Hub of Systems Development and Documentation

Stephen M. Butler
Weyerhaeuser Company

INTRODUCTION

Throughout the traditional application development life cycle information about the project is generated phase by phase in a series of documents. Each phase has to adapt this information into its own format and documents. In some cases, this is a complete rework of the information and the way it is presented. The best examples are the transition from ANALYSIS to DESIGN and the documentation from DEVELOPMENT to PRODUCTION. Some claim that this last is neither valuable nor useful in its current form!

Dictionary support for the various phases of the application life cycle is not new. Generally, such support has been specific to each phase; but, the interfaces still are paper documents. Thus the dictionary support effort of earlier phases are duplicated for each subsequent step. Use of the dictionary for the interface will allow the subsequent phases to build upon the earlier efforts rather than duplicate them.

This paper will first look at the types of dictionary support used at each phase of the application life cycle and what could be done to use the dictionary for the interface between these phases. Then the paper will focus on how this might be implimented using DICTIONARY/3000. Following that will be a recap of specific tools that will be needed. Many of these do not exist today.

APPLICATION LIFE CYCLE--DICTIONARY SUPPORT

This author has chosen to break the application life cycle down into the following phases:

PLANNING
ANALYSIS
DESIGN
CODE
PRODUCTION
END-USER QUERY
MAINTENANCE
REPLACEMENT

This may not be the precise arrangement for any given development; but, they provide a common framework that the reader can tailor to his particular need. **ANALYSIS**

According to DeMarco, the most important product of this phase is the specification document. There are a number of ways to get there two of which (Holland and DeMarco) will be addressed here.

Holland's top down approach focuses on the processes occuring within the business. This approach is primarily used by Weyerhaeuser to define the logical business model. A given business is divided into the main FUNCTIONs needed to support that business. Each FUNCTION is divided into PROCESSes which are further divided into ACTIVITYs. At this point, the entity data requirements are defined for each ACTIVITY.

DeMarco's approach for "Structured Analysis and System Specification" is primarily concerned with DATA FLOWS and secondarily with the PROCESSES that modify the data within the DATA FLOW.

Both of these approaches are heavily graphics oriented. They are also iterative. Thus, there is a lot of maintenance to the graphic representation of the logical model being developed.

In the analysis stage the dictionary definitions and known attributes of the items (FUNCTION, PROCESS, ACTIVITY, ENTITY DATA FLOW, etc.) are continually being refined as the analyst gains additional knowledge regarding the project. Since the relationship of these items to each other as shown in the graphic representation is an attribute of the items, this relationship should be maintained in the dictionary. It then makes sense that the dictionary (or another tool making inquiry to the dictionary) would draw the graphic representation--be it the logical business

model, the data flow diagram (DFD), or the logical data structure diagram (DSD) or entity relationship diagrams.

Dictionary use may be complicated during this phase by the use of an analysis tool that supports one or more of the structured approaches to system analysis. If that is the case, a facility to transfer information bi-directionally between the analysis tool and the dictionary is a necessity.

DESIGN

This phase turns the logical model of the analysis phase into a physical model. Thus the DSD become databases and other files. The DFD is turned into a FUNCTIONAL CHART (STRUCTURE CHART, ORG CHART) which is refined until the result is functionally valid for the physical implementation that is envisioned. The STRUCTURED SPECIFICATION is not readily turned into a useful FUNCTIONAL CHART even when transaction and transform analysis are used. The output of the analysis phase is logically oriented and many of the physical aspects of the system are ignored. Also ignored are the control paths, the exception routines, etc., that the physical model must address.

The dictionary is updated with the additional physical attributes of the system as they become known. In addition, the normalized (hopefully) logical DSD is used to create the physical DSD which is also maintained in the dictionary. It would be advantageous if a tool could be used to make the first cut at the transformation from logical to physical data structure.

The FUNCTIONAL CHART that is one of the outputs of the design phase is a rather physical representation of the final system of programs and procedures. Thus the actual names of the routines, their relation to each other, and the data elements used (files accessed) are known. This information should be put into the DICTIONARY in a form that makes updating fairly easy. The charts could be one of the outputs of the DICTIONARY or some other tool that would make inquiry to the DICTIONARY.

CODE

While changes continue to be made in the design--and these should be updated in the DICTIONARY version of the STRUCTURED SPECIFICATIONS and FUNCTIONAL CHART--the major DICTIONARY support requirements for this phase include (but not limited to):

- Listing the detail specifications for procedures or modules the programmer is coding.
- Code generation for the table handling portions of the system.
- Code generation according to the detailed design specs.
- Creation and maintenance of language specific copylib or include files. It would be better if the individual language compilers would accept directives in the source code from which the compiler would determine what type of inquiry to make into the DICTIONARY and which information to copy from the DICTIONARY into the source.
- Compile stream generator. Most, if not all, of the information needed to compile a program (or the entire system) is in the DICTIONARY (or can be easily added). This would negate the headache of maintaining a seperate set of job streams for compilation.
- Continuing maintenance of the documentation regarding the logical and physical aspects of the system.

PRODUCTION

By this phase most of the DICTIONARY maintenance is done. There are several items that are of use to this phase:

- Batch job stream documentation. Rather than maintain an operator guide book that documents each step of a batch job the documentation should be in the dictionary and made accessible to the operator.
- On-line help facility within the DICTIONARY (or documented where it is maintained).
- User manuals maintained within the DICTIONARY (or documented where it is maintained).

The argument is that all information regarding the system should be documented in the DICTIONARY. This may not be possible for all types of information. In that case, effort should be made to document in the DICTIONARY where those additional items of information can be located. In addition, there should be a minimum of redundant information both within the DICTIONARY and between the DICTIONARY and the additional items of information.

MAINTENANCE

The DICTIONARY is now a repository of information to help the maintenance crew do its job. Inquiry can be made as to what 'things' are affected if changes are made to a specific 'thing'. This phase becomes a mini-life cycle for each change envisioned. The changes are walked through the STRUCTURE

SPECIFICATION to get the impact on the logical structure. Next the FUNCTIONAL CHART is modified to locate the changes on the physical structure. Finally, the related documentation entries are updated to reflect the change, and it is implemented using many of the concepts noted above.

REPLACEMENT

The application life-cycle has now come full circle. The DICTIONARY has one more major function before the replacement system takes over. It serves as the major source of information to be downloaded to the analysis tools. Also, many of the structures and other pieces of information/documentation can be salvaged. Like the phoenix rising from the ashes, the DICTIONARY of the new system is an outgrowth and metamorphosis of its previous self. The following table documents the major items that should be contained in the dictionary.

Planning

- Enterprise Model
  * Function
  * Process
  * Activity
  * Info. requirements
- Documents
- Entities
- Subject databases
- Business data elements

Analysis

- Logical/Functional Spec.
  * Data flows
  * Data processes
  * Logical files (logical DSD)
- Logical data elements

Design

- Physical Spec.
- Transactions
- Modules
- Physical file
- Records
- Screens
- Reports
- Physical data elements

Code

- Procedures
- Jobs
- Jobsteps
- Programs
- Processing data elements

Maintenance

- Additions, changes, and deletions to above specs. and documents
- +/- data element changes

The data elements hold all these documents together. They are the common thread pervading the entire life cycle. As such, it is essential to document their evolution from the conceptual 'Business Data Elements' to the quantified 'Physical Data Elements' and 'Processing Data Elements'.

DICTIONARY/3000 IMPLEMENTATION

Understanding possible methods to impliment the above ideas within DICTIONARY/3000 requires a careful investigation of what is available within the DICTIONARY. The ENTITY USAGE CHART (Figure 1) shows the conceptual arrangement of DICTIONARY/3000. (For the internal arrangement and physical layout see DICTIONARY/3000 Internal Structure--Figure 2; and DICTIONARY/3000 Schema--List 1.)

Just as DICTIONARY is the hub of the application life-cycle, the **ELEMENT** entity is the hub of the dictionary. The ELEMENT entity is oriented toward the physical data elements. Every attribute of ELEMENT has an HP defined usage--most of them tied rather tightly with the RAPID products. The TYPE attribute of ELEMENT can be left undefined (which means the other physical attributes are also undefined).

The **FILE** entity describes the different file structures that are available on the HP 3000. For some of these structures (such as IMAGE database and VPLS form files) the internal members are described as file types which indicate which elements are used and in what order they occur. Every attribute of the FILE entity has an HP defined meaning.

The **CLASS** entity defines the security for the ELEMENT and FILE entries. All security structures of IMAGE are supported. If there is no security defined at the FILE entry level for a particular data set, then the security for that data set is the resultant matrix of the securities for the data elements contained in that data set.

The TYPE attribute has no HP defined meaning--rather, whatever is meaningful to you. The field should be used to organize the security classes according to some classification of their usage. For example, the following types would be for the indicated usage: PROG - programatic access, the program is the end user rather than using a password on behalf of a user.

> The concept is used by the RAPID compilers and especially by INFORM. It doesn't matter what capability the user has, the program itself needs this access in order to do its thing.

MNGR - for the user who has ultimate responsability for the entire database. A program may even have this hard coded but the end result is that the program works on behalf and at the behest of the data base manager.

USER - level of access for some user. The CLASS-NAME would further define what user this password is for. There may even be programs set up for the user which have the password hard coded in them.

MANT - maintenance level. For the person or group that maintains a certain section of the data base. Used for fixups, corrections, etc. above and beyond the USER level. Indeed, this may be the same as MNGR for certain sections of the data base.

UTIL - a low level access for general purpose utilities to do structure checks, etc. These utilities would never change the database and would even have no predefined

xxxx - idea of the database structure. send ideas to the author.

The **GROUP** entity (aka INFORM GROUPS) defines the end user view for ad hoc reporting. It has the extra capability to indicate what file the associated elements reside in. It is only when the entry is related to $MENU or one of its children (grandchildren, etc.) that INFORM will include it in a menu of groups. When not in that structure, GROUP entries could be used to document other types of end user views. These would mainly be ordered lists of elements that reside on a report or other type of document. Remember that screens are documented in the FILE entries. The TYPE attribute is the only field for which HP has not defined a meaning. As noted under the CLASS discussion, the attribute could be used to indicate that a document was being documented. In addition, the TYPE could be used to classify the INFORM GROUPS. That will be left as an excercise for the reader.

LOCATION is little used (if used at all). Every field has an HP defined meaning--though CPU is not as HP oriented as ACCOUNT and GROUP. There is no TYPE field to help classify the LOCATION. If there had been, it could be used to indicate which LOCATION was SOURCE, OBJECT, DATA, USL, etc. Such cannot be done directly. A standard GROUP naming convention will help. This will be addressed further under the PROCEDURE discussion below.

There are two entities left (CATEGORY and PROCEDURE) for which many of the attributes have defined meanings. The usage of these is not well defined and, in many cases, not even attempted. It is in these that the structures mentioned in the Application Life Cycle section will be placed.

CATEGORY will document all the logical information. Things like the BUSINESS MODEL, DATA FLOW, and LOGICAL DATA STRUCTURE are prime candidates. Each entry will need a unique primary name that exists no other place within the CATEGORY structure of DICTIONARY. Thus a DATA FLOW member cannot have the same name as that of a BUSINESS MODEL member.

A more detailed look at CATEGORY shows these attributes:

image CATEGORY - a twenty (20) character field that is the primary name of the member entry. For the BUSINESS MODEL, the name is that of a FUNCTION, a PROCESS, an ACTIVITY, etc. This is upper case alpha-numeric.

CATEGORY-PARENT - the name of a CATEGORY which will serve as a parent in the relationship between two categories.

CATEGORY-CHILD - the name of a CATEGORY that is the child in the relationship between two categories.

CATEGORY-NAME - a fifty (50) character long title for the CATEGORY. This is descriptive of the CATEGORY entry without being part of the description. This is upper case alpha-numeric.

CATEGORY-RESP - a twenty (20) character alpha-numeric field that is the name of the person, department, or other that is responsible for the information represented by the meta-data.

CATEGORY-TYPE - a four (4) character upper case alpha-numeric field used to indicate the classification of this CATEGORY.

NOTE: The DICTIONARY/3000 command to create a CATEGORY is CREATE CATEGORY. To relate two categories together (as noted below) use RELATE CATEGORY. To associate an ELEMENT with a CATEGORY use ADD CATEGORY.

A standard set of CATEGORY-TYPE values are:

BUSINESS MODEL

BSPM - BUSINESS SYSTEM PLAN MODEL. This is the highest level of a BUSINESS MODEL. There is one of these for each business model in DICTIONARY. The lower levels consist of functions, processes, activities, etc.

FCNT - FUNCTION within the BSP. This is a child to a BSPM and parent to ACTN.

ACTN - ACTION within the FUNCTION. As with the FUNCTION, the CATEGORY named in this entry is of this type.

INFO - INFO REQUIREMENTS. This CATEGORY names an information requirement of the action to which this is a child.

The information requirements may be associated with the entries in the ELEMENT entity. This is done by the ADD CATEGORY command within DICTIONARY/3000.

LOGICAL DATA MODEL

LDSD - LOGICAL DATA STRUCTURE DIAGRAM. This is the highest level for the LDSD. The rest of the structure is made up of a table of files containing elements. This is the primary method of showing relational (ie, 3rd/4th Normal Form) data structures.

LDF - LOGICAL DATA FILE. These are children of the corresponding LDSD. They define the logical data files within the data structure.

LDEx - LOGICAL DATA ELEMENT. These are the children of the LDF's. The 'x' defines the type of logical element being defined.

P - This logical element is the primary key. There is only one primary key. It behaves the same as any of the secondary keys but has been chosen by the analyst/designer to serve as the primary way of accessing the data.

S - This logical element is a secondary key. There may be more than one secondary key; but, each is capable of being the identifier as is the primary key.

N - This logical element is not a key.

The data elements that make up each LDEx are associated to the appropriate category. This is the link back to the central hub.

SUBJECT DATA BASE

SDB - SUBJECT DATA BASE. This is the highest level for a given subject data base. It corresponds to the BASE type in the FILE entity.

SDF - SUBJECT DATA FILE. These are the files within the subject data base. They correspond to the AUTO, MAST, or DETL type entries in the FILE entity.

SDEx - SUBJECT DATA ELEMENT. This is the low level. It is similar to the low level of the LOGICAL DATA MODEL in that the rules for 'x' above also apply here with logical element changed to subject element. The data elements within ELEMENT that make up each SDEx are associated via the ADD CATEGORY command.

NOTE: At this point the question is raised--Why both a LOGICAL DATA MODEL and SUBJECT DATA BASE structures? The former is a part of an APPLICATION or BUSINESS SYSTEM; the later is for the entire business. Thus two more TYPES are:

APPL - This defines an APPLICATION. It has one each of:

1. LOGICAL DATA MODEL (LDSD).
2. DATA FLOW DIAGRAM (DFD)--yet to be defined.
3. FUNCTIONAL ORG CHART (FOC)--yet to be defined.
4. other organization documents defined by the user.

BUSS - This defines a BUSINESS. It has one or more of the following:

1. BUSINESS SYSTEM PLAN MODEL (BSPM). (Usually only one).
2. APPLICATION (APPL).

3. SUBJECT DATA BASE (SDB).

4. others not yet defined.

DATA FLOW DIAGRAM

This requires two parallel decompositions. One is for the actual data flows and the other is for the processes. The data flow will be documented in the CATEGORY entity. The process will be documented in the PROCEDURE entity (see below). The documentation for the process will describe how to link these two organizations together.

DFD - DATA FLOW DIAGRAM. The highest level for the data flow (or BUBBLE CHART) documentation within an APPLICATION (APPL).

DFP - DATA FLOW PIPE. The pipe that runs between the processes. This is a child of the DFD.

DFF - DATA FLOW FILE. A special pipe that is a data store; or, as Gane and Sarson state: "Data flows are data structures in motion; data stores are data structures at rest." This is a child of the DFD or of a DFDB.

DFDB - DATA FLOW DATA BASE. A special data store (DFF) that consists of many data stores. That is, a DFDB can have DFF's related to it. This is a child of the DFD.

DFE - DATA FLOW ELEMENT. A decomposition of the DFP, DFF, or DFDB. For DFF or DFDB one may choose to use the 'x' extender as is done for the LOGICAL DATA MODEL.

NOTE: The LOGICAL DATA MODEL is the outgrowth of the DATA FLOW structure. There will be many similarities.

PROCEDURE documents the programs, modules, and processes. The process half of the DATA FLOW DIAGRAM goes here. Also included are the FUNCTIONAL CHART as well as a description of the physical system as delivered to the user.

These attributes are available:

PROCEDURE - a twenty (20) character upper case alpha-numeric field that is the primary name of the entry.

PROCEDURE-PARENT - the name of a PROCEDURE which will serve as a parent in the relationship between two procedures.

PROCEDURE-CHILD - the name of a PROCEDURE that is the child in the relationship between two procedures.

PROCEDURE-NAME - a fifty (50) character upper case alpha-numeric long title for the PROCEDURE.

PROCEDURE-RESP - a twenty (20) character alpha-numeric field that is the name of the person, department, or other that is responsible for the information represented by the meta-data.

PROCEDURE-LANG - a ten (10) character upper case alpha-numeric field that holds the language code that the procedure is written in. These values need to be standardized across the HP environment.

PROCEDURE-TYPE - a four (4) character upper case alpha-numeric field used to indicate the classification of this PROCEDURE.

A standard set of PROCEDURE-TYPE values are:

BUSS - This defines a BUSINESS. The PROCEDURE name must match an identical CATEGORY name of the same type. It is used to hold those PROCEDURE entries together that belong to the business. There may be more that one business documented in DICTIONARY.

APPL - It has the same function for the PROCEDURE as in CATEGORY. There must be a common name of this type in each. There may be many of these for a given

business. Attached to this are the process half of the DFD, the FUNCTIONAL CHART, and the SYSTEM CHART.

DATA FLOW DIAGRAM

DFD - DATA FLOW DIAGRAM. The PROCEUDRE name for this entry must have a corresponding entry for a CATEGORY of like type.

DFPL - DATA FLOW PROCESS LEVEL. The PROCEDURE is the name of a process bubble. The first level DFPL are related to the DFD. Lower level DFPL are related to higher level DFPL.

DFIN - DATA FLOW INPUT. The PROCEDURE name is a name of a CATEGORY entry of type DFP, DFDB, or DFF. This names the data flow that is input to the DFPL to which this is related. The DFIN cannot have any children within the PROCEDURE entries. This is because it is defined within the CATEGORY section of DICTIONARY/3000.

DFOT - DATA FLOW OUTPUT. The PROCEDURE name is a name of a CATEGORY entry of type DFP, DFDB or DFF. This names the data flow that is output from the DFPL to which it is related. Like the DFIN, the DFOT cannot have any children.

DFIO - DATA FLOW INPUT/OUTPUT. The PROCEDURE name is the name of a CATEGORY entry of type DFP, DFDB or DFF. This names the data flow that passes both into and out of (named the same) the DFPL. As with the DFIN and DFOT, the DFIO cannot have any children.

DFEX - DATA FLOW EXTERNAL. This PROCEDURE is either a source or a sink or both. It is used in place of a DFPL when the PROCESS is in reality a SINK or SOURCE. This may have DFIN, DFOT, or DFIO (not likely) as children. They define the data flows to and from the external process (ie, SOURCE or SINK).

FUNCTIONAL CHART

The first level cut of the FUNCTIONAL CHART comes from transform and transaction analysis of the DFD. Thereafter, the designers modify it until the results are acceptable. There may be more than one FUNCTIONAL CHART for an APPLICATION.

FCHT - FUNCTIONAL CHART. The PROCEDURE is the name of the high level module (box) in the FUNCTIONAL CHART.

FCMD - FUNCTIONAL CHART MODULE. The PROCEDURE is the name of the module (any level box) within the FUNCTIONAL CHART. This is a child of the FCHT or another FCMD.

FCDN - FUNCTIONAL CHART DOWNFLOW. The PROCUDURE is the name of the data flow down to the FCMD of which this is a child. (Complicated sounding isn't it! Relax, if your FUNCTIONAL CHART doesn't show the data flow between module levels then don't use it.) This FCDN is a child of an FCMD. That FCMD receives data down from its parent. This FCDN is the name of that flow. Associated to this FCDN are ELEMENTS.

FCUP - FUNCTIONAL CHART UPFLOW. The PROCEDURE is the name of the data flow up from the FCMD of which this is a child. That is, this FCUP is a child of an FCMD. That FCMD sends data back to its parent. This FCUP is the name of that flow. Associated to this FCUP are ELEMENTS (use the ADD PROCEDURE command and give this PROCEDURE name when prompted).

THE SYSTEM

Whatever this is called, it is the finished product of jobstreams, programs, subroutines, etc. that is delivered to the user.

SYS - SYSTEM. This PROCEDURE is the name of the system. It is a child of the BUSS procedure.

SUBS - SUB-SYSTEM. This PROCEDURE is the name of one of the sub-systems within the system.

FCTN - FUNCTION. This PROCEDURE is a function within the SUBS.

JOBS - JOB STREAM. This PROCEDURE is a job stream within the FCTN.

PROG - PROGRAM. This PROCEDURE is a program within either the on-line portion of the FCTN (ie, FCTN procedure is the parent) or within the job stream (JOBS procedure is the parent).

SUBP - SUBPROGRAM. This PROCEDURE is a sub-program within a PROG. This is written and is compiled directly into the same USL as is the PROG.

SLRT - SL ROUTINE (non-system SL). This PROCEDURE is a sub-program that is placed into the account SL.

SLSY - SL SYSTEM ROUTINE. This PROCEDURE is a system level routine.

FILE - FILE. This PROCEDURE is a file (found in the FILE entity entries) used by the PROG, SUBP, or SLRT. The actual ELEMENT entries used by the parent will be associated with the appropriate file. If there is no file, then the elements will be associated with the PROG, SUBP, etc.

This last may seem like a duplication of what the FILE entity entries show. To some extent it is in that all entries here are in the FILE entries also. The difference is that the procedure does not use all the elements within a file. This allows the DBA to document which elements from what files are actually used. These are to be considered as the procedure's sub-schema.

DESCRIPTIONS

The DESCRIPTION entries for each entity, relation, and association have different uses.

The description entry made during the CREATE command of DICTDBM describes the entry in its general case. Documentation that applies across the board to the entry is to be entered here. This description is output during the LIST, DISPLAY, and REPORT commands of DICTDBM.

The description made during the RELATE command describes the relation between the two occurences of the entity. Anything specific to the relation is to be documented at that time.

The description made during the ADD command describes the association between the two different entities. Document the specific information that relates to this association.

This may take some experimentation to understand what is happening. Basically, each entry can have three (3) descriptions. Some can have more. For example, the FILE entity has the following:

CREATE FILE - this description gives information about the file in general. What its purpose is, ...

RELATE FILE - description gives information about the relation between the two files. Any special considerations when this data set is in this base (or this form is in this view file) are documented at this point.

ADD FILE - describes the association between the element and the file. Any special constraints for the element when it is in the file.

ADD FILE-LOC -describes the association between the file and the location.

ADD FILE-CLASS - and the documentation of the association of this file with that class.

SUMMARY

The DICTIONARY must become the interface between the many phases of the application life cycle. With the information internally maintained so that the graphic representation can be produced the DBA is able to utilize tools to insure a consistent description of the developing system.

Likewise, the production support people have at their fingertips the detailed information regarding the system from initial conception until final delivery. The run time manuals are stored in the DICTIONARY to be retrieved on-line as the need arises. The end user has access to some help facility to gain the information that normally would be hid on page x-y of the user manual. Described above is a mechanism to start down that road. With newer and better tools available, the long victorian novels

that concern Tom De Marco will become a thing of the past. The "STRUCTURED ANALYSIS and SYSTEM SPECIFICATION" (Tom De Marco) will become facility of the DICTIONARY. When the programmer is ready to code, all of the supporting information, formats, files, etc. will be in the DICTIONARY ready to be utilized.

My many thanks to members of the Data Administration unit who gave their input. Especially to Larry Rolstad who must do the same for the IBM dictionary.

BIBLIOGRAPHY

Date, CJ. **An Intorduction to Database Systems** Vol 1, 3rd ed. Addison-Wesley Publishing Co., 1981.

De Marco, Tom. **Structured Analysis and System Specification** Yourdon, 1979.

Dowling, Jim. "RAPID is a Relative Term" **MONTREAL PROCEEDINGS**, 1983.

Larson, Orland. "Software Prototyping: Today's Approach to Information Systems Design and Development", **MONTREAL PROCEEDINGS**, 1983.

Puckering, Gary, "Data Dictionaries--A New Era", **MONTREAL PROCEEDINGS**, 1983.

```
**** FIGURE SMBTXT-1 GOES ON THIS PAGE
**** FIGURE SMBTXT-2 GOES ON THIS PAGE
DICTIONARY/3000 Schema--List 1
$CONTROL BLOCKMAX=1024

BEGIN DATA BASE DICT;

PASSWORDS:
     1  HPPRGUSE;   <<HP programatic use password>>
     2  MANAGER;    <<MANAGER access--not realy total; but close>>
     3  PROGRAMR;   <<PROGRAMMER access>>
     4  INFORM;     <<INFORM group creator/maintainer>>
     5  DOCUMENT;   <<DOCUMENTATION access for the documentarian>>
     6  REPORT;     <<REPORT access for those who want reports>>
                    <<about DICTIONARY contents.  No relation >>
                    <<to usage of REPORT/3000.                >>

ITEMS:
     CATEGORY,              U20       (6/1,2,3,4,5);
     CATEGORY-CHILD,        U20       (6/1,2,3,4,5);
     CATEGORY-NAME,         U50       (6/1,2,3,4,5);
     CATEGORY-PARENT,       U20       (6/1,2,3,4,5);
     CATEGORY-RESP,         X20       (6/1,2,3,4,5);
     CATEGORY-TYPE,         U4        (6/1,2,3,4,5);
     CLASS,                 I1        (4,5,6/1,2,3);
     CLASS-NAME,            U50       (4,5,6/1,2,3);
     CLASS-PASSWORD,        X8        (3/1,2);
     CLASS-RESP,            X20       (4,5,6/1,2,3);
     CLASS-TYPE,            U4        (4,5,6/1,2,3);
     CONTROL-KEY,           I1        (6/1,2,3,4,5);
     CONTROL-OPTIONS,       U30       (6/1,2,3,4,5);
     DATE-CHANGE,           U6        (6/1,2,3,4,5);
     DATE-CREATE,           U6        (6/1,2,3,4,5);
     DESCRIPTION-KEY,       I2        (6/1,2,3,4,5);
     DESCRIPTION-LINE,      X50       (6/1,2,3,4,5);
     ELEMENT,               U20       (6/1,2,3,4,5);
     ELEMENT-ACCESS,        U2        (4,5,6/1,2,3);
     ELEMENT-ALIAS,         U20       (5,6/1,2,3,4);
     ELEMENT-ALIAS-C,       U60       (6/1,2,3,4,5);
```

```
        ELEMENT-ALIAS-P,    U30       (6/1,2,3,4,5);
        ELEMENT-CHILD,      U20       (4,5,6/1,2,3);
        ELEMENT-COUNT,      I1        (4,5,6/1,2,3);
        ELEMENT-DEC,        I1        (4,5,6/1,2,3);
        ELEMENT-DISPLAY,    I1        (5,6/1,2,3,4);
        ELEMENT-EDIT,       X30       (4,5,6/1,2,3);
        ELEMENT-ENTRY,      X30       (4,5,6/1,2,3);
        ELEMENT-HEADING,    X30       (4,5,6/1,2,3);
        ELEMENT-KEY,        I2        (6/1,2,3,4,5);
        ELEMENT-LENGTH,     I1        (4,5,6/1,2,3);
        ELEMENT-NAME,       U50       (4,5,6/1,2,3);
        ELEMENT-PARENT,     U20       (4,5,6/1,2,3);
        ELEMENT-POSITION,   I1        (4,5,6/1,2,3);
        ELEMENT-PRIMARY,    I1        (4,5,6/1,2,3);
        ELEMENT-RESP,       X20       (4,5,6/1,2,3);
        ELEMENT-SIZE,       I1        (4,5,6/1,2,3);
        ELEMENT-TYPE,       U2        (4,5,6/1,2,3);
        ELEMENT-UNITS,      X10       (4,5,6/1,2,3);
        FILE,               U20       (6/1,2,3,4,5);
        FILE-ACCESS,        U2        (4,5,6/1,2,3);
        FILE-ALIAS,         U8        (6/1,2,3,4,5);
        FILE-ALIAS-F,       U16       (4,5,6/1,2,3);
        FILE-BLOCK,         I1        (4,5,6/1,2,3);
        FILE-CHILD,         U20       (4,5,6/1,2,3);
        FILE-KEY,           I2        (6/1,2,3,4,5);
        FILE-NAME,          U50       (4,5,6/1,2,3);
        FILE-PARENT,        U20       (4,5,6/1,2,3);
        FILE-PARENT-KEY,    I2        (5,6/1,2,3,4);
        FILE-RESP,          X20       (4,5,6/1,2,3);
        FILE-SIZE,          I2        (6/1,2,3,4,5);
        FILE-TYPE,          U4        (4,5,6/1,2,3);
        GROUP,              U20       (5,6/1,2,3,4);
        GROUP-CHILD,        U20       (5,6/1,2,3,4);
        GROUP-NAME,         U50       (5,6/1,2,3,4);
        GROUP-PARENT,       U20       (5,6/1,2,3,4);
        GROUP-RESP,         X20       (5,6/1,2,3,4);
        GROUP-TYPE,         U4        (5,6/1,2,3,4);
        IDENTITY-CHANGE,    U8        (6/1,2,3,4,5);
        IDENTITY-CREATE,    U8        (6/1,2,3,4,5);
        LINK-VALUE,         I1        (5,6/1,2,3,4);
        LOCATION,           U20       (6/1,2,3,4,5);
        LOCATION-ACCOUNT,   U8        (6/1,2,3,4,5);
        LOCATION-CPU,       U8        (6/1,2,3,4,5);
        LOCATION-GROUP,     U8        (6/1,2,3,4,5);
        LOCATION-NAME,      U50       (6/1,2,3,4,5);
        POSITION,           K2        (6/1,2,3,4,5);
        PROCEDURE,          U20       (6/1,2,3,4,5);
        PROCEDURE-ALIAS,    U8        (6/1,2,3,4,5);
        PROCEDURE-CHILD,    U20       (6/1,2,3,4,5);
        PROCEDURE-LANG,     U10       (6/1,2,3,4,5);
        PROCEDURE-NAME,     U50       (6/1,2,3,4,5);
        PROCEDURE-PARENT,   U20       (6/1,2,3,4,5);
        PROCEDURE-RESP,     X20       (6/1,2,3,4,5);
        PROCEDURE-TYPE,     U4        (6/1,2,3,4,5);
        REPORT,             U6        (/1);
        REPORT-LOC,         U20       (/1);
        REPORT-NAME,        U50       (/1);

SETS:

NAME:   DATA-ELEMENT,     MANUAL    (4,5,6/1,2,3);
ENTRY:  ELEMENT           (8),
        ELEMENT-NAME,
        ELEMENT-TYPE,
        ELEMENT-SIZE,
        ELEMENT-DEC,
        ELEMENT-LENGTH,
```

```
          ELEMENT-COUNT,
          ELEMENT-UNITS,
          ELEMENT-RESP,
          ELEMENT-HEADING,
          ELEMENT-ENTRY,
          ELEMENT-EDIT,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 303;

    NAME:   DATA-FILE,        MANUAL    (4,5,6/1,2,3);
    ENTRY:  FILE              (6),
          FILE-NAME,
          FILE-TYPE,
          FILE-RESP,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 151;

    NAME:   DATA-PROCEDURE,   MANUAL    (6/1,2,3,4,5);
    ENTRY:  PROCEDURE         (4),
          PROCEDURE-LANG,
          PROCEDURE-NAME,
          PROCEDURE-RESP,
          PROCEDURE-TYPE,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 11;

    NAME:   DATA-CATEGORY,    MANUAL    (6/1,2,3,4,5);
    ENTRY:  CATEGORY          (3),
          CATEGORY-NAME,
          CATEGORY-TYPE,
          CATEGORY-RESP,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 101;

    NAME:   DATA-GROUP,       MANUAL    (5,6/1,2,3,4);
    ENTRY:  GROUP             (3),
          GROUP-NAME,
          GROUP-TYPE,
          GROUP-RESP,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 11;

    NAME:   DATA-CLASS,       MANUAL    (3,4,5,6/1,2);
    ENTRY:  CLASS             (2),
          CLASS-NAME,
          CLASS-TYPE,
          CLASS-PASSWORD,
```

```
          CLASS-RESP,
          FILE-KEY,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 67;

    NAME:   DATA-LOCATION,   MANUAL    (6/1,2,3,4,5);
    ENTRY:  LOCATION        (2),
          LOCATION-NAME,
          LOCATION-GROUP,
          LOCATION-ACCOUNT,
          LOCATION-CPU,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY;
    CAPACITY: 11;

    NAME:   DIC-CONTROL,     MANUAL    (6/1,2,3,4,5);
    ENTRY:  CONTROL-KEY     (0),
          DESCRIPTION-KEY,
          FILE-KEY,
          ELEMENT-KEY,
          CONTROL-OPTIONS,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE;
    CAPACITY: 1;

    NAME:   LINK-FILE,       AUTOMATIC (6/1,2,3,4,5);
    ENTRY:  FILE-KEY        (1);
    CAPACITY: 101;

    NAME:   LINK-ELEMENT,    AUTOMATIC (6/1,2,3,4,5);
    ENTRY:  ELEMENT-KEY     (1);
    CAPACITY: 79;

    NAME:   LINK-DESCRIPTION,AUTOMATIC (6/1,2,3,4,5);
    ENTRY:  DESCRIPTION-KEY (1);
    CAPACITY: 611;

    NAME:   DATA-REPORTLOC,  AUTOMATIC (/1);
    ENTRY:  REPORT-LOC      (1);
    CAPACITY: 11;

    NAME:   ELEMENT-ELEMENT, DETAIL    (4,5,6/1,2,3);
    ENTRY:  ELEMENT-PARENT  ( DATA-ELEMENT    (POSITION        )),
          ELEMENT-CHILD   (!DATA-ELEMENT    ),
          ELEMENT-POSITION,
          DATE-CREATE,
          DATE-CHANGE,
          IDENTITY-CREATE,
          IDENTITY-CHANGE,
          DESCRIPTION-KEY,
          POSITION;
    CAPACITY: 11;

    NAME:   FILE-FILE,       DETAIL    (4,5,6/1,2,3);
    ENTRY:  FILE-PARENT     ( DATA-FILE       (POSITION        )),
          FILE-CHILD      (!DATA-FILE       ),
          FILE-ALIAS-F,
          FILE-SIZE,
```

```
        FILE-BLOCK,
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 150;

NAME:   PROCEDURE-PROCED,DETAIL    (6/1,2,3,4,5);
ENTRY:  PROCEDURE-PARENT( DATA-PROCEDURE  (POSITION        )),
        PROCEDURE-CHILD (!DATA-PROCEDURE  ),
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 11;

NAME:   CATEGORY-CATEGOR,DETAIL    (6/1,2,3,4,5);
ENTRY:  CATEGORY-PARENT ( DATA-CATEGORY   (POSITION        )),
        CATEGORY-CHILD  (!DATA-CATEGORY   ),
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 150;

NAME:   GROUP-GROUP,     DETAIL    (5,6/1,2,3,4);
ENTRY:  GROUP-PARENT    ( DATA-GROUP      (POSITION        )),
        GROUP-CHILD     (!DATA-GROUP      ),
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 11;

NAME:   FILE-ELEMENT,    DETAIL    (4,5,6/1,2,3);
ENTRY:  FILE             ( DATA-FILE       (POSITION        )),
        ELEMENT          (!DATA-ELEMENT    ),
        ELEMENT-ALIAS,
        FILE-KEY,
        ELEMENT-KEY,
        ELEMENT-PRIMARY,
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 600;

NAME:   FILE-PATH,       DETAIL    (6/1,2,3,4,5);
ENTRY:  FILE             (!DATA-FILE       ),
        FILE-KEY         ( LINK-FILE       );
CAPACITY: 100;

NAME:   FILE-SORT,       DETAIL    (6/1,2,3,4,5);
ENTRY:  ELEMENT          (!DATA-ELEMENT    ),
        ELEMENT-KEY      ( LINK-ELEMENT    );
CAPACITY: 80;
```

```
NAME:    PROCEDURE-ELEMEN,DETAIL    (6/1,2,3,4,5);
ENTRY:   PROCEDURE        ( DATA-PROCEDURE  (POSITION        )),
         ELEMENT          (!DATA-ELEMENT    ),
         ELEMENT-ALIAS-P,
         DATE-CREATE,
         DATE-CHANGE,
         IDENTITY-CREATE,
         IDENTITY-CHANGE,
         DESCRIPTION-KEY,
         POSITION;
CAPACITY: 11;

NAME:    CATEGORY-ELEMENT,DETAIL    (6/1,2,3,4,5);
ENTRY:   CATEGORY         ( DATA-CATEGORY   (POSITION        )),
         ELEMENT          (!DATA-ELEMENT    ),
         ELEMENT-ALIAS-C,
         DATE-CREATE,
         DATE-CHANGE,
         IDENTITY-CREATE,
         IDENTITY-CHANGE,
         DESCRIPTION-KEY,
         POSITION;
CAPACITY: 200;

NAME:    GROUP-ELEMENT,   DETAIL    (5,6/1,2,3,4);
ENTRY:   GROUP            ( DATA-GROUP      (POSITION        )),
         ELEMENT          (!DATA-ELEMENT    ),
         ELEMENT-ALIAS,
         FILE-KEY,
         FILE-PARENT-KEY,
         LINK-VALUE,
         ELEMENT-DISPLAY,
         DATE-CREATE,
         DATE-CHANGE,
         IDENTITY-CREATE,
         IDENTITY-CHANGE,
         DESCRIPTION-KEY,
         POSITION;
CAPACITY: 11;

NAME:    CLASS-ELEMENT,   DETAIL    (4,5,6/1,2,3);
ENTRY:   CLASS            ( DATA-CLASS      (POSITION        )),
         ELEMENT          (!DATA-ELEMENT    ),
         ELEMENT-ACCESS,
         DATE-CREATE,
         DATE-CHANGE,
         IDENTITY-CREATE,
         IDENTITY-CHANGE,
         DESCRIPTION-KEY,
         POSITION;
CAPACITY: 800;

NAME:    CLASS-FILE,      DETAIL    (4,5,6/1,2,3);
ENTRY:   CLASS            ( DATA-CLASS      (POSITION        )),
         FILE             (!DATA-FILE       ),
         FILE-ACCESS,
         DATE-CREATE,
         DATE-CHANGE,
         IDENTITY-CREATE,
         IDENTITY-CHANGE,
         DESCRIPTION-KEY,
         POSITION;
CAPACITY: 11;

NAME:    FILE-LOCATION,   DETAIL    (6/1,2,3,4,5);
ENTRY:   LOCATION         ( DATA-LOCATION   (POSITION        )),
         FILE             (!DATA-FILE       ),
```

```
        FILE-ALIAS,
        FILE-SIZE,
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 11;

NAME:   PROCEDURE-LOCATI,DETAIL    (6/1,2,3,4,5);
ENTRY:  LOCATION        ( DATA-LOCATION   (POSITION        )),
        PROCEDURE       (!DATA-PROCEDURE  ),
        PROCEDURE-ALIAS,
        DATE-CREATE,
        DATE-CHANGE,
        IDENTITY-CREATE,
        IDENTITY-CHANGE,
        DESCRIPTION-KEY,
        POSITION;
CAPACITY: 11;

NAME:   DESCRIPTION-TEXT,DETAIL    (6/1,2,3,4,5);
ENTRY:  DESCRIPTION-KEY (!LINK-DESCRIPTION(POSITION        )),
        DESCRIPTION-LINE,
        POSITION;
CAPACITY: 1000;

NAME:   REPORT-LIST,     DETAIL    (6/1,2,3,4,5);
ENTRY:  REPORT-LOC      (!DATA-REPORTLOC  (REPORT          )),
        REPORT,
        REPORT-NAME,
        DATE-CREATE,
        IDENTITY-CREATE;
CAPACITY: 11;

END.
```

Hub of Systems Development and Documentation

Stephen M. Butler Weyerhaeuser Company

BIOGRAPHY

Stephen M. Butler is the Sr. Technical Specialist for the HP3000 within the I/S Data Administration unit of Weyerhaeuser Company. His area of expertise for the company is with IMAGE and the RAPID products. He has been with Weyerhaeuser since July, 1981.

Steve has six (6) years of hospital application development experience using both a Honeywell 115 (1 yr) and the HP 3000 (5 yrs). He was the Director of Data Processing during the last three (3) of those years.

In 1975, he obtained a BS degree in chemistry with very strong minors in mathematics and physics.

Steve is married and has three (3) children between the ages of two (last October) and five (last March).
