1993

Software development: As applied to State Fire Protection Assessments Project

Noel Shubert Weaver
The University of Montana

SOFTWARE DEVELOPMENT

As Applied to State Fire Protection Assessments Project

By

Noel Shubert Weaver

B.S., Cook College of Rutgers University, 1979

Presented in partial fulfillment of the requirements

for the degree of

Master of Science

University of Montana

1993

Approved by

[illegible]

Chairman, Board of Examiners

[illegible]

Dean, Graduate School

July 22, 1993

Date

Table of Contents

INTRODUCTION

In the course of developing a microcomputer application for the Montana Department of State Lands, I undertook a hands-on, real-world opportunity to accomplish all phases of the software development cycle. The need for this application was first brought to my attention in early 1989, and to date the application which I completed has been in use for three and one-half years.

What follows are, first, general descriptions of both the setting and the role of the application I developed. Then, stepping through the stages of the software development process, I discuss experiences related to each stage of developing and maintaining this application. Finally, a general discussion summarizes the project and evaluations.

The Setting

Under Montana State Law, owners of forested land[1] must provide adequate fire protection for that land. Landowners may comply with this law by paying the State a fee, in exchange for the State providing fire protection service. The State

[1] "Forest land" has a specific legal definition for fire protection purposes (State Statute 76-13-102(8) MCA): land which has enough timber (standing or down), slash, or brush to constitute, in the judgement of the Department of State Lands, a fire menace to life or property; grassland and agricultural areas are included when those areas are intermingled with or contiguous to and no further than one-half mile from areas of forest land.

has the responsibility of assessing and collecting these fees, and of distributing the monies to the proper protection agency. The Fire Management Bureau of the Montana Department of State Lands, Forestry Division, manages this Fire Protection Assessment Program.

General Description of State-related Wildland Fire Protection

Several administrative programs are used to achieve the general goals of wildland fire protection in the state. The type of fire protection and related management procedures vary among the programs.

<u>Forest Fire District Protection Program</u> The Forest Fire District Protection Program applies to those areas commonly referred to as "Districts." Districts are formed by a vote of the landowners. They are areas of classified forest land with a definable boundary, and may be divided into several "protection units," each of which is protected by a separate protection agency. Through the Fire Protection Assessment Program, all private owners of forested land within in a District's boundaries are charged for the protection received; the assessment is added to their county property tax statement. (Government lands within a District may be protected through a master agreement. Private lands that are classified as nonforest land within the District are offered protection through the Nonforest Lands Protection Agreements Program.)

<u>Affidavit Unit Protection Program</u> The Affidavit Unit Protection Program applies to areas commonly referred to as "Affidavit Units." These are areas where no District has been formed, individual private landowners are interested in protection, and a recognized forest fire protection agency is willing to protect individual private classified forest lands. Private lands within an Affidavit Unit are protected usually only at the landowner's option, and only through a Forest Fire Protection Affidavit. (Under some circumstances, protection is given without an Affidavit.) Through the Fire Protection Assessment Program, owners of protected lands within a unit are charged for the protection received; the assessment is added to their county property tax statement. (Government lands within an Affidavit Unit are protected by a cooperative agreement.)

<u>State-County Cooperative Protection Program</u> The State-County Cooperative Protection Program applies to areas in counties which maintain a cooperative agreement with the State. The boundaries of the area protected under this agreement include all State and private lands within the county except: forest lands within a District, forest lands protected under an Affidavit, and nonforest lands protected under a Nonforest Agreement. This program is designed to provide wildland fire protection to the State and private lands not covered by another form of protection. The funding for this program is obtained through means other than State assessments.

Nonforest Lands Protection Agreements Program The Nonforest Lands Protection Agreements Program is for providing fire protection to nonforest land, ie., land which cannot be classified as forest land. The owner of nonforest lands may have the land protected under this program by entering into a contract with the protection agency. Through the Fire Protection Assessment Program, owners of protected nonforest lands are charged for the protection received; the assessment is added to their county property tax statement.

General Description of Fire Protection Assessments Program

The owners of lands protected under the Forest Fire District Protection Program, the Affidavit Unit Protection Program, and the Nonforest Lands Protection Agreements Program are to be charged fees through the Fire Protection Assessments Program. The total collected fees using various rates must be predictable for financial management purposes. Therefore, six basic purposes of the Fire Protection Assessments Program are as follows:

1). Properly identify lands protected under this program.

Fire Protection Assessments Program personnel must identify the protected lands and the parties responsible for paying assessments. They use maps showing boundaries of protected areas, county landowner records, and Affidavit and Nonforest Lands documents to do so.

2). Collect the necessary information about these protected lands. Once the protected pieces of land, or parcels, are identified, personnel need to record or update relevant data. Data required for calculating proper assessments consists of:

- the name of the person responsible for paying the assessment on the parcel,
- the name of the protection area (District or Affidavit Unit) in which the parcel lies, and
- the size of the parcel, in acres.

Personnel must record or update additional data about each parcel to facilitate the maintenance of complete and accurate records, to aid in communicating the assessments amounts to the counties, and/or to enable the system to provide reports of protected land to protection agencies:

- county of parcel;
- the school district containing the parcel;
- the assessor number assigned to the parcel by the county;
- the name of the parcel landowner (if the parcel is being sold under contract);

- the parcel location by section, township, and range, and in some cases the number of acres lying in each quarter-quarter of the section;
- free-form description of plat or subdivision, etc.;
- whether the land is classified as forested or nonforested; and
- whether or not the land is a mining claim.

3). Calculate the fire protection assessment for protected lands.

Fire protection assessments are calculated based on the identified protected lands and the current assessment fee schedule. For each party owing assessments in a protection area, the owned acreage is summed. The total assessment for each party is then calculated, based on the rate structure in effect. The total assessment is then divided among all parcels involved.

4). Provide counties with the assessment amounts for proper billing.

On an annual basis, the assessment amounts for all protected lands are reported to the counties so that the assessments will appear on the proper real estate tax bills. Each county is provided with a report, on paper or magnetic tape, listing the parcels on which assessments are owed and the amount of the assessment for each parcel. The form, format, and organization of each report corresponds with the recordkeeping system in place at the associated county office.

5). Provide protection agencies with a listing of protected lands.

Each protection agency is provided with a printout listing the lands for which they are required to provide fire protection.

6). Predict total assessments that would be generated under different fee schedules.

Whenever the assessment fee schedule is under review, estimates of revenues are calculated based on new fee schedules.

General Overview of the Tasks Performed by the Fire Protection Assessments Computer System

The Fire Protection Assessments Computer System was designed to manage information about lands protected under the Fire Protection Assessments Program. This information management includes maintaining records, calculating assessments, and generating reports and summaries.

The core of the Fire Protection Assessments Computer System is a large data file, the Mainframe Master File. The Master File contains all current records of lands protected under the Fire Protection Assessments Program. The Fire Protection Assessments Computer System is composed of a combination of mainframe and microcomputer tasks, which provide ways to manage and receive information from the Mainframe Master File.

The Mainframe Master File is maintained on a mainframe in Helena. Personnel in Missoula gather update information, and use a microcomputer to produce data files that are sent to Helena to update the mainframe. The mainframe program produces reports and annually calculates assessments. This basic system has been in place for approximately 23 years, although in 1987 the microcomputer function replaced what originally was keypunching tasks.

The original division of the computerized Fire Protection Assessments tasks is as follows:

Mainframe System:

- maintain safe storage of the Mainframe Master File
- receive data files containing updates to the Mainframe Master File
- generate error-check reports based on updates
- generate large reports
- calculate the assessment for each piece of land in the Mainframe Master File
- calculate assessment subtotals and totals showing revenues that would be generated under different fee schedules
- generate data files on tapes

Microcomputer System:

- provide a way to add to, delete from, and make changes to the set of information describing the protected lands
- generate data files to update the Mainframe Master File

Shortcomings of the Fire Protection Assessments Microcomputer System

Although valuable in its time of creation, the original microcomputer system was very limited: it was little more than a simulation of keypunching. To better-appreciate the potential for improvements, consider an abbreviated description of the then-current process of updating the Mainframe Master File:

> Fire Protection Assessments Program data researchers packed maps and bulky printouts to the county offices, where they would do their best to compare their data set with the current county records; lack of standardization among the counties, in terms of maintenance and reporting of landowner data, complicated their comparisons. When finding a discrepancy, the researchers filled in blanks on a paper form to encode the necessary additions or changes to the existing parcel data. For each addition of a parcel, all non-optional fields of data were transcribed to the form; for each change, usually only a subset of the data needed to be recorded. Accurate completion of the forms required understanding some rather cryptic encoding procedures, knowing proper field formatting (left- or right-justification, zero- or space-filling), remembering which fields were required and which were optional, and knowing the idiosyncrasies particular to each type of data entry (add, change, delete). Next, back in the office, the data on the forms was keyed into the microcomputer. The files created in the computer were

periodically sent to the mainframe, where some types of errors were trapped. The mainframe generated reports to document the updating and errors, and the "error-free" records were used to update the Mainframe Master File.

Concurrent with the development of microcomputers capable of efficiently handling larger amounts of information, the users and data managers began to envision ways to radically increase productivity: develop a new PC-based computer application to mirror the data on the mainframe, to allow quick and easy access to the data, and to perform fundamental error-checking. This is where I entered the scene; my task was to develop the new PC-based computer application.

General Goals of the New Fire Protection Assessments Microcomputer System

To most effectively meet the envisioned goals, the new microcomputer system had to maintain a Master File that would replicate the Mainframe Master File. This PC-based Master File is called the PC Master File. Thus, the primary objectives to be met by the new microcomputer system are summarized as: 1) to provide an efficient and reliable means of updating data in the Mainframe Master File, and 2) to provide a simple and flexible means of accessing the data in the PC Master File.

The division of the basics of the computerized Fire Protection Assessments tasks changed little with the new system (changes shown in italics):

Mainframe System:

- maintain safe storage of the Mainframe Master File
- receive data files containing updates to the Mainframe Master File
- generate error-check reports based on updates
- generate large reports
- calculate the assessment for each piece of land in the Mainframe Master File
- calculate assessment subtotals and totals showing revenues that would be generated under different fee schedules
- generate data files on tapes
- *generate data files (containing subsets of the Mainframe Master File) to restore the microcomputer system*

Microcomputer System:

- provide *efficient* ways to add to, delete from, and make changes to the set of information describing the protected lands
- *provide significant error-trapping*
- generate data files to update the Mainframe Master File
- *generate small reports*

- *receive data files (containing subsets of the Mainframe Master File) from the mainframe system*

Along with these system modifications were some necessary changes in data file structures and, for the mainframe, some modifications to rate calculation procedures. Aside from the technical and accuracy specifications, much emphasis was placed on user-friendliness, efficiency of operations, flexibility in use, and plans for growth.

User-friendliness Based on historical operations, it was predicted that many users would be required to use the final system from time to time. Aside from the expected turnover rate in the position of Fire Protection Assessments Program Manager (3 different persons have held this position in the last 7 years), there is a seasonal need to press other Fire Management Bureau personnel into service in order to complete the data collection/update process; these personnel are both temporary employees and full-time staff. Thus, a simple and intuitive user interface was of critical importance.

Efficiency of Operations By minimizing the number of keystrokes required for performing operations, productivity would be boosted significantly. Most users of the system are not typists. The primary data files updated using this system contain over 40 fields, with a total record length of over 251 characters. Obvious improvements in efficiency are gained by not forcing the user to cursor through every field, by providing short-cuts to entering long strings wherever possible, by allowing the user to easily undo unintentional actions, by having the computer perform real-time error checks wherever feasible, and by having the computer automatically do field formatting.

Flexibility in Use The new system introduced the means of quick access to the data files with which the users worked – a new tool. Only by implementing the means for the user to have several options for working with this data would this tool be most fully utilized. This included, for example, providing multiple indexing options and the means for making changes to groups of records meeting user-specified criteria.

Plans for Growth The specifications for this system included known features that were to be more completely specified and added as time allowed. To the extent feasible, all stages of development were to allow for this growth.

THE SOFTWARE DEVELOPMENT CYCLE

Feasibility Study

Overview

The feasibility study was primarily performed by Fire Management Bureau personnel in order to acquire the necessary budget to perform the improvements, and to block the development of a mainframe-based alternative that was perceived as less desirable. It was clear that benefits would be reaped by making major improvements to the then-20-year-old system; the predicted 33 percent reduction in personnel-hours required to perform related assessments duties would free time for making the data files more accurate and complete, resulting in increases in efficiency and revenue. Furthermore, the new system would allow users to accomplish many procedures previously unrealistic, such as performing massive data standardizations.

Methods

The Fire Protection Assessments Program Manager and Fire Management Bureau Computer Specialist produced a 27-page report, *Department of State Lands Assessment Computer System Alternative Analysis and Funding Justification*. This report addresses the fundamental changes needed in the computer system; responds to a proposal for a mainframe-based revamp of the existing system; proposes, alternatively, a microcomputer-based upgrade to the system; elaborately analyzes the projected costs of alternatives; compares alternatives on both cost and benefit bases; and provides recommendations. In conjunction with the Operational Services Bureau Supervisor, the same personnel also prepared an 18-page report, *Fire Management Bureau Fire Assessments Program Project Plan*, which summarizes the microcomputer system project, specifies cost estimates in terms of current budgeting, outlines estimates of person-hours required for completion of various phases, and presents an overall time-frame analysis.

Discussion and Evaluation

A feasibility study needs to examine the technical, economic, and operational feasibility of the project. In a government setting such as this, these aspects are potentially quite complex. For example, there may be restrictions on the type or brand of hardware or software purchased, there may be funds available for equipment but not for personnel, and there may be internal politics governing certain types of decisions.

In this circumstance, since there were personnel involved who are fully capable of examining and understanding the technical feasibility as well, it was appropriate that State personnel undertake the feasibility study. Furthermore, not only were these people intimately familiar with the idiosyncrasies of working within the State system, with the Fire Protection Assessments Program, and with the existing computer system, they also had much more experience than I in planning projects of this nature, and thus could, for example, better-estimate time frames.

A possible disadvantage of this arrangement is the loss of a "fresh look" that an outsider (such as myself) might bring to the plan. By not being entrenched in the current system, an outsider potentially has the advantage of seeing problems and solutions in a different context. Admittedly, some solutions initially imagined by an outsider may be totally inappropriate with respect to existing constraints, but this would be discovered in the course of the feasibility study.

It is notable that in reviewing the original documents with 20/20 hindsight, the original plan was not followed precisely, and the estimates of time and expense were often "extremely coarse." This reemphasizes both the difficulty in making these types of projections and the importance in realizing that these are *just estimates*. The process of thinking through the stages and details (to the extent possible) is nevertheless important, however, to have a general handle on the project, particularly in comparison with alternatives. Where necessary time or financial constraints exist, it is important to recognize the essential elements of the system and to plan for their assured development. In this project, many goals were identified, and priorities were given to various aspects as the development progressed. This approach ensured the development of the critical elements, allowed for the development of other elements as time allowed, and assured that the computer system would be planned and created in such a way as to allow for the future addition of unmet goals without the need to redo the structure of the system.

In summary, the feasibility study for this size of a project in this type of setting ideally should be the product of coordinated efforts of the client's personnel and the computer system developer. Input from the client's personnel is most likely to ensure the understanding, anticipation, and consideration of the relevant constraints, context, and long-range goals. Input from the system developer opens the door to some fresh perspectives. Thus, this arrangement is most likely to maximize the endurability of the final product.

Requirements and Specifications

Overview

The level of specificity required in this stage of software development depends largely on the project. In this case, the product was to bridge the efforts of Fire Protection Assessments personnel with the mainframe database. Concurrent with this project, the mainframe personnel were developing a new interface in the mainframe program to receive data in the "new format" required by this system. New fields were being added to the database on the mainframe, and updates were to be sent as entire records rather than changed-fields-only.

The Mainframe Interface The portion of this project that produced the "update data file" for the mainframe was necessarily very specific in requirements from the onset. Field widths, justifications, and valid values were planned and data file formats were agreed upon. Since this involved little change from the pre-existing situation, this was relatively simple, and was primarily handled in the next stage (see page 29).

The User Interface This portion of the project was less-specifically defined ahead of time. Basic requirements were specified, desired characteristics were expressed, and the form was allowed to evolve. This permitted a lot of freedom in design and programming, yet also required obtaining continuous feedback from the end user. Developing and fine-tuning the user interface was by far the most time-consuming aspect of the project.

The *Requirements and Specifications* document for this project (see Appendix I) served as a communication tool, describing in considerable detail the functions that the microcomputer application was to perform. Intended for use by both the user and myself, this is not a highly computer-technical document; it is a common meeting ground. Although some of the items were changed through the course of the software development process, the unchanged document served its purpose well as a reference describing exactly what features were to be implemented.

The document specifies how the user is able to lookup, change, add, and delete records in the PC Master File. Key fields, indexed field-combinations that the user may use to retrieve a particular record, are listed.

Time-saving features are described for clarification. These include, but are not limited to:

- Code tables

  Some of the data in the Data Files is represented by codes. Also, codes may be used to enter frequently-used landowner names for improved speed, accuracy, and consistency. These codes are easily retrieved from code tables, which can be activated with a function key. Also, the code tables are easy to edit (where appropriate).

- Automatic-add defaults

  When adding a series of parcels that have one or more fields in common, one may set default values for those fields. (The default values are checked for validity before the user can use them.) The data will automatically be entered in an "ADD" screen, and can be overridden if so desired. This can improve both accuracy and speed.

- Parcel duplication

  This is especially helpful if a landowner splits a parcel into several smaller parcels, or if a new parcel is very similar to an existing parcel. Once a parcel is found in the data file, it may be "Split," creating an "ADD" with the same data in the fields, ready to be edited. The user makes the necessary changes, saving data entry on most of the fields.

- Error-checking

  Upon completion of an Add or Edit screen, many of the values are checked to be sure they are valid. If any are not, the computer beeps and the fields are flagged on the screen. The user must make the corrections or abort the operation.

- Short forms

  If only certain fields must be changed in a group of records, the user may select a "Short Form" which only allows access to those fields (although the entire record is visible). This minimizes accidental change of other fields, and helps the user speed through the data entry screen. A function key may be used to pop the user into a full-edit mode, should other occasional changes be required.

- Speedy ways to move about the screen

  There are about 40 fields on the PC Master File data entry screen. Rather than having to key through every field, certain keys may be used to skip several fields.

- Automatic-fill of quarter-quarters

  If a parcel describes an entire section, or most of one, the quarter-quarters may be filled automatically with "40" each. (The user can edit these if necessary.) This saves up to 32 keystrokes. Similarly, quarter-quarters may be zeroed.

- Group change function

  This allows for the *en masse* change of some element(s) in a set of records which match user-specified values.

- Checking disk space

  Although not directly a data entry problem, it most surely can affect the integrity of the data file. At appropriate times, the disk space and file sizes are monitored to ensure smooth operation.

- A "LOOK" function separate from the "EDIT" function

  Designed for when the user is "just looking," this minimizes accidental change of data. A function key may be used to pop into an edit mode should the user spot something that needs correction.

- Automatic field formatting

  Fields requiring left- or right-justification and/or padding with zeros or spaces are appropriately formatted automatically.

- Multiple indexes

  The user may select one of six indexes, thereby matching the sequence found at any county. Once locating a desired parcel (the key depending of course on the index), one can then step forward or backward through the file according to the chosen sequence. Furthermore, while examining the data of one parcel, the index scheme may be changed without moving from that parcel. Thus, for example, one can locate a parcel by landowner name, then change to the legal land description index to examine the neighboring properties.

- User manual

  To ensure the program's usefulness through the changes in personnel, a user manual was deemed essential.

Many other significant features are addressed in the document as well, in an effort to completely explain the expectations of the computer system.

Methods

In preparation for creating the *Requirements and Specifications* document, I studied the reports and related material describing the purpose and function of the Fire Protection Assessments Program. I pored over the documentation of the existing computer system, examined data entry forms and computer-generated reports, and read the preliminary documents leading to the decision to revamp the computer system. Additionally, I worked with Fire Protection Assessments Program personnel to learn, through participation and demonstration, of the steps required to perform the day-to-day tasks involved in the data collection and management processes. I asked a lot of questions.

Fire Management Bureau personnel involved in the assessments program generated a four-page document outlining their primary needs and priorities, a useful tool for preliminary communications. Although this document specified most of the basic concepts of the program requirements and specifications, much more detail and clarification was necessary. Working with this document, I met extensively with the Fire Protection Assessments Program Manager and Fire Management Bureau Computer Specialist to discuss the particulars. Together we combed through the list, discussing the rationale, the scope, and the envisioned results relating to each item.

As a result of my investigations, I prepared a draft *Requirements and Specifications* document which detailed the objectives as I understood them. This initial document aided in identifying features that weren't quite clear. After carefully reviewing the draft with the appropriate Fire Protection Assessments Program personnel, I filled in necessary details, obtained approval for the final version, and moved on to the next stage.

Discussion and Evaluation

Learning about the Client's Needs

It is appropriate now to introduce one of the most important aspects of the whole business of software development: communication. Here, effective communication must occur between the client and the computer expert. The systems developer should, initially, acquire a thorough grasp of the needs of the client, as perceived by the client. This is aside from the computer environment, parameters, and constraints; I am referring to the "end results" envisioned by the client. A useful step in this learning process is to work with the client in doing the job as it is currently being done. This is not program design, creation, implementation, or anything learned in the classroom; basically, it is learning another's job. I cannot overstate the importance of listening and working hard to understand the needs of the client. New ideas may be welcomed and warranted, but it is only secondary to seeing the situation through the client's eyes.

In the case of the project I completed, I learned about all relevant aspects of the background and the day-to-day tasks of the Fire Protection Assessments personnel as they related to the tasks being computerized in the new system. I worked with the people using the system, listened carefully, and asked a multitude of questions. This listening and learning is a continuous process, not limited to the requirements and specifications phase of the software development cycle.

Fine-tune the Specific Requirements

During this phase, I developed a better understanding of the specific needs of the assessments program personnel, and the client realized the choices they had to make. For example, an initial objective was to be able to use the program to "...sort the data by *any* category..." with the unrealized implication of over 40 index files being continuously maintained. We were able to trim the list to seven meaningful indexes (some have since been added and changed), leaving the rest to *ad hoc* file manipulation outside the scope of the program under development.

An important question during this phase pertained to the frequency of performing particular tasks. This was usually not specified in the requirements, but the understanding of this influenced the nature of the solutions implemented. Emphasis was placed on optimally streamlining the most frequent tasks, where the payoff, in terms of user productivity, would be greatest.

Plan for Unanticipated Requirements

During this project, it became very evident to me that, in the case of major computer systems development, often the client is not able to anticipate many specifics of the desired system until at least some of the system exists. I believe this is simply because the context for new uses is non-existent; "solutions" or "improvements" are based on the current context (the method presently employed), with little or no basis for realistically imagining beyond the immediately-identifiable fixes. Additionally, simply by relieving pre-existing bottlenecks, the new system shifts bottlenecks to different stages of the process. The best way to prepare for this type of "growth" in the system, it seems, is to design a highly flexible and modularized program. Minimize hard-coding to the extent possible and reasonable. This requires more thought and effort initially, but the payoff is tremendous when modifications are implemented.

Implementation Language Considerations

By this stage, Fire Management Bureau personnel had expressed a desire to have the program written in Clipper; this would facilitate standardization (both in code maintenance and in user interface) with other programs in use within the Bureau. Having written one of those other programs myself, I had the advantage of better-understanding the type of interface desired. More importantly, having already worked with Clipper to a great extent gave me better insight for knowing how simple or complex it would be to implement some of the desired features.

In some situations it may be true that many preliminary steps of software development may be achieved without a specific coding language in mind. However, it was my experience in this case that it was of great significance to know the language of implementation. This allowed me to assess in advance the feasibility of particular features, given the time and financial constraints.

Program Analysis and High-level Design

Overview

For this portion of the software development, I developed a set of high-level data flow diagrams, a data dictionary, a file dictionary, a cross-reference table of data elements and files, a list of valid codes of certain data elements, and a description of the record format for the mainframe (see Appendices II and III). I prepared these documents using the information I learned in the requirements and specifications stage.

Methods

I created the set of high-level data flow diagrams using techniques described in Tom DeMarco's *Structured Analysis and System Specification.*[2] The diagrams serve as a graphical representation of the system, defining the scope, primary processes, sources and sinks of information, and the interfaces between these.

For the data dictionary and file dictionary, I devised a notation that both uses ideas from the same DeMarco source and incorporates features that clarify data

[2] DeMarco, T. 1978. *Structured Analysis and System Specification.* Yourdon Press, Prentice Hall Co., Englewood Cliffs, NJ. 352 pp.

characteristics, thus completely defining the data elements (see Figure 1). In this way, I was able to unambiguously describe every data element that was related to the system requirements.

* *	used to offset comments	
(TJFL)	shows code for Type, Justification, Fill character, and Length of data in field, according to the following:	
	Types:	C - Character
	Justification:	R = Right Justify
		L = Left
		_ = does not apply
	Fill Characters:	Z = Zero
		S = Space
		_ = does not apply
	Length:	number is field size
=	indicates "is equivalent to"	
{ }	indicates "zero or more iterations of" whatever appears inside the { }	
+	indicates "and"	
-	indicates "without"	
[|]	indicates "or"; eg., [This | That | Other] means "either This or That or Other"	
...	indicates the continuation of given series	

Figure 1 Data Dictionary and File Dictionary notation

The data dictionary describes the entries, both the "plain english" significance and the technical definition. When defining an element (the smallest data component, such as a field in a data file), the technical definition specifies the format and, if there is a logically finite set of valid values, the valid values; in cases where the value is a free-form user entry, the definition gives a brief description of the logical contents (e.g., "memos"). For non-element terms which are a combination of elements, the definition shows the appropriate combination of elements. Similarly, some non-element terms are most simply defined using the appropriate combination of non-element terms.

The file dictionary describes the contents and, where applicable, the index keys of the data files used in the application. It is to be used in conjunction with the data dictionary, which further defines the terms used in the file dictionary.

The cross-reference table of data elements and files, "Data Element and File Quick-reference," summarizes the similarities and differences between the data files in terms of the pieces of data included in each file.

Example

To illustrate how the tools work, consider the following. The PC Master File is defined: = { *County_File* }, literally meaning it consists of zero-or-more county files; the program user may decide how many county files there are. The PC's version of the Master File is actually a set of files, one for each county, and are collectively referred to as the PC Master File.

A County_File is defined: = { *Parcel* }, meaning zero-or-more parcels.

The Parcel has a much lengthier definition, since it includes the 42 fields, or elements, that describe a piece of land being assessed. To simplify for this example, suppose it consists of just two elements, Descript_1 and Township. The definition of Parcel would then be: = *Descript_1* + *Township*. To determine the meaning of these elements, one examines the definition of each.

In the data dictionary, Descript_1 is defined:

> * *(C_S32) Free-form land description* *
>
> = *[land description*
>
> *| Blank]*

meaning that it is a character string of length 32, padded with spaces. It is either a "land description" (which is not further defined) or blank. This definition indicates that any string of characters will be a "valid" (though perhaps not useful or even meaningful) entry.

The definition for Township, on the other hand, is very specific:

> * *(CRZ4) Legal land designation of parcel's Township; must be within the spread of Montana Townships.* *
>
> = *[[01 | 02 | 03 | ... | 35 | 36 | 37]*
>
> *+ [0 | 5] + N*
>
> *| [01 | 02 | 03 | ... | 15 | 16 | 17]*
>
> *+ [0 | 5] + S]*

This element is a right-justified character string, length 4, padded with zeros. More precisely, it indicates that it will contain three numerals followed by *N* or *S*, and that only certain sets of numerals are valid. The definition is patterned after the logical goal: if it is a *N*orth township, the first two digits must be within the range of *01* to *37*; the first two digits of a *S*outh township must be within the range *01* to *17*. Based on this definition, the program's validity check ensures the prevention of data errors that indicate a township number lying beyond Montana's borders. According to the definition, a third digit, either *0* or *5*, follows the first two digits. When this digit is *5*, it indicates a "half township." Only a few of these exist in Montana but, as shown in this definition, the program accepts any township designated as a half township. This type of error, deemed unlikely in part due to the design of the user interface, is not trapped.

An important aspect of writing these definitions is keeping them as readable as possible. The township definition could have been accurately been written:

```
=  / 0 + / 1 | 2 | 3 | ... | 7 | 8 | 9 / + / 0 | 5 / + / N | S /
   | 1 + / 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 / + / 0 | 5 / + / N | S /
   | 1 + / 8 | 9 / + / 0 | 5 / + N
   | 2 + / 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 / + / 0 | 5 / + N
   | 3 + / 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 / + / 0 | 5 / + N /
```

but this requires considerably more study to understand.

Similarly, it is important to create readable definitions by breaking them into logical elements. The PC Master File as defined in this example is more easily understood than would be the equivalent, but briefer:

```
{{  [       land description
            | Blank              ]
    +       [ [ 01 | 02 | 03 | ... | 35 | 36 | 37 ]
            +  [ 0 | 5 ]  +  N
            | [ 01 | 02 | 03 | ... | 15 | 16 | 17 ]
            +  [ 0 | 5 ]  +  S ]  } }
```

Compounding definitions in this way can also lead to the problems associated with redundancy, in cases where elements or sets of elements occur in many files.

Discussion and Evaluation

This stage was very important in terms of clarifying the technical details of the application. It can be summarized as a transformation of information gathered primarily in the requirements and specifications stage. Originally in the form of text in supporting documents, notes from verbal communications, and derived concepts, the technical details of the specifications are redefined via this process to a concise, consistent, and structured format. Thus, a significant value of this stage is that it reveals gaps and holes that may be lurking in the complexity of the system.

The data flow diagrams were helpful in clearly and simply describing the scope and basic functions of the system. Although the diagrams were of limited reference value once constructed (with respect to those of us intimately familiar with the system), the exercise of creating them was helpful because it forced me to consider each of the relevant aspects on an individual basis. Without constructing the diagrams, it was difficult to visualize a clear overview of the application and remember the important aspects of every component in the system. Aside from the construction process, these diagrams are also helpful in summarizing the system to someone who needs to learn about it.

The data dictionary proved to be extremely helpful. In constructing this tool, I found that I often needed more clarification and discussion to complete the definitions. Ultimately, every data element was individually examined in terms of logical content, technical form, and valid values. By inspecting and documenting these details, the client and I could confirm that we were on a common track.

Although the data dictionary was at times tedious to draft, I later appreciated my efforts. The data was not especially complex, but I found that I often referenced this document in subsequent stages of the application development. Thus, the ultimate correctness of the application relied on the accuracy of this document.

The cross-reference table of data elements and files was a well-used tool. Although repetitive of dictionary entries, thus increasing maintenance requirements, the tool earned its keep because of its simplicity; with so many data elements and files involved in this system, it otherwise would be difficult to readily see their relationships.

Detailed Design

Overview

During the creation of the program, as I applied the use of prototypes, I tried various methods of maintaining a program design document. My initial efforts were in the form of structure charts. These diagrams quickly became outdated, and were difficult to maintain current and neat without spending an inordinate amount of time drafting new versions. I modified the technique, next using labelled sticky-notes to represent the modules; these were more easily changed and moved around without creating such a mess. Alas, this method also proved unsatisfactory as the program became more complex.

Finally, I created a relatively detailed design using *Brackets*TM[3]. The resulting document is largely a mirror of the program structure with some of the details omitted for simplification.

Methods

Brackets is easy to use, functioning much like a spreadsheet in terms of its user interface. To continue the analogy, module names are inserted in the cells. Using

[3]*Brackets 2.0*TM is a computer-aided software engineering tool from Ken Orr & Associates, Inc. © Copyright 1986, 1987 TLA Systems and Education Ltd. All Rights Reserved.

brackets that project from the module names and specific codes between module names, one can describe the relationships among the modules, such as sequence, iteration, and alternation.

The brackets, containing subtrees or leaves of modules, can be expanded or compressed, allowing the user to more easily examine different levels of the structure within the limitations of a display monitor.

An essential feature of *Brackets* is the "dupe" (duplicate) function, which allows one to attach an individual module (and its bracket containing its substructure) to many places in the structure. This simplifies the entering and editing of the duplicated module and its associated substructure by keeping all of its occurrences identical.

Although *Brackets* has more features when used for COBOL programs, for my purposes the uses were limited to entering and illustrating a program design, navigating through the design interactively, and printing a representation of the design. Figure 2 shows a sample of the screen display; a corresponding page of printed output is in Appendix IV.

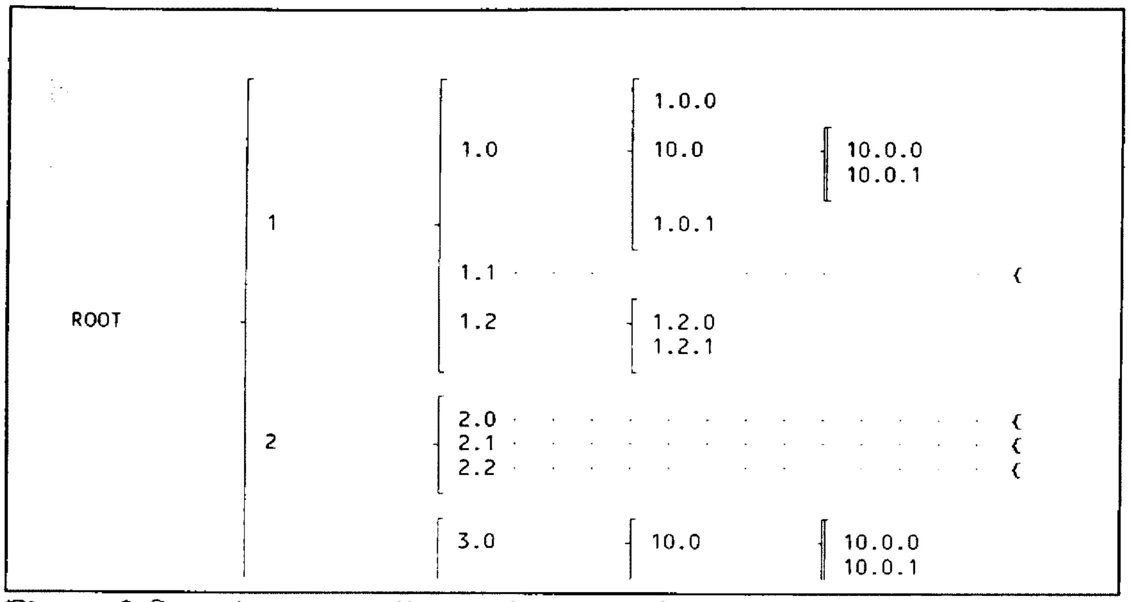


Figure 2 Sample screen display from *Brackets*.

Discussion and Evaluation

The intent of this design stage is to develop and show the structure of the program itself. It can be used to aid the code developer in knowing *a priori* the function and interface of each module to be written; in code maintenance it can be used to learn more about the system or find the correct place to implement a code addition or change. The final product should serve as a simplified representation of the code. It should illustrate the structure of the program, and it may include information about the interfaces between the modules.

The program in its final form contains approximately 350 modules. If a detailed design (structure chart) showed one module for each code module, and this structure chart could logically display an average of seven modules on each page (not including connectors), this would result in a 50-page structure chart. Although many of the modules relating utility functions such as screen windowing or checking for disk space could be eliminated, I feel that this still significantly underestimates the number of pages that would be required to logically present the program structure. Nearly one-third of the 350 modules are called by more than one module; these "callees" may be leaves or roots of sub-trees of the chart. A logical presentation of many of these modules requires beginning a new page, which would be referenced on the pages of calling modules. Similarly, the "tree" structure of the *Brackets* output breaks and uses a reference to a module that is called by more than one module.

The point here is not really the number of pages, but instead the difficulty of representing this program with a useful detailed design which can be reasonably maintained along with development and changes to code. The thought of flipping through over 50 pages of diagrams to find a module does not ring of convenience; furthermore, the user would likely feel lost if following many references to find a particular portion of the program. If one builds an index to facilitate this, then the index must also be maintained.

An implied irony is that it is most practical to build and maintain a detailed design only for smaller programs, where such elaboration would seem to be less necessary! More appropriate, I think, is to construct detailed design only under certain circumstances, such as where it aids in clearly expressing a particularly complex section of the program.

None of the various methods I used for a detailed structure proved to be of much use to me in this project. Additionally, another programmer, who recently began to learn the structure and code of this program, found the output of *Brackets* to be useless. Primarily recasts of the program itself (or vice versa), and since they existed entirely independent of the program (i.e., not integrated with code production), these methods did little more than create a burden of maintenance. An addition to or nontrivial change in the program required making the appropriate changes to both the code and this design document to keep it current. I often felt as though I was writing the program twice!

An ideal program-design system, to avoid maintenance problems such as this, should be able to read the program code and output a design. The resulting design should be useful, meaning it should be more than just a report that describes all relationships and interfaces – ideally, it should be accessible interactively, allowing the user to readily navigate through the structure and query about specific elements along the way. Such a "design" system, though, would address only the maintenance problems after the code is written; the user must design from scratch initially, then write the code, then use the design system to regenerate the design. For coarse designs this is an improvement, but better yet is a system that generates code from the design. In effect, using this type of system, the "design stage" is all that the user would need to modify. This design function, then, would need to accommodate much detail that typically is omitted from designs produced using simpler systems. The line becomes blurred: is this powerful system a higher-level programming language or a sophisticated design tool? Either way, the potential seems great for simplifying the creation and maintenance of effective programs. Although excited about the development of such products, I don't expect to find extremes of simplicity and flexibility in the same product.

To summarize, in cases such as this, where the program language lends itself to writing readable code, and where no automated system exists for generating code from a design document and for maintaining the design when the code is changed, I think that creating and maintaining such detailed design is not time well spent.

Code Development

Overview

I wrote the program in Clipper (by Nantucket; Summer '87 release). At the time I began writing the program (before the existence of dBASE IV), Clipper was thought of as being primarily a compiler for dBASE. The languages have in common most commands and functions, although each has some commands and functions absent from the other.

Using prototypes heavily in the development of this program, I began writing code after the high-level design, concurrent or alternating with the detailed design. During this stage, I also began writing the user guide.

Methods

My choice of text editors for writing the code was WordPerfect.

The prototypes I used in the course of creating this system were not "test-and-scratch" programs; they were stages of the program under development. After writing new "visible" sections of code, I would confer with the client to obtain feedback and offer suggestions regarding the forthcoming stages.

An important aspect to discuss with respect to coding methods relates to writing human-readable, self-documenting code. While writing this program I learned many tricks about this, and was able to fully appreciate them as the complexity of the program developed. Code appearance, complexity, choice of variable and procedure names, and parameter-passing all influence readability, and are applicable regardless of language.

Code Appearance It goes without saying that conventional rules of indentation should be adopted and used consistently. This makes the code structures obvious to the reader. It is also important, for readability, to employ a consistent method of how capitalization is used. For example, I capitalized all letters in a Clipper keyword, and used mixed case for variable and procedure names. The use of mixed case allowed for guiding the user in how to read compound names, e.g. GetKeyValues (as opposed to GETKEYVALUES or getkeyvalues), and is easier than using "_" to separate the root words.

Complexity Brief modules are so much easier to understand.

Choice of Variable and Procedure Names Although more typing is involved, long descriptive names are by far easier to work with because they are easier to understand. I also found it very helpful to adopt naming conventions that can aid the reader in immediately understanding the nature of the variable name: for example, I used a lower-case "f" to prefix variable names whose value is the name of a file.

Parameter-Passing Two aspects are of importance here: globals versus locals and variable-parameters versus value-parameters. In classroom exercises, one learns that passing and correctly specifying parameters helps reduce the likelihood of inadvertently altering data. There is another important benefit: the source code is much easier to read and understand when the use of globals is minimized and the nature of passed parameters is correctly identified.

Discussion and Evaluation

Although WordPerfect may be inferior to text editors that are custom-made for writing in a particular language, there was no such thing text editor for Clipper, and I found there to be many advantages to using WordPerfect. It simplified my life since I was already familiar with WordPerfect, and I found that additional skills I acquired while writing the code could be transferred to other uses. Some of the features of WordPerfect that I found most helpful were: easy creation and modification of simple or complex macros for repetitive tasks; within- and across-document searches for text strings; the ability to insert page breaks so that I could page-down to easily scan through the modules contained within one file; and the ease with which blocks or columns of text can be copied and moved.

The use of prototypes in the development of a system such as this, where the type of user interface is exceptionally critical, is extremely helpful. I found that working with prototype systems enabled me to more clearly discuss concepts and choices with the user, and it allowed the user to better imagine his needs within the context of the new system. Because of this approach, additional simple features were easily implemented on-the-fly rather than requiring a revamp of a completed system. Also, since the actual program-under-development served as the prototypes, this method did not require additional effort to produce tools useful only for communication.

When I began to write the program, the formalities of some of the guidelines for writing readable code seemed like just formalities. As the program grew, they became essential, even for merely my own use. Considering my experiences in modifying other programmers' code, I do not hesitate to conclude that cryptic, poorly documented code has a brief lifetime of use, if modifications are to be made.

Beginning to write a user manual at this stage helps to emphasize a logical, user-oriented interface in the development of the program. If done in limited detail at this stage, to minimize necessary changes, the later-completion of this manual is then easier.

Testing and Implementation

Overview

The testing and implementation phase was a coordinated effort involving the Fire Protection Assessments Program Manager, the Fire Management Bureau Computer Specialist, and me. Since prototyping was used throughout the software development, much of the testing had been performed by the time the code was completed. We needed, however, to test specifically the entire functional system, running test data through the entire process, from data entry to mainframe reporting, to uploading from the mainframe.

Methods

We produced test data sets that included carefully planned combinations of valid and invalid data. We also used a small county of existing data to upload from the mainframe. No testing was comprehensive with respect to "taking all possible paths of execution," and the thought of doing this would be absurd. Actually, the personnel for whom the program was intended happened to be excellent testers; they tried to use the program in what seemed to be every way imaginable, which included ways that originally I had not envisioned while coding! I soon learned that to ensure robustness, I had to anticipate virtually any combination of user keystrokes; for example, a module called by a "hot key" must first deactivate the "hot key" to avoid reaching the eventual limits of recursion.

Discussion and Evaluation

Generally speaking, the results were pretty much as expected, with one major exception: when comparing the data files of the mainframe with those of the PC, 16 of the data fields were not similarly sequenced. This was due, primarily, to a difference in our communication language. Part of the data was a series of 16 2-character fields, each holding the value of the number of acres in a particular quarter-quarter of the section-field data. The fields were named according to a concatenation of "quarter quadrant" position (NW, NE, SW, SE) and "quarter-quarter quadrant" position, resulting in the names NWNW, NWNE, ... SESE. There was nothing new about this convention – the mainframe data fields already had these names, and the Fire Protection Assessments personnel had been using this shorthand all along. Unbeknownst to any of us, there were two conventions in use: NWNE meant "northwest quarter of northeast quarter" to the end user, but it meant "northeast quarter of northwest quarter" to the mainframe people. Luckily, we were in the testing phase when this potentially easily-overlooked error was discovered. An important lesson here is that one must strive for careful and complete communication, and plan test sets that will reveal program errors even if you "know" the errors don't exist.

CONCLUSION

As a result of the cooperative efforts described herein, the developed system clearly has been resulting in higher productivity and efficiency. Although many of the enhancements that resulted in this were planned from the beginning, there were several whose need became apparent along the way:

- Menus that are easy to use

  One can cursor to the selection and press <Enter>, or simply press the first letter of the menu choice (making it more like "command driven" once you are familiar with the menus). Upon making most menu selections, part of the previous menus remain visible to help the user keep track of progression through the program.

- On-screen reminders

  "Information lines" describe menu choices, function key use (where appropriate) and how to abort or proceed with an operation. An "environment window" informs the user of program status (LOOK, EDIT, WAIT, ERR), name of file in use, index in use, and other useful information.

- Use of memory variables for data entry

  If a record is to be edited, the data is loaded from the data file into memory variables. The user can then make whatever changes desired. Upon completing the screen, the user can accept, redo, or reject the changes. Nothing is written to the data file until the user accepts the entries.

- Leaving "deleted" records in the data file

  Packing a file is very time consuming when the file is large. By making the screen obviously indicate when a record is "deleted," one can safely leave them there. The user may "pack" the file later, when it won't interfere with data entry.

- Multiple or single parcel views

  The user may view all of one parcel at a time, or some subset of many parcels on the screen. A function key can be used to pop the user from one mode to another. Some of the fields may be edited while in the "Multiple" mode, making standardization easier.

- Keeping track of the file pointer

  A change made to a value in a key field can wreak havoc on a user working through a file. In many programs, the file pointer ends up on the same record in the new location. In this program, the file pointer moves to whatever *was* the "next" record before the "current" record was moved.

- Remote workstations

  In many cases, it is extremely handy to take the computer into the county office and enter data there. For these purposes, a "remote workstation" can be set up on a portable computer. This workstation operates much like the main program. Upon return from the field, the data is loaded into the PC Master File and the Transaction File.

Note, however, there are many occasions and situations which may seem like they *should*, but *won't*, be improved. As an example from this project, there was early and brief discussion regarding the awkward nature of the then-current data file, specifically the fields BilleeName, OtherName, and BuyerFlag; to simplify, the definitions of the contents of the fields BilleeName and OtherName are determined by the contents of the field BuyerFlag. It was determined that it would be too large of a project to modify the existing data file format to any major extent. Thus, a lot of the bathwater was retained right along with the baby. The best that could be done under the circumstances was to make the user-interface allow the user to be free of dealing with the BuyerFlag field, and indicate the meaning of these "name" fields in a more intuitive way.

To summarize the project in terms of the stages of software development methods, different stages and methods proved to have different levels of usefulness for this particular project. The feasibility study was useful, but primarily for comparing this project to other solutions. The document resulting from the requirements and specifications stage was first an effective communication tool, then a reference that lasted through the much of the development of the program. The documents, particularly the dictionaries, resulting from the program analysis and high-level design were especially helpful, both in communication and later reference, remaining as a very useful tool when revising the program. The results of the frustrating detailed design stage were disappointing; I was at a loss to find a useful tool or end-product. Prototyping was facilitated by the modular code development, resulting in a successful product that has been used, enjoyed, and enhanced. The process of final testing is not to be overlooked, in that it is a "last chance" to trap mistakes; in this case, it prevented some major problems with data accuracy.

This entire process of software development was an effort toward an optimal balance among flexibility, speed, ease-of-use, practicality, and built-in accuracy for the fire assessment program personnel. It was a rewarding experience, and a useful experience. As time passes, I continue to make enhancements to the code, and am able to appreciate the significance of the hard work that went into this product.

APPENDIX I

REQUIREMENTS AND SPECIFICATIONS

FIRE ASSESSMENT MICROCOMPUTER SYSTEM
REQUIREMENTS & SPECIFICATIONS DOCUMENT

TABLE OF CONTENTS

General Functions

The primary goal of the microcomputer system under development for the Fire Protection Assessment Program is to fulfill the requirements as described in these Requirements & Specifications Documents. In addition to the software to achieve the microcomputer system goals, products of this project include a user manual and documentation for each stage of software development. Also, program documentation will be provided in accordance with Montana Department of State Lands Information Processing Policy.

The Fire Protection Assessments Program maintains a large database of information on the state's mainframe system in Helena. For purposes of discussion, this mainframe data file is referred to as the "Master File." When the new microcomputer system is in use, it will maintain its own version of the Master File, herein after referred to as the "PC Master File." The PC Master File is actually a set of data files, one for each county in the state. For purposes of this discussion, a county data file will be referred to as a "County File."

The general goals of the microcomputer system include the following:

1. Provide an efficient and reliable means of updating data in the Master File.
2. Provide a simple and flexible means of accessing the data in the PC Master File.

More specifically, the goal of the new microcomputer system is to meet the requirements as outlined on the following pages.

1. PC Master File Operations. The system allows the following types of operations to be performed on the PC Master File:

Lookup	Change
Addition	Duplication
Deletion	

These operations, which occur on a regular basis, are to work as follows:

A. Lookups allow the user to find all or part of the information about one or more Parcels contained in the PC Master File. A lookup operation displays the first Parcel matching the description entered by the user. After specifying the County, one of the following types of descriptions may be used in a lookup:

1) Billee_Name and Other_Name
2) Township, Range, Section, Descript1 and Descript2
3) Assess_Num
4) Unique_Num
5) School_Dist, Title_Owner, Buyer_Flag, Title_Owner continuation or Buyer
6) Date_Stamp
7) Attn_Flag

B. Additions allow the user to add new Parcels to the PC Master File. This operation includes the capability of retrieving a set of information from the Aff_Review File, to simplify those types of additions.

C. Duplications allow the user to make a copy (except for the Unique_Num field) of an existing Parcel in the PC Master File. A duplication is preceded by a lookup, an addition, or a change operation. It is most often followed by a change operation. If the duplicate Parcel is saved, it is treated (by the system) as an addition (for error-checking, etc.).

D. Deletions allow the user to remove Parcels from the PC Master File. A deletion requires doing one of the lookup operations first.

E. Changes allow the user to alter or add information about an existing Parcel in the PC Master File. Each change operation, or each group of change operations, requires doing one of the lookup operations first. Each change fits into one of the following classes, and is restricted to the type of change(s) specified. The user also has the option of making "unrestricted" changes to an individual Parcel, while processing a series of restricted changes.

1) Restricted to change in Billee_Name and Other_Name, and Buyer_Flag only.
2) Restricted to change in Assess_Num only.
3) Restricted to change in Prot_Area only.
4) Restricted to change in School_Dist only.
5) Restricted to change in Co_Locator only.
6) Restricted to change in Descript1, Descript2, and Co_Locator only.
7) Unrestricted change in Parcel: All permitted changes in a Parcel.

The system allows the processing of Groups with minimal effort. These will allow the user to confirm each change before it is made. Two specific Group features are:

(1) a simple way to change the same information (eg., Assess_Num) in a series of Parcels.

(2) a simple way to change a set of similar Parcels (eg., same Township and Range) in an identical way (eg., assign them all the same Prot_Area code). (Other examples: all 'John Smith' Billee_Name and/or Other_Name to 'Jane Doe'; all School_Dists of Parcels with Township '05N', Range '10W', Section '05' from '44' to '66') This can be used, for example, if ownership of an unsold subdivision is transferred to another party.

2. Indexing Criteria. The system supports lookup operations by allowing the user to access any County File in sequential order based on one of the following indexing criteria:

   A. Billee_Name and Other_Name
   B. Township, Range, Section, Descript1 and Descript2
   C. Assess_Num
   D. Unique_Num
   E. School_Dist, Title_Owner, Buyer_Flag, Title_Owner continuation or Buyer
   F. Date_Stamp
   G. Attn_Flag

3. Error Checking. The system supports all operations which modify the PC Master File (Parcel additions, deletions, changes) with error checking.

   A. Parcel deletions: Any deletion of an entire Parcel allows the user to view the Parcel before the deletion, and to confirm the deletion.

   B. Parcel changes: Any change to a Parcel allows the user to view the Parcel before the change, and to view the new version before choosing to confirm or cancel the entered changes. For multiple changes to a single Parcel, there is just one confirmation or cancellation.

   C. Parcel additions and changes: Information added to the PC Master File is checked for errors to the extent feasible. Some error checking occurs at the time of a Parcel addition/change. Additionally the user may check the PC Master File for errors. A check of the PC Master File is reported to the screen or the printer, at the user's choice. The following types of error-checking are available:

      1) These data are checked for being within the range of valid values:

County	Range
Prot_Area	Section
Township	Action_Code

      2) Tot_Acres: must equal sum of Quarter-Quarters, unless all Quarter-Quarters are empty. If Tot_Acres is greater than 640, the system alerts the user. Zero is permitted as a temporary entry, and results in the system setting the Attn_Flag.

3) A Parcel <u>must</u> contain the following the following data before it can be saved in the PC Master File:

Unique_Num	Township
Billee_Name	Range
County	Section
Prot_Area	

4) The system may include a feature, which the user can turn on or off, that alerts the user that the following data is missing:

Assess_Num	School_Dist

5) Other_Name: must be non-Blank in Parcel if Buyer_Flag = X.

6) Action_Code: must be included in Trans_Parcel.

7) Attn_Flag: A Trans_Parcel cannot be sent to the mainframe until the Attn_Flag is Blank. The Attn_Flag is not Blank if: (1) Tot_Acres = 0, or (2) Tot_Acres does not equal total of Quarters (unless all Quarters are zero), or (3) the user chooses to set the Attn_Flag.

D. Parcel <u>duplications</u>: (This option will be decided upon later, depending on the potential for accidental duplication of identical Parcel information.) The system warns the user unless there has been some change made in both the original and the duplicate.

4. <u>Efficiency of Data Entry Operations</u>. Efficiency of Parcel addition and change entries is facilitated in three major ways:

A. In some cases, the system automatically selects the proper information to add to the PC Master File based on the entries made by the user. These automatic operations include the following:

1) The user may enter a code (up to 4 characters) to indicate the description of the Billee_Name and/or Other_Name. This is used for Names that occur frequently in the PC Master File.

2) If the user enters '640' in for the Parcel's Tot_Acres, then the system automatically enters '40' for each Quarter-Quarter field.

3) The user may insert any Parcel information in a free format. The system will handle necessary justification and/or insertion of leading zeroes.

4) The user enters his/her initials at the beginning of a data entry session; this, along with the computer system's time and date is automatically added to the Parcel's information in the PC Master File whenever a Parcel is added or changed.

B. For Parcel addition operations, the user can select certain default values that are to be automatically entered into appropriate fields on the screen. In all cases, the user may overwrite these values for a particular Parcel.

C. The user has a means of "skipping ahead" over several fields in a given Parcel (those to be left untouched) to get to a field requiring data entry.

D. The user can set the Attn_Flag to call attention to a Parcel which requires further correction. (This is for problems that cannot be immediately resolved, and cannot be identified by the computer.) The user should include an explanation in Remarks. When corrections are made, the user will clear the Attn_Flag.

5. Unique_Num Management. The system handles Unique_Num assignment and ensures "uniqueness" of Unique_Nums. The system will assign the next unused number (in sequence) to any addition to the PC Master File. Unique_Nums of deleted Parcels will not be reused.

6. Affidavit Monitoring. The system includes a means of monitoring affidavit information. (Development of this set of features is not high priority, so will proceed only as time allows.)

A. This allows the user to enter and maintain all of the information needed for an affidavit parcel, should it become included in the Fire Assessments Program.

B. The following operations are available for the Aff_Review File just as they are for the PC Master File:

Lookup	Change
Addition	Duplication
Deletion	

C. This may include the collection of statistics about processed affidavits.

D. In addition, there is a means of easily transferring the Aff_Parcel data to the PC Master File.

7. Maintenance of Related Files. The system includes access to viewing the following information:

A. Billee_Name and Other_Name and corresponding code, for the large landowners. The user may also make additions, deletions, and changes to the file containing this information (Owners File).

B. Protection District or Affidavit Unit name, corresponding Prot_Area code, and corresponding type (Affidavit or District). The user may also make additions, deletions, and changes to the file containing this information (Prot_Area File).

8. Report Features. The system includes report features as follows:

A. Where screen width allows, reports are formatted for output to the screen or to the printer, at the user's choice.

B. Various types of county report forms can be generated to aid the user in data collection at the county offices.

1) These are formatted to resemble the organization of information at the county covered by the report.

2) These will include a flag or a date showing which Parcels have been updated since the last major mainframe reports were printed.

C. A library of report formats may be available for subsets of the PC Master File that have certain characteristics in common. Some may provide a means for counting Parcels and totalling acreages. Different printers and printing styles may be accommodated. Further specifics still need to be defined.

D. Reports of single Parcels are available.

9. General Error Checks. The system includes the following general error checks:

   A. The disk is checked for sufficient disk space before additions are made to any data file.

10. Updating and Downloading Mainframe Data. The system provides a means of transferring data to and from the mainframe.

   A. Only modifications to the PC Master File are sent to the mainframe. These modifications are tracked in the Transaction File.

   B. The system maintains the Transaction File; this includes keeping track of (1) all Parcel additions, deletions, and changes; (2) the changed/added Parcels which are in any part recognized as "incomplete"; and (3) the "send-status" of the Trans_Parcels. Only the most recent modification to a given Parcel is retained among the "unsent" in the Transaction File; this ensures that only current information is received by the mainframe.

   C. The system transforms records from the Transaction File into ASCII text, and into the agreed upon record format.

   D. There is a means of transforming and uploading the entire set of Master File information from the mainframe. There is also a means of checking the PC Master File data against the mainframe data to ensure consistency.

   E. There is a means of sending current Prot_Area File information to the mainframe so that it may be used in reports produced by the mainframe.

   F. Issues relating to timing of these data transfers will be addressed, and guidelines will be provided to ensure data integrity.

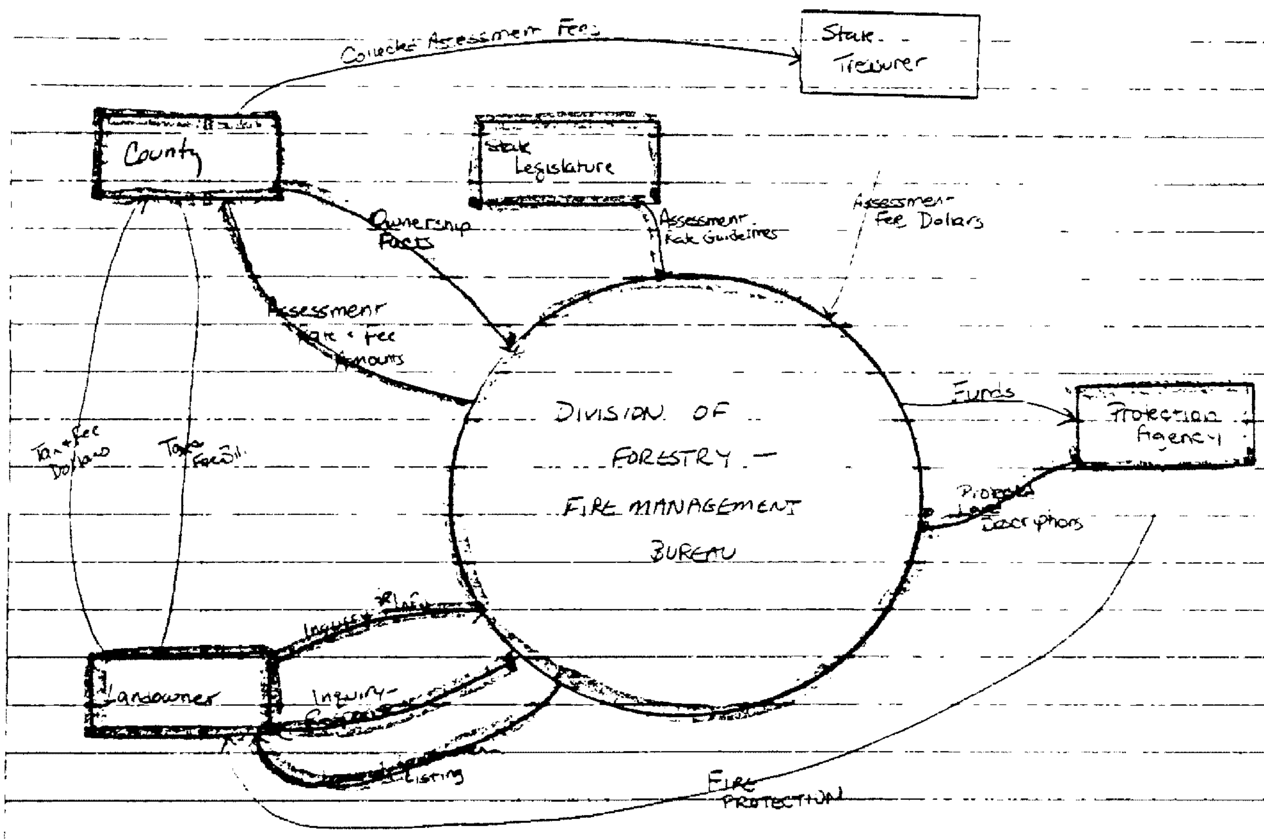
11. Backups and Disaster Recovery. The system includes a means for efficiently maintaining backups of local data, and reminds the user to maintain current backups. This issue will also be fully addressed in the User's Guide. It will also include a means of tracking and

automatically "reinstalling" changes, as deemed necessary, in case of computer failure.

12. Miscellaneous. The system will perform simple totals (of Parcels, Billee_Names, or Acres in County or Prot_Area). The system will not perform any other analysis or produce statistics on the PC Master File information; that is the responsibility of the mainframe end. There may be some processing available for recording assessment rates and estimating assessments.

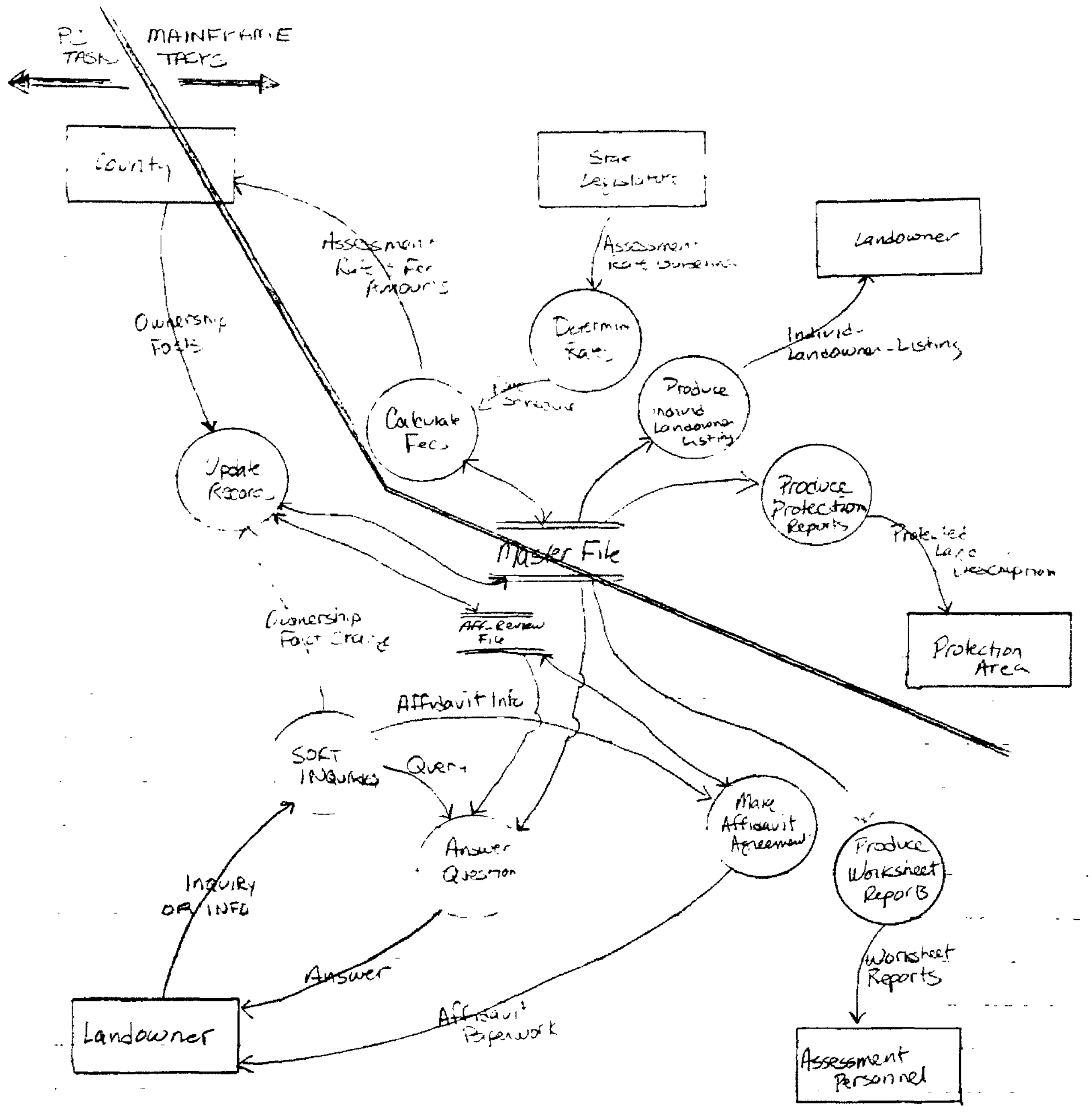
APPENDIX II

DATA FLOW DIAGRAMS

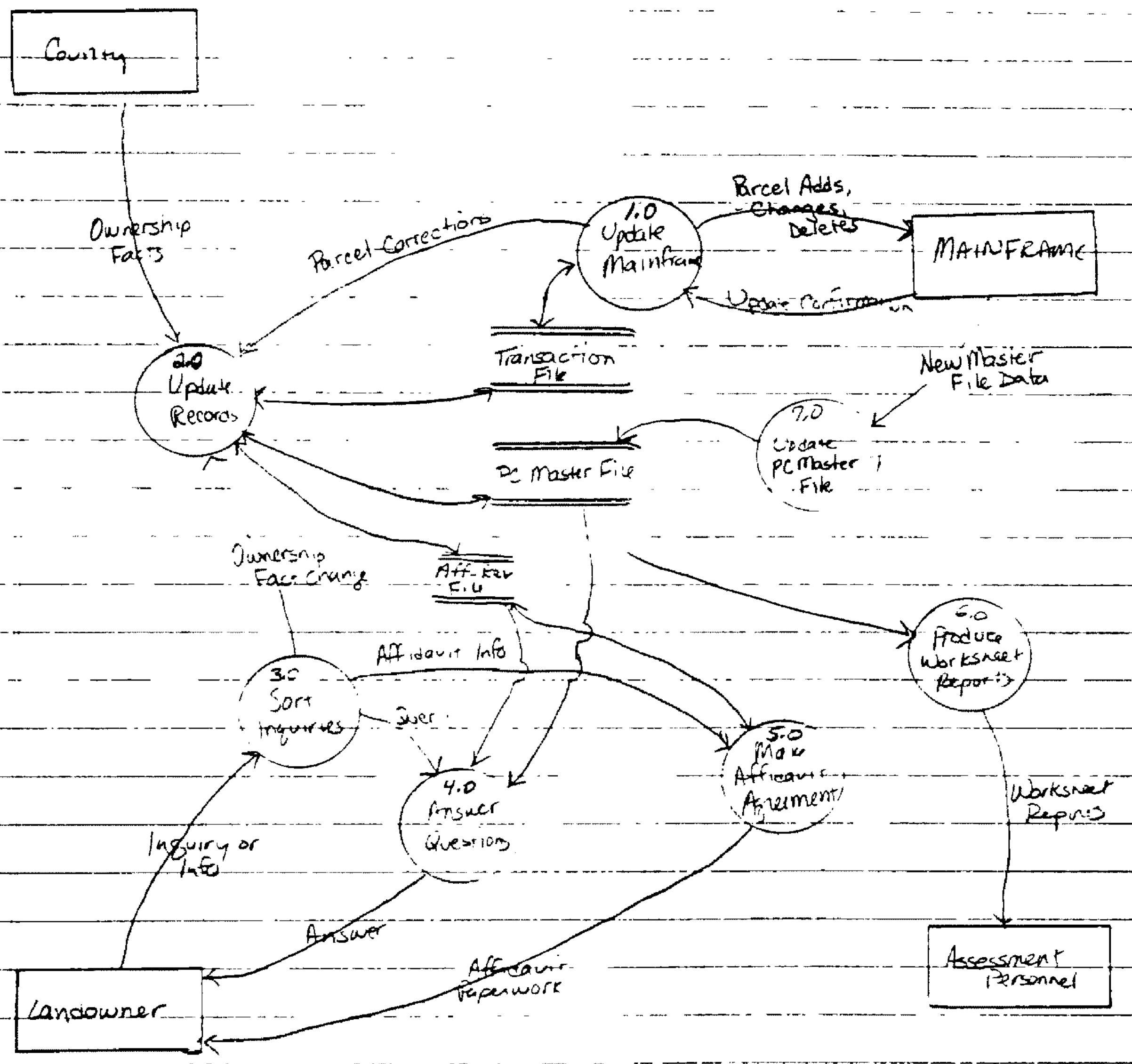


FIRE ASSESSMENTS PROGRAM

▬ Scope of Assessment Personnel concern



FIRE ASSESSMENTS PROGRAM:
SCOPE OF ASSESSMENT PERSONNEL CONCERN;
TASKS & DIVISION BETWEEN PC & MAINFRAME



FIRE ASSESSMENT PROGRAM:
PRIMARY PC TASKS

APPENDIX III

DATA DICTIONARY, FILE DICTIONARY, AND DATA ELEMENT AND FILE QUICK-REFERENCE

DATA DICTIONARY

Symbols used in the DATA DICTIONARY and FILE DICTIONARY:

* *	---->	used to offset comments
(TJFL)	---->	shows code for Type, Justification, Fill character, and Length of data in field, according to the following:

Types:
C = Character

Fill Characters:
Z = Zero
S = Space
_ = does not apply

Justification:
R = Right Justify
L = Left
_ = does not apply

Length:
number is field size

=	---->	indicates "is equivalent to"
{ }	---->	indicates "zero or more iterations of" whatever appears inside the { }
+	---->	indicates "and"
-	---->	indicates "without"
[¦]	---->	indicates "or"; eg., [This ¦ That ¦ Other] means "either This or That or Other"
...	---->	indicates the continuation of given series

Pay particular attention to the following: Where data is specifically defined to have a particular value, the system checks (during data entry, if error checking is sufficiently speedy) to be sure that a valid value is in that field. Where data is not defined to have a particular value, the system allows anything to be placed in that field. All cases where fields may be left empty are so designated ("Blank" or "Zero").

```
Action_Code         :  * (C__1)  Designates type of record
                          change *

                =   [ A * add *
                    | C * change *
                    | D * delete * ]

Addition
 _From_Pending      :  *  Parcel data for new Affidavit *

                =   Parcel


Address             :  * (CLS30)  Mailing (street or PO box)
                          address *

                =   [ mailing address
                    | Blank          ]


Aff_Action          :  * (C__6)  Date of initial action on
                          affidavit *

                =   [ Valid_Date
                    | Blank          ]


Aff_Remarks         :  * (CLS35)  User's memos about affidavit *

                =   [ memos
                    | Blank          ]


Aff_Parcel          :
                =   Parcel

                -   Non_Assess
                -   Tot_Assess
                -   Assess_Num

                +   Prev_Uni_Num
                +   Prev_Name
                +   Address
                +   City
                +   State
                +   Zip_Code
                +   Aff_Status
                +   Aff_Action
                +   Followup1
                +   Followup2
                +   Aff_Signed
                +   Aff_Remarks    ]
```

```
Aff_Signed          :  * (C__6)  Date of Billee_Name signature
                          on affidavit *

                 =     [ Valid_Date
                       | Blank            ]


Aff_Status          :  * (C__1) Designates status of record in
                          Aff_Review File *

                 =     [ P      * pending *
                       | U      * updating *
                       | S      * signed *  ]


Affidavit_
  Info              :  *  Landowner information relating to
                          establishing an Affidavit Agreement *


Affidavit_
  Paperwork         :  *  Forms and Assessment Program
                          information relating to establishing an
                          Affidavit Agreement *


Answer              :  *  Response to request for
                          information/correction by landowner *

                 =     [ {element of Parcel}
                        | {element of Aff_Parcel} ]


Assess_Num          :  * (CRZ10)  Numeric code assigned to
                          Parcel by County Assessor *

                 =     [ assessor number
                       | UNKNOWN
                       | Blank             ]


Attn_Flag           :  * (C__1) Code indicating there are
                          incomplete elements of record; to be
                          explained in Remarks *

                 =     [ X      * Parcel needs attention *
                       | Blank  * Parcel is complete *      ]
```

```
Billee_Name       :  * (CLS30)  Name of person(s) paying
                        assessment; is Buyer if Parcel is being
                        purchased under Contract for Deed,
                        otherwise is Title_Owner *

               =   Valid_Name


Blank             :  * Full field of spaces *


Buyer             :  * Person(s) purchasing Parcel under
                        Contract for Deed *


Buyer_Flag        :  * (C__1)  Identifies status of
                        Billee_Name *

               =   [ X     * Billee_Name is Buyer *
                   | Blank * Billee_Name is Title_Owner *     ]


City              :  * (CLS15) Name of mailing address city *

               =   [ city name
                   | Blank      ]


Co_Locator        :  * (CLS15)  Code (possibly Geocode) used
                        by counties to describe Parcel *

               =   [ county locator code
                   | Blank                ]


Co_Name           :  * (CLS13)  Name of a Montana county *


County            :  * (CRZ2) Code for the county of the
                        Parcel (See Appendix A) *


Date_Stamp        :  * (C__6) Date that the parcel was added
                        to file; or, if changes to the parcel
                        have been made, date of the most recent
                        change. *

               =   Valid_Date
```

```
Descript_1        :  * (C_S32) Free-form land description *

             =    [ land description
                  ¦ Blank          ]

Descript_2        :  * (C_S32) Free-form land description *

             =    [ land description
                  ¦ Blank          ]

Followup1         :  * (C__6)  Date of first followup on
                       affidavit *

             =    [ Valid_Date
                  ¦ Blank        ]

Followup2         :  * (C__6)  Date of second followup on
                       affidavit *

             =    [ Valid_Date
                  ¦ Blank        ]

Freq_Name1        :  * (CLS30)  First part (or all) of name of
                       frequently-occurring landholder for
                       which there is a Name_Code.  To be
                       inserted into Billee_Name if there is no
                       Buyer of Parcel. To be inserted in
                       Other_Name if there is a Buyer.  (See
                       Appendix A.) *

             =    Valid_Name

Freq_Name2        :  * (CLS30)  Second part of name of
                       frequently-occurring landholder for
                       which there is a Name_Code.  To be
                       inserted into Other_Name if there is no
                       Buyer of Parcel. To be omitted if there
                       is a Buyer.  (See Appendix A.) *

             =    [ Valid_Name
                  ¦ Blank      ]

Group             :  * set of Parcels having the same County
                       which require the same type of change *
```

```
Inquiry_or_
  Info            :  *  Request for information/correction by
                        landowner *

              =      [ Co_Name + Billee_Name
                     | Co_Name + Other_Name
                     | Co_Name + Section + Township + Range ]
                     +
                     [ {element of Parcel}
                      | {element of Aff_Parcel} ]


Inquiry_
 Response         :  *  Response to request for
                        information/correction or fire
                        protection by landowner *

              =      [ Answer
                      | Affidavit_Paperwork ]


Mining_Flag       :  * (C__1)  Indicates whether or not land
                        is a mining claim *

              =      [ X     * mining claim *
                     | Blank * not a mining claim * ]


Name_Code         :  * (CLS4)  Code identifying frequently-
                        occurring landholder.  See Appendix A. *


Nego_Flag         :  * (C__1)  Indicates whether Nego_Rate is
                        a total assessment or a by-acre rate *

              =      [ T     * total assessment *
                     | A        * by-acre rate *
                     | Blank * no Nego_Rate * ]


Nego_Rate         :  * (CRZ6)  Indicates negotiated amount of
                        assessment *

              =      [ six-digits
                     | Blank *  rate to be calculated on
                              mainframe  *    ]
```

```
New_Master_
  File_Data        : *  The entire contents of the mainframe's
                        Master File, in ASCII text, split into
                        files by County. *

                   =  { Parcel }


NE_NE              :  * (CRZ2)  Number of acres of this Parcel
                        in this Quarter-Quarter. (Quarter
                        designated first, then Quarter-Quarter)
                        (Normally only up to 40) *

                   =  [ [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 97 ¦ 98 ¦ 99 ]
                      ¦ Blank     * if no acres * ]


NE_NW              :  (see NE_NE)


NE_SE              :  (see NE_NE)


NE_SW              :  (see NE_NE)


NFZ                :  * (C__1)  Designates whether or not
                        parcel is classified Non-Forest-Zone *

                   =  [ X      * non-forest-zone *
                      ¦ Blank * forest-zone *       ]


Non_Assess         :  * (C__1)  code describing a non-assessed
                        protected land *

                   =  [ X      * non-assessed land *
                      ¦ Blank * assessed land   *     ]


NW_NE              :  (see NE_NE)


NW_NW              :  (see NE_NE)


NW_SE              :  (see NE_NE)


NW_SW              :  (see NE_NE)
```

```
Operator          :  * (C__3)  Upper case initials of operator
                      who added or most recently changed the
                      data. *


Other_Name        :  * (CLS30)  If Buyer_Flag = "X", this is
                      the Title_Owner and cannot be Blank.
                      Otherwise, this may be a continuation of
                      Title_Owner or Blank. *

              =   [ Valid_Name
                  | Blank       ]


Ownership_Facts :  *  Current Parcel data from County used
                      in updating PC Master File *

              =  [ { element of Parcel }
                 | Remark  ]


Ownership_
  Fact_Change     :  *  Current Parcel data from landowner
                      used in updating PC Master File *

              =  [ { element of Parcel }
                 | Remark  ]
```

```
Parcel          :
             =   Unique_Num
             +   County
             +   Buyer_Flag
             +   Billee_Name
             +   Other_Name
             +   Prot_Area
             +   Section + Township + Range
             +   School_Dist
             +   NE_NE + NE_NW + NE_SE + NE_SW + NW_NE +
                 NW_NW + NW_SE + NW_SW + SE_NE + SE_NW +
                 SE_SE + SE_SW + SW_NE + SW_NW + SW_SE +
                 SW_SW +
             +   Tot_Acres
             +   Assess_Num
             +   Descript_1 + Descript_2
             +   Tot_Assess
             +   Nego_Rate
             +   Nego_Flag
             +   Non_Assess
             +   NFZ
             +   Co_Locator
             +   Subdiv
             +   Mining_Flag
             +   Attn_Flag
             +   Operator + Date_Stamp + Time_Stamp


Parcel_Adds_
  Changes_
    Deletes     :  *  Updates to Master File;  Trans_Parcels
                      which have Send_Status = R   *

             =   { Trans_Parcel }


Parcel_
  Corrections   :  *  Information about Parcel which was not
                      successfully added to Master File *

             =   { { element of Trans_Parcel } + error
                 message }


Parcel_Deletion
  _Or_Change    :  *  Parcel data and corrected information
                      *

             =   Parcel
             +   corrected information about some part of
                 Parcel
```

```
Prev_Uni_Num     :  * (CRZ6)  If Affidavit update, is
                      Unique_Num of parcel before updating was
                      required; otherwise is blank *

             =    [ Unique_Num
                  ¦ Blank          ]


Prev_Name        :  * (CLS30)  If Affidavit update, is
                      Billee_Name of parcel before updating
                      was required; otherwise is blank *

             =    [ Valid_Name
                  ¦ Blank          ]


Prot_Area        :  * (CLS3)  Code identifying Fire
                      Protection District or Affidavit Unit
                      which protects the parcel.  (See
                      Appendix A) *


Prot_Name        :  * (CLS43)  Name of Fire Protection
                      District or Affidavit Unit which has a
                      corresponding Prot_Area (code).  (See
                      Appendix A.)  *


Prot_Type        :  * (C__1)  Designates Fire Protection
                      District or Affidavit Unit  *

             =    [ D  * Fire Protection District *
                            ¦ U  * Affidavit Unit *    ]


Quarter-Quarter :  * Part of legal land description.  First
                     quarter designation is Quarter of
                     Section, second is quarter of Quarter.
                     Eg., NW_NE refers to the NE ¼ of the NW
                     ¼ of the Section. *

             =    [ NE_NE ¦ NE_NW ¦ NE_SE ¦ NE_SW ¦ NW_NE ¦
                  NW_NW ¦ NW_SE ¦ NW_SW ¦ SE_NE ¦ SE_NW ¦
                  SE_SE ¦ SE_SW ¦ SW_NE ¦ SW_NW ¦ SW_SE ¦
                  SW_SW ]


Quarter          :  * One fourth of a Section; the total of
                     all Quarter-Quarters beginning with the
                     same Quarter.
                     eg:  NE_NE + NE_NW + NE_SE + NE_SW  *
```

```
Query            :  *  Inquiry about Fire Assessment related
                       issue *

Range            :  * (CRZ4)  Legal land designation of
                       Parcel's Range; must be within the
                       spread of Montana Ranges.  *

              =    [ [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 60 ¦ 61 ¦ 62 ]
                       + [ 0 ¦ 5 ] + E
                   ¦ [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 33 ¦ 34 ¦ 35 ]
                       + [ 0 ¦ 5 ] + W ]

Remarks          :  * (CLS100)  Notes made by user about a
                       Parcel.  Cannot be Blank.  *

School_Dist      :  * (C_S7)  Indicates school district of
                       parcel.  The first 2 positions are
                       numeric, with leading zero if necessary.
                       Any subsequent unused spaces are left
                       blank. *

              =    [    [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 97 ¦ 98 ¦ 99 ]
                          + [ other school district data ¦
                       spaces ]
                   ¦ UNKNOWN                              ]

SE_NE            :  (see NE_NE)

SE_NW            :  (see NE_NE)

SE_SE            :  (see NE_NE)

SE_SW            :  (see NE_NE)

Section          :  * (CRZ2)  Legal land designation of
                       parcel's Section *

              =    [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 34 ¦ 35 ¦ 36 ]
```

```
Send_Status        :  * (C__1)  Indicates whether or not
                        Trans_Parcel has been processed for
                        sending to mainframe *

                =     [ N * not ready *
                      | R  * ready *
                      | D  * done *
                      | C  * confirmed *  ]

Sent_Date          :  * (C__6)  Date on which Trans_Parcel was
                        processed to be sent to mainframe *

                =     [ Valid_Date
                      | Blank      ]

State              :  * (C__2)  national two-letter state code
                        corresponding to  mailing address *

                =     [ state code
                      | Blank      ]

Subdiv             :  * (C??5)  subdivision identification;
                        Department of Revenue subdivision codes
                        *

                =     [ subdivision code
                      | Blank            ]

SW_NE              :  (see NE_NE)

SW_NW              :  (see NE_NE)

SW_SE              :  (see NE_NE)

SW_SW              :  (see NE_NE)

Time_Stamp         :  * (C)  Time that the parcel was added;
                        or, if changes to the parcel have been
                        made, time of the most recent change *

Title_Owner        :  * The person(s) who has legal title to
                        the parcel. *
```

```
Tot_Acres         :  * (CRZ4)  Total acreage of parcel.  If
                      any Quarter-Quarter shows acreage,
                      Tot_Acres must equal sum of Quarter-
                      Quarters unless Attn_Flag = "X".  May be
                      zero only if Attn_Flag = "X". *

             =    [ 0000 ¦ 0001 ¦ 0002 ¦ ... ¦ 9998 ¦ 9999 ]


Tot_Assess        :  * (CRZ6)  Value of assessment on that
                      Parcel *

             =    [ six-digits
                  ¦ Blank * Assessment not yet calculated *
                          ]


Township          :  * (CRZ4)  Legal land designation of
                      parcel's Township;  must be within the
                      spread of Montana Townships.  *

             =    [ [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 35 ¦ 36 ¦ 37 ]
                     +  [ 0 ¦ 5 ]  +  N
                  ¦ [ 01 ¦ 02 ¦ 03 ¦ ... ¦ 15 ¦ 16 ¦ 17 ]
                     +  [ 0 ¦ 5 ]  +  S ]


Trans_Parcel      :
             =    Parcel
             +    Action_Code
             +    Send_Status
             +    Sent_Date


Unique_Num        :  * (CRZ6)  A unique number assigned by the
                      system to a Parcel for identification
                      purposes. *

             =    six-digits


Update_
  Confirmation    : * Confirmation from mainframe that updates
                      are in place *

                          =   { Trans_Parcel }
                      + { error messages }
```

```
Valid_Date          :  * a 6-digit representation of date -
                          YYMMDD *

              =     [ 00 | 01 | 02 | ... | 97 | 98 | 99 ]
                    + [ [ 01 | 03 | 05 | 07 | 08 | 10 | 12 ]
                          + [ 01 | 02 | 03 | ... | 29 | 30 | 31 ]
                      | [ 04 | 06 | 09 | 11 ]
                          + [ 01 | 02 | 03 | ... | 28 | 29 | 30 ]
                      | 02 + [ 01 | 02 | 03 | ... | 27 | 28 | 29
                    ] ]

Valid_Name          :  * (CLS30)  Name(s) of one or more persons
                          or an organization.  Must not begin with
                          a space.  Considerable effort is
                          required to ensure consistency in the
                          method used to record these names.
                          However, since the format will not be
                          monitored by the computer, it is a
                          matter to be addressed in the User
                          Guide. *

Worksheet_
  Reports           :  *  Reports generated by system;  details
                          to be determined *

Zip_Code            :  * (C__5)  5 digit zip code for mailing
                          address *

              =     [ zip code
                    | Blank      ]

Zero                :  * Full field of zeros *
```

FILE DICTIONARY

```
Aff_Review File :  * Affidavit info needing attention *
              =    { Aff_Parcel }

                   INDEX KEYS  :   County + Billee_Name
                                   County + Fn1(Township) +
                                   Fn2(Range) + Section
                                   County + Unique_Num
                                   County + Date_Stamp


Counties File   :  * County codes & corresponding names *
              =    { County + Co_Name }

                   INDEX KEY   :   Co_Name


Owners File     :  * Name_Codes & corresponding names *
              =    { Name_Code + Freq_Name1 + Freq_Name2 }

                   INDEX KEY   :   Freq_Name1


PC Master File  :  * A convenient term for referring to the
                     primary data files for microcomputer
                     system *

              =    { County_File }


County File     :  * Data file of Parcels in same County *
              =    { Parcel }

                   INDEX KEYS  :   Billee_Name + Other_Name
                                   Township + Range + Section +
                                       Descript1 + Descript2
                                   Assess_Num
                                   Unique_Num
                                   School_Dist +
                                       Fn(Billee_Name,
                                       Buyer_Flag, Other_Name)
                                   Date_Stamp
                                   Attn_Flag


Prot_Area File  :  * Protection Districts or Affidavit Units
                     & corresponding names *
              =    { Prot_Area + Prot_Name + Prot_Type }

                   INDEX KEY   :   Prot_Name
```

```
Remarks File      :  * Remarks about a Parcel *
              =   { Unique_Num + Remark }

                  INDEX KEY    :   Unique_Num


Sent File         :  *  Trans_Parcels (microcomputer file
                        format) that have been sent to mainframe
                        but not yet confirmed *
              =   { Trans_Parcel }

                  INDEX KEY    :   County + Sent_Date


Transaction File:    * Resulting forms of Parcels which have
                        been added, changed, or deleted but not
                        archived *
              =  { Trans_Parcel }

                  INDEX KEYS   :   County + Unique_Num
                                   County + Date_Stamp
```

DATA ELEMENT AND FILE QUICK-REFERENCE

Files:

M	=	PC Master File
T	=	Transaction File & Sent File
R	=	Remarks File
A	=	Aff_Review File
C	=	County File
P	=	Prot_Area File
O	=	Owners File

	M	T	R	A	C	P	O	
Action_Code		X						
Address				X				
Aff_Action				X				
Aff_Remarks				X				
Aff_Signed				X				
Aff_Status				X				
Assess_Num	X	X						
Attn_Flag		X	X		X			
Buyer_Flag	X	X		X				
Billee_Name	X	X		X				
City					X			
Co_Locator	X	X		X				
Co_Name					X			
County	X	X		X	X			
Date_Stamp	X	X		X				
Descript_1	X	X		X				
Descript_2	X	X		X				
Followup1					X			
Followup2					X			
Freq_Name1							X	
Freq_Name2							X	
Mining_Flag	X	X		X				
Name_Code								X
Nego_Flag		X	X		X			
Nego_Rate		X	X		X			
NE_NE	X	X		X				
NE_NW	X	X		X				
NE_SE	X	X		X				
NE_SW	X	X		X				
NFZ	X	X		X				
Non_Assess	X	X						
NW_NE	X	X		X				
NW_NW	X	X		X				
NW_SE	X	X		X				
NW_SW	X	X		X				

DATA ELEMENT AND FILE QUICK-REFERENCE (continued)

Files:

M	=	PC Master File
T	=	Transaction File & Sent File
R	=	Remarks File
A	=	Aff_Review File
C	=	County File
P	=	Prot_Area File
O	=	Owners File

	M	T	R	A	C	P	O
Operator		X	X		X		
Other_Name	X	X		X			
Prev_Uni_Num					X		
Prev_Name					X		
Prot_Area		X	X		X		X
Prot_Name							X
Prot_Type							X
Range	X	X		X			
Remarks			X				
School_Dist	X	X		X			
SE_NE	X	X		X			
SE_NW	X	X		X			
SE_SE	X	X		X			
SE_SW	X	X		X			
Section	X	X		X			
Send_Status		X					
Sent_Date			X				
State				X			
Subdiv	X	X		X			
SW_NE	X	X		X			
SW_NW	X	X		X			
SW_SE	X	X		X			
SW_SW	X	X		X			
Time_Stamp	X	X		X			
Tot_Acres		X	X		X		
Tot_Assess	X	X					
Township		X	X		X		
Unique_Num	X	X	X	X			
Zip_Code					X		

APPENDIX A

VALID CODES OF DATA ELEMENTS

```
County         :   * (CRZ2) Code for the county of the Parcel *
               =
               [    01   * Beaverhead *
               |    02   * Bighorn *
               |    03   * Blaine *
               |    04   * Broadwater *
               |    05   * Carbon *
               |    06   * Carter *
               |    07   * Cascade *
               |    08   * Chouteau *
               |    09   * Custer *
               |    10   * Daniels *
               |    11   * Dawson *
               |    12   * Deer Lodge *
               |    13   * Fallon *
               |    14   * Fergus *
               |    15   * Flathead *
               |    16   * Gallatin *
               |    17   * Garfield *
               |    18   * Glacier *
               |    19   * Golden Valley *
               |    20   * Granite *
               |    21   * Hill *
               |    22   * Jefferson *
               |    23   * Judith Basin *
               |    24   * Lake *
               |    25   * Lewis & Clark *
               |    26   * Liberty *
               |    27   * Lincoln *
               |    28   * McCone *
               |    29   * Madison *
               |    30   * Meagher *
               |    31   * Mineral *
               |    32   * Missoula *
               |    33   * Musselshell *
               |    34   * Park *
               |    35   * Petroleum *
               |    36   * Phillips *
               |    37   * Pondera *
               |    38   * Powder River *
               |    39   * Powell *
               |    40   * Prairie *
               |    41   * Ravalli *
               |    42   * Richland *
               |    43   * Roosevelt *
               |    44   * Rosebud *
               |    45   * Sanders *
```

```
            |   46   * Sheridan *
            |   47   * Silver Bow *
            |   48   * Stillwater *
            |   49   * Sweet Grass *
            |   50   * Teton *
            |   51   * Toole *
            |   52   * Treasure *
            |   53   * Valley *
            |   54   * Wheatland *
            |   55   * Wibaux *
            |   56   * Yellowstone * ]



Name_Code        :  * (CLS4)  Code identifying frequently-
                      occurring landholder *
            =
            [   BIA      * Bureau of Indian Affairs *
            |   BLM      * Bureau of Land Management *
            |   CT       * Champion Timberlands
                           Champion Intl Corp *
            |   FWP      * MT Dept Fish Wildlife & Parks *
            |   PC       * Plum Creek Timber Co *
            |   SF       * State of Montana
                           Forests *
            |   SL       * State of Montana
                           Lands *
            |   SLL      * Stoltze Land & Lumber Co *
            |   USFS     * U S Forest Service *
            |   others to be defined        ]
```

```
Prot_Area          :  * Code identifying Fire Protection
                        District or Affidavit Unit which
                        protects the parcel. *
                =
                [    AD   * Avon District *
                |    APD  * Anaconda Forest Protection District *
                |    AU   * Anaconda Unit *
                |    B    * Blackfoot Forest Protection District *
                |    BB   * Bitterroot-Blackfoot *
                |    BF   * Beaverhead National Forest *
                |    BRF  * Bitterroot National Forest *
                |    BU   * Big Fork District *
                |    CF   * Custer National Forest *
                |    DF   * Deer Lodge National Forest *
                |    EU   * Eureka Unit *
                |    FN   * Flathead-Northern Montana *
                |    FF   * Flathead National Forest *
                |    GBD  * Gallatin Bridger District *
                |    GF   * Gallatin National Forest *
                |    GRD  * Gallatin River District *
                |    HB   * Helena-Blackfoot *
                |    HCD  * Helena Continental Divide *
                |    HD   * Helena District *
                |    HF   * Helena National Forest *
                |    HFD  * Helena Forest District *
                |    HU   * Helena Unit *
                |    I    * BIA - Flathead *
                |    K    * Northern Montana Forest Protection
                               District *
                |    KF   * Kootenai National Forest *
                |    KN   * Kootenai-Northern Montana *
                |    LB   * Lolo-Blackfeet *
                |    LCD  * Lincoln Continental District *
                |    LCF  * Lewis & Clark National Forest *
                |    LF   * Lolo National Forest *
                |    LN   * Lolo-Northern Montana *
                |    MB   * BLM - Butte *
                |    ML   * BLM - Lewistown *
                |    MM   * BLM - Miles City *
                |    SCD  * State Continental Divide *
                |    SN   * Stillwater-Northern Montana *
                |    SU   * Stillwater District *
                |    SW   * Swan District *
                |    YD   * Yellowstone District * ]
```

APPENDIX B

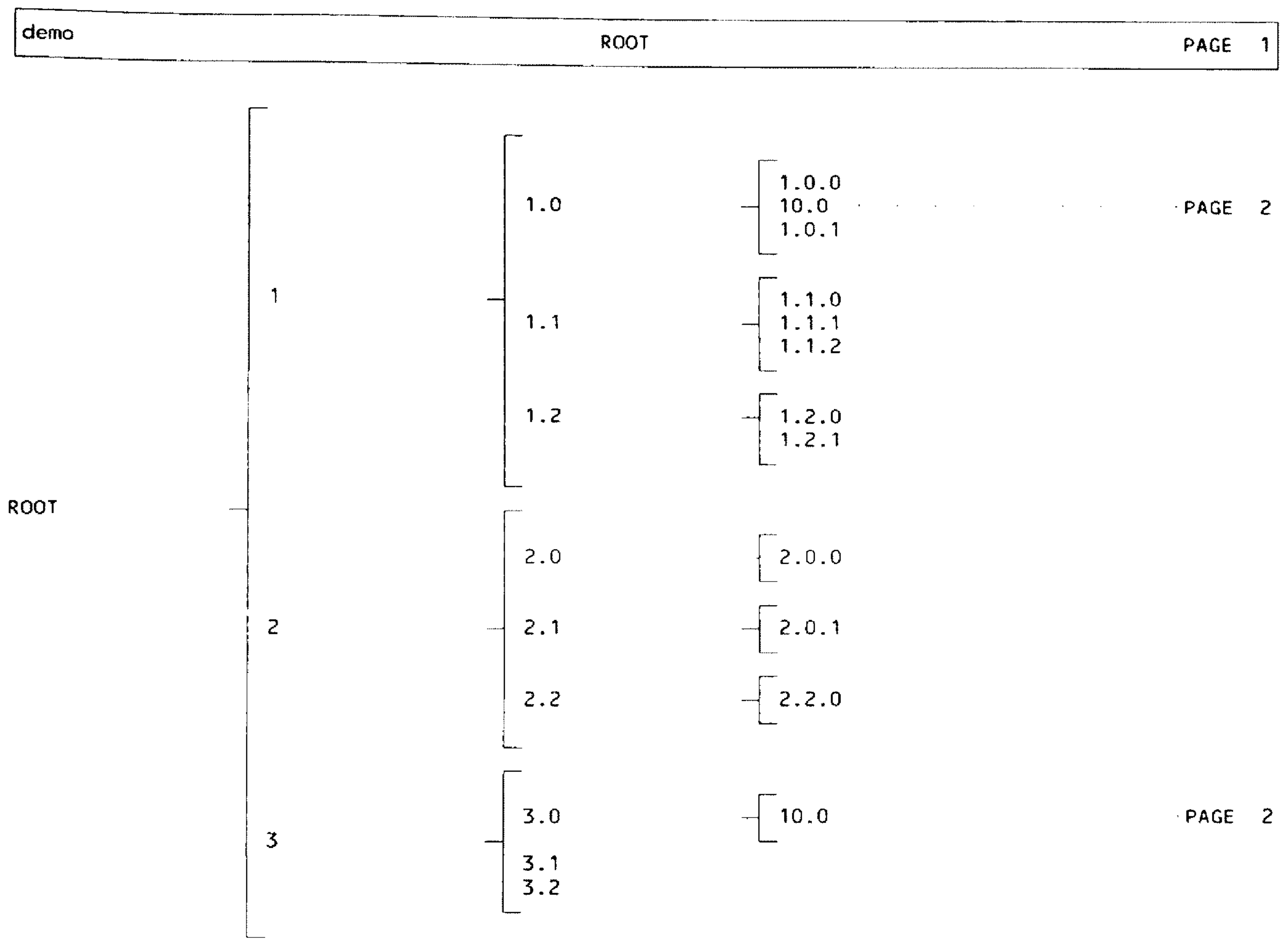
RECORD FORMAT FOR MAINFRAME

FIELD NUM	POS	LEN	TYPE	DESCRIPTION
1	1	6	C	Unique_Num
2	7	2	C	County
3	9	1	C	Buyer_Flag
4	10	30	C	Billee_Name
5	40	30	C	Other_Name
6	70	3	C	Prot_Area
7	73	2	C	Section
8	75	4	C	Township
9	79	4	C	Range
10	83	7	C	School_Dist
11	90	2	C	NE_NE
12	92	2	C	NE_NW
13	94	2	C	NE_SE
14	96	2	C	NE_SW
15	98	2	C	NW_NE
16	100	2	C	NW_NW
17	102	2	C	NW_SE
18	104	2	C	NW_SW
19	106	2	C	SE_NE
20	108	2	C	SE_NW
21	110	2	C	SE_SE
22	112	2	C	SE_SW
23	114	2	C	SW_NE
24	116	2	C	SW_NW
25	118	2	C	SW_SE
26	120	2	C	SW_SW
27	122	4	C	Tot_Acres
28	126	10	C	Assess_Num
29	136	32	C	Descript_1
30	168	32	C	Descript_2
31	200	6	C	Tot_Assess
32	206	6	C	Nego_Rate
33	212	1	C	Nego_Flag
34	213	1	C	Non_Assess
35	214	1	C	NFZ
36	215	15	C	Co_Locator
37	230	5	C	Subdiv
38	235	1	C	Mining_Flag
39	236	1	C	Attn_Flag
40	237	3	C	Operator
41	240	6	C	Date_Stamp
42	246	4	C	Time_Stamp
43	250	1	C	Action_Code
44	251	1	C	Send_Status
45	252	6	C	Sent_Date
	258	<--------		TOTAL RECORD LENGTH

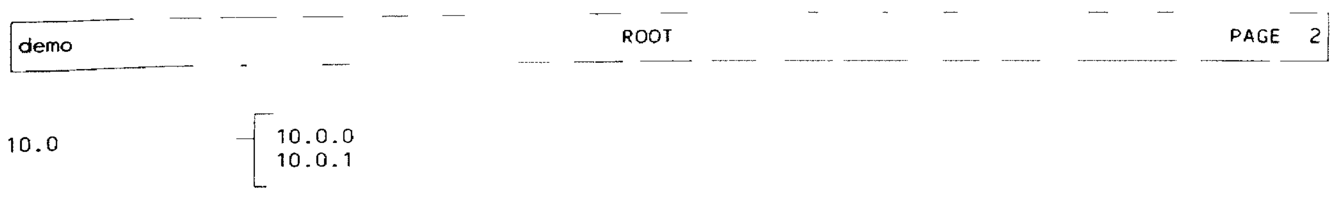
APPENDIX IV

BRACKETS OUTPUT SAMPLE

Page 1 of output:



Page 2 of output:


demo
ROOT
PAGE 2
10.0
10.0.0
10.0.1