WHEAT Software Development Toolkit

Gregory W Pouch
Geohydrology Section
Kansas Geological Survey

Abstract

The Windows-based Hydrogeologic Exploration and Appraisal Toolkit (WHEAT) is a set of user-friendly programs developed at the KGS for the retrieval, analysis, manipulation and display of information on natural resources, especially groundwater. The principle goals in development of WHEAT have been: ease of use; hardware independence; applicability to resource management problems; end-user extendibility; and the ability to exchange information with other analysis packages. This document meets the goal of end-user extendibility and strengthens WHEAT's ability to exchange information by explaining how WHEAT programs store data in databases and providing source code for retrieving and manipulating data.

This document provides a brief overview of EMAPKGS2, provides code for reading and write WHEAT XYChains, explains the SQL statements used by WHEAT, and provides some advice on database design.

This document is intended for scientist/engineer-programmers who wish to write extension programs to use with WHEAT databases, such as a hypothetical program that would convert survey notes to WHEAT-format XYChains.

This manual assumes a working knowledge of structures and relational databases, database programming using Visual Basic under Microsoft Windows.

Kansas Geological Survey
Open-file Report

Disclaimer

The Kansas Geological Survey does not guarantee this document to be free from errors or inaccuracies and disclaims any responsibility or liability for interpretations based on data used in the production of this document or decisions based thereon. This report is intended to make results of research available at the earliest possible date, but is not intended to constitute final or formal publication.

Disclaimer

The computer program/processing software described in this document is the work of the Kansas Geological Survey, University of Kansas. The Kansas Geological Survey, University of Kansas, makes no warranty or representation, either express or implied, with respect to the documentation or interpretations or decisions based on data processed using this software, including their quality, performance, merchantability, or fitness for a particular purpose. In no event will the Kansas Geological Survey, University of Kansas, be liable for direct, indirect, special, incidental, punitive, or consequential damages arising out of the use of or inability to use the software or documentation whether based upon contract, negligence, strict liability or otherwise.

The Kansas Geological Survey, University of Kansas, does not warrant that this software will meet your requirements or that it will operate in an error free manner.

If you use this software, you are accepting the terms of this disclaimer (i.e., use at your own risk.). If you do not or cannot agree to these terms, return the software immediately.

Under no circumstances should you re-distribute this software apart from this disclaimer.

Table of Contents

Table of Code Listings

List of Tables

Introduction

WHEAT is a set of user-friendly programs for electronic mapping. Its centerpiece is EMAPKGS2, which is a computer-based mapping program that allows the user to design, view, and print maps, as well as perform spatial queries. WHEAT includes programs for data import/export, contouring of point data, and legend design. WHEAT was developed at the KGS for the retrieval, analysis, manipulation and display of information on natural resources, especially groundwater. It was designed to run under the Microsoft Windows 3.1 operating system (WHEAT has been used under Windows for Workgroups 3.11, Windows 95, and Windows NT 3.51.) and is intended to be easy to use. The principle goals in development of WHEAT have been: ease of use; hardware independence; applicability to resource management problems; end-user extendibility; and the ability to exchange information with other analysis packages.

No software package satisfies every need that will eventually arise. An analyst often needs direct access to coordinates or other attributes to process the data in some unforeseen way and will thus want to extend the WHEAT system. WHEAT solves this sort of problem in two ways. First, the data are stored in a few simple formats in a database that the user can access with programs. (WHEAT stores data in Microsoft Access format databases.) Second, WHEAT includes a number of programs for import, export, and data manipulation, so that the user can produce plain text files containing the coordinates and process them in other programming languages, or even on other machines.

This document explains the data structures used by WHEAT, explains how to manipulate them, and includes source-code for some of the more useful programs. It assumes familiarity with Microsoft Access, particularly table and query design, and familiarity with either Visual Basic or Access Basic, and the database features of those languages.

Advice on Choosing a Computer

Although the target machine for WHEAT programs should be a small, low cost computer, this does not mean that programming of WHEAT extensions should be done on slow machines. The programming environment needed to develop graphical user interface programs with database features puts a load on the computer far beyond what the program alone will. Developers need fast computers with lots of RAM, and lots of disk space. You should always test programs on slow machines to see what they do under low-memory condition.

Advice on Choosing Software Development Packages

The following does not constitute an official endorsement of any products by the Kansas Geological Survey, the University of Kansas, or the State of Kansas, but merely represents the author's personal experience with some software development packages.

First, I offer these general pieces of advice. READ THE DIRECTIONS. If a software development package does not do what you need, get another one: wasting your time trying to work with the wrong tools does not save you or your employer money.

For serious work, the user-interface should be written in Microsoft Visual Basic; get the Professional Edition if at all possible. Microsoft Access includes a subset of Visual Basic and the ability to write forms for a user-interface, but Access forms are harder to program than Visual Basic forms. Unless you already are proficient in C++, do not be lured into it: it is very easy to spend huge amounts of time trying to learn C++ and still not accomplish anything. I have never used Delphi, and so have no opinion on this.

For heavy number crunching, use Fortran. I like Microsoft Fortran Power Station 1.0. MS Fortran 5.1, which is a 16-bit compiler, has been used, but has a rather cumbersome interface. I have heard good things about MS Fortran Power Station 4.0, but have not yet tried it.

Database design should be done in Microsoft Access. Data imports should be done in Microsoft Access. At each installation that uses WHEAT, such as a department or workgroup, there should be at least one copy of MSAccess for maintenance, data import, and database design. The Access Development Kit does not seem to provide much functionality that you would not already have in Visual Basic.

WHEAT fonts were designed in CorelDraw 3.0. I like it as an illustration package.

Acknowledgments

The author wishes to thank all those who contributed support to the development of WHEAT, especially the staff of the Equus Beds Groundwater Management District Number 2. For useful advice, suggestions, and encouragement, I thank my co-workers at the Kansas Geological Survey, particularly Martin Smith, Steve Yoder, Rich Sleazer, and Marios Sophocleous, and the staffs of GMDs 4 and 5. I particularly wish also to thank my fiancée, Becky Roesner, for putting up with my obsessive programming.

Overview of WHEAT EMAPKGS2

WHEAT is centered around mapping, so a brief overview of EMAPKGS2 , the electronic mapping tool included in WHEAT, is in order before beginning. All WHEAT programs use a database for storing map data. EMAPKGS2 allows the analyst to

interactively construct maps from data stored in a relational database, perform point-oriented spatial queries such as locating all wells within a specified radius, perform geographic overlays, and export data to other programs for further analysis.

Maps in EMAPKGS2 consist of a series of overlaid themes, which are loaded into memory in response to user actions. A theme is a set of geographic features retrieved from a table or view in the database: all features in a theme must be of the same geometric type–point, line, polygon, tile, or text label; for example, the user could have a theme for rivers (lines), for lakes (polygons), or for wells (points). Theme definitions can be designed interactively, saved to disk, and loaded in later sessions. Once the themes for a map are loaded into memory, the user interactively repositions the viewing area to a desired location in a variety of ways, including scroll bars, menu items, and dialog boxes.

From the user's perspective, a theme is a set of geographic features with similar properties, plotted in a "particular way". (For example, a theme might be Streams or Wells.) The "particular way" refers to the plotting styles of each feature, which can come from fields in the source table or from user-selected constants.

From the program's perspective, a theme on disk is a set of information for constructing a theme in memory. A theme in memory is a set of geographic features and their plot styles, along with some information such as the source of the data and instructions on handling spatial queries. On disk, a theme is a record in the table zWHEATtblThemes: a theme has a geometric type (ThemeType), a user-assigned name (ThemeName), a program-assigned serial number (ThemeSerialNumber), and two SQL statements, one for the map (MapSource) and one for spatial queries (DBSource). The MapSource SQL and the DBSource SQL statements need not retrieve data from the same table, although they should have a common field (zqFeatureID) used to link the two. A later section explains the fields in MapSource and DBSource SQL statements in more detail. (EMAPKGS2 allows the user to design and load theme definitions. THEMEEDIT is another program for editing WHEAT-useable SQL statements. It is basically all of the theme dialogs in a stand-alone program that allows easy editing of the ThemeDefinitions already stored on disk.)

This manual does not include a detailed description of EMAPKGS2, as the program is currently over a megabyte of source code. However, a brief overview of its internal workings might be helpful. There are three main tasks in EMAPKGS2: loading data, displaying data, and running spatial queries.

On disk, map data is stored in tables. In memory, these data are stored in User-Defined Types (UDTs in Visual Basic, also known as structures or records). There is one, huge array with all the geographic points in a Windows global memory block. For each geometric type (such as lines), there is an array of UDTs that contains plotting style, Serial Number, and indexes into global memory block of the first and last point for each feature. Themes are stored in array of UDTs, which contains the name, type, SQL statements, and indexes of the first and last elements of the theme (the first line and last line, for instance). There is also an array of Theme index numbers stored at the form level that controls plot order and theme visibility.

When EMAPKGS2 displays a theme – on screen, to a printer, or to a metafile – it sweeps through the array of themes to display. For each theme, it sweeps through the array of features. For each feature, it determines whether it is even partially in the user-

controlled ViewWindow. For points, textlabels, and tiles, it simply displays them. For lines and polygons, it first clips geographic features to the ViewWindow using a modified Cohen-Sutherland clipping algorithm, and then displays them using Windows API calls.

For spatial queries, the user selects a theme, a search type, and some optional search parameters. When the user double-clicks on the maps, EMAPKGS2 uses the coordinates of the mouse to perform a spatial query, mainly by using subroutines and functions in the WhtGeom*N*.DLL dynamic link library.

General Description of WHEAT Data Storage

WHEAT uses data stored in a Microsoft Access format relational database for both the attributes and coordinates of various geographic features. The Microsoft Access database format is also the native database format for Visual Basic 3.0 and 4.0.

WHEAT uses a relational database to store information about geographic features, both descriptive (or tabular) attributes and locational attributes. The location(s) of features are described as X (east-west) and Y (north-south). WHEAT does not use any particular coordinate system, except that X and Y will be plotted as equal distances on the screen and on printing, and X increases to the right and Y increases to the top. (WHEAT uses a right-handed, isotropic coordinate system. There might be information about the database's coordinate system in the table ztblWHEAT_LOG, in the UNITS entry.)

The attributes and locational information can all be stored in one table for each set of features, or they can be stored in separate tables and related in a query. Additionally, plotting instructions (like color, line width, and symbol) can be stored in a table or defined as constants.

Feature Attributes

Attributes of geographic features that are not locational are simply stored in regular fields and should be accessed using standard Access/Visual Basic methods.

Locational Data

There are two general categories of features on a map: those defined by a single coordinate pair, such as the location of a house or a weather station, and those defined by a series of ordered coordinate pairs, such as a road or a lake. For retrieving single point locations, WHEAT uses two fields, one for X and one for Y. For retrieving multi-point features, WHEAT uses one field containing the locations in what is hereafter called XYChain format, which is simply an ordered array of (X, Y) locations stored in the database as a binary large object (or BLOB). This section explains how to store and retrieve location data using Microsoft Visual Basic 3.0. Virtually identical techniques can be used from Microsoft Access and Excel, and similar techniques could be used from Fortran, C, and other languages. Code for this section appears in Appendix 1.

Internally, WHEAT stores point locations in a structure referred to as a RealPoint, which is simply X and Y as two four-byte floating point numbers (in Basic, singles; in Fortran, REAL*4's, in C, short floats). (See declaration of Type RealPoint in Listing 1.) An XYChain is an array of RealPoints (or ordered set of point locations), and a RealRect is two RealPoints in the order upper left, lower-right. All geographic features in WHEAT are stored as some combination of RealPoint, RealRect, or XYChain.

Point Locations

For point locations, used for point symbols, text labels, and, as a pair, for tiles, WHEAT simply gets the values from two separate fields in the table. If a WHEAT program seems unable to find the field, make sure they start with X_ and Y_

XYChains

Unfortunately, MSAccess and other relational database systems usually do not provide an easy way to store arrays. Although it could be argued that an XYChain could be stored in some other table as (X, Y, Order, ParentSerialNumber), this is slow, error prone, and doubles the amount of data to be stored, all to no obvious benefit. To avoid these problems, WHEAT hides the array of RealPoints in a string which is stored in the database in a Binary Large Object (BLOB) field, also known as an OLE Object field.

WHEAT programs convert between RealPoints and XYBLOBs in one of two ways. The safe method consists of getting each 8-byte (=1 RealPoint) chunk of the string, storing that 8-byte string in a structure, and copying the structure containing the "string" RealPoint into an actual RealPoint structure using Visual Basic's LSET operator, which copies one structure into another without type checking or conversions. The fast method uses the Windows API to perform a direct memory copy: only the safe method is described here.

Listing 2 shows the RealPoint-to-BLOB and the BLOB-to-RealPoint conversions using the safe method. The declarations in Listing 1 and the routines in Listing 2 are all that need to be included to gain direct access to WHEAT XYChains to program a line/polygon processing scheme.

There is an additional user-defined-type used in these routines called a String8, that is essentially a fixed width string . It contains one member, a fixed-width, eight-byte string. String assignments are to/from this member. LSET assignment are to/from variables of type String8 or of type RealPoint.

WHEAT Data Model

In geographic information system (GIS) nomenclature, a data model describes how geographic are represented in the geographic database. For example, some geographic information systems (GISs), such as Arc/Info from ESRI, organize geographic data according to the arc-node data model, which stores points; directed, jointed lines (arcs); end-points which are connected to arcs (nodes); and complex topological information to

assemble derived features such as polygons, which in the arc-node data model is a region enclosed by a set of arcs traversed in a particular way and containing a certain label point.

WHEAT does not use arc-node topology, nor anything else the user would likely recognize as a data model. A point is represented as a pair of coordinates (X,Y), a linear feature such as a road is represented by an array of points (an XYChain) on the linear feature, a polygon is represented as an array of points (an XYChain) on the perimeter of the polygon, and a tile is represented by two coordinate pairs. For lack of a better term, this will be called chain topology. "Chain topology" is the same "data model" used by most humans to describe geometric features and by most computer graphics packages, such as the Windows API, to represent geometric features.

Users are generally unaware that WHEAT has a data model, and it is the author's belief that this situation is desirable. As with jokes, if you have to explain a data model, it's not going over well.

Advice on Database Design

There are several approaches to designing a database, and the differences often reflect personal style more than rules that must be followed. The following should be taken as editorial opinion based on experience designing and using databases at the Kansas Geological Survey.

Because MSAccess allows easy use of attached tables, there is no need to keep all the data in a single database. It is safer to have the data in topical databases, such as transportation and hydrology, and have a central database that attaches to tables in all the subsidiary. There are two immediate advantages to this: the databases are smaller, and more likely to be backed up regularly, and a catastrophic failure of one database will not ruin the entire set. (In MSAccess 2.0, all long fields (Memo and LongBinary) are stored in one huge table, hidden from the user. Sometimes this gets damaged, and data is lost. The smaller the database this happens to, the smaller the effort to restore the database.)

When naming fields in databases, use full names liberally, and use the caption and description properties whenever the name is not self-explanatory. If the data is somehow encoded, create a decoding table and keep it in the database. Do not rely on your memory for these things.

Any Access-allowed field name can be used by WHEAT, except that field names should not begin with ZQ. To preserve portability in case you should decide to move to another database management system, it is best to limit field names to letters and numbers and use no punctuation other than underscores (_). In particular, avoid commas, periods, parentheses, brackets, spaces, and anything that might look like a mathematical function or operator. The word "date" is reserved as an Access keyword, but "date_" works fine. Avoiding spaces in field names, and using capitalization to indicate word breaks is useful as well (e.g., name a field AppropriationDate rather than Appropriation Date).

You do not pay money for the length of names, so do not try saving on characters. Use full, descriptive names whenever possible. Do not give fields names like Name and SerialNumber, give fields names like HydrologyLineSerialNumber and CityName. Otherwise, you will eventually try joining two tables with fields named SerialNumber that

do not join on that field, and MSAccess will try joining them automatically. If you plan on exporting the data to another database, it may be safest if the lengths of field names are kept below 16 characters or the first 16 characters are unique.

Whenever there is some standard numbering or naming convention available, use it. For example, use the Federal FIPS codes for county serial numbers in a counties table, rather than some arbitrary integer. If there is a broader standard available, use that one: e.g., use the Federal FIPS county codes rather than the Kansas county numbers.

Do not get clever with serial numbers. They should be used ONLY for providing a unique ID, not encoding the location in latitude-longitude, the owner's name, the county, or what you had for dinner last night. If there is not already some standard numbering system for the objects at hand, begin at one, proceed to two and so on. Serial numbers are numbers, not strings. They should not contain letters, or this will dramatically slow down the database. User either Long Integers (4-byte integers) or Counters for serial numbers; short integers tend to cause all sorts of problems.

Every table should have a primary key, preferably single-column.

It is better to have many easy-to-read fields than a few fields that require a program to decipher. Each field should contain only one type of information.

WHEAT and Unique ID Numbers

Once you have displayed a theme with EMAPKGS2, you may want to perform spatial queries, such as finding the river closest to a given point, or assemble a list of all points within a five-mile radius. With WHEAT EMAPKGS2, you do this by running a spatial query.

Unique ID Numbers Needed for Spatial Querying

When EMAPKGS2 loads data on a feature into a theme, it stores a feature name. When EMAPKGS2 performs a spatial query, it gets the feature IDNumber from the theme, then runs a query looking for all data in records where that feature name occurs and displays the results in the datagrid and spatial query results window. Because of this, it is important that tables contain a column of data suitable for performing this link. This should be one column with unique values, such as a serial number. THE FEATURE ID NUMBER SHOULD BE UNIQUE or you will end up returning incorrect results. A one-field primary key is best.

One-To-Many Spatial Queries

When the user saves a theme definition, WHEAT writes the SQL statement that defines the graphical data and the SQL statement that defines the tabular data to be retrieved by a spatial query in fields MapSource and DBSource in table zWHEATtblThemes. You can modify these definitions to suit your needs. For instance, if we had county outlines that we wished to display from a table called Counties, but census data in a table called CountiesCensus which contains census data for each decade

for each county (a one-to-many relate between counties and census data), and we want to be able to select a county and retrieve census data, we could add a new record to the zWHEATtblThemes table, set the MapSource field to

```
SELECT DISTINCTROW [County line] AS zqPolygonOutline, [PlotColor] AS zqFillColor, 0 AS zqFillStyle, 0 AS
  zqOutlineColor, 2 AS zqOutlineWidth, format([CountyName]) AS zqFeatureID  FROM [Counties]  ORDER BY [PlotColor];
```

and set the DBSource field to

```
SELECT DISTINCTROW *, format([CountyName]) AS zqFeatureID FROM [CountiesCensus]  ORDER BY [CensusYear];
```

and the ThemeName field to "Counties with Census Data", then loading that theme will allow us to perform the desired one-to-many spatial query. This technique is useful whenever many observations might be associated with a single geographic feature, such as water levels with a well or soil layers with a soil series polygon.

Field Naming Strategies and Automatic Field Selection

When using EMAPKGS2 or THEMEEDIT, the program guesses at what fields might contain coordinates or plotting instructions based on the field name and the search strings contained in the [DefaultFields] section of the WHEAT.INI file. By carefully choosing field names, EMAPKGS2 can be very intuitive for the user.

Each of the keywords in the DefaultFields section ends with LookFor or ShowAlways. Not surprisingly, the LookFor entries contain strings that will be searched for in the list of fields and ShowAlways entries contain constants that will be included below the fields in the list. The constants and strings are separated by spaces, and each line should have one space after the last word.

When a user chooses a new table, the program finds the names of all fields in that table. For each choice the user has to make, the program compares the list of LookFor sub-strings to the list of available fields, and, if a match is found, adds the field's name to the appropriate list. It then adds all the constants to the lists as well. Finally, it chooses the first item in the list.

By having X_ as the first entry on the XLookFor line and naming all fields containing X coordinates X_Center or X_Well or even just X_, the program will correctly choose those fields as the default X coordinate where an X coordinate is called for. If the field name does not appear in the proper lists, the user can still drag the field name to the list.

Listing 3 shows the automatic field selection routines used in EMAPKGS2 and THEMEEDIT. Listing 4 shows the default values for the ___ShowAlways and ___LookFor routines.

Import Programs

WHEAT includes few programs for the import of attribute files. The import functions of MSAccess and Excel are far superior to anything I could write. If Access cannot import some file, it is usually best to import it into Excel, label the fields/columns in the first row, save it, then use the MSAccess import from Excel option.

WHEAT does include programs for importing line/polygon data, because these must be converted to WHEAT's XYChain format. For a simple example of conversion between

XYChains and ASCII text, the user is directed to the source code for BLOBMEMO.EXE. A complicated example is the ARCS_IN4.EXE program: most of the complication is due to reading the Arc/Info generate format.

A better example of an import-export program can be found in the source code for GENIMPEX.EXE (or Generic Import Export). GENIMPEX was written for three purposes: as an example import-export program for this report; to provide a fast, easy way to export or import WHEAT XYChains in MSAccess databases from or to its own .WXY format, so that other programs can process this data; and as an intermediate format for translation programs (for example, the user could write a program to translate DXF files into WXY format, then use GenImpEx to move those files into WHEAT.).

Putting data into WXY format is relatively easy and is possible with almost any compiler. A WXY file is a pure binary format: there are no record delimiters or record length indicators. The "records" are each 8 bytes long. The first record in the file must be the word WHEATCHN. All subsequent records are used for storing XYChain data. For each XYChain, there is one header record containing two 4-byte integers, one for a serial number and one for the number of coordinate pairs. There are then nPairs records, each containing two 4-byte floating point numbers (REAL*4s, singles) giving the points coordinates. The routines WXY_Read and WXY_Write can be used for XY files. There is also an example Fortran program that reads WXY files, called TESTWXY.FOR. (TESTWXY.FOR is a Microsoft Fortran program: you may need to modify the open statements to get a byte-stream file in other compilers.)

Theme Map SQL Statements

A theme definition is a set of information for constructing a theme in memory. A theme in memory is a set of geographic features and their plot styles, along with some information such as the source of the data and instructions on handling spatial queries, stored in an array of UDTs. Each element in each array of UDTs corresponds to a record in a table, and a theme definition gives the correspondence.

A theme definition on disk is a record in the table zWHEATtblThemes: a theme definition has a geometric type (ThemeType), a user-assigned name (ThemeName), a program-assigned serial number (ThemeSerialNumber), and two SQL statements, one for the map (MapSource) and one for spatial queries (DBSource). The MapSource SQL and the DBSource SQL statements need not retrieve data from the same table, although they should have a common field (zqFeatureID) used to link the two. This section explains the fields in MapSource and DBSource SQL statements in detail, to allow you to write programs that generate theme definitions independently of the current set of WHEAT programs.

EMAPKGS2 and THEMEEDIT both include a series of forms that let the user interactively design SQL statements describing a theme and how to plot it. (The user generally has no inkling he/she is designing SQL queries.) This section gives an example SQL statement for each of the currently-supported theme types and explains the fields. For the most part, the names are self-explanatory. Values are usually those used by Visual Basic Professional 3.0.

Table 1 Theme Types

Geometric Type	Theme Type Code	Named Constant	Comments
Symbols	1	OnePoint	single points, plotted with a symbol
Lines	2	JointedLine	linear features such as rivers, highways
Tiles	6	TileBlock	cardinally-oriented rectangle, used for ViewBounds, USPLSS sections, and such
Polygons	9	SolidPolygon	polygonal feature, like state, geologic outcrop area, or lake. Related to line.
Text Labels	8	LabelText	text, like the name of a highway or lake or town
Raster Grid	10	RasterGrid	Weakly supported in current version. I am phasing this out in favor of a standalone raster GIS that will be compatible.
Photo Mosaic	32	PhotoMosaic	NOT USED in this version, but reserved for future use. This will be a set of subset bitmap images.
	0	SkipThis	NOT USED. Included to allow the deletion of features without having to shuffle memory.

Table 2 Example Map SQL Statements

Theme Type	MapSource	DBSource
Symbols	SELECT DISTINCTROW [X_UTM14] AS zqX , [Y_UTM14] AS zqY , 8 AS ZqSize , 0 AS ZqColor , 50 AS ZqSymbol , 2 AS ZqFontCode , [PlaceNameSerialNumber] AS zqFeatureID FROM [USPopulatedPlaceNames] ;	SELECT *, [PlaceNameSerialNumber] AS zqFeatureID FROM [USPopulatedPlaceNames]
Text Labels	SELECT DISTINCTROW [X_UTM14] AS zqX , [Y_UTM14] AS zqY, 8 AS zqFontSize , 0 AS zqColor , 1 AS zqFontCode , [PlaceName] AS zqLabelText, 0 AS zqAlignment , 0 AS zqOpaque, [PlaceNameSerialNumber] AS zqFeatureID FROM [USPopulatedPlaceNames] ;	SELECT *, [PlaceNameSerialNumber] AS zqFeatureID FROM [USPopulatedPlaceNames]
Lines	SELECT DISTINCTROW [XYChain_UTM14] AS zqXYBlobData , 1 AS zqWidth , 0 AS zqColor , 0 AS zqStyle , [LineIDNumber_Roads] AS zqFeatureID FROM [USRoads] ;	SELECT *, [LineIDNumber_Roads] AS zqFeatureID FROM [x Road Attributes Better QDF]
Polygons	SELECT DISTINCTROW [XYChain_UTM14] AS zqPolygonOutline , [RandomColor] AS zqFillColor , 0 AS zqFillStyle , 0 AS zqOutlineColor , 1 AS zqOutlineWidth , [SerialNumber] AS zqFeatureID FROM [USStates] WHERE ([State2Ltr] IN ('KS','MO','NE','CO','OK')) ORDER BY [RandomColor] ;	SELECT *, [SerialNumber] AS zqFeatureID FROM [USStates]
Tiles	SELECT DISTINCTROW [X_Centroid] AS zqMinX , [X_Centroid] AS zqMaxX , [Y_Centroid] AS zqMinY , [Y_Centroid] AS zqMaxY , 0 AS zqOutlineStyle , 255 AS zqOutlineColor , 1 AS zqOutlineWidth , 0 AS zqFillColor , 1 AS zqFillStyle , [kssas_SerialNumber] AS zqFeatureID FROM [Sections QNWR Area] ;	SELECT *, [kssas_SerialNumber] AS zqFeatureID FROM [Sections QNWR Area]

Table 3 Interpretation of Fields in EMAPKGS2

Field Name	Meaning	Notes
zqFeatureID	Unique ID Number	
zqX zqMinX zqMaxX	East-west coordinate	
zqY zqMinY zqMaxY	North-south coordinate	
ZqSize zqFontSize	Size of text in points	FontSize in VBasic
ZqColor zqFillColor zqOutlineColor	Color	ForeColor or FillColor in VBasic
ZqSymbol	Ascii character code for a symbol	Print chr$(SymbolCode)
ZqFontCode	Font Code	WHEAT font code. See section "Font Code"
zqLabelText	Text	Print Text (after CurrentX=X, CurrentY=Y)
zqAlignment	Text Alignment	Used by Windows API SetAlignment
zqOpaque		Not presently used
zqXYBlobData zqPolygonOutline	XYChain	An XYChain
zqWidth zqOutlineWidth	Outline Width	DrawWidth in VBasic
zqStyle	Line style	DrawStyle in VBasic
zqFillStyle	Fill style	FillStyle in VBasic

Symbol Code

The symbol code used to select the symbol to plot at a point is the ASCII character number. For example, symbol code 65 corresponds to "A". Symbol codes must represent printable characters (no control codes like backspace or such.) Only numbers in the ranges 33-126 and 161-255 represent printable characters and can be valid symbol codes.

The symbol is plotted with its baseline and left edge aligned with the point's coordinates. The characters in the fonts included with WHEAT were specially designed to have the logical center of the symbol at this location. Most fonts have the lower, left corner of a typical uppercase letter at that location. If you use fonts that did not come with WHEAT, the symbol will be offset slightly from the point's location.

The WHEAT fonts were created using CorelDraw 3.0. The samples includes some example files with the .CDR extension that can be used as templates for font creation CorelDraw.

Color Code

The color code used to define the line, symbol, or fill color in a WHEAT theme is a four-byte integer containing 1-byte for each of red, green, and blue. When loading a theme, clicking on the word Color to the left of the color listbox will bring up the standard Windows color dialogue box, which lets you choose a color visually and adds it to the list of colors.

Table 4 Some named colors

Black	0
Dark Red	128
Bright Red	255
Dark Green	32768
Bright Green	65280
Dark Blue	8388608
Bright Blue	16711680
Dark Purple	8388736
Bright Purple	16711935
Dark Gray	4210752
Medium Gray	8421504
Light Gray	12632256
White	16777215
Yellow	65535
Orange	33023
Brown	5243008

You can specify colors analytically. In the following discussion, it is assumed that color components are scaled from 0 to 255. Given the red, green, and blue components of a color, the color code can be calculated as

```
COLOR_CODE = 255*Red_Value + 255*255*Green_Value + 255*255*255*Blue_Value
```

Both MSAccess and Visual Basic produce a compatible number using the RGB( ) function. You can use the MSAccess RGB( ) function to set colors in an Update Query in MSAccess.

Line Width

The width of the line or outline in screen pixels. When printed, EMAPKGS2 will use three printer pixels for every screen pixel in setting line thickness, except for line thicknesses of 1, which will remain one pixel thick. A line width of zero leads to no line being drawn. The user can override the screen-to-print thickness ratio when printing.

Line Style

The line style used in drawing lines and outlines of filled figures is a code number indicating how to draw the line. It is the DrawStyle in MSAccess and Visual Basic.

Table 5 Line Styles

0	Solid
1	Dash
2	Dot
3	Dash-dot
4	Dash-dot-dot
5	Invisible
6	Inside solid

Fill Style

The fill style is used to determine how to shade in filled geographic features, such as polygons and tiles. It is the Fill Style in Visual Basic and MSAccess. The values are listed below.

Table 6 Fill Styles

0	Opaque
1	Transparent
2	Horizontal Line
3	Vertical Line
4	Upward Diagonal
5	Downward Diagonal
6	Cross
7	Diagonal Cross

Symbol Size and Font Size

For points and text labels, the size is the size of the symbol or text in points. On-screen, the size may be rounded up or down, depending on your monitory and video mode. Printers, which generally have much finer resolution, will usually print at the exact size.

Font Code

The font code used by E-MAP is a code number representing a particular font. The low 14-bits of the font code are the serial number of the font as found in the table zWHEATtblFontsList. (This table consists of fonts and a serial number. If WHEAT cannot find a font code entered, it will choose to use Arial.) The two high bits are used to represent bold and italic. The following code is from EMAPKGS2 and illustrates use of WHEAT font code values.

Text Alignment

Text labels can be aligned in several orientations relative to the point specified by the labels coordinates. For labeling features with an areal extent, like counties and soil units, it is best to use Center-Baseline, which centers the text horizontally at the X coordinate and aligns the text baseline to the Y coordinate. For point features, just about any combination will work.

You can also label a point several times by having labels with different alignments. For example, if you want to label wells with the formation from which water comes, a head value, and a serial number, you could have the formation code in the upper left by choosing right-bottom; the head in the upper left by choosing left-bottom; and the serial number in the lower right by choosing left-top.

The actual value corresponds to the value in the Windows SetTextAlign function.

Table 7 Alignment Values

0	Left Top
2	Right Top
6	Center Top
8	Left Bottom
10	Right Bottom
14	Center Bottom
24	Left Baseline
26	Right Baseline
30	Center Baseline

Criteria

The user can enter criteria to limit which features in the source table or query are to be loaded. The criteria consist of conditions on the fields chosen by the user. As an example, suppose you had a table about counties, and it contained fields containing the county line ([CountyLine] ,an XYBLOB), the county population ([Population] as long integer), the average income ([AvgIncome] as floating point number), and the county name ([Name] as text), and you only want to display counties with a population above 10000. Then you would enter the criterion

```
( [Population] > 10000 )
```

in the criteria box. If you wished to further limit the counties displayed to those with an average income above 22000, then you would enter the criteria

```
( [Population] > 10000 ) and ( [AvgIncome]>22000)
```

in the criteria box.

The criteria used in WHEAT are put directly into the WHERE clause of a SQL statement to define the data, so any allowable SQL criteria can be used in WHEAT. (See

the MSAccess manual or the Visual Basic Manual for a complete reference of allowable SQL statements.)

Program Activity Log Entries

Whenever your program changes the database contents, by modifying data or importing data, you should make an entry in the table ztblWHEAT_LOG. The subroutines in the next listing are designed to do that. I must admit, my own programs do not always log their activities, and I often regret this fact months later. Please try to be better than me about this.

Sample Programs and Files

Accompanying this report is a disk or file containing a series of sample programs and sample data files for you to use in developing your own WHEAT extensions. These are in a pkZip file and should be un-zipped using pkUnzip -d , to recreate the directory structure. When there are two files of the same name, the newer one is better.

You are free to copy these files and incorporate them into your programs. Please give proper academic credit by citing this publication.

XYBLOB05.BAS Utility XYChain functions and WHEAT log functions

WHEATBAS.MDB A Microsoft Access database, designed for use as an add-in, containing code in MSAccess Basic for use with WHEAT. There are sample forms showing how to program for WHEAT in Access. To use this as an add-in, copy it to the directory containing MSAccess.EXE as WheatBAS.MDA, and install it using the Add-In manager on the File menu.

BLOBMEMO\ Directory containing three Visual Basic projects

- BLOBMEMO BLOB–Memo converts XYChains as Binary Large Objects to/from XYChains in ASCII format in Memo fields in a table in a database.
- BLOB2PTS BLOB To Points takes a table containing XYChains and IDNumbers and produces a new table containing points as IDNumber, SequenceNumber, X, Y records.
- PTS2BLOB Points To BLOB takes a table containing records as IDNumber, SequenceNumber, X, Y records and produces a new table containing XYChains and IDNumbers

CHNMASSAG\ Assorted XYChain processing functions in one program. Illustrates XYChain reading and processing and field creation for results.

GENIMPEX\	Generic Import-Export reads/writes WXY format files. This can serve as a skeleton program for writing your own import/export programs. TESTWXY.FOR is a Fortran program that reads WXY format files.
THMEDTPP\	THEMEEDIT allows interactive design of WHEAT theme definitions
WHT2ARC	WHEAT To Arc/Info converts exports WHEAT XYChain data into Arc/Info generate format for use with GENERATE and exports other attribute data into dBase IV files for use with DBASEINFO.
WHTFRARC\	WHEAT from Arc (ARCS_IN4.MAK) imports Arc/Info generate files.

Code Listings

Listing 1 Type Declarations for Geometric Types.

```
Option Explicit
Type RealPoint
    X As Single
    Y As Single
End Type
Type String8
    f As String * 8
End Type
Dim RlPt As RealPoint
Dim S8 As String8
Dim SampleXYChain() as RealPoint
Type RealRect
   Left As Single
   Top As Single
   Right As Single
   Bottom As Single
End Type
Dim SampleRect as RealRect
```

Listing 2 Safe Routines to Covert Between "String" and Structure XYChains

```
Function RealPoints2BLOB (DataPts() As RealPoint) As String
'A function to convert an array of RealPoints to an XYChain in 'BLOB (Binary
   Large Object) format.
Dim n As Long,BigTemp As String ' Temporary storage of results
Dim RlPt As RealPoint    'Temporary RealPoint
Dim S8 As String8        'Fixed-width string structure
BigTemp = ""
For n = LBound(DataPts) To UBound(DataPts)
    'For each point, copy the point into a temporary RealPoint,
    ' copy that into the fixed-width string type using LSet,
    ' then append the string member to the function return
    RlPt = DataPts(n)
    LSet S8 = RlPt
    BigTemp = BigTemp & S8.f
Next n
'Return XYBLOB that accumulated in BigTemp
RealPoints2BLOB = BigTemp
End Function

Sub BLOB2RealPoints (BLOBin As String, DataPts() As RealPoint)
' A subroutine to convert an XYChain in (BLOB) format to an
' array of RealPoints  It would be nice to make this a function,
' but a function cannot return an array.
Dim n As Long, nPoints As Long, NPos As Long
Dim RlPt As RealPoint    'Temporary RealPoint
Dim S8 As String8        'Fixed-width string structure
nPoints = Len(BLOBin) \ 8 '8 Bytes to a RealPoint
ReDim DataPts(1 To nPoints)
NPos = 1  'NPos will keep track of where in the BLOB we are
For n = 1 To nPoints
```

```
    S8.f = Mid$(BLOBin, NPos, 8)
    LSet RlPt = S8
    DataPts(n) = RlPt
    NPos = NPos + 8
Next n
End Sub
```

Listing 3 Automatic Field Selection Routines

```
Sub cmbTblQryList_Click ()

 If Me!ACTION.Caption = "wait" Then Exit Sub
 If Not flgIgnoreMapSQLChange Then
   'For usual case, set the SpatialQuerySQL and the theme name
   txtSpatialQuerySQL = "SELECT * FROM " &
   AddBracketsAsNeeded(CStr(cmbTblQryList))
   ListOfFieldsForDBForm = "*"
   txtThemeName = "POLYGONS: " & cmbTblQryList & "   " & Now
 End If

Dim TableName As String
On Error GoTo ERRcmbTblQryList_Click
 TableName = cmbTblQryList
 Set snpFieldList = MapDB.CreateSnapshot(TableName)
 nFields = snpFieldList.Fields.Count
 cmbFieldList.Clear
 cmb_zqPolygonOutline.Clear
 ReDim FieldsInCurrentQT(1 To nFields), FieldListWBrackets(1 To nFields)

 For i = 1 To nFields
    If snpFieldList.Fields(i - 1).Type = DB_LONGBINARY Then
        'Put XYBLOBs (stored in OLE/BinaryLargeObjectFields) in the
        ' outline list box
        cmb_zqPolygonOutline.AddItem "[" & snpFieldList.Fields(i - 1).Name &
   "]"
       Else
        'Put everything else in the general field list.
        cmbFieldList.AddItem snpFieldList.Fields(i - 1).Name
        FieldListWBrackets(i) = "[" & snpFieldList.Fields(i - 1).Name & "]"
    End If
    FieldsInCurrentQT(i) = snpFieldList.Fields(i - 1).Name
 Next i
 snpFieldList.Close
 cmbFieldList.ListIndex = 0
 txtCriteria = ""

  If cmb_zqPolygonOutline.ListCount = 0 Then
    Beep
    MsgBox "This field contains no line data." & CRLF & "Please choose a new
   query or table that does."
    cmbTblQryList.SetFocus
    SendKeys "%{DOWN}"
    Exit Sub
   Else
    cmb_zqPolygonOutline.ListIndex = 0
   End If
```

```
 cmb_zqFeatureID.Clear
 FillCMBFromSGSTLST FieldListWBrackets(), FeatureNameLookFor,
   FeatureNameShowAlways, cmb_zqFeatureID
 cmb_zqFeatureID.ListIndex = 0

 cmb_zqFillColor.Clear
 FillCMBFromSGSTLST FieldListWBrackets(), ColorLookFor, ColorShowAlways,
   cmb_zqFillColor
 cmb_zqFillColor.ListIndex = 0

 cmb_zqOutlineColor.Clear
 FillCMBFromSGSTLST FieldListWBrackets(), ColorLookFor, ColorShowAlways,
   cmb_zqOutlineColor
 cmb_zqOutlineColor.ListIndex = 0

 cmb_zqOutlineWidth.Clear
 FillCMBFromSGSTLST FieldListWBrackets(), LineWidthLookFor, "0<=>No_Outline
   " & LineWidthShowAlways, cmb_zqOutlineWidth
 cmb_zqOutlineWidth.ListIndex = 0

 cmb_zqFillStyle.Clear
 FillCMBFromSGSTLST FieldListWBrackets(), FillStyleLookFor,
   FillStyleShowAlways, cmb_zqFillStyle
 cmb_zqFillStyle.ListIndex = 0


 FlagActivateDragDrop = 0
Exit Sub
ERRcmbTblQryList_Click:
Select Case Err
   Case 380
   'Invalid value in property
   '(setting the listindex to 0 when there are no items.
   'Indicates an empty list.
     Beep
     Resume Next
   Case Else
      MsgBox "Error #" & Err & " occurred in cmbTblQryList_Click." & CRLF &
   Error
      Err = 0
      Exit Sub
      Resume Next
   End Select
End Sub

Sub FillCMBFromSGSTLST (strSearchIn() As String, ByVal strLookFor As String,
   ByVal strShowAnyway As String, CMB As ComboBox)
'Takes a bunch of strings, in strSearchIn(), and adds whichever of those
   contain strings in strLookFor to the CombBox.
'It always adds the strings in strAddAnyway.
'For instance, if strSearchIn contained New Jersey, Texas, New Mexico, X in
   UTM13, X in SPCS
' and strLookFor contained "X", and strAddAnyway contained "3", "8/2", then
'  the combobox would end up containing Texas, New Mexico ,X in UTM13, "3",
   "8/2", and X in SPCS.
'if strLookFor contained Jersery, Mexico, the combobox would get
' New Jersey, New Mexico, "3", "8/2"
Dim n As Long, m As Long, l As Long, nLookFor As Long

strLookFor = UCase$(strLookFor)
```

```
nLookFor = 0
ReDim LookFor(nLookFor) As String
FillArrayWithDelimitedStrings strLookFor, " ", LookFor()
nLookFor = UBound(LookFor)
ReDim Used(LBound(strSearchIn) To UBound(strSearchIn))  As Integer
Dim Temp As String
For m = 1 To nLookFor
    For n = LBound(strSearchIn) To UBound(strSearchIn)
       If Not Used(n) Then
        If InStr(strSearchIn(n), LookFor(m)) <> 0 Then
            CMB.AddItem strSearchIn(n)
            Used(n) = True
        End If
       End If
    Next n
Next m
'Add the constant items
DlmtdStngsToComboBox " ", strShowAnyway, CMB

End Sub

Sub FillArrayWithDelimitedStrings (StringIn As String, Delimiter As String,
   strArrayOut() As String)
Dim nStrings As Long, n As Long
nStrings = CountOccurrences(StringIn, Delimiter)
ReDim strArrayOut(nStrings)
For n = 1 To nStrings
    strArrayOut(n) = ExtractDelimitedString(StringIn, Delimiter, n)
Next n

End Sub

Function ExtractDelimitedString (ByVal InString As String, ByVal Delimiter
   As String, ByVal nth As Long) As String
'  Pulls the nth Delimiter-delimited string out of InString.
'Greg Pouch KGS 13 January 1994


  Dim Temp1 As String, ThisField As String
  Dim n As Long
Dim DelimLength As Long, NextDelimiter As Long

If nth < 1 Then Exit Function
DelimLength = Len(Delimiter)
If DelimLength < 1 Then Exit Function
If CountOccurrences(InString, Delimiter) < nth Then Exit Function

Temp1 = InString
n = 0
Do
    NextDelimiter = InStr(Temp1, Delimiter)
    ThisField = Mid$(Temp1, 1, NextDelimiter - 1)
    Temp1 = Mid$(Temp1, NextDelimiter + DelimLength)
    n = n + 1
Loop While (n < nth)
ExtractDelimitedString = ThisField
End Function

Function CountOccurrences (ByVal MamaString As String, ByVal BabyString As
   String) As Long
```

```
Dim n As Long
Dim BabyLength As Long
Dim NextBabyAt As Long
BabyLength = Len(BabyString)

If BabyLength < 1 Then Exit Function

Dim CurrentPos As Long
CurrentPos = 1
n = 0
NextBabyAt = InStr(CurrentPos, MamaString, BabyString, 1)
Do Until NextBabyAt = 0
 n = n + 1
 CurrentPos = NextBabyAt + BabyLength
 NextBabyAt = InStr(CurrentPos, MamaString, BabyString, 1)
Loop
CountOccurrences = n
End Function


Sub DlmtdStngsToComboBox (Delimiter As String, StringIn As String, ComboOut
   As ComboBox)
Dim nStrings As Long, n As Long
nStrings = CountOccurrences(StringIn, Delimiter)
For n = 1 To nStrings
    ComboOut.AddItem ExtractDelimitedString(StringIn, Delimiter, n)
Next n

End Sub
```

Listing 4 Default Substrings and Constants for Automatic Field Selection

The following subroutine initializes the substrings that will be searched for in field names and the constants that will be shown to the user. As can be seen, the user can override these values by modifying the WHEAT.INI file.

```
Sub InitializeWHEATEnv (INIFile As String)
Dim Buffer  As String, intSizeBuffer As Integer, nBytesBack As Integer
Buffer = Space$(1024)
'The LOOKFOR and SHOWALWAYS strings for each variable for which one is
'  provided are to be kept in the [DEFAULT FIELDS] section of the wheat.ini
   file.

Dim SectionName As String, KeyString As String, DefaultValue As String
    SectionName = "DefaultFields"

'SizeLookFor = "FONTSIZE HEIGHT WIDTH SIZE FONT "
   KeyString = "SizeLookFor"
   DefaultValue = "FONTSIZE HEIGHT FONT "
   Buffer = Space$(1024)
   nBytesBack = GetPrivateProfileString(SectionName, ByVal KeyString,
   DefaultValue, Buffer, 1024, INIFile)
   SizeLookFor = Left$(Buffer, nBytesBack) & " "

'SizeShowAlways = "10 12 18 24 36 "
    Buffer = Space$(1024)
    KeyString = "SizeShowAlways"
    DefaultValue = "10 12 18 24 36 8 6 "
```

```
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    SizeShowAlways = Left$(Buffer, nBytesBack) & " "
'ColorLookFor
    Buffer = Space$(1024)
    KeyString = "ColorLookFor"
    DefaultValue = "COLOR COLOUR TINT HUE "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    ColorLookFor = Left$(Buffer, nBytesBack) & " "

'ColorShowAlways
    Buffer = Space$(1024)
    KeyString = "ColorShowAlways"
    DefaultValue = "0<=>Black 255<=>BrightRed 65280<=>BrightGreen
   16711680<=>BrightBlue 8421504<=>Gray 16777215<=>White 65535<=>Yellow
   33023<=>Orange "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    ColorShowAlways = Left$(Buffer, nBytesBack) & " "

'SymbolCodeLookFor
    Buffer = Space$(1024)
    KeyString = "SymbolCodeLookFor"
    DefaultValue = "SYM CODE "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    SymbolCodeLookFor = Left$(Buffer, nBytesBack) & " "

'SymbolCodeShowAlways
    Buffer = Space$(1024)
    KeyString = "SymbolCodeShowAlways"
    DefaultValue = "45 51 33 43 "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    SymbolCodeShowAlways = Left$(Buffer, nBytesBack) & " "

'XLookFor
    Buffer = Space$(1024)
    KeyString = "XLookFor"
    DefaultValue = "X_ EAST WEST LONG "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    XLookFor = Left$(Buffer, nBytesBack) & " "
  XShowAlways = "" 'Ignore/overwrite user prefernce for having a constant X
   in some map layer.

'YLookFor
    Buffer = Space$(1024)
    KeyString = "YLookFor"
    DefaultValue = "Y_ NORTH SOUTH LAT "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    YLookFor = Left$(Buffer, nBytesBack) & " "
  YShowAlways = "" 'Ignore/overwrite user prefernce for having a constant X
   in some map layer.

'FeatureNameLookFor
    Buffer = Space$(1024)
    KeyString = "FeatureNameLookFor"
```

```
    DefaultValue = "SERIAL NUM ID # "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FeatureNameLookFor = Left$(Buffer, nBytesBack) & " "
  FeatureNameShowAlways = "" 'Ignore/overwrite user prefernce for having a
   constant NAME in some map layer.

'XYLookFor
    Buffer = Space$(1024)
    KeyString = "XYLookFor"
    DefaultValue = "XY BLOB CHAIN LINE POLY BORDER BOUNDARY "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    XYLookFor = Left$(Buffer, nBytesBack) & " "
  XYShowAlways = "" 'Ignore/overwrite user prefernce for having a constant
   NAME in some map layer.

'FontCodeLookFor
    Buffer = Space$(1024)
    KeyString = "FontCodeLookFor"
    DefaultValue = "FONTCODE TYPE "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FontCodeLookFor = Left$(Buffer, nBytesBack) & " "

'FontCodeLookFor
    Buffer = Space$(1024)
    KeyString = "FontCodeShowAlways"
    DefaultValue = "0 1 2 3 "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FontCodeShowAlways = Left$(Buffer, nBytesBack) & " "

'FeatureTextLookFor
    Buffer = Space$(1024)
    KeyString = "FeatureTextLookFor"
    DefaultValue = "TEXT LABEL NAME DESC "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FeatureTextLookFor = Left$(Buffer, nBytesBack) & " "

'FeatureTextShowAlways
    Buffer = Space$(1024)
    KeyString = "FeatureTextShowAlways"
    DefaultValue = "PutYourMessageHere,In DoubleQuotes"
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FeatureTextShowAlways = Left$(Buffer, nBytesBack) & " "

'LineWidthLookFor
    Buffer = Space$(1024)
    KeyString = "LineWidthLookFor"
    DefaultValue = "LINEWIDTH WID THICK "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    LineWidthLookFor = Left$(Buffer, nBytesBack) & " "

'LineWidthShowAlways
    Buffer = Space$(1024)
    KeyString = "LineWidthShowAlways"
```

```
    DefaultValue = "1 2 3 4 5 6 "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    LineWidthShowAlways = Left$(Buffer, nBytesBack) & " "

'LineStyleLookFor
    Buffer = Space$(1024)
    KeyString = "LineStyleLookFor"
    DefaultValue = "LINESTYLE STYLE "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    LineStyleLookFor = Left$(Buffer, nBytesBack) & " "

'LineStyleShowAlways
    Buffer = Space$(1024)
    KeyString = "LineStyleShowAlways"
    DefaultValue = "0<=>Solid 1<=>Dash 2<=>Dot 3<=Dash-Dot 4<=>Dash-Dot-Dot
   5<=>Transparent 6<=>Inside_Solid "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    LineStyleShowAlways = Left$(Buffer, nBytesBack) & " "

'FillStyleLookFor
    Buffer = Space$(1024)
    KeyString = "FillStyleLookFor"
    DefaultValue = "FILLSTYLE FILL STYLE INTERIOR INSIDE"
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FillStyleLookFor = Left$(Buffer, nBytesBack) & " "

'FillStyleShowAlways
    Buffer = Space$(1024)
    KeyString = "FillStyleShowAlways"
    DefaultValue = "0<=>Solid 1<=>Transparent 2<=>Horizontal 3<=>Vertical
   4<=>Upward_Diagonal 5<=>DownwardDiagonal 6<=>HVCross(+)
   7<=>DiagonalCross(x) "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    FillStyleShowAlways = Left$(Buffer, nBytesBack) & " "

'AlignmentLookFor
    Buffer = Space$(1024)
    KeyString = "AlignmentLookFor"
    DefaultValue = "ALIGN ORIENT "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    AlignmentLookFor = Left$(Buffer, nBytesBack) & " "
'AlignmentShowAlways
    Buffer = Space$(1024)
    KeyString = "AlignmentShowAlways"
    DefaultValue = "0<=>Left_Top 2<=>Right_Top 6<=>Center_Top
   8<=>Left_Bottom 10<=>Right_Bottom 14<=>Center_Bottom 24<=>Left_Baseline
   26<=>Right_Baseline 30<=>Center_Baseline "
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    AlignmentShowAlways = Left$(Buffer, nBytesBack) & " "

'[SpatialQueries]
SectionName = "SpatialQueries"
    Buffer = Space$(1024)
```

```
    KeyString = "HighlightColor"
    DefaultValue = "255"
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    DefaultHighLightColor = Val(Buffer)

    Buffer = Space$(1024)
    KeyString = "HighlightRadius"
    DefaultValue = "400"
    nBytesBack = GetPrivateProfileString(SectionName, KeyString,
   DefaultValue, Buffer, 1024, INIFile)
    DefaultHighLightRadius = Val(Buffer)
    If DefaultHighLightRadius <= 0 Then DefaultHighLightRadius = 200
End Sub
```

Listing 5 WXY Functions from XYBLOB4.BAS from GENIMPEX

```
Option Explicit
Type RealPoint
    X As Single
    Y As Single
End Type
Type String8
    f As String * 8
End Type
Dim RlPt As RealPoint
Dim S8 As String8
Type RealRect
   Left As Single
   Top As Single
   Right As Single
   Bottom As Single
End Type

Function WXY_Read (strFileName As String, DB As Database, strTableName As
   String) As Long
'WXY_Read (strFileName As String, DB As Database, strTableName As String) As
   Long
'WXY_Read (strFileName , DB , strTableName )
'' A routine to read a .WXY file, used for easy import/export to/from WHEAT.
' The return value is the number of chains read.
' This routine may change strTableName if necessary to get a unique name.


'The WXY format is used as a transmission format for XY Chains.
' It is a binary format, with no record delimiters.
'The intended purpose is as an intermediate import/export format,
' or for use by external processing routines.

'Each "record" consists of 8-bytes.
'The first record MUST be
'1   WHEATCHN        note that this is upper case
'The rest of the file is in blocks, one for each line or polygon.
' The first record in a block consists of two 4-byte integers
' IDNumber, nPairs    where IDNumber is a serial number, to be later
'                       used for linking, and nPairs is the number of
   coordinate pairs in the line/polygon
```

```
'After the header record for a line, all subsequent records consist
' of a single X,Y pair in IEEE85 format (default for 4-byte float for most
   compilers, now)

On Error GoTo ERR_WXYRead

Dim strCheck As String
Dim IDNumber As Long, nPairs As Long, n As Long, nChains As Long
Dim X As Single, Y As Single
Dim RlPts() As RealPoint

Close 10
Open strFileName For Binary As 10
strCheck = Space$(8)
Get 10, , strCheck
If strCheck <> "WHEATCHN" Then
   'Not a valid file
   WXY_Read = -1
   Exit Function
End If


'This is a valid file, so create the table requested.
Dim tdf As New tabledef, nfID As New field, nfXYChain As New field
Dim tbl As table, fldID As field, fldXY As field

On Error Resume Next
'First try opening the table to make sure it isn't there.  If it is,
' automatically append the time to the table name to make it unique.
  Set tbl = DB.OpenTable(strTableName)
  If Err = 0 Then
   Err = 0
   strTableName = strTableName & Format(Now, "YYMMDDhhmmss")
  End If


On Error GoTo ERR_WXYRead

nfID.Name = strTableName & "IDNumber": nfID.Type = DB_LONG
nfXYChain.Name = "XYChain": nfXYChain.Type = DB_LONGBINARY

tdf.Fields.Append nfID
tdf.Fields.Append nfXYChain

tdf.Name = strTableName
LBL_AppendTable:
DB.TableDefs.Append tdf

Set tdf = Nothing: Set nfID = Nothing: Set nfXYChain = Nothing



Set tbl = DB.OpenTable(strTableName)
Set fldID = tbl.Fields(strTableName & "IDNumber")
Set fldXY = tbl.Fields("XYChain")
nChains = 0
Do
   Get 10, , IDNumber
   Get 10, , nPairs
  If nPairs > 0 Then
```

```
    ReDim RlPts(1 To nPairs)
    For n = 1 To nPairs
       Get 10, , RlPts(n).X
       Get 10, , RlPts(n).Y
    Next n
    tbl.AddNew
       fldID = IDNumber
       fldXY = RealPoints2BLOB(RlPts())
    tbl.Update
    nChains = nChains + 1
   End If


Loop Until EOF(10)


WXY_Read = nChains
tbl.Close
Set tbl = Nothing
Set fldID = Nothing: Set fldXY = Nothing




Exit Function
ERR_WXYRead:
Select Case Err
   Case Else
      MsgBox "Error #" & Err & CRLF & Error
      Exit Function
      Resume
End Select


End Function

Function WXY_Write (dnsIn As Dynaset, strXYField As String, strIDField As
   String, strFileName As String) As Long
'WXY_Write (dnsIn As Dynaset, strXYField As String, strIDField As String,
   strFileName As String) As Long
'WXY_Write (dnsIn , strXYField , strIDField , strFileName )
'' A routine to write a .WXY file, used for easy import/export to/from
   WHEAT.
' The return value is the number of chains written.
' This routine may change strTableName if necessary to get a unique name.


'The WXY format is used as a transmission format for XY Chains.
' It is a binary format, with no record delimiters.
'The intended purpose is as an intermediate import/export format,
' or for use by external processing routines.

'Each "record" consists of 8-bytes.
'The first record MUST be
'1   WHEATCHN        note that this is upper case
'The rest of the file is in blocks, one for each line or polygon.
' The first record in a block consists of two 4-byte integers
' IDNumber, nPairs    where IDNumber is a serial number, to be later
'                       used for linking, and nPairs is the number of
   coordinate pairs in the line/polygon
```

```
'After the header record for a line, all subsequent records consist
' of a single X,Y pair in IEEE85 format (default for 4-byte float for most
   compilers, now)

On Error GoTo ERR_WXYWrite
Dim strTemp As String, nChains As Long
  Dim nPairs As Long, IDNumber As Long

'Create the file
Close 11
Open strFileName For Binary As 11

strTemp = "WHEATCHN"
Put 11, , strTemp


Dim fldID As field, fldXY As field
Set fldID = dnsIn.Fields(strIDField)
Set fldXY = dnsIn.Fields(strXYField)



dnsIn.MoveFirst

nChains = 0
Do
  IDNumber = fldID
  nPairs = fldXY.FieldSize() \ 8
  If nPairs * 8 <> fldXY.FieldSize() Then
     strTemp = Left$(fldXY, 8 * nPairs)
    Else
     strTemp = fldXY
  End If
  Put 11, , IDNumber
  Put 11, , nPairs
  Put 11, , strTemp
  nChains = nChains + 1
  dnsIn.MoveNext

Loop Until dnsIn.EOF


 WXY_Write = nChains

'Clean up
  dnsIn.MoveFirst
  Set fldXY = Nothing
  Set fldID = Nothing



Exit Function
ERR_WXYWrite:
Select Case Err
   Case Else
      MsgBox "Error #" & Err & CRLF & Error
      Exit Function
      Resume
End Select
End Function
```

Listing 6 Font Codes

```
Type typFontLookup
    Number As Integer          'an Integer*2 ID numbers
    Name As String             'The name of the font
    ExtraSpace As String * 4   'Extra space, for whatever comes up
End Type
Global strListOfFonts() As String, FontTable()  As typFontLookup
Dim nF As Integer, nFonts As Integer, nFnt As Integer
Global Const FONT_BOLD = -32768
Global Const FONT_ITALIC = 16384
Const FONT_MASKIN = 16383

Sub SetFontCodeInPIC (ByVal FontCode As Integer, PIC As PictureBox)
    On Error Resume Next
    PIC.FontName = FontNameFromID(FontCode)
    PIC.FontBold = IsBold(FontCode)
    PIC.FontItalic = IsItalic(FontCode)
End Sub

Function FontNameFromID (ByVal IDNumber As Integer) As String
  Dim tmp As String, tmpCode As Integer
  tmpCode = MASK14LOBITS And IDNumber
  tmp = "Arial" 'Set default font to Arial
  For nF = 1 To UBound(FontTable)

      If FontTable(nF).Number = tmpCode Then
        tmp = FontTable(nF).Name
        Exit For
      End If
  Next nF
  FontNameFromID = tmp
End Function

Function IsBold (FontCode As Integer) As Integer
 If (FontCode And FONT_BOLD) <> 0 Then
    IsBold = True
   Else
    IsBold = False
 End If
End Function

Function IsItalic (FontCode As Integer) As Integer
If (FontCode And FONT_ITALIC) <> 0 Then
    IsItalic = True
 Else
    IsItalic = False
End If
End Function
```

Listing 7 Log Entries for WHEAT

```
Sub WheatLog_AddEntry (DBName As String, strInputs As String, strOutputs As
   String, strProgName As String, strUserName As String)
```

```
'''Call WheatLog_AddEntry (DBName,strInputs,strOutputs,strProgName,
   strUserName)
'' Adds an entry to the WheatLog Table.
Dim DB As Database
'Dim strInputs As String, strOutputs As String, strProgName As String,
   strUserName As String
'call WheatLog_AddEntry (DBName , strInputs, strOutputs , strProgName ,
   strUserName )
On Error Resume Next
Dim tbl As table

Set DB = OpenDatabase(DBName, False, False)
Set tbl = DB.OpenTable("ztblWHEAT_LOG")
 If Err <> 0 Then
   Call WheatLog_DefineTable(DB)
   DB.Close
   Set DB = OpenDatabase(DBName, False, False)
   Set tbl = DB.OpenTable("ztblWHEAT_LOG")
 End If
 Err = 0

 tbl.AddNew
   tbl("Inputs") = strInputs
   tbl("Outputs") = strOutputs
   tbl("ProgramName") = strProgName
   tbl("UserName") = strUserName
   tbl("TimeStamp") = Now
 tbl.Update
'    "WHEAT_Log_SerialNumber" Serial number not set.
   tbl.Close : Set tbl = Nothing
   DB.Close : Set DB = Nothing

End Sub

Sub WheatLog_DefineTable (DBIn As Database)
On Error GoTo ERRDefine_ztblWHEATLOG
'GWPouch 940830
'Defines a table store program log entries


Dim NewTable As New tabledef
NewTable.Name = "ztblWHEAT_LOG"

Dim nfSerial As New field
nfSerial.Type = DB_LONG
nfSerial.Name = "SerialNumber"
nfSerial.Attributes = 49'Auto-increment, fixed width, updateable
NewTable.Fields.Append nfSerial

Dim nfWhen As New field
nfWhen.Type = DB_DATE
nfWhen.Name = "TimeStamp"
NewTable.Fields.Append nfWhen

Dim nfInputs As New field
nfInputs.Type = DB_MEMO
nfInputs.Name = "Inputs"
NewTable.Fields.Append nfInputs

Dim nfOutputs As New field
```

```
nfOutputs.Type = DB_MEMO
nfOutputs.Name = "Outputs"
NewTable.Fields.Append nfOutputs

Dim nfProgramName As New field
nfProgramName.Type = DB_TEXT
nfProgramName.Size = 255
nfProgramName.Name = "ProgramName"
NewTable.Fields.Append nfProgramName

Dim nfUserName As New field
nfUserName.Type = DB_TEXT
nfUserName.Size = 255
nfUserName.Name = "UserName"
NewTable.Fields.Append nfUserName

DBIn.TableDefs.Append NewTable

Exit Sub


ERRDefine_ztblWHEATLOG:
Select Case Err
 Case 3010 'Table of that name already exists
    Err = 0
    Exit Sub
    Resume
 Case 3125
    'TableOutName = InputBox$(TableOutName & " is not a valid name.  Please
   choose a new one.", "Saving Contours to Table", TableOutName)
    'NewTable.Name = TableOutName
    'Resume
 Case Else
    MsgBox "Error #" & Err & " occured in WheatLog_DefineTable." & CRLF &
   Error
    Err = 0
    Exit Sub
    Resume
End Select

End Sub
```