

Visualization and Management of Non-Spatial, Time-Series Data in ArcMap

Michael F. Liberti

ABSTRACT

An ArcMap toolbar was developed using ArcObjects to visually analyze and manage time-series water level records. Tucson Water maintains continuously recording dataloggers in over 60 monitoring wells. Many of these wells are affected by interference from other nearby pumping wells. The issue was to develop a tool for Hydrology staff to visually evaluate continuous water level data to eliminate non-static, pumping water level measurements.

The toolbar accesses a database to return a water level recordset for a given well. The water level records are converted into Cartesian XY data using the Julian date for X and the water level depth for Y. The converted XY data are manifested in ArcMap as a point event layer in Cartesian space (depth vs time). Point event data representing one or more database records can be identified and selected. Selected point records can then be flagged in the database as pumping water levels.

INTRODUCTION

Portions of this introduction were adapted from the Tucson Water: Water Plan: 2000-2050 (Tucson Water, 2004)



Setting

The City of Tucson is located in the northern semi-arid reaches of the Sonoran Desert in eastern Pima County, Arizona approximately 100 miles SE of Phoenix (Figure 1). As shown on Figure 2, the City is situated in the center of the Tucson basin which is a broad, desert valley surrounded by the Santa Catalina, Rincon, Santa Rita, Sierrita, Tortolita, and Tucson Mountains. Basin elevations in the City of Tucson range from about 2,300 feet above mean sea level (AMSL) in the northwest to about 3,200 feet AMSL in the southeast; the surrounding mountains range in elevation from approximately 4,700 to over 9,100 feet AMSL.

Figure 1. Location Map

Various ephemeral washes and rivers are located throughout the Tucson region. When storm flows occur, the Santa Cruz River runs north-northwest, and Rillito Creek runs from west-northwest. The Rillito is formed by the confluence of Tanque Verde Creek and Pantano Wash. In Avra Valley, Brawley Wash is the primary channel and merges with the Santa Cruz River near the Pima County/Pinal County line.

The average daily minimum temperature in the City ranges from about 39°F in January to 73°F in July while the average daily maximum temperature ranges from 65°F in January to 100°F in July. The local area annually averages about 12 inches of precipitation in the valleys and about 25 inches in the higher elevations. In 2000, the City of Tucson's population was 486,699, making it the second largest city in Arizona.

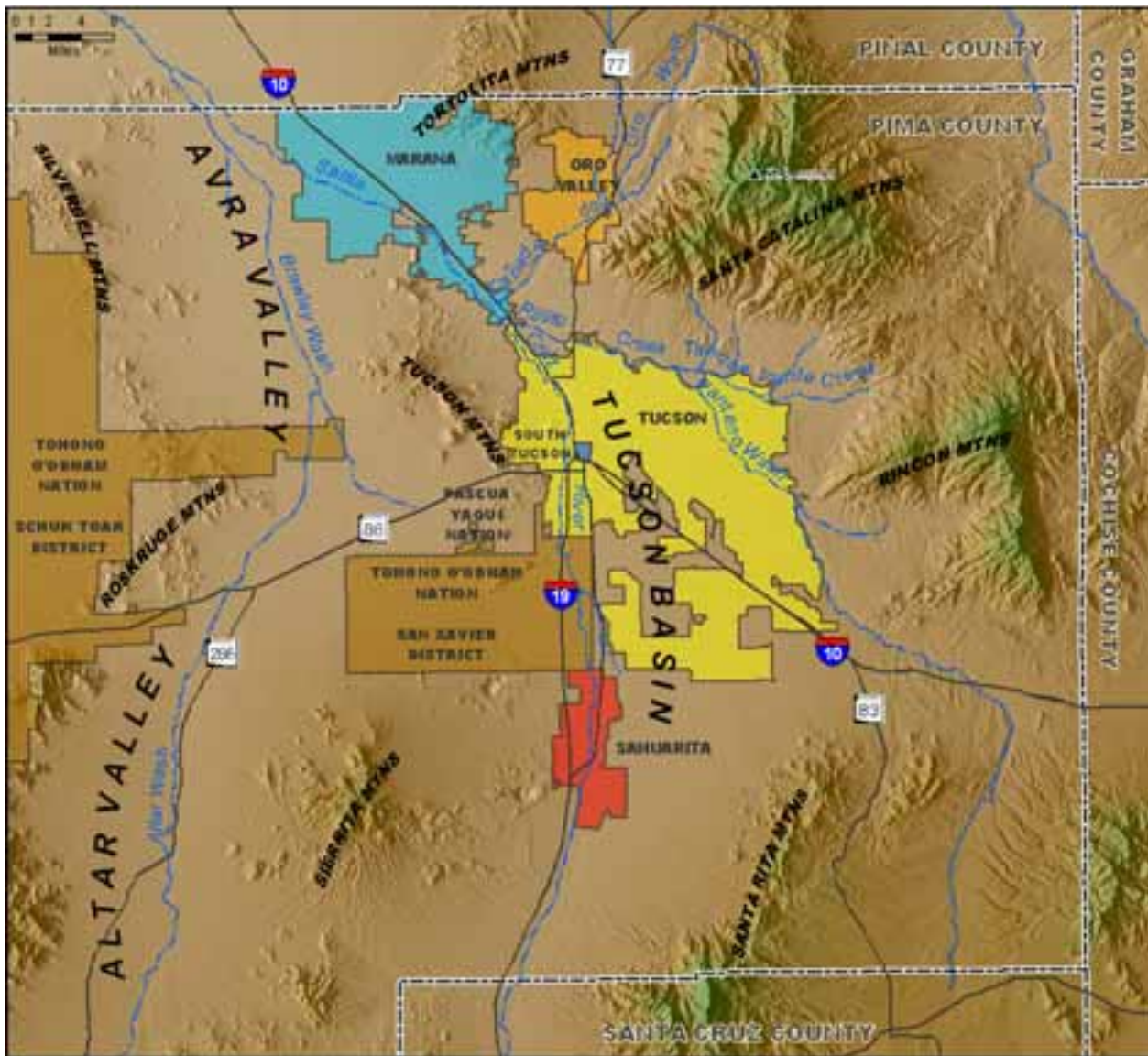


Figure 2. Tucson Region Site Map

Water Utility

Tucson Water (TW) is a municipal water provider owned and operated by the City of Tucson. The Utility is subject to the authority of the City of Tucson Mayor and Council, and the Director of Tucson Water is subject to the authority of the City Manager. Tucson Water is self-supporting and relies primarily on revenues generated from water connection fees and water sales.

Tucson Water is the largest water provider in southeastern Arizona and one of at least 20 water providers within the Tucson basin. In 2000, the Utility served a population of 638,936 within a 300-square-mile service area. Approximately 40% of Tucson Water customers reside outside the jurisdictional boundary of the City (Figure 3). The Utility operates a dual water system that serves potable water and reclaimed water for non-potable use or irrigation. Tucson Water currently has 4,300 miles of pipelines that

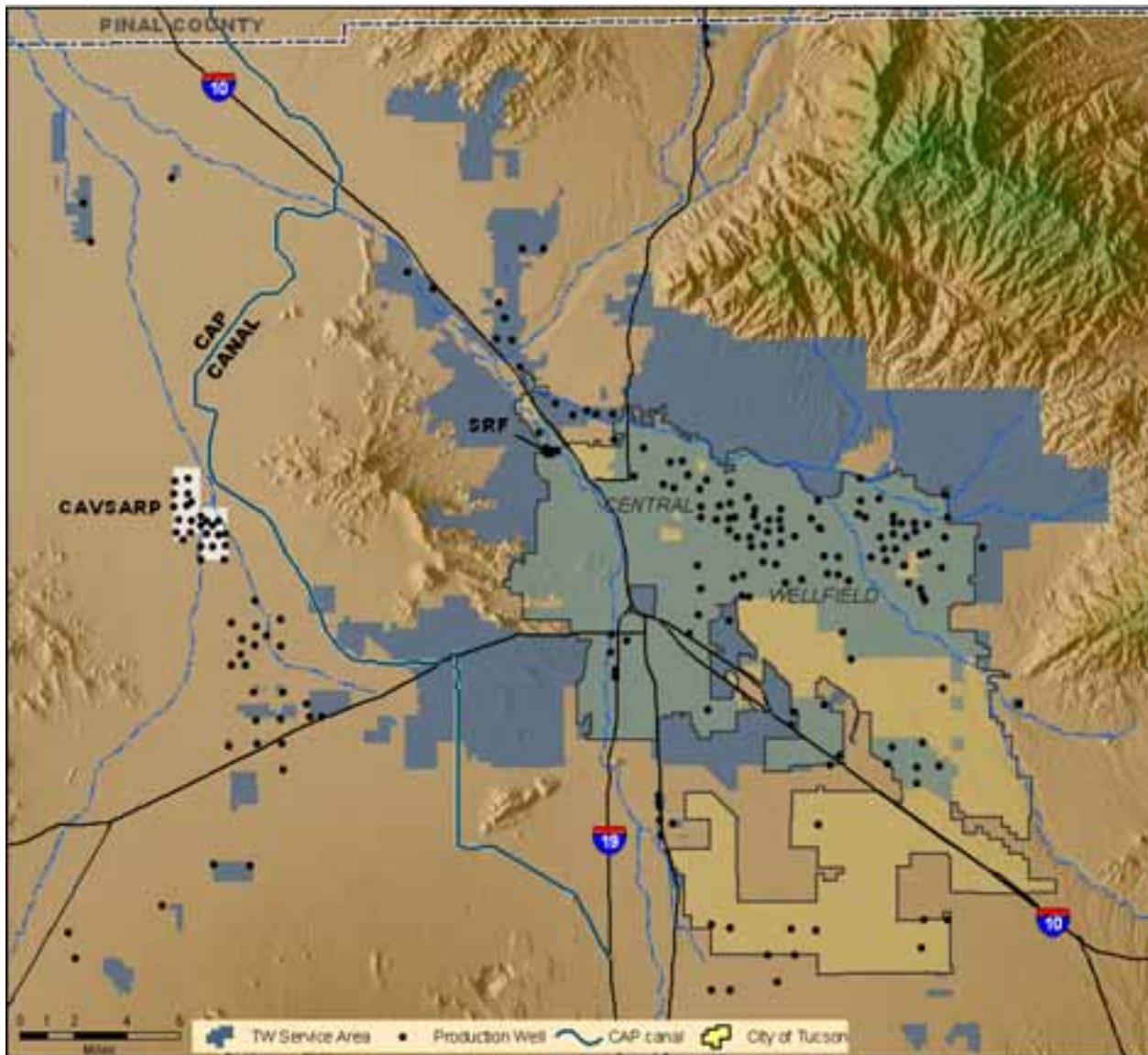


Figure 3. Current Tucson Water Service Area, Water Resources and Production Wells

convey potable and reclaimed water to customers over a 300 square-mile service area. Tucson Water has recently diversified its use of water resources to include not only ground water but also Colorado River water and reclaimed water.

In 2005 Tucson Water delivered about 123,000 ac-ft of potable water to customers. To meet the potable demand, about 72,300 ac-ft of groundwater was pumped from over 200 production wells. About 70,000 ac-ft of Colorado River water was recharged and 50,700 ac-ft was recovered from the Central Avra Valley Storage and Recovery Project (CAVSARP). Aside from potable water, Tucson Water delivered almost 8,000 ac-ft of non-potable water to turf irrigators. Reclaimed water demand is met by recovering tertiary quality effluent that is either treated through dual-media filtration or through recharge and recovery (soil aquifer treatment) at the Sweetwater Recharge Facilities (SRF).

Water Resources

The Tucson basin, as well as much of southwest Arizona, southeast California and southern Nevada is located in the Basin and Range geologic province characterized by high mountain ranges separated by broad alluvial valleys. The development of this geologic province began with the uplift of mountain ranges as a result of volcanic and seismic activity. The uplifted mountains were pulled apart by normal faulting that formed deep valleys between mountain ranges. Over time, subsequent erosional degradation of the mountains (topographic highs) filled the valleys (topographic lows) with alluvial sediments. These valleys or alluvial basins also filled with groundwater that accumulated over geologic time from surface waters percolating deep into the deposited alluvium. The result, as seen in the Tucson basin, is a deep, thick, regional alluvial aquifer system.

Prior to the early 1990s, the Tucson community had relied almost exclusively on ground water to meet water demand. Due to rapidly growing demand associated with population increases following World War II, the regional ground-water system transitioned from one in approximate equilibrium, where a balance existed between ground-water withdrawals and natural recharge, to one of accelerating depletion. Despite implementation of demand management programs, ground-water withdrawals continued to increase due to continuing growth through 2000. Rapidly declining water levels in the metropolitan area as well as in surrounding areas have resulted in measurable land subsidence, increased pumping costs, and the gradual loss of natural habitat along local riparian corridors. Figure 4 shows the 50-year regional groundwater decline.

It was increasingly recognized that ground water could no longer be relied on as the sole source for municipal supply. Ground-water levels were declining at an accelerated rate and measurable land subsidence was being documented. In 1984, Tucson Water was one of the first water utilities in the western United States to develop a tertiary wastewater treatment and delivery system. This system produces reclaimed sewage water for urban irrigation and industrial use to conserve ground water for higher quality uses and to reduce ground-water pumping. Plans also were in place to utilize imported Colorado River water via the Central Arizona Project canal by 1992 to ensure the community would have a renewable source of supply to supply future growth.

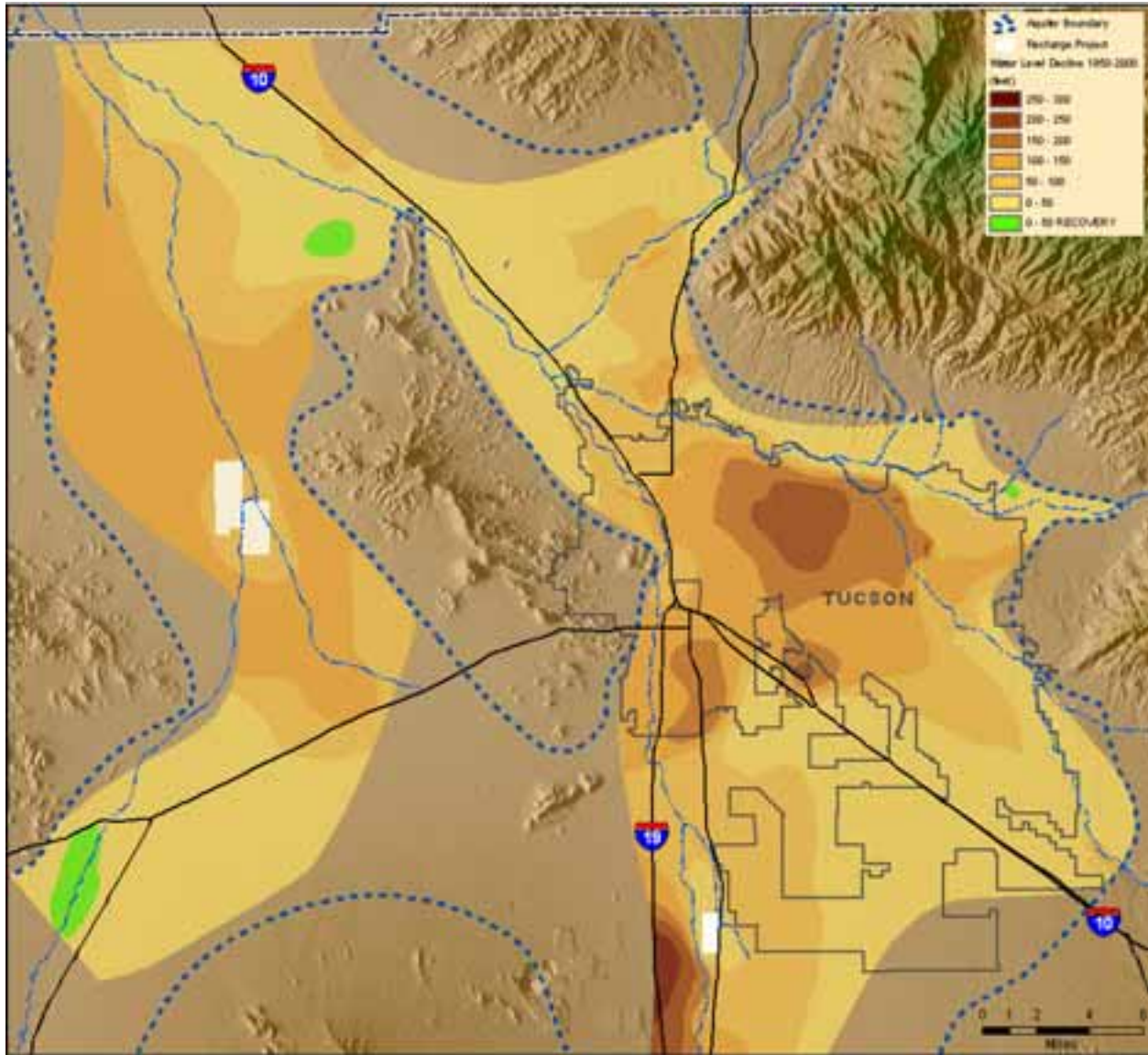


Figure 4. 50-year Groundwater Decline (1950-2000)

Tucson Water began deliveries of the surface water blended with groundwater in 2001. Colorado River water is diverted from the Central Arizona Project (CAP) and is conveyed to 330 acres of water-spreading infiltration basins at CAVSARP. Colorado River water infiltrates through the basin bottoms and percolates downward through hundreds of feet of sediments to the water table. The percolating water benefits from natural filtration and treatment until it recharges the aquifer and mixes with ground water. Recovery wells located nearby extract the blend of Colorado River water and ground water for municipal supply. CAVSARP is designed to deliver about 60,000 acre-feet of blended water to customers per year. The facility has allowed Tucson Water to cut back on ground-water pumping in the metropolitan area and to reduce the community's dependence on ground water for municipal supply. As a result of both artificial recharge and a decreased dependency of groundwater withdrawal from the central wellfield, water levels recently began to recover on a regional level (Figure 5).

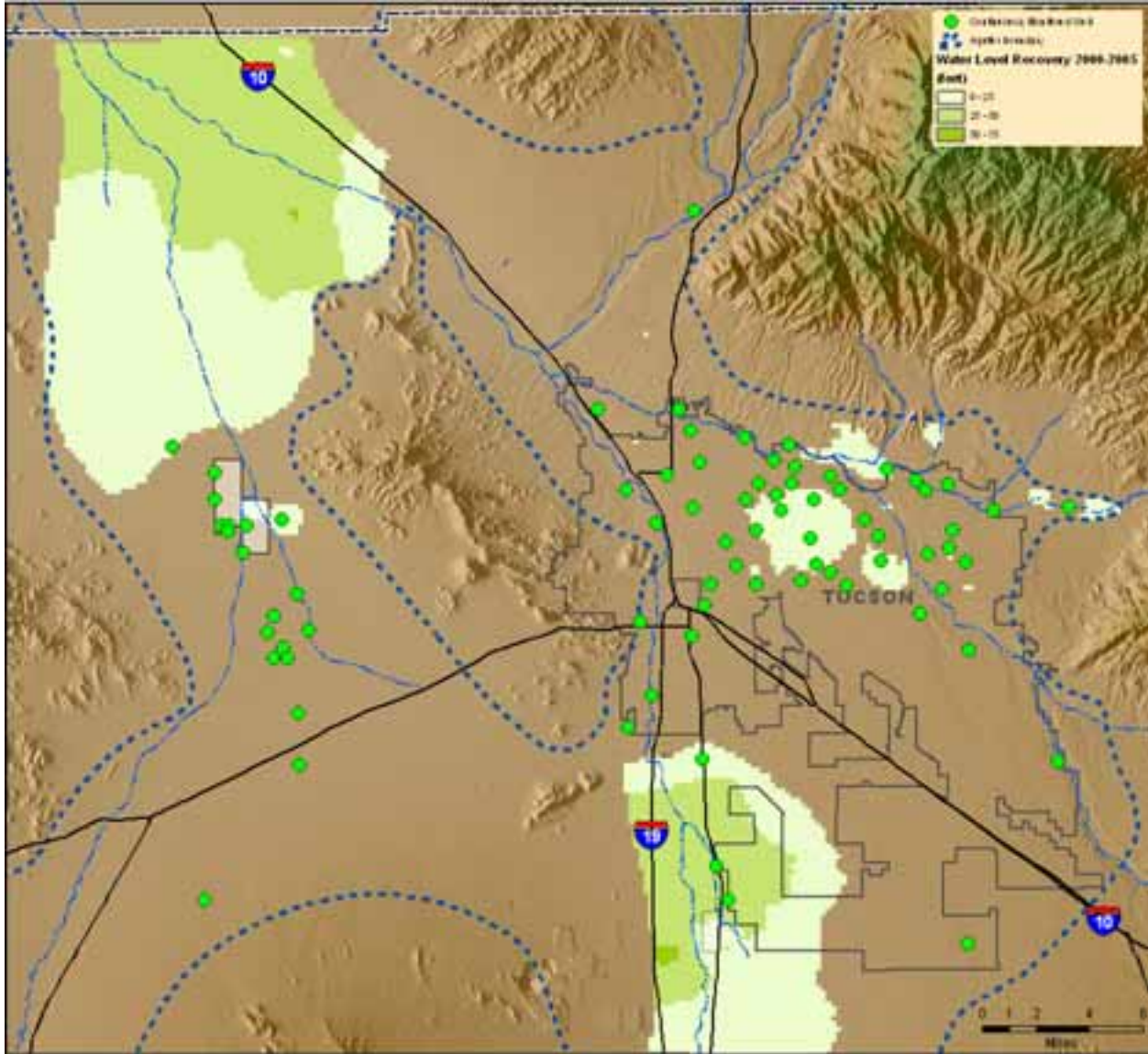


Figure 5. 5-year Groundwater Level Recovery (2000-2005)

Monitoring Program

Beginning in the early 1980s Tucson Water Hydrology staff began a continuous groundwater monitoring program to study the effects of regional groundwater decline, aquifer compaction, seasonal changes in regional pumpage regimes and seasonal changes in shallow groundwater levels near intermittent washes. More recently, continuous monitoring has been expanded to monitor local groundwater responses to artificial surface water recharge, as well as regional groundwater recovery in the central wellfield (see Figure 3). Today, Hydrology staff has equipped more than 75 wells with vibrating wire pressure transducers with either of two brands of continuous recording dataloggers: Campbell Scientific or Geokon. Water level measurements are collected on the hour either hourly or every four hours. However, only the one measurement per day is imported into the water level table in the enterprise Hydrology database.

PROBLEM

The primary difficulty with managing thousands of hourly or daily measurements that are downloaded from dataloggers is the tedious process of quality control. Figure 6 illustrates raw continuous water level data collected from a monitor well that exhibits interference from either its own pumpage, or a nearby well. Graphing the hydrograph in MS Excel (Figure 7) shows a 40ft difference between static and pumping water levels.

```
737, 2005, 257, 1600, 0, 3.04, 39.3, 335.06, 19.1, 5190
737, 2005, 258, 0000, 0, 2.97, 21.1, 335.02, 19.1, 54906
737, 2005, 258, 0800, 0, 2.96, 21.1, 335.96, 19.1, 25687
737, 2005, 258, 1600, 0, 3.03, 39.6, 335.93, 19.1, 21036
737, 2005, 259, 0000, 0, 2.97, 21.5, 375.95, 19.1, 45124 = Pumping Water Level
737, 2005, 259, 0800, 0, 2.96, 23.0, 335.91, 19.1, 60778
737, 2005, 259, 1600, 0, 3.02, 38.8, 335.89, 19.1, 39925
:
```

Figure 6. Raw Datalogger Data, Depth Measurements are Bold.

It is inherently difficult for staff to screen out undesirable pumping water levels from a long list of comma-separated values from the raw text files. Traditional graphing software is capable of visually graphing data, but individual data points cannot be physically manipulated from the graph. In the past, Visual Basic for Applications (VBA) software development within Microsoft Access was developed by Hydrology, Database Management Unit staff to import raw datalogger files. This application graphs data alongside the table of values. However, when a bad data value occurred in the graph, the user was required to visually match errant water level values from the graph with the corresponding value in the table. The user had to potentially scroll through the entire data table to select the appropriate record to flag.

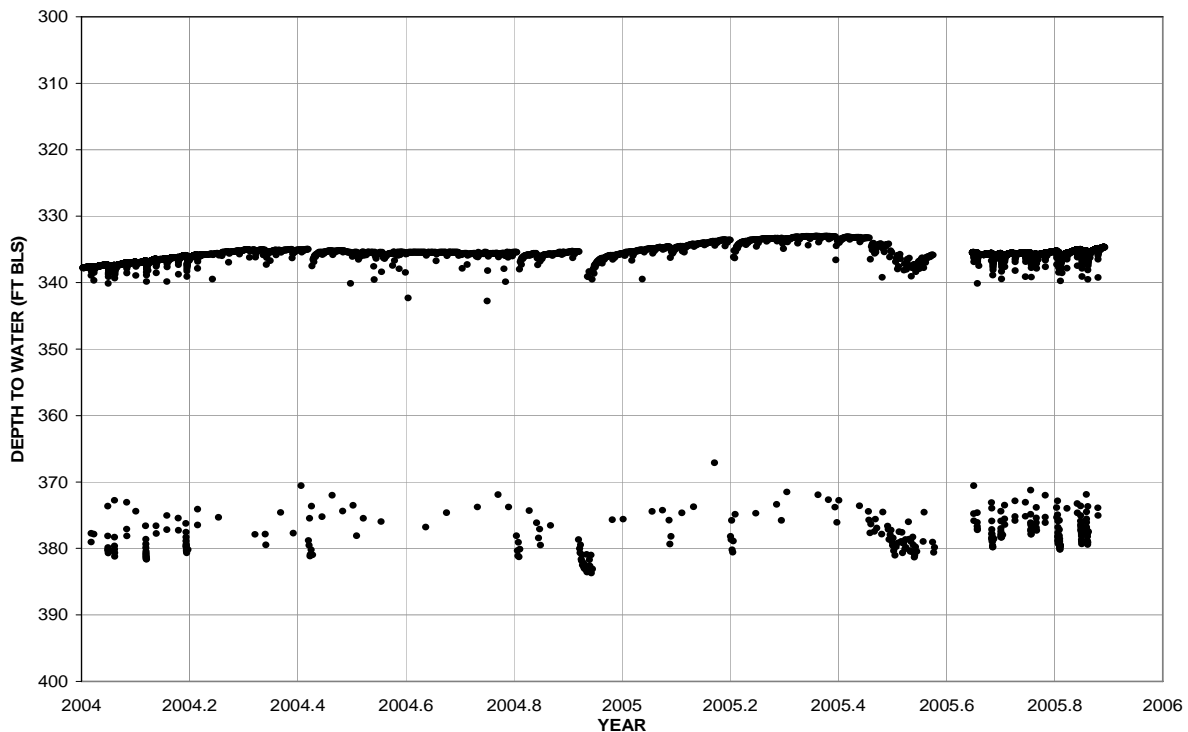


Figure 7. Water Level Hydrograph Showing Pumping Water Levels as Graphed in MS Excel.

This tedious nature of the QA/QC procedures in such a case often results in less than timely data management. Depending on the frequency and duration of measurements, large datasets may take months to clean. It therefore became the goal of the Hydrology section's spatial database developer to create a user interface that will allow the visual management of continuous records by manually selecting suspect data points in a hydrograph to manipulate the physical data record.

A GIS application was developed for ArcGIS desktop that takes advantage of feature-linked data within X,Y coordinate space. The premise of the GIS tool is to import comma-delimited data from a text file from the ArcMap interface into a temporary database table. The records in this table are then manifested in Cartesian space as point event features. To produce a hydrograph in Cartesian map space, in general the date and hour of each measurement is the X coordinate and the depth-to-water is the Y coordinate. The point event features, representing individual water level records, then behave in ArcMap as a regular geographic dataset, such that any point can be identified, queried, rendered, updated or selected dynamically between the ArcMap interface and the temporary table.

METHODS

A demo application was written and documented so that users can change the code to suite their particular data needs, input file parameters or order of parameter fields. Although an analogous tool was developed for Tucson Water Hydrology staff to analyze groundwater level measurements affected by pumpage, it can be modified to handle continuous data from any discipline. Both the Campbell and Geokon dataloggers are set up to store the well ID, year, Julian day, hour and depth. Other data, like temperature and voltage, are ignored. It should be noted that various brands of dataloggers may store data in different columns or may store the date and time information in different forms.

The general process starts by importing a comma-delimited text file into a temporary, event-source table (**temp_xy**) in a Microsoft (MS) Access database (**Demo1.MDB**). While the data are imported, an X and Y coordinate are calculated for each record such that the fractional or decimal year is the X parameter and the negative of depth is the Y parameter. The event table's records are then manifested in the ArcMap coordinate space as a point event layer such that the table is dynamically linked to the map. Once data are rendered in map space, a tool is used to select any points that represent undesirable data. An empty Boolean field in the event table of the corresponding point of the event features is updated to true for each selected point. In the case for Tucson Water Hydrology, the Boolean field represents a pumping water level. Once all undesirable table data have been selected, the records are loaded into the enterprise **WaterLevel** table. For hydrological research, it is important for pumping water levels to be stored in the database for pumpage efficiency and transmissivity calculations. However, pumping water levels must not be incorporated in static water level hydrographs or external data requests. The application uses ArcObject to manipulate map data and Active-X Data Objects (ADO) recordsets for Access database (DB) transactions. Note that ADO transactions associated with an Access DB take place by

the Admin user. Therefore any table to be accessed by ADO needs to have the security set so that Admin has administrative privileges.

RESULTS

Figure 8 illustrates the layout of the Pumping Water Level (PWL) toolbar that was developed by Tucson Water Hydrology staff. A diagram of the processes is provided in Appendix 1 and the application code is provided in Appendix 2. Specific descriptions of the toolbar controls and associated procedures are given as follows:

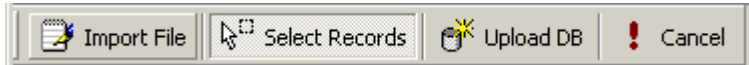


Figure 8. Pumping Water Level (PWL) Toolbar

1. Import file button (Appendix 2, code Domain 1)

-OPEN FILE DIALOG

The first step of the import process uses the Open File Dialog control to navigate to and select the datalogger text file that is to be analyzed. However, many Microsoft controls are not available for use in ESRI software forms. This is true for the Open File Dialog control. Therefore, an API call to the Open File Dialog is performed programmatically to store the name and location of the file. The results are the same, but this unfortunately complicates the code. Once the file and directory of the import file are stored, the path is used to format the ADO connection properties. The file is not technically open, but the directory and file name are injected into the ADO connection object and recordset SQL statement properties, respectively.

The OLE DB ADO objects are used throughout the application, but the connection object for importing the fields from the text file is a unique extension of the MS Access file import drivers. Although a text file is not an MS Access database, the connection string still uses the *Microsoft Jet 4 Engine* as the Provider property but the *file directory* as the Data Source property. This works because MS Access drivers are capable of importing text files. In case of a genuine MS Access transaction, the connection object Data Source property would be the database file path and the SQL query string would select from a table in the DB. In this unique case, the SQL query string selects from the text file directly.

ADO COLUMN	0	1	2	3	4	5	6	7	8
	<i>constant</i>	YEAR	JULIAN DAY	HOUR	<i>seconds</i>	DTW	<i>temp</i>	<i>volts tot</i>	WELLNUM
CAMPBELL	3	2005	312	1600	0.25	404.9	29.07	14.32	2683
	WELLNUM	YEAR	JULIAN DAY	HOUR	<i>seconds</i>	<i>volts solar</i>	<i>volts bat</i>	DTW	<i>temp</i>
GEOKON	331	2003	365	2300	0	2.88	9.74	337.797	29.1

Figure 9. ADO Fields for Both Brands of Dataloggers.

Since Tucson Water uses both Campbell and Geokon dataloggers, a decision about the make of the datalogger is performed when the first record is imported. Although both dataloggers store the date and time information in the same ADO fields (Figure 9), the well identifier and the water level are stored in different fields. Campbell dataloggers store the well ID in ADO field 8 and the depth value in ADO field 5, the Geokon dataloggers store the *WellNum* in ADO field 0 and the depth to water value (*DTW*) in

ADO field 7. The decision about datalogger type is based on ADO field 0. The import file is from a Campbell if the value in the first column (ADO field 0) equals "3". Otherwise the first column represents the *WellNum* of a Geokon record. This particular comparison works for Tucson Water only because the well with ID = 3 is backfilled (i.e. well #3 can never be equipped with a datalogger).

-CONVERT DATA AND LOAD EVENT TABLE

Two ADO recordsets must be instantiated for importing records into the event table. One recordset enumerates through the rows of data in the text file. The other recordset is declared to first delete any existing, temporary records in the event table and then to open the empty table to sequentially add new records from the first recordset. The transactions of records out of the text file and into the event table occur simultaneously on a record-by-record basis.

temp_xy	
RecordNum	
WellNum	
X	
Y	
DTW	
PWL	
Date	
Hour	

Figure 10 shows the schema of the temporary event table, **temp_xy** in the **Demo1.MDB** database. The *RecordNum* field is used to uniquely identify each water level record and will be used as the foreign key between the table records and the GIS event layer attributes. The *WellNum* field stores the unique well identifier. The source field of the well number depends on the datalogger type. The *WellNum* is the primary key for the entire Tucson Water enterprise well DB management system (DBMS).

Because the measurement date/time information is not stored as a Date/Time value, two conversions need to be made from the imported fields. First, the fraction of the year is calculated from the Year, Julian Day and Hour fields (e.g. 6/1/2005 12:00AM = 2005.50000). The fraction of a whole year is used as the X coordinate because a date/time values would not work in Cartesian space. Second, a Date/Time value is calculated for each record. This process is performed in a separate function (Appendix 2, code Domain 5). Both converted date values are stored in the event table for each measurement in the Y and *Date* fields, respectively. Furthermore, both date/time conversions are aware of leap years. Finally, the *Hour* attribute from the import file is copied to the *Hour* field in the event table without any change.

A negative depth measurement is stored as the Y coordinate. The negative of the depth is used simply so that the hydrographs plots logically on a reverse axis (0 depth or ground level on top) in Cartesian space. The actual depth measurement is also copied to the *DTW* field in the event table.

-GRAPH AND RENDER THE DATA (Appendix 2, code Domain 2)

Graphing the XY data in ArcMap as an XY Event layer occurs in a subroutine so that it can be called when needed. One complication of the code for graphing data is that the objects do not exist and must be instantiated. The rest of the ArcObjects programming in this application accesses map objects that already exist. Therefore all of the properties of the objects used for rendering data need to be set.

Dynamically rendering data from an Access table into ArcMap requires in general, three new objects: a GeoFeatureLayer, a UniqueValueRenderer and a WorkspaceFactory. In order to display the selected (undesirable) records differently, the points are rendered directly from the **temp_xy** event table such that any selected record will appear as a large, red dot. By default the records are rendered as small black dots (Figure 11). In other words, if the *PWL* value of a given record is FALSE, a small, black dot is rendered. Conversely, a large, red dot is rendered for any record where *PWL* = TRUE.

A new GeoFeatureLayer is instantiated to be filled by the **temp_xy** records within the WorkspaceFactory code and to accept the UniqueValueRenderer formatting. The UniqueValueRenderer object simply formats the GeoFeatureLayer point symbology so that the layer is categorized by the value of *PWL* in the event table (i.e. TRUE = large, red; FALSE = small, black).

A new AccessWorkspaceFactory object is instantiated and the properties are set such that the **Demo1** database is opened as the feature workspace and the **temp_xy** is opened as the table source of XY data to be displayed as a point event feature class in the ArcMap interface.

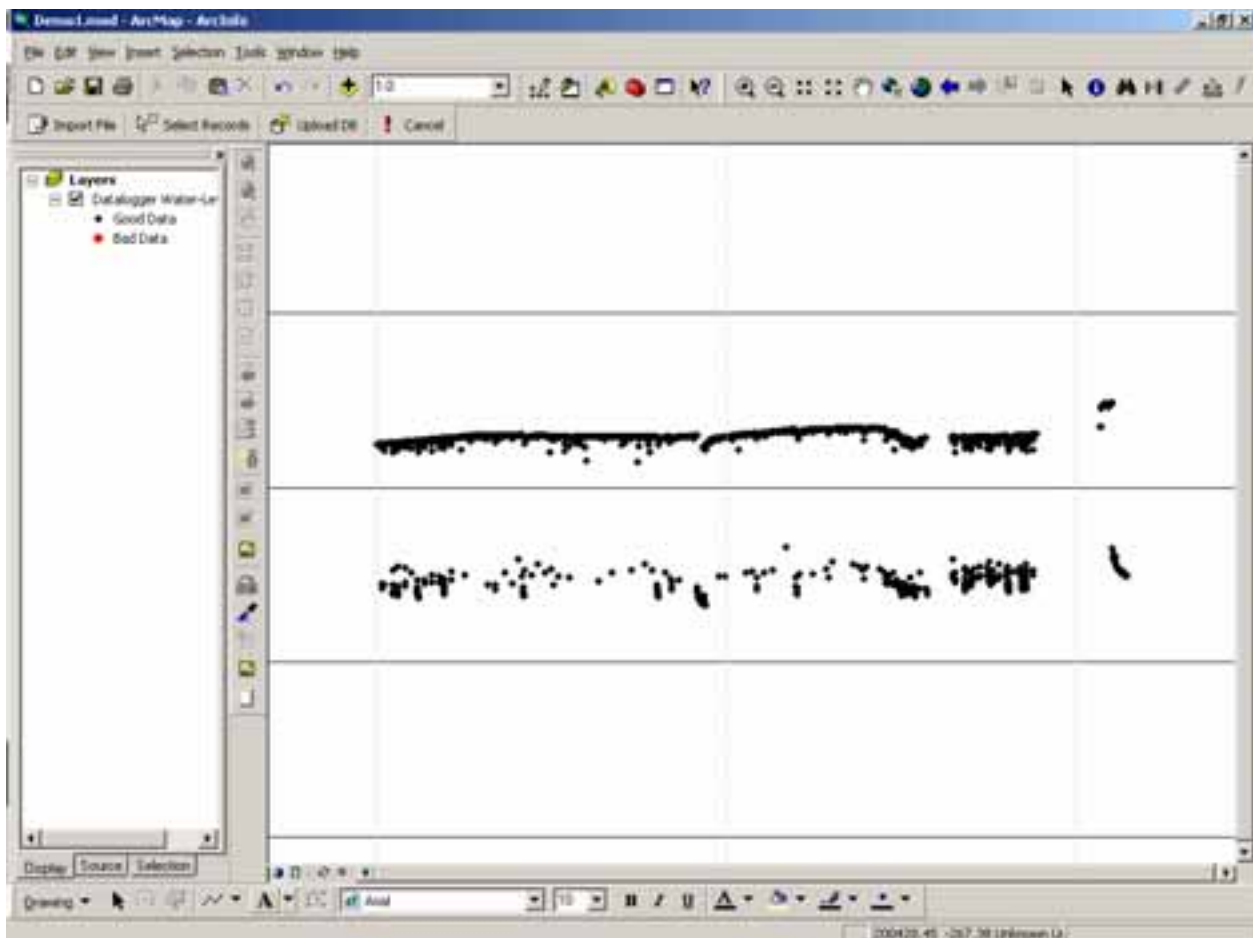


Figure 11. Raw Datalogger Data in the ArcMap Interface

2. Select records tool (*Appendix 2, code Domain 3*)

The Select Records tool is used by the user to lasso any points that represent outliers (pumping water levels) of the hydrograph data. The tool uses a SpatialFilter object within a geometric envelope (the lasso rectangle) to enter any point feature within the envelope into a FeatureCursor object. For each point in the feature cursor, an enumeration process then occurs such that an ADO recordset, with a single record, is sequentially created for each point. The *RecordNum* value of each successive feature in the cursor is used as a query filter for a second, single-record ADO recordset. In other words, for every point that is selected on the map, the matching record in the **temp_xy** table is selected by matching the *RecordNum* fields; the *PWL* field is updated to TRUE. The tool can be use multiple times until all bad data are selected. The previous graphing routine is called each time points are selected in order to re-render selected data as large, red dots in the ArcMap interface (Figure 12).

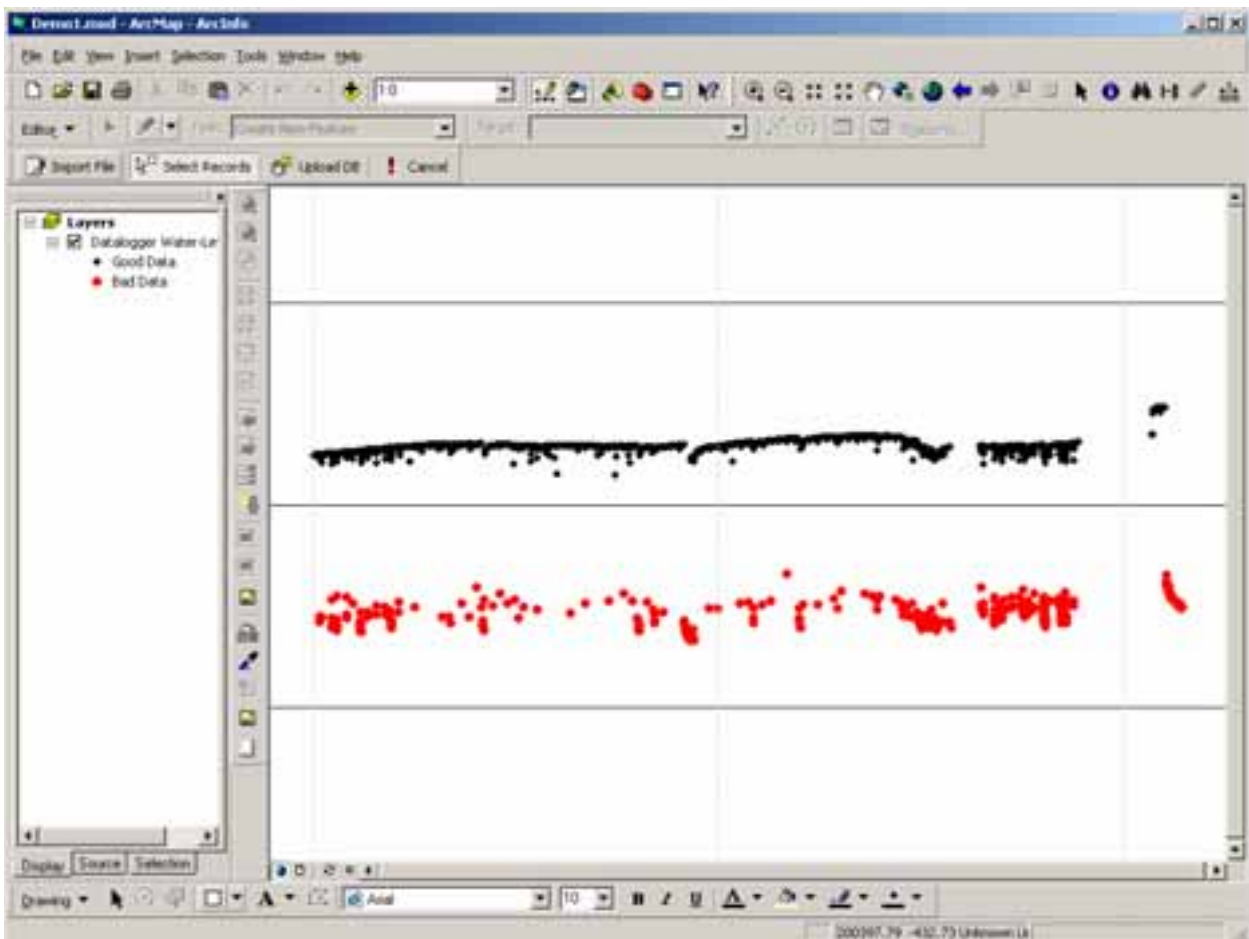


Figure 12. Selected, Pumping WaterLevel Events Rendered as Red in the ArcMap Interface

3. Upload DB button (*Appendix 2, code Domain 4*)

-LOAD TEMP DATA INTO PERMANENT TABLE

The process of loading the corrected records from the event table into the enterprise water level table is similar to the process of loading the records from the import file into the event table. Two ADO recordsets are declared: one to select records out of the

temp_xy table; the other to open the **WaterLevel** table to sequentially append the records from the **temp_xy** table. First, a decision is made by the user about the amount of data to enter into the **WaterLevel** table. In the case of Tucson Water, only the 8:00AM reading are entered into the **WaterLevel** table. Hourly readings are generally unnecessary for hydrologic evaluation and also take up large amounts of disk space. However, on occasion as dataloggers batteries begin to fail, the dataloggers may collect readings on uneven time scales, or record hourly but on a minute other than minute 0. In these cases, the application will allow the user to import only the 8:00AM readings or all of the records. This affects the first ADO recordset's SQL query. If only 8:00AM readings are desired, the SQL string selects from the **temp_xy** table where the *Hour* field equals 800 (8:00AM). Otherwise all of the records are queried from the **temp_xy** table (i.e. there is no WHERE clause in the SQL).

From the **temp_xy** table, only the *WellNum*, *Date*, *DTW* and the Boolean *PWL* field values are entered into the **WaterLevel** schema. This interaction occurs on a record-by-record basis. For every record in **temp_xy**, a new record is added to the **WaterLevel** table until *EOF* (End of File). The other fields in the **temp_xy** table are only temporary and not imported. These fields are used only for graphing (X,Y) and for cross-reference (*RecordNum*). Several other fields in the **WaterLevel** table are also updated. Text is written to the *Comments* field to indicate that the records were collected from a datalogger, and also notes the brand of datalogger. The *EnterDate* field is simply updated by the *Now()* function to yield the date and time each record is appended. Finally, the logged-in, MS Windows user is written to the *EnteredBy* fields of each record. The code to acquire the Windows user's ID from the PC also uses API calls. Again, this complicates the code, but prevents a user of the application from needing to log into a database.

-ARCHIVE AND RENAME THE IMPORTED FILE

The final step after the **temp_xy** data are loaded into the **WaterLevel** table is to rename and archive the imported file. As the **temp_xy** table ADO recordset is enumerated, two global variables are updated to store the date of the first and last datalogger reading. A File System Object (FSO) is created in order to extract the parts of the import file path as strings, e.g. file name, file extension and directory, needed to rename and move the file to an archive folder. The first date and last date are concatenated onto the end of the file name, between the file directory and the file extension. For example if the first reading is on 03/04/2004 and the last reading is on 06/31/2005, the resulting file will be moved and renamed: <path>\archive\<filename>_03042004_06312005.TXT.

DISCUSSION

A Demo application using this technology was developed for both a presentation at the 2006 ESRI User Conference but also for adaptation by any users who could benefit from a free application solution to a significant data management problem.

The Demo

The project demo will be provided to ESRI for users to download from the EDN website. The code is fully contained in an MXD document and is used in conjunction with the **Demo1** database that is required to be located within the same workspace folder.

This demo application and other versions of this application have a flaw with rendering the point features. Although the point feature rendering works fine when stepping through the code in the debugger, the map does not always refresh properly. Even though there are several repeated programmatic calls to the map view refresh event, the rendering of the large, red points does not always occur after refreshing. The user must frequently manually refresh the map.

Another improvement to the code can be made to the way in which the database is virtually connected to the ArcMap interface. The existing application uses a somewhat static connection to the DB. The application must open the database and query the records many, if not thousands of times during the application procedures. However, an (Open Database Connectivity) ODBC connection to the database would increase the efficiency of the database transactions and feature rendering. This will be important for large datasets. An ODBC connection to the **Demo1** DB would allow a more direct, dynamic link to the database table. This would eliminate many of the ADO connections and recordset operations. In general, instead of a new point event feature class being created from the **temp_xy** table each time the data are rendered, the **temp_xy** table would be manually rendered only once. Since the database is always open when using an ODBC connection, any changes to the data table would automatically manifest in ArcMap. In other words, instead of needing to create and format the an ArcMap point layer from the table (e.g. using the Display X,Y tool) each time the map is refreshed, the table is always connected and formatted directly in ArcMap, with or without attributes. The empty point event feature class is always reflecting the attributes in the **temp_xy** table. Changing the table data (i.e. importing a new datalogger file) will automatically change the features. The use of an ODBC connection would also potentially rectify the aforementioned flaw with refreshing of the rendered features.

Modifying the Code

Included in Appendix 1 is a diagram of the process for importing, managing and loading data into a DBMS. Spatial database developers with a basic training in VB programming or VBA application development/automation and ADO can alter the database interactions to analyze a wide range of continuous data. In general the adaptation of the import code would require changing the ADO connection string Data Source property, the SQL statement and the ADO field references to refer to the formatting of the import text file. This is also true for the similar code for uploading the data into an enterprise DBMS.

There are two levels of ArcObjects programming skills needed to modify the ArcMap objects. Those with a cursory exposure to ArcObjects programming should be able to understand the basic interactions used for interrogating the existing map objects (e.g. document, map and layer properties). However, an advanced knowledge of the multi-interface, ArcMap object model will be needed to re-develop or modify the code used to render new layers and to format the WorkspaceFactory and feature class objects. The modification of the ArcObjects code may not be necessary, however.

The code for this application is provided in Appendix 2 for reference. The code not only provides examples of VBA interactions between ArcObjects feature cursors enumerations and ADO table records, but also includes useful code for date and time conversions as well as File System Objects code for moving and renaming files. These VBA universal code routines can be used for any application.

Modifying the Axes

The X and Y axes and labels in the ArcMap interface are simply graphics. Axis lines were added using an exact X,Y position and width in the line graphic properties. Users are able to add or change axis lines either by copy-and-paste or by manually drawing lines, keeping in mind that the X axis (year) is multiplied by 100 to spread out the points and the Y axis is negative. For example January 1, 1990 would appear as 199000. The length of the axis can also be manually set so that the length is the number of years times 100. Therefore, to represent 20 years, the width will be 2000.

Versions

There are several versions of this application being developed by Hydrology staff. The version described in this demo is used as a dataloader. However, a second version accesses existing water level data from the enterprise DBMS and loads the complete water level record for any well. The goal of this particular application is to manage existing water level records. The user can choose to view the data on two different scales. For continuous data, similar to the case of this demo, the X coordinates are multiplied by 100. If an annual display is desired, the X coordinate is only multiplied by 10. Annual readings do not need to be spread out like continuous data. Furthermore, this application makes use of a Well Database Object Model. Instead of running ADO transactions within the application, water level recordsets are returned by declaring a well object, setting the object to point to the chosen well, and calling a method to return the non-pumping water levels. With this application, water level records can be rendered to be symbolized either by the agency that collected the water level, or by the project/permit for which the water level was collected.

A third version of this application is being developed for managing SCADA data. Many of the CAVSARP recovery wells are equipped with SCADA systems. Aside from managing water levels, the application will allow the user to select different water parameters to analyze. Currently, Hydrology staff collects continuous runtime, water levels, flow, volume and power consumption for wells that are monitored through the SCADA system. This third version may be extended to manage and import the different water quality parameters (dissolved oxygen, TDS, Ph and conductivity) collected from continuous surface water quality recorders.

CONCLUSIONS

The results of the preliminary deployment of this tool for Hydrology staff have been positive. Not only does it increase the efficiency of data management, it also gives field staff an opportunity to work with ESRI GIS software. Unfortunately, GIS skills can be lost if staff do not regularly use ArcMap. Datalogger data are downloaded in the field by Hydrology technician staff and stored on PDAs (Personal Digital Assistants). Once in the office, field staff can access the datalogger files directly from their docked PDAs. This eliminates the buildup of raw files within the Hydrology file servers. Since the application copies the files and renames files according to the start and end dates of the records, the need for manual file management is eliminated. Because the raw files always have the same name when downloaded, the concern for confusing or overwriting files that have not yet been uploaded may be eliminated.

ACKNOWLEDGEMENTS

The author would like to give his appreciation to the entire Tucson Water Hydrology Staff, as well as colleagues in the Information Services and Mapping and GIS sections.

APPENDIX 1

Process Diagram

IMPORT FILE (CODE DOMAIN 1)

open

POINT TO TRANSDUCER DATA TEXT (*.txt) FILE TO LOAD INTO ADO RECORDSET
USES API CALL FOR OPEN FILE DIALOG BOX

?

DETERMINE TRANSDUCER TYPE

If column 0 = 3 then CAMPBELL, else GEOKON

ADO COLUMN	0	1	2	3	4	5	6	7	8
	CONSTANT	YEAR	JULIAN DAY	HOUR	seconds	DTW	temp	volts bat+solar	WELLNUM
CAMPBELL	3	2005	312	1600	0.25	404.9	29.07	14.32	2683
	WELLNUM	YEAR	JULIAN DAY	HOUR	seconds	volts solar	volts bat	DTW	temp
GEOKON	331	2003	365	2300	0	2.88	9.74	337.7971	29.1

initialize

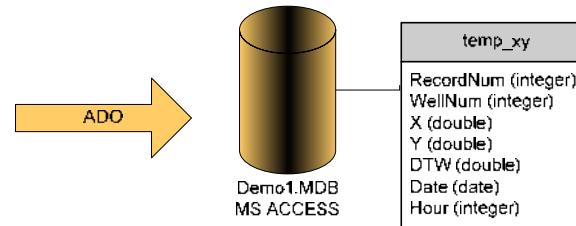
INITIALIZE TEXT FILE AND **temp_xy** TABLE RECORDSETS

CLEAR **temp_xy** TABLE IN **Demo1** DATABASE (CONNECTION OBJECT COMMAND)

AN ADO RECORDSET IS DECLARED TO ACCEPT RECORDS FROM TEXT FILE

A SECOND ADO RECORDSET IS DECLARED TO SEQUENTIALLY FILL **temp_xy** WITH FIRST, CONVERTED ADO RECORDSET

3, 2005, 312, 0000, 0.28, 120.4, 23.74, 14.32, 2034
 3, 2005, 312, 0400, 0.28, 120.3, 23.74, 14.32, 2034
 3, 2005, 312, 0800, 0.28, 120.4, 23.74, 14.32, 2034
 3, 2005, 312, 1200, 0.28, 120.5, 23.74, 14.32, 2034
 3, 2005, 312, 0600, 0.28, 120.4, 23.74, 14.32, 2034
 3, 2005, 312, 2000, 0.28, 120.4, 23.74, 14.32, 2034



load

CONVERT AND LOAD TRANSDUCER PARAMETERS INTO **temp_xy** TABLE SUCH THAT:

RECORDNUM = temporary sequential integer

WELLNUM = **WELLNUM**

X = convert **YEAR, JULIANDAY** and **HOUR** into decimal date * 100 (CODE DOMAIN 5)

Y = **-DTW** (negative depth to water)

DTW = **DTW** (depth to water)

DATE = convert **YEAR, JULIANDAY** and **HOUR** into serial date value (MM/DD/YY HH:MM) (CODE DOMAIN 5)

HOUR = **HOUR**

graph

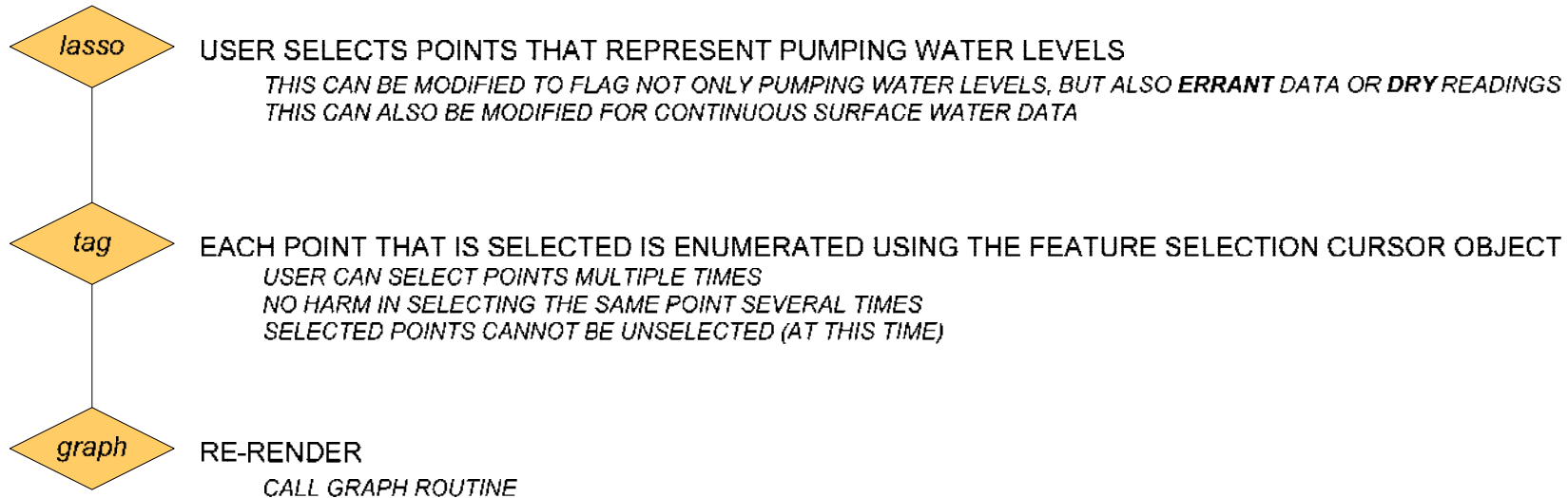
DISPLAY AND RENDER RECORDS FROM **temp_xy** AS EVENTS IN CARTESIAN SPACE (DOMAIN 2)

A WORKSPACE FACTORY OBJECT IS ENSTANTIATED TO ACCEPT THE **temp_xy** TABLE AS AN XYEVENTSOURCE USING **x** AND **y** AS COORDINATES

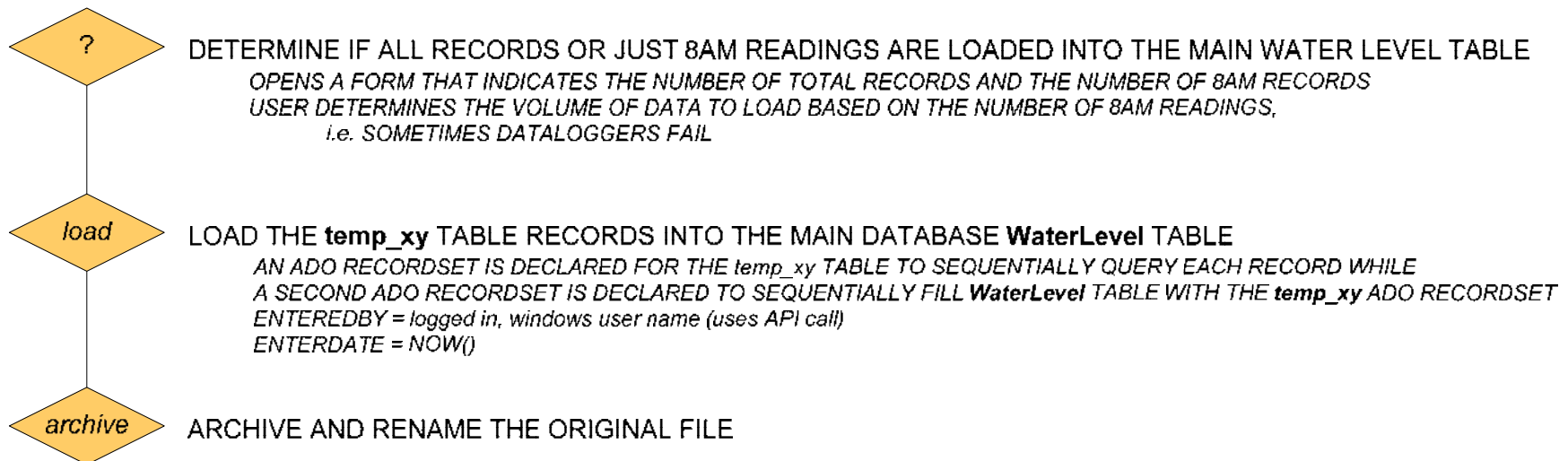
THE **temp_xy** TABLE IS DYNAMICALLY MANIFESTED AS AN XY-SCATTERPLOT IN THE ARCMAP INTERFACE

A UNIQUEVALUE RENDERER OBJECT IS ENSTANTIATED SO THAT "GOOD DATA" ARE RENDERED AS SMALL **BLACK** DOTS; "BAD DATA" ARE EVENTUALLY RENDERED AS LARGE **RED** DOTS

SELECT RECORDS (CODE DOMAIN 3)



UPLOAD DB (CODE DOMAIN 4)



APPENDIX 2

Code

```

' *****
' MICHAEL F LIBERTI
' CITY OF TUCSON
' TUCSON WATER DEPARTMENT
' 06/23/2006
' 2006 ESRI UC PAPER UC1290
' DEMO1.MXD
' *****
Option Explicit

' Global variables
Public loggertype As String
Public FilePath As String
Public StartDate As String 'for renaming the import file
Public EndDate As String 'for renaming the import file

' Declaration for GetLoginUserName()
' FROM GOOGLE GROUPS
' API calls written within the advapi32.DLL to determine the windows user
Private Declare Function GetUserName Lib "advapi32.dll" _
    Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long

' Declaration for Open File Dialog box
Private Declare Function GetSaveFileName Lib "comdlg32.dll" _
    Alias "GetSaveFileNameA" (pOpenfilename As OpenFilename) As Long

' API calls to Open File Dialog box
Private Type OpenFilename
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustFilter As Long
    iFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type

' API calls to Open File Dialog box
Private Enum OFNFlagsEnum
    OFN_ALLOWMULTISELECT = &H200
    OFN_CREATEPROMPT = &H2000
    OFN_ENABLEHOOK = &H20
    OFN_ENABLETEMPLATE = &H40
    OFN_ENABLETEMPLATEHANDLE = &H80
    OFN_EXPLORER = &H80000
    OFN_EXTENSIONDIFFERENT = &H400
    OFN_FILEMUSTEXIST = &H1000
    OFN_HIDEREADONLY = &H4
    OFN_LONGNAMES = &H200000
    OFN_NOCHANGEDIR = &H8
    OFN_NODEREFERENCELINKS = &H100000
    OFN_NOLONGNAMES = &H40000
    OFN_NONETWORKBUTTON = &H20000
    OFN_NOREADONLYRETURN = &H8000
    OFN_NOTESTFILECREATE = &H10000
    OFN_NOVALIDATE = &H100
    OFN_OVERWRITEPROMPT = &H2
    OFN_PATHMUSTEXIST = &H800

```

```

    OFN_READONLY = &H1
    OFN_SHAREAWARE = &H4000
    OFN_SHAREFALLTHROUGH = 2
    OFN_SHARENOWARN = 1
    OFN_SHAREWARN = 0
    OFN_SHOWHELP = &H10
End Enum

' Uses the API for the open file dialog
' *****
' Show the common dialog to select a file to save. Returns the path of the
' selected file or a null string if the dialog is canceled
' Parameters:
' - sFilter is used to specify what type(s) of files will be shown
' - sDefExt is the default extension associated to a file name if no one is
' specified by the user
' - sInitDir is the directory that will be open when the dialog is shown
' - lFlag is a combination of Flags for the dialog. Look at the Common
' Dialogs' Help for more informations
' - hParent is the handle of the parent form
' *****
Function ShowSaveFileDialog(ByVal sFilter As String, Optional ByVal sDefExt As _
    String, Optional ByVal sInitDir As String, Optional ByVal lFlags As Long, _
    Optional ByVal hParent As Long) As String
    Dim OFN As OpenFilename

    On Error Resume Next
    ' set the values for the OpenFileName struct
    With OFN
        .lStructSize = Len(OFN)
        .hwndOwner = hParent
        .lpstrFilter = Replace(sFilter, "|", vbNullChar) & vbNullChar
        .lpstrFile = Space$(255) & vbNullChar & vbNullChar
        .nMaxFile = Len(.lpstrFile)
        .flags = lFlags
        .lpstrInitialDir = sInitDir
        .lpstrDefExt = sDefExt
    End With

    ' show the dialog
    If GetSaveFileName(OFN) Then
        ' extract the selected file path and return string to complete the function
        ShowSaveFileDialog = left$(OFN.lpstrFile, InStr(OFN.lpstrFile, _
            vbNullChar) - 1)
    End If
End Function

' function returns Windows User as string
' MUST have API declaration at beginning of code
Function GetLoginUserName() As String
    Dim UserName As String
    Dim NameLen As Long

    NameLen = 2048
    UserName = String(NameLen, 0)
    GetUserName UserName, NameLen
    GetLoginUserName = left(UserName, NameLen)
End Function

' TOOLBAR BUTTON
Private Sub cmdCancel_Click()
    MxDocument_OpenDocument
End Sub

' DOMAIN 1 -----
' OPEN FILE
' TOOLBAR BUTTON
Private Sub cmdOpenFile_Click()
    On Error GoTo Err_handler

    Dim sFilter As String

```

```

sFilter = "Text files (*.txt)|*.txt|All files (*.*)|*.*"
' calls open file dialog and returns the filepath
FilePath = ShowSaveFileDialog(sFilter, "txt", "C:\")

' load data from datalogger text file into an ADO recordset
Dim incn As New ADOConnection
Dim inrs As New ADORecordSet
' OLE DB connection to the text file folder
incn.Open ("Provider = Microsoft.Jet.OLEDB.4.0;Data Source=" & CurDir(FilePath) & _
"; Extended Properties=""text;HDR=NO;FMT=Delimited""")
With inrs
.ActiveConnection = incn
.CursorType = adOpenDynamic
.LockType = adLockOptimistic
End With
inrs.Open "SELECT * FROM " & FilePath & ""

' convert parameters and load datalogger recordset into temp_XY table
Dim outcn As New ADOConnection
Dim outrs As New ADORecordSet
With outcn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.Properties("Jet OLEDB:System Database") = "c:\WINNT\system32\system.mdw"
.Open CurDir & "\demol.mdb" ' userid, password
End With
With outrs
.ActiveConnection = outcn
.CursorType = adOpenDynamic
.LockType = adLockOptimistic
End With
outcn.Execute "DELETE FROM temp_xy" ' deletes existing data in the temp_XY table
outrs.Open "SELECT * FROM temp_xy" ' opens the empty temp_xy table for appending

' determine if the the data was collected from a Geokon or Campbell datalogger
' Campbell datalogger data have a 3 in field(0)
loggertype = "geokon"
inrs.MoveFirst
If inrs.Fields(0) = 3 Then
loggertype = "campbell"
End If

' loops through input recordset (text file) and adds to output recordset (temp xy)
Dim decdate As Double
Dim leapyear As Boolean
leapyear = False
Dim outdate As String
Dim outtime As String
Dim outdatetime As Date
Dim i As Integer ' record number counter
i = 1
Do While Not inrs.EOF
If inrs.Fields(1).Value Mod 4 = 0 Then
leapyear = True
End If
outrs.AddNew
'convert input date (day of year) to DATE value (CONVERTTODATE FUNCTION)
If inrs.Fields(3).Value = 2400 Then
outtime = TimeSerial(0, Right((inrs.Fields(3).Value), 2), 0)
Else
outtime = TimeSerial((inrs.Fields(3).Value / 100), _
Right((inrs.Fields(3).Value), 2), 0)
End If
outdate = Me.convtoDate(inrs.Fields(2).Value, inrs.Fields(1).Value)
outdatetime = outdate & " " & outtime
outrs.Fields("date").Value = outdatetime

' convert input date (day of year) to a julian date (decimal date)
If leapyear Then
decdate = (inrs.Fields(1).Value + (inrs.Fields(2).Value / 366) _
+ (inrs.Fields(3).Value / 2400 / 366))
Else

```

```

        decdate = (inrs.Fields(1).Value + (inrs.Fields(2).Value / 365) _
        + (inrs.Fields(3).Value / 2400 / 365))
    End If
    outrs.Fields("x").Value = decdate * 100 ' *100 to spread out points

    ' fill the rest of the fields
    If loggertype = "campbell" Then
        outrs.Fields("wellnum").Value = inrs.Fields(8).Value
        outrs.Fields("y").Value = -(inrs.Fields(5).Value)
        outrs.Fields("dtw").Value = (inrs.Fields(5).Value)
    Else
        outrs.Fields("wellnum").Value = inrs.Fields(0).Value
        outrs.Fields("y").Value = -(inrs.Fields(7).Value)
        outrs.Fields("dtw").Value = (inrs.Fields(7).Value)
    End If
    outrs.Fields("recordnum").Value = i
    outrs.Fields("Hour").Value = inrs.Fields(3).Value
    i = i + 1
    outrs.Update
    outrs.MoveNext
    inrs.MoveNext
Loop
inrs.Close
incn.Close
outrs.Close
outcn.Close
GraphData

Exit Sub
Err_handler:
    MsgBox "Invalid input file", vbCritical, "Error"
End Sub
' END DOMAIN 1-----

' DOMAIN 2 -----
' GRAPHING SUB
Public Sub GraphData()
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Set pMxDoc = ThisDocument
    Set pMap = pMxDoc.FocusMap
    Dim pActiveView As IActiveView
    Set pActiveView = pMxDoc.ActiveView

    pMap.ClearLayers
    pActiveView.Clear
    pActiveView.Refresh
    pMxDoc.UpdateContents

    Dim pFeatLayer As IGeoFeatureLayer
    Set pFeatLayer = New FeatureLayer

    ' unique value renderer
    Dim pSimpleMarkerSym As ISimpleMarkerSymbol
    Dim pUVRenderer As IUniqueValueRenderer
    Dim pColor As IRgbColor

    Set pUVRenderer = New UniqueValueRenderer
    With pUVRenderer
        .FieldCount = 1
        .Field(0) = "PWL"
        .UseDefaultSymbol = False
    End With

    Set pColor = New RgbColor
    pColor.RGB = RGB(0, 0, 0)
    Set pSimpleMarkerSym = New SimpleMarkerSymbol
    With pSimpleMarkerSym
        .Style = esriSMSCircle
        .Outline = False
        .Color = pColor
    End With

```



```

        .size = 4
    End With
    With pUVRenderer
        .AddValue 0, "", pSimpleMarkerSym
        .Label(0) = "Good Data"
        .Symbol(0) = pSimpleMarkerSym
    End With

    pColor.RGB = RGB(255, 0, 0)
    Set pSimpleMarkerSym = New SimpleMarkerSymbol
    With pSimpleMarkerSym
        .Style = esriMSCircle
        .Outline = False
        .Color = pColor
        .size = 6
    End With
    With pUVRenderer
        .AddValue 1, "", pSimpleMarkerSym
        .Label(1) = "Bad Data"
        .Symbol(1) = pSimpleMarkerSym
    End With

' ASSIGNS THE TEMP TABLE TO BE MANIFESTED AS AN EVENT LAYER
Dim pFact As IWorkspaceFactory
Dim pFeatWS As IFeatureWorkspace
Dim pTable As ITable
Set pFact = New AccessWorkspaceFactory
Set pFeatWS = pFact.OpenFromFile(CurDir & "\demo1.mdb", 0)
Set pTable = pFeatWS.OpenTable("temp_xy")

' ASSIGNS WHICH FIELDS IN THE TEMP TABLE ARE X AND Y
Dim pXYEvent2FieldsProperties As IXEvent2FieldsProperties
Set pXYEvent2FieldsProperties = New XYEvent2FieldsProperties
With pXYEvent2FieldsProperties
    .XFieldName = "x"
    .YFieldName = "y"
End With

' ASSIGNS A GIS EVENT TABLE OBJECT TO THE TEMP TABLE
Dim pDataSet As IDataset
Dim pTableName As IName
Set pDataSet = pTable
Set pTableName = pDataSet.FullName

Dim pXYEventSourceName As IXEventSourceName
Set pXYEventSourceName = New XYEventSourceName
With pXYEventSourceName
    Set .EventProperties = pXYEvent2FieldsProperties
    Set .EventTableName = pTableName
End With

Dim pName As IName
Dim pXYEventSource As IXEventSource
Set pName = pXYEventSourceName
Set pXYEventSource = pName.Open

Set pFeatLayer.FeatureClass = pXYEventSource
Set pFeatLayer.Renderer = pUVRenderer

' ASSIGNS THE LAYER LABEL AND ADD THE EVENT LAYER TO THE MAP
pFeatLayer.Name = "Datalogger Water-Level Data"
pMap.AddLayer pFeatLayer
pActiveView.Refresh
pActiveView.Refresh
pActiveView.Refresh
pActiveView.Refresh
pActiveView.Refresh
End Sub
' END DOMAIN 2 -----
Private Function MxDocument_OpenDocument() As Boolean

```

```

    'removes previous layers
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Set pMxDoc = ThisDocument
    Set pMap = pMxDoc.FocusMap
    pMap.ClearLayers
    pMxDoc.UpdateContents
    pMxDoc.ActiveView.Refresh
End Function

' DOMAIN 3 -----
' SELECT RECORDS
' TOOLBAR BUTTON
Private Sub toolSelect_MouseDown(ByVal button As Long, ByVal shift As Long, _
    ByVal x As Long, ByVal y As Long)
    'SET UP POINTERS TO THE OBJECTS THAT CONTROL THE AREA THAT IS SELECTED
    'WHEN THE CURSOR IS DRAGGED
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    Dim pActiveView As IActiveView
    Set pActiveView = pMxDoc.ActiveView
    Dim pEnvelop As IEnvelope
    Dim pRubberBand As IRubberBand
    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap
    Set pRubberBand = New RubberEnvelope
    Set pEnvelop = pRubberBand.TrackNew(pActiveView.ScreenDisplay, Nothing)

    'SETS UP WHICH LAYER TO SELECT WITHIN THE ENVELOPE
    Dim pLayer As ILayer
    Set pLayer = pMap.Layer(0) ' the first layer, the only layer
    Dim pFeatLayer As IFeatureLayer
    Set pFeatLayer = pLayer
    Dim pFeatClass As IFeatureClass
    Set pFeatClass = pFeatLayer.FeatureClass

    'SPATIALLY SELECTS THE POINTS IN THE ENVELOPE
    Dim pSpatialFilter As ISpatialFilter
    Set pSpatialFilter = New SpatialFilter
    Set pSpatialFilter.Geometry = pEnvelop
    pSpatialFilter.SpatialRel = esriSpatialRelIntersects

    'PUTS THE SELECTED FEATURES IN A CURSOR
    Dim pFeatureCursor As IFeatureCursor
    Set pFeatureCursor = pFeatClass.Search(pSpatialFilter, True)
    Dim pFeat As IFeature
    Set pFeat = pFeatureCursor.NextFeature

    'OPENS THE DB SO THAT THE SELECTED RECORDS CAN BE FLAGGED
    Dim cn As New ADODB.Connection
    Dim rs As New ADODB.RecordSet

    With cn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        '.Properties("Jet OLEDB:System Database") = "c:\WINNT\system32\system.mdw"
        .Open CurDir & "\demol.mdb"
    End With
    With rs
        .ActiveConnection = cn
        .CursorType = adOpenDynamic
        .LockType = adLockOptimistic
    End With

    'LOOPS THROUGH THE CURSOR RECORDS
    Do Until pFeat Is Nothing
        rs.Open "SELECT * FROM temp_xy WHERE RecordNum = " & pFeat.Value(0) & " "
        rs.Fields("PWL").Value = True ' FLAGGED PWL = YES
        Set pFeat = pFeatureCursor.NextFeature
        rs.Update
        rs.Close
    Loop

```

```

cn.Close

pMap.DeleteLayer pLayer
Me.GraphData
pActiveView.Refresh
End Sub
' END DOMAIN 3 -----

' DOMAIN 4 -----
' UPLOAD DB
' TOOLBAR BUTTON
Private Sub cmdUpdate_Click()
' "IN" RECORDSET QUERIES RECORDS FROM THE temp_xy TABLE
Dim cn_in As New ADODB.Connection
Dim rs_in As New ADODB.RecordSet
With cn_in
.Provider = "Microsoft.Jet.OLEDB.4.0"
.Properties("Jet OLEDB:System Database") = "c:\WINNT\system32\system.mdw"
.Open CurDir & "\demol.mdb"
End With
With rs_in
.ActiveConnection = cn_in
.CursorType = adOpenDynamic
.LockType = adLockOptimistic
End With

' QUERY USER FOR QUANTITY OF RECORDS TO BE UPLOADED
Dim Answer As Integer
Answer = MsgBox("Would you like to load only the 8AM readings? " & _
"Otherwise all data will be loaded", vbYesNo, "Data Volume")
If Answer = vbYes Then
rs_in.Open "SELECT * FROM temp_xy WHERE Hour = 800"
Else
rs_in.Open "SELECT * FROM temp_xy"
End If

' "OUT" RECORDSET OPENS THE ACTUAL WATER LEVEL TABLE FOR APPENDING
' "IN" RECORDS FROM THE temp_xy TABLE
Dim cn_out As New ADODB.Connection
Dim rs_out As New ADODB.RecordSet
With cn_out
.Provider = "Microsoft.Jet.OLEDB.4.0"
.Properties("Jet OLEDB:System Database") = "c:\WINNT\system32\system.mdw"
.Open CurDir & "\demol.mdb"
End With
With rs_out
.ActiveConnection = cn_out
.CursorType = adOpenDynamic
.LockType = adLockOptimistic
End With
rs_out.Open "SELECT * FROM waterlevel"

Dim FirstRecord As Boolean ' flag to update the StartDate variable only once in loop
FirstRecord = True

Dim inmonth As String
Dim inday As String

Do While Not rs_in.EOF
rs_out.AddNew
rs_out.Fields("WellNum").Value = rs_in.Fields("WellNum").Value
rs_out.Fields("Date").Value = rs_in.Fields("Date").Value
rs_out.Fields("Pumping").Value = rs_in.Fields("PWL").Value
rs_out.Fields("DTW").Value = rs_in.Fields("DTW").Value
rs_out.Fields("Comments").Value = "Data from " & loggertype & " datalogger"
rs_out.Fields("EnteredBy").Value = GetLoginUserName ' windows user
rs_out.Fields("EnterDate").Value = Now ' today
rs_out.Update

' update date variables to be used to rename the file in archive
If month(rs_in.Fields("Date").Value) < 10 Then

```

```

        inmonth = 0 & month(rs_in.Fields("Date").Value)
    Else
        inmonth = month(rs_in.Fields("Date").Value)
    End If
    If day(rs_in.Fields("Date").Value) < 10 Then
        inday = 0 & day(rs_in.Fields("Date").Value)
    Else
        inday = day(rs_in.Fields("Date").Value)
    End If

    If FirstRecord Then
        StartDate = inmonth & inday & year(rs_in.Fields("Date").Value)
        ' only updated the first time
    End If
    EndDate = inmonth & inday & year(rs_in.Fields("Date").Value)
    ' updates until finished, i.e. keeps value of last record

    rs_out.MoveNext
    rs_in.MoveNext
    FirstRecord = False ' StartDate will no longer update after the first loop
Loop
rs_in.Close
cn_in.Close
rs_out.Close
cn_out.Close
MsgBox ("Data upload complete")

' use the file system object to rename and move the input file
Dim fi As New Scripting.FileSystemObject
Dim InFileDirectory As String
InFileDirectory = CurDir(FilePath)
Dim InFileName As String
InFileName = fi.GetBaseName(FilePath)
Dim InFileExt As String
InFileExt = fi.GetExtensionName(FilePath)

Dim OutFile As String ' appends the first and last records dates to input file name
OutFile = CurDir & "\Archive\" & InFileName & "_" & StartDate & "_" & EndDate & _
    "." & InFileExt

' rename and copy original file to archive
Name FilePath As OutFile

MsgBox (InFileName & "." & InFileExt & " has been moved to archive")
End Sub
' END DOMAIN 4 -----

' DOMAIN 5 -----
' DATE CONVERSION FUNCTION
Public Function convtodate(dayofyear As Integer, inyear As Integer) As Date
    ' A BIT VERBOSE, BUT IT WORKS!
    Dim month As Integer
    Dim day As Integer
    Dim leapyear As Boolean
    leapyear = False
    If inyear Mod 4 = 0 Then
        leapyear = True
    End If
    Select Case leapyear
        Case False ' not leap year
            Select Case dayofyear
                Case Is < 32
                    month = 1 ' january
                    day = dayofyear
                Case Is < 60
                    month = 2 ' february
                    day = dayofyear - 31
                Case Is < 91
                    month = 3 ' march
                    day = dayofyear - 59
                Case Is < 121

```

```

        month = 4 ' april
        day = dayofyear - 90
Case Is < 152
        month = 5 ' may
        day = dayofyear - 120
Case Is < 182
        month = 6 'june
        day = dayofyear - 151
Case Is < 213
        month = 7 ' july
        day = dayofyear - 181
Case Is < 244
        month = 8 ' august
        day = dayofyear - 212
Case Is < 274
        month = 9 ' sept
        day = dayofyear - 243
Case Is < 305
        month = 10 ' oct
        day = dayofyear - 273
Case Is < 335
        month = 11 ' nov
        day = dayofyear - 304
Case Is < 366
        month = 12 'dec
        day = dayofyear - 334
End Select
Case Else ' leap year
Select Case dayofyear
Case Is < 32
        month = 1 ' january
        day = dayofyear
Case Is < 61
        month = 2 ' february
        day = dayofyear - 31
Case Is < 92
        month = 3 ' march
        day = dayofyear - 60
Case Is < 122
        month = 4 ' april
        day = dayofyear - 91
Case Is < 153
        month = 5 ' may
        day = dayofyear - 121
Case Is < 183
        month = 6 'june
        day = dayofyear - 152
Case Is < 214
        month = 7 ' july
        day = dayofyear - 182
Case Is < 245
        month = 8 ' august
        day = dayofyear - 213
Case Is < 275
        month = 9 ' sept
        day = dayofyear - 244
Case Is < 306
        month = 10 ' oct
        day = dayofyear - 274
Case Is < 336
        month = 11 ' nov
        day = dayofyear - 305
Case Is < 367
        month = 12 'dec
        day = dayofyear - 335
End Select
End Select
convtodate = DateSerial(inyear, month, day)
End Function
' END DOMAIN 5 -----

```

REFERENCE

City of Tucson Water Department. 2004. Water Plan: 2000-2050 (*Mayor and Council Final Draft, November 22, 2004*). <http://www.tucsonaz.gov/water/waterplan.htm>

AUTHOR INFORMATION

Michael F. Liberti
Hydrologist
City of Tucson Water Department
(Tucson Water)

PO BOX 27210
Tucson, AZ 85726

(520)791-2689
(520)701-3293 (fax)

michael.liberti@tucsonaz.gov