vbYellow
        BufferedMap.FillColor = vbYellow
    Else
        BufferedMap.ForeColor = Color
        BufferedMap.FillColor = BufferedMap.ForeColor
    End If

    ' Draw the appropriate symbol

    BufferedMap.FillStyle = vbFSSolid
    BufferedMap.DrawWidth = Width
    BufferedMap.Line (UTrX(x1), UTrY(y1))-(UTrX(x2), UTrY(y2))
    BufferedMap.DrawWidth = 1

End Sub

'====================================================================
' Subroutine:   TrX
' Description:  Converts an x coordinate from twips to map units
' Arguments:    1. Twip x coordinate
' Return Value: Map units x value
'====================================================================
Public Function TrX(x As Single) As Single
    TrX = ((x / Map.ScaleWidth) * (Bound(3) - Bound(1))) + Bound(1)
End Function

'====================================================================
' Subroutine:   TrY
' Description:  Converts a y coordinate from twips to map units
' Arguments:    1. Twip y coordinate
' Return Value: Map units y value
'====================================================================
Public Function TrY(y As Single) As Single
    TrY = -(((y / Map.ScaleHeight) * (Bound(4) - Bound(2))) + Bound(2))
End Function

'====================================================================
' Subroutine:   UTrX
' Description:  Converts an x coordinate from map units to twips
' Arguments:    1. Map units x coordinate
' Return Value: Twip x value
'====================================================================
Public Function UTrX(x As Single) As Single
    UTrX = ((x - Bound(1)) / (Bound(3) - Bound(1))) * (Map.ScaleWidth)
End Function

'====================================================================
' Subroutine:   UTrY
' Description:  Converts a y coordinate from map units to twips
' Arguments:    1. Map units y coordinate
' Return Value: Twip y value
'====================================================================
Public Function UTrY(y As Single) As Single
    UTrY = ((-y - Bound(2)) / (Bound(4) - Bound(2))) * (Map.ScaleHeight)
End Function

'====================================================================
' Subroutine:   GetBounds
' Description:  Loads the boundaries of the map into memory
' Arguments:    1. x minimum reference (set by subroutine)
'               2. y minimum reference (set by subroutine)
'               3. x maximum reference (set by subroutine)
'               4. y maximum reference (set by subroutine)
' Return Value: None (reference values set)
'====================================================================
Public Sub GetBounds(ByRef xmin As Single, ByRef ymin As Single, ByRef xmax As Single, _
ByRef ymax As Single)

    Dim sqlString As String
    Dim rs As New ADODB.Recordset

    sqlString = "select * from Catalog_BoundBox"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    xmin = CSng(rs("XMIN").Value)
    ymin = CSng(rs("YMIN").Value)
    xmax = CSng(rs("XMAX").Value)
    ymax = CSng(rs("YMAX").Value)
    rs.Close

End Sub

'====================================================================
' Subroutine:   GetClassType
' Description:  Returns the feature class type (point or line)
' Arguments:    1. Class name
' Return Value: Feature class type
'====================================================================
Public Function GetClassType(ClassName As String) As String
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_FeatureClass where name = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetClassType = rs("TYPE").Value
    rs.Close
End Function

'====================================================================
' Subroutine:   GetSymbolColor
' Description:  Returns the symbol color associate with the specified
'               feature class (Long from RGB())
' Arguments:    1. Class name
' Return Value: Symbol color
'====================================================================
Public Function GetSymbolColor(ClassName As String, ClassType As String) As Long
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_Symbology where featureclass = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetSymbolColor = RGB(rs("COLOR_RED").Value, rs("COLOR_GREEN").Value, rs("COLOR_BLUE").Value)
    rs.Close
End Function

'====================================================================
' Subroutine:   GetSymbolType
' Description:  Returns the symbol type associate with the specified
'               feature class (circle, rectangle, etc.)
' Arguments:    1. Class name
' Return Value: Symbol type
'====================================================================
Public Function GetSymbolType(ClassName As String) As String
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_Symbology where featureclass = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetSymbolType = rs("SYMBOL").Value
    rs.Close
End Function

'====================================================================
' Subroutine:   GetSymbolHeight
' Description:  Returns the symbol height associate with the specified
'               feature class (for rectangular markers)
' Arguments:    1. Class name
' Return Value: Symbol height
'=====================================================================
Public Function GetSymbolHeight(ClassName As String) As Single
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_Symbology where featureclass = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetSymbolHeight = rs("SIZE_HEIGHT").Value
    rs.Close
End Function

'=====================================================================
' Subroutine:   GetSymbolWidth
' Description:  Returns the symbol width associate with the specified
'               feature class (for rectangular markers)
' Arguments:    1. Class name
' Return Value: Symbol width
'=====================================================================
Public Function GetSymbolWidth(ClassName As String) As Single
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_Symbology where featureclass = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetSymbolWidth = rs("SIZE_WIDTH").Value
    rs.Close
End Function

'=====================================================================
' Subroutine:   GetSymbolLineWidth
' Description:  Returns the line width associate with the specified
'               feature class
' Arguments:    1. Class name
' Return Value: Symbol width
'=====================================================================
Public Function GetSymbolLineWidth(ClassName As String) As Single
    Dim rs As New ADODB.Recordset
    Dim sqlString As String
    sqlString = "select * from Catalog_Symbology where featureclass = """ & ClassName & """"
    rs.open sqlString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    GetSymbolLineWidth = rs("SIZE_WIDTH").Value
    rs.Close
End Function

'====================================================================
' Subroutine:   SelectFeatures
' Description:  Selects features based on either an XPath query or
'               on a selected location, filtered by temporal
'               existence.  Adds a list of selected feature IDs to
'               the SelectedFeatures list box.
' Arguments:    1. Query string or "" if spatial select
'               2. Optional x coordinate of location
'               3. Optional y coordinate of location
' Return Value: None
'====================================================================
Public Sub SelectFeatures(QueryString As String, Optional x As Variant, Optional y As Variant)

    Dim x0 As Single
    Dim y0 As Single
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim l As Single
    Dim r As Single
    Dim d(2) As Single
    Dim rs As New ADODB.Recordset
    Dim CurrentFeature As Long
    Dim LastFeature As Long
    Dim ClassType As String

    ' Clear the selected feature IDs list

    SelectedFeatures.Clear

    ' If no x and y coordinates are specified, use the
    ' SQL query expression, and check the date the
    ' feature existed.

    If (IsMissing(x) And IsMissing(y)) Then
        If (QueryString <> "") Then
            QueryString = "SELECT * FROM " & ClassName(CurrFC).Tag & " WHERE " & QueryString
            QueryString = QueryString & " and born <= #" & ReferenceDate.Value & "# and retired > #" _
               & ReferenceDate.Value & "#"
            rs.open QueryString, Conn, adOpenKeyset, adLockOptimistic
            If (rs.RecordCount > 0) Then
                rs.MoveFirst
            Do While Not rs.EOF
                SelectedFeatures.AddItem rs("FEATURE_ID").Value
                rs.MoveNext
            Loop
        End If
        rs.Close
    Else
        ShowError "No selection criteria specified."
    End If

' Otherwise do a spatial search by checking within the search
' tolerance of the point clicked.  Loop through each coordinate
' and check to be sure it exists at that point in time.

Else

    LastFeature = 0
    ClassType = GetClassType(ClassName(CurrFC).Caption)
    If (ClassType = "point") Then
        QueryString = "SELECT * FROM " & ClassName(CurrFC).Tag & "_Shape WHERE born <= #" & _
           ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & "# order by feature_id"
    Else
        QueryString = "SELECT * FROM " & ClassName(CurrFC).Tag & "_Shape WHERE born <= #" & _
           ReferenceDate.Value & "# and retired > #" & ReferenceDate.Value & _
           "# and vertex > 0 order by feature_id, vertex"
    End If
    rs.open QueryString, Conn, adOpenKeyset, adLockOptimistic
    rs.MoveFirst
    If (rs.RecordCount > 0) Then
        If (ClassType <> "point") Then
            CurrentFeature = rs("FEATURE_ID").Value
            x1 = rs("X").Value
            y1 = rs("Y").Value
            x0 = x1
            y0 = y1
            Do While Not rs.EOF
                If (x >= (x1 - SelectTolerance) And (x <= (x1 + SelectTolerance))) And _
                (y >= (y1 - SelectTolerance) And (y <= (y1 + SelectTolerance))) Then
                    If (CurrentFeature > LastFeature) Then
                        SelectedFeatures.AddItem CurrentFeature
                        LastFeature = CurrentFeature
                    End If
                Else
                    If (rs("FEATURE_ID").Value <> CurrentFeature) Then
                        If (ClassType = "polygon") Then
                            x2 = x0
            y2 = y0
            l = Sqr((x2 - x1) ^ 2 + (y2 - y1) ^ 2)
            If (l <> 0) Then
              r = ((y1 - y) * (y1 - y2) - (x1 - x) * (x2 - x1)) / l ^ 2
              If (r >= 0) And (r <= 1) Then
                d(1) = x1 + r * (x2 - x1)
                d(2) = y1 + r * (y2 - y1)
                If (x >= (d(1) - SelectTolerance) And _
               (x <= (d(1) + SelectTolerance))) And _
               (y >= (d(2) - SelectTolerance) And _
               (y <= (d(2) + SelectTolerance))) Then
                    If (CurrentFeature > LastFeature) Then
                        SelectedFeatures.AddItem CurrentFeature
                        LastFeature = CurrentFeature
                    End If
                End If
              End If
            End If
        End If
        CurrentFeature = rs("FEATURE_ID").Value
        x1 = rs("X").Value
        y1 = rs("Y").Value
        x0 = x1
        y0 = y1
    Else
        x2 = rs("X").Value
        y2 = rs("Y").Value
        l = Sqr((x2 - x1) ^ 2 + (y2 - y1) ^ 2)
        If (l <> 0) Then
          r = ((y1 - y) * (y1 - y2) - (x1 - x) * (x2 - x1)) / l ^ 2
          If (r >= 0) And (r <= 1) Then
            d(1) = x1 + r * (x2 - x1)
            d(2) = y1 + r * (y2 - y1)
            If (x >= (d(1) - SelectTolerance) And (x <= (d(1) + SelectTolerance))) _
           And (y >= (d(2) - SelectTolerance) And (y <= (d(2) + SelectTolerance))) Then
                If (CurrentFeature > LastFeature) Then
                    SelectedFeatures.AddItem CurrentFeature
                    LastFeature = CurrentFeature
                End If
            End If
          End If
        End If
        x1 = x2
        y1 = y2
    End If
End If
```

```
                    rs.MoveNext
                Loop
            Else
                Do While Not rs.EOF
                    x1 = rs("X").Value
                    y1 = rs("Y").Value
                    If (x >= (x1 - SelectTolerance)) And (x <= (x1 + SelectTolerance)) And _
                    (y >= (y1 - SelectTolerance) And (y <= (y1 + SelectTolerance))) Then
                        SelectedFeatures.AddItem rs("FEATURE_ID").Value
                    End If
                    rs.MoveNext
                Loop
            End If
            rs.Close
        End If

    End If

    ' Set the selected feature to be the first item in the list

    SelectedFeatures.Text = SelectedFeatures.List(0)

End Sub

'====================================================================
' Subroutine:   ClearSelection
' Description:  Clears the selection and resets the interface
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub ClearSelection()
    Record = 0
    SelectedFeatures.Clear
    Attributes.Nodes.Clear
    DrawFeatures
    RefreshForm
End Sub
```

# APPENDIX R

## PROGRAM LISTING FOR XBROWSER.FRM

```
'====================================================================
' Copyright (c) 2003 Erik Shepard
'
' Permission to use, copy, modify, distribute, and sell this software
' and its documentation for any purpose is hereby granted without fee,
' provided that (i) the above copyright notices and this permission
' notice appear in all copies of the software and related
' documentation, and (ii) the name of Erik Shepard may not be used in
' any advertising or publicity relating to the software without the
' specific, prior written permission of Erik Shepard.
'
' THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
' EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
' WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
'
' IN NO EVENT SHALL ERIK SHEPARD BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
' INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES
' WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR
' NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF
' LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
' PERFORMANCE OF THIS SOFTWARE.
'====================================================================
' Form:         xBrowser
' Description:  The main interface for the application, which
'               supports the following:
'
'               1.  Drawing features from feature classes
'               2.  Dynamically enabling / disabling drawing feature
'                   classes
'               3.  Set a reference date to be the "current
'                   operational date" for operations in the system
'               4.  Map extent controls (zoom in and out, pan, full
'                   extent)
'               5.  Select features graphically or through XPath query
'               6.  View and edit attributes on a point or line
'
'               The key feature of this application is that there are
'               no user defined data structures; XML is the dynamic
'               data structure for all operations.
'====================================================================

Option Explicit                     ' Require declaration of variables

'====================================================================
' Import GDI function for double buffered drawing
'====================================================================
Private Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal x As Long, ByVal y As Long, ByVal
nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal xSrc As Long, ByVal ySrc As Long, ByVal
dwRop As Long) As Long

'====================================================================
' Variable declarations
'====================================================================
Private ButtonTool As String        ' Current tool from the button bar
Private Bound(4) As Single          ' Map xmin,ymin,xmax,ymax
Private ScaleFactor As Single       ' Current map scale (1:SF)
Private SelectTolerance As Single   ' Selection tolerance from 0-2'
Private Click(2) As Single          ' Last click x and y coordinates
Private BufferedMap As PictureBox   ' Temporary picturebox to hold
                                    ' picture for double buffering
Private Catalog As DOMDocument40    ' Persistent copy of the catalog
                                    ' for efficient access
Private Table As DOMDocument40      ' Current feature class XML node
Private CurrFCName As String        ' Current feature class name
Private Record As IXMLDOMNode       ' Current feature record XML node

'====================================================================
' Subroutine:   Form_Load
' Description:  Initialize interface at application start
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub Form_Load()

    Dim Name As IXMLDOMNode
    Dim i As Integer
```

```
' Set a reference to the PictureBox Maps for feature drawing

If Not (xBrowser Is Nothing) Then Set BufferedMap = Map

' Load the XML Catalog

LoadCatalog

' Load feature class names into the table of contents

i = 0
For Each Name In Catalog.selectNodes("catalog/featureclass/name")
    DrawClass(i).Tag = Name.Text
    DrawClass(i).Visible = True
    ClassName(i).Caption = Name.Text
    ClassName(i).Tag = Catalog.selectSingleNode("catalog/featureclass[name = """ & _
           Name.Text & """]/table").Text
    ClassName(i).Visible = True
    ClassImage(i).Picture = LoadPicture("bitmaps\" & _
           Catalog.selectSingleNode("catalog/featureclass[name = """ & Name.Text & _
           """]/tocimage").Text)
    ClassImage(i).Visible = True
    i = i + 1
Next Name
ClassImage(0).BorderStyle = vbFixedSingle

' Set map extent

GetBounds Bound(1), Bound(2), Bound(3), Bound(4)
ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
DisplayScale

' Set the initial tolerances and status variables

SelectTolerance = 5
SelectToleranceLabel.Caption = "5.000"
ButtonTool = "Select"

' Set the classes tab to be active

Tabs.Tab = 0

' Set the reference date to today's date

ReferenceDate.Value = Now
```

```
    ' Set the feature class and draw the map

    ChangeFeatureClass 0
    DrawFeatures
    RefreshForm

End Sub

'====================================================================
' Subroutine:   Toolbar_ButtonClick
' Description:  Event handler for button clicks on the toolbar
' Arguments:    1. Toolbar button
' Return Value: None
'====================================================================
Private Sub Toolbar_ButtonClick(ByVal Button As MSComctlLib.Button)

    ' Set the tool to be equal to the button's key (if appropriate)
    ' and operate on the button

    Select Case Button.Key

        Case "Select"

            ButtonTool = Button.Key

        Case "ZoomIn" ' Toggle zooming in

            If ButtonTool = Button.Key Then
                ButtonTool = "None"
            Else
                ButtonTool = Button.Key
            End If

        Case "ZoomOut" ' Toggle zooming out

            If ButtonTool = Button.Key Then
                ButtonTool = "None"
            Else
                ButtonTool = Button.Key
            End If

        Case "Pan" ' Toggle panning

            If ButtonTool = Button.Key Then
                ButtonTool = "None"
            Else
```

```
                ButtonTool = Button.Key
            End If

        Case "FullExtent" ' Redraw the full map

            GetBounds Bound(1), Bound(2), Bound(3), Bound(4)
            ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
            DisplayScale
            DrawFeatures
            DrawSelected

    End Select

    ' Update the form

    RefreshForm

End Sub

'====================================================================
' Subroutine:   DrawClass_Click
' Description:  Event handler for checkbox indicating whether or not
'               to draw the associated feature class, redraws the
'               screen after enabling or disabling
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub DrawClass_Click(index As Integer)
    DrawFeatures
    DrawSelected
    RefreshForm
End Sub

'====================================================================
' Subroutine:   ClassImage_Click
' Description:  Event handler for changing feature classes.  Feature
'               classes can be changed either by clicking the image
'               or the name of the feature class.  Redraws the screen
'               after changing feature classes (if a feature was
'               selected).
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub ClassImage_Click(index As Integer)
    ChangeFeatureClass index
    If (SelectedFeatures.ListCount > 0) Then
```

```
        ClearSelection
    Else
        RefreshForm
    End If
End Sub

'====================================================================
' Subroutine:   ClassName_Click
' Description:  Event handler for changing feature classes.  Feature
'               classes can be changed either by clicking the image
'               or the name of the feature class.  Redraws the screen
'               after changing feature classes (if a feature was
'               selected).
' Arguments:    1. Position in the control array (0 based)
' Return Value: None
'====================================================================
Private Sub ClassName_Click(index As Integer)
    ChangeFeatureClass index
    If (SelectedFeatures.ListCount > 0) Then
        ClearSelection
    Else
        RefreshForm
    End If
End Sub

'====================================================================
' Subroutine:   ReferenceDate_CloseUp
' Description:  Event handler for changing the reference date, redraws
'               the map and refreshes the form after setting the
'               current record to nothing
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub ReferenceDate_CloseUp()
    ClearSelection
End Sub

'====================================================================
' Subroutine:   SelectedFeatures_Click
' Description:  Event handler for click on the selected feature IDs
'               list box, sets the current record, updates the
'               form with attributes and redraws the map
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub SelectedFeatures_Click()
```

```
Dim i As Integer
Dim AttributeNode As IXMLDOMNode
Dim ShapeNode As IXMLDOMNode
Dim CatalogNode As IXMLDOMNode
Dim RelatedTable As DOMDocument40
Dim RelatedNode As IXMLDOMNode
Dim OldRecord As IXMLDOMNode
Dim RelatedObjectClass As String
Dim KeyField As String
Dim KeyValue As String

' Set a reference to the current record

Set OldRecord = Record
Set Record = Table.selectSingleNode("featureclass/feature[@id=""" & SelectedFeatures.Text & """]")

' Check to see if a feature was selected before continuing

If (Record Is Nothing) Then
    Attributes.Nodes.Clear
    If (Not OldRecord Is Nothing) Then
        DrawFeatures
    End If
    DrawSelected
    RefreshForm
    Exit Sub
End If

' Loop through and place the attributes in the attribute tree list
' Note that shapes must be handled separately from other
' attributes because are complex typed.  Points have a single
' coordinate while lines have many coordinates and a bounding
' box and length attribute

Attributes.Nodes.Clear
For Each AttributeNode In Record.selectNodes("*[@born <= """ & RefDate & """ and @retired > """ _
& RefDate & """]")
    If (AttributeNode.nodeName = "shape") Then
        Attributes.Nodes.Add , , "SHAPE", "SHAPE"
        i = 0
        For Each ShapeNode In AttributeNode.selectNodes("*")
            If (ShapeNode.nodeName = "coordinate") Then
                i = i + 1
                Attributes.Nodes.Add "SHAPE", tvwChild, "COORDINATE " & CStr(i), "COORDINATE"
                Attributes.Nodes.Add "COORDINATE " & CStr(i), tvwChild, "X " & CStr(i), "X"
                Attributes.Nodes.Add "X " & CStr(i), tvwChild, , _
                   CSng(ShapeNode.selectSingleNode("x").Text)
                Attributes.Nodes.Add "COORDINATE " & CStr(i), tvwChild, "Y " & CStr(i), "Y"
                Attributes.Nodes.Add "Y " & CStr(i), tvwChild, , _
                   CSng(ShapeNode.selectSingleNode("y").Text)
            ElseIf (ShapeNode.nodeName = "boundbox") Then
                Attributes.Nodes.Add "SHAPE", tvwChild, "BOUNDBOX", "BOUNDBOX"
                Attributes.Nodes.Add "BOUNDBOX", tvwChild, "XMIN", "XMIN"
                Attributes.Nodes.Add "XMIN", tvwChild, , CSng(ShapeNode.selectSingleNode("xmin").Text)
                Attributes.Nodes.Add "BOUNDBOX", tvwChild, "YMIN", "YMIN"
                Attributes.Nodes.Add "YMIN", tvwChild, , CSng(ShapeNode.selectSingleNode("ymin").Text)
                Attributes.Nodes.Add "BOUNDBOX", tvwChild, "XMAX", "XMAX"
                Attributes.Nodes.Add "XMAX", tvwChild, , CSng(ShapeNode.selectSingleNode("xmax").Text)
                Attributes.Nodes.Add "BOUNDBOX", tvwChild, "YMAX", "YMAX"
                Attributes.Nodes.Add "YMAX", tvwChild, , CSng(ShapeNode.selectSingleNode("ymax").Text)
            Else
                Attributes.Nodes.Add "SHAPE", tvwChild, UCase(ShapeNode.nodeName), _
                   UCase(ShapeNode.nodeName)
                Attributes.Nodes.Add UCase(ShapeNode.nodeName), tvwChild, , ShapeNode.Text
            End If
         Next ShapeNode
    Else
        Attributes.Nodes.Add , , UCase(AttributeNode.nodeName), UCase(AttributeNode.nodeName)
        Attributes.Nodes.Add UCase(AttributeNode.nodeName), tvwChild, , AttributeNode.Text
    End If
Next AttributeNode

' Because the XPointer/XLink standards are not yet final and not supported by the MSXML 4.0
' parser, we'll have to link our related table manually

Set CatalogNode = Catalog.selectSingleNode("catalog/relationship[source_featureclass = """ & _
    CurrFCName & """]")
If (Not CatalogNode Is Nothing) Then
    RelatedObjectClass = CatalogNode.selectSingleNode("destination_objectclass").Text
    KeyField = CatalogNode.selectSingleNode("key_field").Text
    Attributes.Nodes.Add , , "RELATED " & UCase(RelatedObjectClass) & " DATA", "RELATED " & _
           UCase(RelatedObjectClass) & " DATA"
    Set RelatedTable = LoadClass(RelatedObjectClass, RelatedObjectClass & ".xml")
    KeyValue = Record.selectSingleNode(LCase(KeyField & "[@born <= """ & RefDate & _
           """ and @retired > """ & RefDate & """]")).Text
    Set RelatedNode = RelatedTable.selectSingleNode("objectclass/object[@id = """ & KeyValue & """]")
    For Each AttributeNode In RelatedNode.selectNodes("*[@born <= """ & RefDate & _
           """ and @retired > """ & RefDate & """]")
        If (UCase(AttributeNode.nodeName) <> UCase(KeyField)) And _
           (UCase(AttributeNode.nodeName) <> "BORN") And _
           (UCase(AttributeNode.nodeName) <> "RETIRED") Then
```

```
                Attributes.Nodes.Add "RELATED " & UCase(RelatedObjectClass) & " DATA", tvwChild, _
                       UCase(AttributeNode.nodeName), UCase(AttributeNode.nodeName)
                Attributes.Nodes.Add UCase(AttributeNode.nodeName), tvwChild, , AttributeNode.Text
            End If
        Next AttributeNode
    End If

    ' Redraw the map and the form

    DrawFeatures
    DrawSelected
    RefreshForm

End Sub

'====================================================================
' Subroutine:   XPathQuery_KeyPress
' Description:  Event handler for the query string text box.  Pressing
'               enter will call select features with the specified
'               query string.
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub XPathQuery_KeyPress(KeyAscii As Integer)
    If (KeyAscii = 13) Then
      SelectFeatures XPathQuery.Text
    End If
End Sub

'====================================================================
' Subroutine:   SelectToleranceSlider_Scroll
' Description:  Responds to select tolerance slide event, sets the
'               selection tolerance and updates the form
' Arguments:    None
' Return Value: None
'====================================================================
Private Sub SelectToleranceSlider_Scroll()
    SelectTolerance = SelectToleranceSlider.Value / 1000
    SelectToleranceLabel.Caption = Format(SelectTolerance, "#0.000")
End Sub

'====================================================================
' Subroutine:   Map_MouseDown
' Description:  Handle the mouse down click event for selection and
'               map extent tools
' Arguments:    1. Button (left or right mouse)
'               2. Shift (true or false)
'               3. x click on map
'               4. y click on map
' Return Value: None
'====================================================================
Private Sub Map_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
    Dim y1 As Single
    Dim i As Integer

    ' Set the coordinates and operate on the selected button.  Also
    ' create a new point out of the coordinates to work with later.

    x1 = TrX(x)
    y1 = TrY(y)
    DisplayPosition x1, y1

    ' Act on the appropriate button tool

    Select Case ButtonTool

        Case "Select"

            ' Search for features at the click point

            If (Button = vbLeftButton) Then
                SelectFeatures "", x1, y1
            Else
                ClearSelection
            End If

        Case "ZoomIn", "ZoomOut"

            ' Handle map extent tools.  For moving points use the
            ' window location rather than the map location for
            ' smore accurate movement tracking

            If Button = vbLeftButton Then
                ButtonTool = ButtonTool & "Active"
                Click(1) = x
                Click(2) = y
            End If
            Exit Sub

    End Select
End Sub

'====================================================================
' Subroutine:   Map_MouseUp
' Description:  Handle mouse up event for selection and map extent
'               tools on the map
' Arguments:    1. Button (left or right mouse)
'               2. Shift (true or false)
'               3. x click on map
'               4. y click on map
' Return Value: None
'====================================================================
Private Sub Map_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
    Dim y1 As Single
    Dim cx As Single
    Dim cy As Single
    Dim dx As Single
    Dim dy As Single

    ' Set the coordinates and operate on the selected button.  Also
    ' create a new point out of the coordinates to work with later.

    x1 = TrX(x)
    y1 = TrY(y)
    DisplayPosition x1, y1

   ' Act on the selected button tool

    Select Case ButtonTool

        Case "ZoomInActive"

            If Button = vbLeftButton Then

                If Abs(Click(1) - x) > Abs(Click(2) - y) Then
                    If (Click(1) < x) Then
                        Bound(1) = TrX(Click(1))
                        Bound(3) = x1
                    Else
                        Bound(1) = x1
                        Bound(3) = TrX(Click(1))
                    End If
                    If (Click(2) > y) Then
                cy = TrY(Click(2) - (Abs(Click(2) - y) / 2))
            Else
                cy = TrY(y - (Abs(y - Click(2)) / 2))
            End If
            ScaleFactor = Abs(Bound(1) - Bound(3)) / Map.ScaleWidth
            Bound(2) = -cy - (Map.ScaleHeight * ScaleFactor / 2)
            Bound(4) = -cy + (Map.ScaleHeight * ScaleFactor / 2)
        Else
            If (Click(2) < y) Then
                Bound(2) = -TrY(Click(2))
                Bound(4) = -y1
            Else
                Bound(2) = -y1
                Bound(4) = -TrY(Click(2))
            End If
            If (Click(1) > x) Then
                cx = TrX(Click(1) - (Abs(Click(1) - x) / 2))
            Else
                cx = TrX(x - (Abs(x - Click(1)) / 2))
            End If
            ScaleFactor = Abs(Bound(2) - Bound(4)) / Map.ScaleHeight
            Bound(1) = cx - (Map.ScaleWidth * ScaleFactor / 2)
            Bound(3) = cx + (Map.ScaleWidth * ScaleFactor / 2)
        End If
        ButtonTool = "ZoomIn"
        DisplayScale
        DrawFeatures
        DrawSelected
        RefreshForm
        Exit Sub

    End If

Case "ZoomOutActive"

    If Button = vbLeftButton Then

        If Abs(Click(1) - x) > Abs(Click(2) - y) Then
            ScaleFactor = ScaleFactor * Map.ScaleWidth / Abs(Click(1) - x)
        Else
        End If
        If (Click(2) > y) Then
            cy = TrY(Click(2) - (Abs(Click(2) - y) / 2))
        Else
            cy = TrY(y - (Abs(y - Click(2)) / 2))
        End If
```

```
                If (Click(1) > x) Then
                    cx = TrX(Click(1) - (Abs(Click(1) - x) / 2))
                Else
                    cx = TrX(y - (Abs(x - Click(1)) / 2))
                End If
                Bound(1) = cx - (Map.ScaleWidth * ScaleFactor / 2)
                Bound(3) = cx + (Map.ScaleWidth * ScaleFactor / 2)
                Bound(2) = -cy - (Map.ScaleHeight * ScaleFactor / 2)
                Bound(4) = -cy + (Map.ScaleHeight * ScaleFactor / 2)
                ButtonTool = "ZoomOut"
                DisplayScale
                DrawFeatures
                DrawSelected
                RefreshForm
                Exit Sub

            End If

        Case "Pan"

            If Button = vbLeftButton Then

                dx = x1 - ((Bound(1) + Bound(3)) / 2)
                dy = y1 + ((Bound(2) + Bound(4)) / 2)

                Bound(1) = Bound(1) + dx
                Bound(2) = Bound(2) - dy
                Bound(3) = Bound(3) + dx
                Bound(4) = Bound(4) - dy
                DisplayScale
                DrawFeatures
                DrawSelected
                RefreshForm

            End If

    End Select

End Sub

'====================================================================
' Subroutine:   Map_MouseMove
' Description:  Handles mouse move click event for map extent tools
' Arguments:    1. Button (right or left mouse button)
'               2. Shift (true or false)
'               3. x click on map
'                4. y click on map
' Return Value: None
'====================================================================
Private Sub Map_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)

    Dim x1 As Single
    Dim y1 As Single

    ' Set the coordinates

    x1 = TrX(x)
    y1 = TrY(y)
    DisplayPosition x1, y1

    ' Operate on the appropriate button tool

    If (ButtonTool = "ZoomInActive") Or (ButtonTool = "ZoomOutActive") Then
        Map.Cls
        Map.ForeColor = RGB(50, 50, 50)
        Map.FillStyle = vbFSTransparent
        Map.Line (Click(1), Click(2))-(x, y), , B
    End If

End Sub

'====================================================================
' Subroutine:   ChangeFeatureClass
' Description:  Loads a new feature class table, resets the record
'               and redraws the screen.
' Arguments:    1. Table of contents index of feature class
' Return Value: None
'====================================================================
Public Sub ChangeFeatureClass(index As Integer)
    CurrFCName = ClassName(index).Caption
    Set Table = LoadClass(CurrFCName, ClassName(index).Tag)
    Set Record = Nothing
    RefreshForm
End Sub

'====================================================================
' Subroutine:   RefreshForm
' Description:  Updates the buttons, mouse pointers and table of
'               contents and refreshes the screen
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub RefreshForm()

    Dim Btn As Button
    Dim i As Integer

    ' Set all buttons unselected

    For Each Btn In Toolbar.Buttons
        Btn.Value = tbrUnpressed
    Next Btn

    ' Process the tools selection

    Select Case ButtonTool
        Case "Select"
            Toolbar.Buttons("Select").Value = tbrPressed
            DisplayMessage "Click to select an feature of the " & _
               "specified feature type; right click to " & "unselect selected features."
        Case "ZoomIn", "ZoomInActive"
            Toolbar.Buttons("ZoomIn").Value = tbrPressed
            DisplayMessage "Click to zoom in around the click rectangle."
        Case "ZoomOut", "ZoomOutActive"
            Toolbar.Buttons("ZoomOut").Value = tbrPressed
            DisplayMessage "Click to zoom out around the click rectangle."
        Case "Pan", "PanActive"
            Toolbar.Buttons("Pan").Value = tbrPressed
            DisplayMessage "Drag the map to pan."
        End Select

    ' Process the current feature class

    For i = 0 To 8
        If (CurrFCName = ClassName(i).Caption) Then
            ClassImage(i).BorderStyle = 1
        Else
            ClassImage(i).BorderStyle = 0
        End If
    Next i

    ' Update the mouse pointer

    Select Case ButtonTool
        Case "Select"
            Map.MousePointer = vbDefault
        Case "ZoomIn", "ZoomInActive"
            Map.MouseIcon = imlPointers.ListImages(1).Picture
            Map.MousePointer = vbCustom
        Case "ZoomOut", "ZoomOutActive"
            Map.MouseIcon = imlPointers.ListImages(2).Picture
            Map.MousePointer = vbCustom
        Case "Pan", "PanActive"
            Map.MouseIcon = imlPointers.ListImages(3).Picture
            Map.MousePointer = vbCustom
    End Select

    ' Refresh the form

    Me.Refresh

End Sub

'====================================================================
' Subroutine:   DisplayMessage
' Description:  Writes a specified message to the status bar
' Arguments:    1. Message string
' Return Value: None
'====================================================================
Public Sub DisplayMessage(Message As String)
    UpdateStatusBar Message, 1
End Sub

'====================================================================
' Subroutine:   DisplayScale
' Description:  Writes the current scale factor to the status bar
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DisplayScale()
    UpdateStatusBar "1:" & CStr(ScaleFactor), 2
End Sub

'====================================================================
' Subroutine:   DisplayPosition
' Description:  Writes the current position to the status bar
' Arguments:    1. x coordinate in map units
'               2. y coordinate in map units
' Return Value: None
'====================================================================
Public Sub DisplayPosition(x As Single, y As Single)
    UpdateStatusBar x & ", " & y, 3
End Sub
```

```
'====================================================================
' Subroutine:   UpdateStatusBar
' Description:  Displays a user specified message in the specified
'               status bar panel (1 - 3)
' Arguments:    1. User specified message
'               2. Status bar panel number
' Return Value: None
'====================================================================
Public Sub UpdateStatusBar(Message As String, Panel As Integer)
    StatusBar.Panels.Item(Panel).Text = Message
End Sub

'====================================================================
' Subroutine:   ShowError
' Description:  Displays a dialog with an error message
' Arguments:    1. Error string
' Return Value: None
'====================================================================
Public Sub ShowError(Error As String)
    MsgBox "Error : " & Error, vbOKOnly + vbExclamation, "Error Message"
End Sub

'====================================================================
' Subroutine:   DrawFeatures
' Description:  Draws features from the XML tables
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DrawFeatures()

    Dim j As Integer
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim fc As DOMDocument40
    Dim s As IXMLDOMNode
    Dim cNode As IXMLDOMNode
    Dim XPathString As String
    Dim Color As Long
    Dim MarkerWidth As Single
    Dim MarkerHeight As Single
    Dim LineWidth As Integer
    Dim Symbol As String

    ' Set the hourglass since drawing takes a little bit of time
' We set both the map and the form since we don't which input came
' from.  The mousepointer of the input handle will remain as
' drawing commences

Map.MousePointer = vbHourglass
Me.MousePointer = vbHourglass

' Store the image in memory for more efficient drawing

Map.AutoRedraw = True

' Check to see if all of the map has been initialized;
' if not then exit

If (ScaleFactor = 0) Then Exit Sub

' Clear the device

Map.Cls

' Loop through each of the feature classes

For j = 8 To 0 Step -1

    If (DrawClass(j).Value = 1) Then

        ' Get the symbology

        If (GetClassType(ClassName(j).Caption) = "point") Then
            Color = GetSymbolColor(ClassName(j).Caption)
            Symbol = GetSymbolType(ClassName(j).Caption)
            If (Symbol = "circle") Then
                MarkerWidth = GetSymbolSize(ClassName(j).Caption)
                MarkerHeight = GetSymbolSize(ClassName(j).Caption)
            Else
                MarkerWidth = GetSymbolWidth(ClassName(j).Caption)
                MarkerHeight = GetSymbolHeight(ClassName(j).Caption)
            End If
        Else
            Color = GetSymbolColor(ClassName(j).Caption)
            LineWidth = GetSymbolSize(ClassName(j).Caption)
        End If

        ' Redraw the features

        XPathString = "featureclass/feature/shape[@born <= """ & RefDate & _
```

```
               """ and @retired > """ & RefDate & """]"
            Set fc = LoadClass(ClassName(j).Caption, ClassName(j).Tag, True)
            If (GetClassType(ClassName(j).Caption) <> "point") Then
                For Each s In fc.selectNodes(XPathString)
                    Set cNode = s.selectSingleNode("coordinate[1]")
                    If (Not cNode Is Nothing) Then
                        x1 = CSng(cNode.selectSingleNode("x").Text)
                        y1 = CSng(cNode.selectSingleNode("y").Text)
                        For Each cNode In s.selectNodes("coordinate[position() > 1]")
                            x2 = CSng(cNode.selectSingleNode("x").Text)
                            y2 = CSng(cNode.selectSingleNode("y").Text)
                            DrawLine ClassName(j).Caption, x1, y1, x2, y2, Color, LineWidth
                            x1 = x2
                            y1 = y2
                        Next cNode
                    End If
                Next s
            Else
                For Each s In fc.selectNodes(XPathString)
                    DrawPoint ClassName(j).Caption, CSng(s.selectSingleNode("coordinate/x").Text), _
                       CSng(s.selectSingleNode("coordinate/y").Text), Color, MarkerWidth, _
                       MarkerHeight, Symbol
                Next s
            End If
        End If

    Next j

    ' BitBlt the Map contents to Map and refresh the screen

    BitBlt Map.hDC, 0, 0, Map.ScaleWidth, Map.ScaleHeight, BufferedMap, 0, 0, &HCC0020
    Map.Refresh
    Map.AutoRedraw = False

    ' Reset the mousepointer for the form

    Me.MousePointer = vbDefault

End Sub

'====================================================================
' Subroutine:   DrawSelected
' Description:  Draws the selected feature
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub DrawSelected()

    Dim cNode As IXMLDOMNode
    Dim sNode As IXMLDOMNode
    Dim XPathString As String
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim Color As Long
    Dim MarkerWidth As Single
    Dim MarkerHeight As Single
    Dim LineWidth As Integer
    Dim Symbol As String

    ' Get the symbology

    If (GetClassType(CurrFCName) = "point") Then
        Color = GetSymbolColor(CurrFCName)
        Symbol = GetSymbolType(CurrFCName)
        If (Symbol = "circle") Then
            MarkerWidth = GetSymbolSize(CurrFCName)
            MarkerHeight = GetSymbolSize(CurrFCName)
        Else
            MarkerWidth = GetSymbolWidth(CurrFCName)
            MarkerHeight = GetSymbolHeight(CurrFCName)
        End If
    Else
        Color = GetSymbolColor(CurrFCName)
        LineWidth = GetSymbolSize(CurrFCName)
    End If

    ' Draw the currently selected (or edited) feature

    If (Not Record Is Nothing) Then
        XPathString = "shape[@born <= """ & RefDate & """ and @retired > """ & RefDate & """]"
        Set sNode = Record.selectSingleNode(XPathString)
        If (GetClassType(CurrFCName) = "line") Then
            Set cNode = sNode.selectSingleNode("coordinate[1]")
            If (Not cNode Is Nothing) Then
                x1 = CSng(cNode.selectSingleNode("x").Text)
                y1 = CSng(cNode.selectSingleNode("y").Text)
                For Each cNode In sNode.selectNodes("coordinate[position() > 1]")
                    x2 = CSng(cNode.selectSingleNode("x").Text)
                    y2 = CSng(cNode.selectSingleNode("y").Text)
                    DrawLine CurrFCName, x1, y1, x2, y2, Color, LineWidth, True
                    x1 = x2
                    y1 = y2
                Next cNode
            End If
        Else
            DrawPoint CurrFCName, CSng(sNode.selectSingleNode("coordinate/x").Text), _
               CSng(sNode.selectSingleNode("coordinate/y").Text), Color, MarkerWidth, MarkerHeight, _
               Symbol, True
        End If
    End If

End Sub

'====================================================================
' Subroutine:   DrawPoint
' Description:  Draws a feature point with the appropriate symbology.
'               Draws a selected feature point in cyan.
' Arguments:    1. Feature class name
'               2. x coordinate of the point
'               3. y coordinate of the point
'               4. Optionally whether the point is currently selected
' Return Value: None
'====================================================================
Public Sub DrawPoint(ClassName As String, x As Single, y As Single, Color As Long, Width As Single, _
Height As Single, Symbol As String, Optional Selected As Boolean = False)

    ' Set the color

    If Selected Then
        BufferedMap.ForeColor = vbYellow
        BufferedMap.FillColor = vbYellow
    Else
        BufferedMap.ForeColor = vbBlack
        BufferedMap.FillColor = Color
    End If

    ' Draw the appropriate symbol

    BufferedMap.FillStyle = vbFSSolid
    If (Symbol = "circle") Then
        BufferedMap.Circle (UTrX(x), UTrY(y)), Width
    Else
        BufferedMap.Line (UTrX(x) - Width, UTrY(y) - Height)-(UTrX(x) + Width, UTrY(y) + Height), , B
    End If

End Sub
```

```
'====================================================================
' Subroutine:   DrawLine
' Description:  Draws a feature line segment with the appropriate
'               symbology. Draws a selected feature line in cyan.
' Arguments:    1. Feature class name
'               2. first x coordinate of the line segment
'               3. first y coordinate of the line segment
'               4. last x coordinate of the line segment
'               5. last y coordinate of the line segment
'               6. Optionally whether the feature that the line
'                  segment is part of is currently selected
' Return Value: None
'====================================================================
Public Sub DrawLine(ClassName As String, x1 As Single, y1 As Single, x2 As Single, y2 As Single, _
Color As Long, Width As Integer, Optional Selected As Boolean = False)

    ' Set the color

    If Selected Then
        BufferedMap.ForeColor = vbYellow
        BufferedMap.FillColor = vbYellow
    Else
        BufferedMap.ForeColor = Color
        BufferedMap.FillColor = BufferedMap.ForeColor
    End If

    ' Draw the appropriate symbol

    BufferedMap.FillStyle = vbFSSolid
    BufferedMap.DrawWidth = Width
    BufferedMap.Line (UTrX(x1), UTrY(y1))-(UTrX(x2), UTrY(y2))
    BufferedMap.DrawWidth = 1

End Sub

'====================================================================
' Subroutine:   TrX
' Description:  Converts an x coordinate from twips to map units
' Arguments:    1. Twip x coordinate
' Return Value: Map units x value
'====================================================================
Public Function TrX(x As Single) As Single
    TrX = ((x / Map.ScaleWidth) * (Bound(3) - Bound(1))) + Bound(1)
End Function
```

```
'====================================================================
' Subroutine:   TrY
' Description:  Converts a y coordinate from twips to map units
' Arguments:    1. Twip y coordinate
' Return Value: Map units y value
'====================================================================
Public Function TrY(y As Single) As Single
    TrY = -(((y / Map.ScaleHeight) * (Bound(4) - Bound(2))) + Bound(2))
End Function

'====================================================================
' Subroutine:   UTrX
' Description:  Converts an x coordinate from map units to twips
' Arguments:    1. Map units x coordinate
' Return Value: Twip x value
'====================================================================
Public Function UTrX(x As Single) As Single
    UTrX = ((x - Bound(1)) / (Bound(3) - Bound(1))) * (Map.ScaleWidth)
End Function

'====================================================================
' Subroutine:   UTrY
' Description:  Converts a y coordinate from map units to twips
' Arguments:    1. Map units y coordinate
' Return Value: Twip y value
'====================================================================
Public Function UTrY(y As Single) As Single
    UTrY = ((-y - Bound(2)) / (Bound(4) - Bound(2))) * (Map.ScaleHeight)
End Function

'====================================================================
' Subroutine:   LoadCatalog
' Description:  Loads the catalog XML into memory for fast access.
' Arguments:    None
' Return Value: None
'====================================================================
Public Sub LoadCatalog()
    Set Catalog = New DOMDocument40
    Catalog.validateOnParse = True
    Catalog.Load "data\Catalog.xml"
    Catalog.setProperty "SelectionLanguage", "XPath"
    If (Catalog.parseError.errorCode <> 0) Then
        ShowError Catalog.parseError.reason
        Unload Me
    End If
End Sub
```

```
'====================================================================
' Subroutine:   GetBounds
' Description:  Loads the boundaries of the map into memory
' Arguments:    1. x minimum reference (set by subroutine)
'               2. y minimum reference (set by subroutine)
'               3. x maximum reference (set by subroutine)
'               4. y maximum reference (set by subroutine)
' Return Value: None (reference values set)
'====================================================================
Public Sub GetBounds(ByRef xmin As Single, ByRef ymin As Single, ByRef xmax As Single, _
ByRef ymax As Single)
    xmin = CSng(Catalog.selectSingleNode("catalog/boundbox/xmin").Text)
    ymin = CSng(Catalog.selectSingleNode("catalog/boundbox/ymin").Text)
    xmax = CSng(Catalog.selectSingleNode("catalog/boundbox/xmax").Text)
    ymax = CSng(Catalog.selectSingleNode("catalog/boundbox/ymax").Text)
End Sub

'====================================================================
' Subroutine:   GetClassType
' Description:  Returns the feature class type (point or line)
' Arguments:    1. Class name
' Return Value: Feature class type
'====================================================================
Public Function GetClassType(ClassName As String) As String
    Dim XPathString As String
    XPathString = "catalog/featureclass[name = """ & ClassName & """]/type"
    GetClassType = Catalog.selectSingleNode(XPathString).Text
End Function

'====================================================================
' Subroutine:   GetSymbolColor
' Description:  Returns the symbol color associate with the specified
'               feature class (Long from RGB())
' Arguments:    1. Class name
' Return Value: Symbol color
'====================================================================
Public Function GetSymbolColor(ClassName As String) As Long
    Dim XPathString As String
    XPathString = "catalog/featureclass[name=""" & ClassName & """]"
    XPathString = XPathString & "/symbology/color"
    GetSymbolColor = RGB(Catalog.selectSingleNode(XPathString & "/red").Text, _
        Catalog.selectSingleNode(XPathString & "/green").Text, _
        Catalog.selectSingleNode(XPathString & "/blue").Text)
End Function
```

```
'====================================================================
' Subroutine:   GetSymbolType
' Description:  Returns the symbol type associate with the specified
'               feature class (circle, rectangle, etc.)
' Arguments:    1. Class name
' Return Value: Symbol type
'====================================================================
Public Function GetSymbolType(ClassName As String) As String
    Dim XPathString As String
    XPathString = "catalog/featureclass[name=""" & ClassName & """]"
    XPathString = XPathString & "/symbology/symbol"
    GetSymbolType = Catalog.selectSingleNode(XPathString).Text
End Function

'====================================================================
' Subroutine:   GetSymbolSize
' Description:  Returns the symbol size associate with the specified
'               feature class (for circular markers)
' Arguments:    1. Class name
' Return Value: Symbol size
'====================================================================
Public Function GetSymbolSize(ClassName As String) As Single
    Dim XPathString As String
    XPathString = "catalog/featureclass[name=""" & ClassName & """]"
    XPathString = XPathString & "/symbology/size"
    GetSymbolSize = CSng(Catalog.selectSingleNode(XPathString).Text)
End Function

'====================================================================
' Subroutine:   GetSymbolHeight
' Description:  Returns the symbol height associate with the specified
'               feature class (for rectangular markers)
' Arguments:    1. Class name
' Return Value: Symbol height
'====================================================================
Public Function GetSymbolHeight(ClassName As String) As Single
    Dim XPathString As String
    XPathString = "catalog/featureclass[name=""" & ClassName & """]"
    XPathString = XPathString & "/symbology/size/height"
    GetSymbolHeight = CSng(Catalog.selectSingleNode(XPathString).Text)
End Function

'====================================================================
' Subroutine:   GetSymbolWidth
' Description:  Returns the symbol width associate with the specified
'               feature class (for rectangular markers)
' Arguments:    1. Class name
' Return Value: Symbol width
'====================================================================
Public Function GetSymbolWidth(ClassName As String) As Single
    Dim XPathString As String
    XPathString = "catalog/featureclass[name=""" & ClassName & """]"
    XPathString = XPathString & "/symbology/size/width"
    GetSymbolWidth = CSng(Catalog.selectSingleNode(XPathString).Text)
End Function

'====================================================================
' Subroutine:   LoadClass
' Description:  Loads the XML table for a feature or object class
' Arguments:    1. Feature class name
'               2. Optional parameter to suppress validation (for
'                  speeding up the drawing process
' Return Value: XML DOM Document for the XML table
'====================================================================
Public Function LoadClass(ClassName As String, TableName As String, _
Optional NoValidate As Boolean = False) As DOMDocument40
    Set LoadClass = New DOMDocument40
    LoadClass.validateOnParse = Not (NoValidate)
    LoadClass.Load CStr("data\" & TableName)
    LoadClass.setProperty "SelectionLanguage", "XPath"
    If (LoadClass.parseError.errorCode <> 0) Then
        ShowError LoadClass.parseError.reason & " for featureclass " & ClassName & "."
        Unload Me
    End If
End Function

'====================================================================
' Subroutine:   CurrentFeatureID
' Description:  Returns the feature ID of the currently selected
'               feature.
' Arguments:    None
' Return Value: Feature ID
'====================================================================
Public Function CurrentFeatureID() As Integer
    If (Record Is Nothing) Then
        CurrentFeatureID = 0
    Else
        CurrentFeatureID = CInt(Record.Attributes.getNamedItem("id").Text)
    End If
End Function

'====================================================================
' Subroutine:   SelectFeatures
' Description:  Selects features based on either an XPath query or
'               on a selected location, filtered by temporal
'               existence.  Adds a list of selected feature IDs to
'               the SelectedFeatures list box.
' Arguments:    1. Query string or "" if spatial select
'               2. Optional x coordinate of location
'               3. Optional y coordinate of location
' Return Value: None
'====================================================================
Public Sub SelectFeatures(QueryString As String, Optional x As Variant, Optional y As Variant)

    Dim tRecord As IXMLDOMNode
    Dim CoordinateNode As IXMLDOMNode
    Dim x1 As Single
    Dim y1 As Single
    Dim x2 As Single
    Dim y2 As Single
    Dim l As Single
    Dim r As Single
    Dim d(2) As Single

    ' Clear the selected feature IDs list

    SelectedFeatures.Clear

    ' If no x and y coordinates are specified, prompt the user
    ' for an XPath query expression, and check the date the
    ' feature existed.

    If (IsMissing(x) And IsMissing(y)) Then
        If (QueryString <> "") Then
            QueryString = "featureclass/feature[" & QueryString
            QueryString = QueryString & " and shape/@born <= """ & RefDate & """ and shape/@retired > """ _
               & RefDate & """]"
            For Each tRecord In Table.selectNodes(QueryString)
                SelectedFeatures.AddItem tRecord.Attributes.getNamedItem("id").Text
            Next tRecord
        Else
            ShowError "No selection criteria specified."
        End If

    ' Otherwise do a spatial search by checking within the search
    ' tolerance of the point clicked.  Loop through each coordinate
    ' and check to be sure it exists at that point in time.
```

```
Else

    If (GetClassType(CurrFCName) = "line") Then
        For Each tRecord In Table.selectNodes("featureclass/feature[shape/@born <= """ & _
         RefDate & """ and shape/@retired > """ & RefDate & """]")
            Set CoordinateNode = tRecord.selectSingleNode("shape/coordinate[1]")
            x1 = CSng(CoordinateNode.selectSingleNode("x").Text)
            y1 = CSng(CoordinateNode.selectSingleNode("y").Text)
            If (x >= (x1 - SelectTolerance) And (x <= (x1 + SelectTolerance))) And _
                   (y >= (y1 - SelectTolerance) And (y <= (y1 + SelectTolerance))) Then
                SelectedFeatures.AddItem tRecord.Attributes.getNamedItem("id").Text
            Else
                For Each CoordinateNode In tRecord.selectNodes("shape/coordinate[position() > 1]")
                    x2 = CSng(CoordinateNode.selectSingleNode("x").Text)
                    y2 = CSng(CoordinateNode.selectSingleNode("y").Text)
                    If (x >= (x2 - SelectTolerance) And (x <= (x2 + SelectTolerance))) And _
                    (y >= (y2 - SelectTolerance) And (y <= (y2 + SelectTolerance))) Then
                        SelectedFeatures.AddItem tRecord.Attributes.getNamedItem("id").Text
                        Exit For
                    End If
                    l = Sqr((x2 - x1) ^ 2 + (y2 - y1) ^ 2)
                    If (l <> 0) Then
                      r = ((y1 - y) * (y1 - y2) - (x1 - x) * (x2 - x1)) / l ^ 2
                      If (r >= 0) And (r <= 1) Then
                        d(1) = x1 + r * (x2 - x1)
                        d(2) = y1 + r * (y2 - y1)
                        If (x >= (d(1) - SelectTolerance) And (x <= (d(1) + SelectTolerance))) And _
                        (y >= (d(2) - SelectTolerance) And (y <= (d(2) + SelectTolerance))) Then
                            SelectedFeatures.AddItem tRecord.Attributes.getNamedItem("id").Text
                            Exit For
                        End If
                      End If
                    End If
                    x1 = x2
                    y1 = y2
                Next CoordinateNode
            End If
        Next tRecord
    Else
        For Each tRecord In Table.selectNodes("featureclass/feature[shape/@born <= """ & _
         RefDate & """ and shape/@retired > """ & RefDate & """]")
            x1 = CSng(tRecord.selectSingleNode("shape/coordinate/x").Text)
            y1 = CSng(tRecord.selectSingleNode("shape/coordinate/y").Text)
            If (x >= (x1 - SelectTolerance)) And (x <= (x1 + SelectTolerance)) And _
           (y >= (y1 - SelectTolerance) And (y <= (y1 + SelectTolerance))) Then
                SelectedFeatures.AddItem tRecord.Attributes.getNamedItem("id").Text
                End If
            Next tRecord
        End If

    End If

    ' Set the selected feature to be the first item in the list

    SelectedFeatures.Text = SelectedFeatures.List(0)

End Sub

'=====================================================================
' Subroutine:   ClearSelection
' Description:  Clears the selection and resets the interface
' Arguments:    None
' Return Value: None
'=====================================================================
Public Sub ClearSelection()
    Set Record = Nothing
    SelectedFeatures.Clear
    Attributes.Nodes.Clear
    DrawFeatures
    RefreshForm
End Sub

'=====================================================================
' Subroutine:   RefDate
' Description:  Formats the reference date as yyyymmdd because XPath
'               1.0 does not support date comparisons yet.
' Arguments:    None
' Return Value: Formatted Date
'=====================================================================
Public Function RefDate() As String
    RefDate = Format(ReferenceDate.Value, "yyyymmdd")
End Function
```